การใช้ไมโครคอนโทรลเลอร์กับที่อยู่อาศัย

Microcontroller application in habitats

โดย

นาย ชูชัย ศรีศันสนีย 321080

นาย พรชัย อาชีวะพนิช 321202

นาย สุทธิศักดิ์ หวังพัฒนศิริกุล 321378

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2535

ปริญญานิพนธ์ปีการศึกษา 2535

ภาควิชา วิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประยุกต์ใช้ไมโครคอนโทรลเลอร์กับที่อยู่อาศัย

ผู้จัดทำ

นาย ชูชัย	ศรีคันสนีย	321080
นาย พรชัย	อาชีวะพนิช	321202
นาย สุทธิศักดิ์	หวังพัฒนศิริกุล	321378

อ. เกียรติวรรณ ทรงสัตย์ อาจารย์ที่ปรึกษา

(.....)

การประยุกต์ใช้ไมโครคอนโทรลเลอร์กับที่อยู่อาศัย

นาย ชูชัย ศรีศันสนีย 321080
นาย พรชัย อาชีวะพนิช 321202
นาย สุทธิศักดิ์ หวังพัฒนศิริกุล 321378

อาจารย์ที่ปรึกษา
อาจารย์เกียรติวรรณ ทรงสัตย์

บทคัดย่อ

ความเสียหายของเครื่องมือ และอุปกรณ์ไฟฟ้า มีสาเหตุมาจากความผิดปกติของระบบไฟฟ้า ซึ่งเกิดจากความบกพร่องของอุปกรณ์ไฟฟ้า หรือแหล่งจ่ายไฟฟ้า จะทำให้เกิดความเสียหายต่างๆ ได้แก่ อายุการใช้งานของอุปกรณ์สั้นลง การเสื่อมสภาพของฉนวนไฟฟ้า อันเป็นสาเหตุให้เกิดอัคคีภัย เป็นต้น จึงมีการออกแบบป้องกันเครื่องมือและอุปกรณ์ไฟฟ้า ซึ่งประกอบด้วยฟิวส์ และเซอร์กิตเบรกเกอร์

ฟิวส์ และเซอร์กิตเบรกเกอร์ จะทำหน้าที่เปิดวงจรออก เพื่อป้องกันอันตรายและความเสียหาย ที่จะเกิดกับเครื่องมือ และอุปกรณ์ที่ต่ออยู่กับเซอร์กิตเบรกเกอร์และฟิวส์นั้น ในการนำเอาฟิวส์ และเซอร์กิตเบรกเกอร์มาใช้งาน มีเงื่อนไขที่สำคัญ คือ ฟิวส์หรือเซอร์กิตเบรกเกอร์ จะทำการเปิดวงจรเฉพาะส่วนที่ผิดปกติเท่านั้น ในเครื่องมือ และอุปกรณ์ไฟฟ้าอย่างอื่นยังคงทำงานได้ปกติ ในการทำงานของฟิวส์และเซอร์กิตเบรกเกอร์ ไม่มีหน้าที่อื่นนอกจากการเปิดวงจรเมื่อเกิดความผิดปกติขึ้น ดังนั้น หากการทำงานของอุปกรณ์ป้องกัน สามารถโปรแกรมหน้าที่การทำงานได้ ก็สามารถทำการเปิด และปิดวงจรได้ตามที่ต้องการ โดยการโปรแกรมหน้าที่การทำงานที่ต้องการลงไป ซึ่งจะสร้างความยืดหยุ่นของการทำงานของระบบได้ดีขึ้น หากมีการขยายเพิ่มเติมระบบไฟฟ้า ก็สามารถต่อเติมเพิ่มขยายได้โดยง่าย เพียงทำการแก้ไขเพิ่มเติมโปรแกรมการทำงาน ให้สอดคล้องกับความต้องการของระบบ ที่ขยายออกไป

Microcontroller application in habitats

Mr. Chuchai srisansanee 321080

Mr. Pornchai arechewapanit 321202

Mr. Sutthisak wangpattanasirikul 321378

Advisor

Mr.Kiettiwan songsataya

Abstract

The damage of electrical devices has caused from disoder of electrical system emerged from blemish of electrical devices or electrical-supply source. It makes much defect as decreasing in the age of electrical devices or deteriorating in electrical insulators. Principally it cause burning ; hence , protections are designed to devices as fuses , circuit breakers.

Fuses and circuit breakers will open circuits to protect damage that effect devices connected to circuit breakers and fuses. To use fuses and circuit breakers , there are some vital conditions that either fuses or circuit breakers are to open circuits only disoder parts while others continue its functions. In operation of both fuses and circuit breakers no other functions are done except openning circuits when it has defect. Consequently if operations of devices can program functions , it can open circuit ,or close circuir by rogramming function needed to devices. So it make system performance flexible . If having expands electrical systems , it will easily adjust by changing programs to correspond with demands of expanded system.

กิตติกรรมประกาศ

ขอขอบคุณ อาจารย์ เกียรติวรรณ ทรงสัตย์ เป็นอย่างยิ่ง ที่ได้กรุณาให้คำแนะนำ และคำปรึกษา ตลอดจนการแก้ปัญหาขั้นต้น ขอขอบคุณพี่ประเมษฐ์ ประนยานันทน์ ที่ให้คำปรึกษาทางด้านโปรแกรม พี่อนุตรชัยที่แนะนำการใช้ซอฟแวร์ที่สนับสนุนการทำงานในครั้งนี้ และพี่เทพจิตร์ที่อำนวยความสะดวกในการทำงาน

ขอขอบคุณ กฤษดา ที่ให้ใช้คอมพิวเตอร์

สารบัญ

สารบัญภาพ

สารบัญภาพ

การป้องกันระบบไฟฟ้าและอุบกรณ์ที่ใช้

ในการออกแบบระบบไฟฟ้านั้น สิ่งที่ต้องการก็คือต้องการให้ระบบสามารถที่จะทำงานได้อย่างต่อเนื่อง โดยจะต้องมีพลังงานไฟฟ้าพร้อม ที่จะจ่ายให้แก่เครื่องมือและอุบกรณ์ทางไฟฟ้าได้ตลอดเวลา ดังนั้นความเชื่อถือจึงเป็นสิ่งสำคัญที่สุด ถ้าในระบบเกิดกระแสที่มีขนาดมากกว่าปกติ หรือ กระแสเกิน (over current) ไหลผ่านแล้วย่อมจะเกิดอันตรายมาก กระแสนี้อาจเกิดจากหลายสาเหตุด้วยกัน และอาจมีค่ากระแสมากกว่า 100 kA ก็ได้

แต่ที่นิยมแบ่งสาเหตุของกระแสเกินคือ

1.โหลดเกิน (overload)

2.ลัดวงจร (short circuit)

ดังนั้น เมื่อมีกระแสเกินเกิดขึ้นในวงจร จะต้องมีอุบกรณ์ที่จะเปิดวงจรออก แต่ต้องเปิดวงจรเฉพาะ วงจรที่จำเป็น และ น้อยที่สุด แต่มีความปลอดภัยมากที่สุด เพื่อที่จะให้อุบกรณ์ ที่ไม่เกี่ยวข้องกับ การเกิดกระแสเกินมีพลังงานไฟฟ้าใช้ตลอดเวลา ในกรณีที่มีความผิดปกติของแรงดันไฟฟ้า ก็นำมาซึ่งความเสียหายต่อเครื่องมือและอุบกรณ์ทางไฟฟ้าได้ โดยเฉพาะเครื่องมือและอุบกรณ์ ที่ประกอบด้วยมอเตอร์ไฟฟ้า ลักษณะของความผิดปกติของแรงดันไฟฟ้า ก็คือ ไฟฟ้าตก ไฟฟ้าเกิน ดังนั้น เมื่อเกิดความผิดปกติของแรงดันไฟฟ้าขึ้นในวงจร จึงจำเป็นต้องมีอุบกรณ์ที่จะเปิดวงจรเฉพาะอุบกรณ์และเครื่องมือที่จะเกิดความเสียหายจากความผิดปกติของแรงดันไฟฟ้า

<u>โหลดเกิน</u>

โหลดเกิน หมายถึง กระแสที่มีขนาดมากกว่ากระแสปกติ และถ้ากระแสไฟฟ้าไหลผ่านในวงจรไฟฟ้านานๆ ย่อมจะทำให้เกิดความร้อนขึ้นอย่างมาก ซึ่งจะเป็นอันตรายต่อวงจรไฟฟ้า ในกรณีของโหลดเกินนั้น อาจเกิดจากหลายสาเหตุด้วยกัน เช่น ในกรณีมอเตอร์ ถ้าแบริ่งของมอเตอร์ไม่ได้รับการหล่อลื่น และยังให้มอเตอร์ทำงานอยู่ ย่อมจะทำให้เกิดความร้อนขึ้นบนแกน(shaft) เพลาของมอเตอร์ ซึ่งส่งผลตามมา คือความเร็วของมอเตอร์จะไม่ได้ตามต้องการ และอาจหยุดหมุนที่สุด ในกรณีที่

มอเตอร์ ต้องพยายามให้มีความเร็วเท่ากับกรณีปกติ ทำให้ต้องรับกระแสมากเกินขนาด ดังนั้นย่อมทำให้เป็นอันตรายต่อมอเตอร์ได้ถ้าใช้เป็นเวลานานๆ จึงต้องมีอุปกรณ์มาตัดวงจรเมื่อมีกระแสเกินที่เกิดจากโอเวอร์โหลดนี้ไหลผ่าน อุปกรณ์นี้ คือ อุปกรณ์ป้องกันกระแสเกิน (over current protective device)

โดยทั่วไป กระแสเกินจะมีขนาดไม่เกิน 5-6 เท่าของกระแสปกติ เพราะว่าถ้าเกินมากกว่านี้ ก็อาจกล่าวได้ว่าเป็นกระแสที่เกิดจากการลัดวงจร

การลัดวงจร

การลัดวงจร หมายถึง กระแสที่แตกต่างจากกระแสปกติ ซึ่งบางกรณีอาจจะมีขนาดไม่มากกว่ากระแสโหลดก็ได้ แต่โดยปกติจะมีขนาดเป็นพันเท่าของกระแสปกติ ดังนั้นจึงพอกล่าวได้ว่า สาเหตุใหญ่ที่ทำให้เกิดการลัดวงจร คือ เกิดจากฉนวนเบรกดาวน์ (breakdown) และเมื่อเกิดการลัดวงจรขึ้น จะมีผลตามมาคือ

การอาร์ค

ผลของการเกิดผิดปกตินี้ จะเกิดอย่างเร็วมาก ระยะความยาวของตัวนำอาจจะหลอมละลาย และ กระจายออกไปยังพื้นที่รอบๆ นอกจากนี้ในบางกรณีฉนวนที่หุ้มสายตัวนำอยู่อาจเกิดการลุกไหม้ขึ้นได้

เกิดความร้อน

เนื่องจากการลัดวงจร ย่อมมีกระแสที่มีขนาดสูงเกิดขึ้น ซึ่งจะทำให้เกิดความร้อนขึ้นในตัวนำนั้นถ้ามีกระแสไหลผ่านไป เช่น กระแสลัดวงจรขนาด 15 kA ไหลผ่านตัวนำเบอร์ 6 ซึ่งเป็นทองแดง จะมีอุณหภูมิเพิ่มขึ้นมากกว่า 400F โดยใช้เวลาเพียง 1 ไซเคิลเท่านั้น

เกิดความเค้นเนื่องจากสนามแม่เหล็ก

เพราะว่าสนามแม่เหล็ก จะเกิดขึ้นรอบๆตัวนำเมื่อมีกระแสไหลผ่านตัวนำนั้น ดังนั้นในกรณีที่มีกระแสลัดวงจร ซึ่งมีขนาดเป็นพันๆแอมแปร์ไหลผ่าน ก็จะเกิดสนามแม่เหล็กที่มีขนาดขยายเพิ่มขึ้น ซึ่งจะทำให้เกิดความเค้นขึ้นเมื่อสนามมีขนาดมากขึ้น

แรงขนาดนี้ย่อมทำให้บัสบาร์เกิดแรง ที่จะทำให้สามารถหลุดจากการยึดได้ซึ่ง

ถ้าตัวจับยึดเสียหาย ก็จะทำให้บัสบาร์อาจเกิดการลัดวงจรกัน หรือ ลงกราวน์ก็ได้ ผลอันนี้ทำให้เป็นปัญหาต่อการลัดวงจรทั้งสิ้น

เงื่อนไขในกรณีที่เกิดการลัดวงจรขึ้นในวงจรไฟฟ้า

เงื่อนไขในกรณีที่เกิดการลัดวงจรขึ้นนั้น จะแตกต่างจากกรณีปกติ เช่น ในระบบไฟกระแสสลับในกรณีที่เป็นสภาวะปกติ กระแสที่ไหลผ่านวงจรจะเป็นสัดส่วนกับแรงดันที่จ่ายให้แก่วงจรหรือระบบไฟฟ้านั้น และขึ้นอยู่กับค่าอิมพีแดนซ์ของโหลดในวงจรนั้น แต่เมื่อเกิดการลัดวงจรขึ้นค่าอิมพีแดนซ์จะมีค่าลดลง โดยปกติค่าอิมพีแดนซ์จะเป็นค่าจากแหล่งแรงดันไปยังจุดที่เกิดการผิดปกติขึ้น และค่าอิมพีแดนซ์ของหม้อแปลงก็จะรวมอยู่ในวงจรที่เกิดการผิดปกติด้วย เมื่อค่าอิมพีแดนซ์มีค่าลดลง ก็จะทำให้กระแสมีขนาดสูงขึ้น

การหาค่ากระแสลัดวงจร

กระแสลัดวงจรที่เกิดขึ้นไม่ได้เกิดจากจุดที่เกิดการผิดปกติ แต่เกิดจากทุกๆอุปกรณ์ในวงจรไฟฟ้านั้น ซึ่งจะเป็นตัวที่ทำให้เกิดกระแสลัดวงจรขึ้น อุปกรณ์ต่างๆนี้อาจถือว่าเป็นแหล่งจ่ายแรงดันก็ย่อมได้ ได้แก่

1.ระบบสร้างแหล่งจ่ายกำลังไฟฟ้าซึ่งถือว่าเป็นแหล่งที่จ่ายกำลังไฟฟ้ามากสุด

2.มอเตอร์ที่เป็นแบบเหนี่ยวนำ

3.เครื่องกำเนิดไฟฟ้าที่นำมาใช้เป็นตัวจ่ายระบบแหล่งจ่ายกำลังไฟฟ้า

4.ซิงโครนัสมอเตอร์

5.ซิงโครนัสคอนเดนเซอร์

นอกจากนี้กระแสผิดปกติ ที่เกิดจากระบบแหล่งจ่ายกำลังไฟฟ้าจะถือว่ามีขนาดไม่จำกัด ซึ่งถือว่าเป็นเงื่อนไขที่ไม่ดี (โดยปกติกระแสลัดวงจรที่เกิดจากระบบแหล่งจ่ายกำลังไฟฟ้านั้น ควรที่จะรู้ค่าของระบบว่ามีขนาดเท่าใด

ในช่วงเวลาไซเคิลแรก ของการเกิดผิดปกติขึ้นนั้น กระแสที่เกิดมาจากมอเตอร์แบบเหนี่ยวนำไม่สามารถที่จะตัดทิ้งได้ เพราะเนื่องจากเมื่อเกิดการผิดปกติขึ้นในระบบนั้น มอเตอร์เหนี่ยวนำจะยังคงหมุนไปได้ด้วยแรงเฉื่อยของโหลด ในระยะ

เวลาช่วงนี้จะคล้ายๆกับว่ามอเตอร์แบบเหนี่ยวนำ ได้กลายเป็นเครื่องกำเนิดไฟฟ้า ดังนั้นเอาพุทที่เกิดจากมอเตอร์แบบเหนี่ยวนำนี้ จะอยู่ในช่วงระยะเวลาอันสั้น ประมาณไซเคิลแรกหรือไซเคิลที่สองเท่านั้น และกระแสที่เกิดขึ้นนี้เรียกว่า กระแสลัดวงจรที่มีอิทธิพลของมอเตอร์สนับสนุน

<u>การลดค่ากระแสลัดวงจร</u>

การที่จะลดกระแสลัดวงจร ทำได้โดยลดขนาดหม้อแปลง และเพิ่มระยะห่างของโหลดจากหม้อแปลงให้มากขึ้น เช่น หม้อแปลงขนาด 750 kVA 480 V พร้อมกับมอเตอร์ที่มีโหลด 100 % และกำลังไฟฟ้า ทางด้านแหล่งจ่ายไม่จำกัดจำนวนระยะทางจากหม้อแปลงไปยังโหลด

<u>อุปกรณ์ป้องกันกระแสเกิน</u>

ลักษณะของการเกิดกระแสเกิน หรือ การลัดวงจรที่เกิดขึ้นในระบบไฟฟ้านั้น จะต้องมีอุปกรณ์ที่จะป้องกันอันตรายจากกรณีกระแสเกินและลัดวงจรได้ อุปกรณ์เหล่านี้จะเรียกว่าอุปกรณ์ป้องกันกระแสเกิน ตามหลักการแล้วอุปกรณ์นี้จะต้องสามารถป้องกันตัวนำและอุปกรณ์ไฟฟ้าต่างๆ โดยจะต้องเปิดวงจรเมื่อมีกรณีที่กระแสไหลผ่านตัวมันมากกว่าค่าที่ตั้งไว้ ซึ่งกระแสนี้ถ้าปล่อยไว้จะทำให้เกิดอันตราย โดยทำให้เกิดความร้อนในสายตัวนำหรือฉนวนได้

อุปกรณ์ป้องกันกระแสเกินควรที่จะมีหลักการข้อกำหนดในการทำงานดังนี้

1.การทำงานควรเป็นไปอย่างยั่งยืน

2.ในกรณีที่กระแสปกติไหลผ่านตัวมัน ไม่ควรจะมีการเปิดวงจร

3.จะต้องเปิดวงจรทันทีถ้ามีการเกิดกระแสเกินเกิดขึ้นและไหลผ่านตัวมัน

4.จะต้องสามารถเปลี่ยนหรือปรับได้สะดวก

5.จะต้องมีความปลอดภัยในการใช้งาน ทั้งในกรณีปกติและกรณีกระแสเกินไหลผ่านตัวมัน

อุปกรณ์ที่นำมาใช้ในการป้องกันกระแสเกิน โดยทั่วๆไปก็มี

1.เซอร์กิตเบรกเกอร์

2.ฟิวส์

นอกจากนี้ยังมีอุปกรณ์อื่นๆอีก ซึ่งสามารถนำมาใช้ร่วมกับอุปกรณ์ข้างบนก็ได้

<u>เซอร์กิตเบรกเกอร์</u>

เซอร์กิตเบรกเกอร์สำหรับระบบไฟฟ้าที่สูงกว่า 600 V จะสามารถแบ่งออกได้เป็น 4 พวกใหญ่ๆ คือ

1.เซอร์กิตเบรกเกอร์แบบตัดวงจรระบบอากาศ (air breaker)

2.เซอร์กิตเบรกเกอร์แบบตัดวงจรระบบสุญญากาศ (vacum breaker)

3.เซอร์กิตเบรกเกอร์แบบตัดวงจรระบบน้ำมัน (oil breaker)

4.เซอร์กิตเบรกเกอร์แบบตัดวงจรระบบใช้แก๊สเติม (gas-filled breaker)

เซอร์กิตเบรกเกอร์เหล่านี้จะใช้ร่วมกับรีเลย์ป้องกัน เพื่อการทำงานเป็นไปอย่างอัตโนมัติ เพราะโดยปกติแล้วจะไม่มีอุปกรณ์ ที่มีความรู้สึกต่อการเกินของกระแสในตัวมันเองได้แต่จะมีเครื่องมือหรืออุปกรณ์บางอย่าง เป็นตัวทำให้เซอร์กิตเบรกเกอร์ได้ตัดวงจรออกไปเมื่ออยู่ในภาวะปกติ

<u>ฟิวส์สำหรับระบบที่มีแรงดันไฟฟ้ามากกว่า 600 V</u>

ในการใช้งานของฟิวส์ต่างๆนั้น จะมีฟิวส์กำลังที่ใช้กับงานที่มีระดับแรงดัน2400V หรือสูงกว่านี้ โดยสามารถแบ่งออกได้เป็น 4 ชนิดคือ

1.ฟิวส์กำลังชนิดจำกัดกระแส

2.ฟิวส์กำลังชนิดไม่จำกัดกระแส

3.ฟิวส์คัตเอาต์ชนิดน้ำมัน

4.ฟิวส์ที่ใช้ในระบบจำหน่ายไฟฟ้าเพื่อใช้เป็นฟิวส์คัตเอาต์

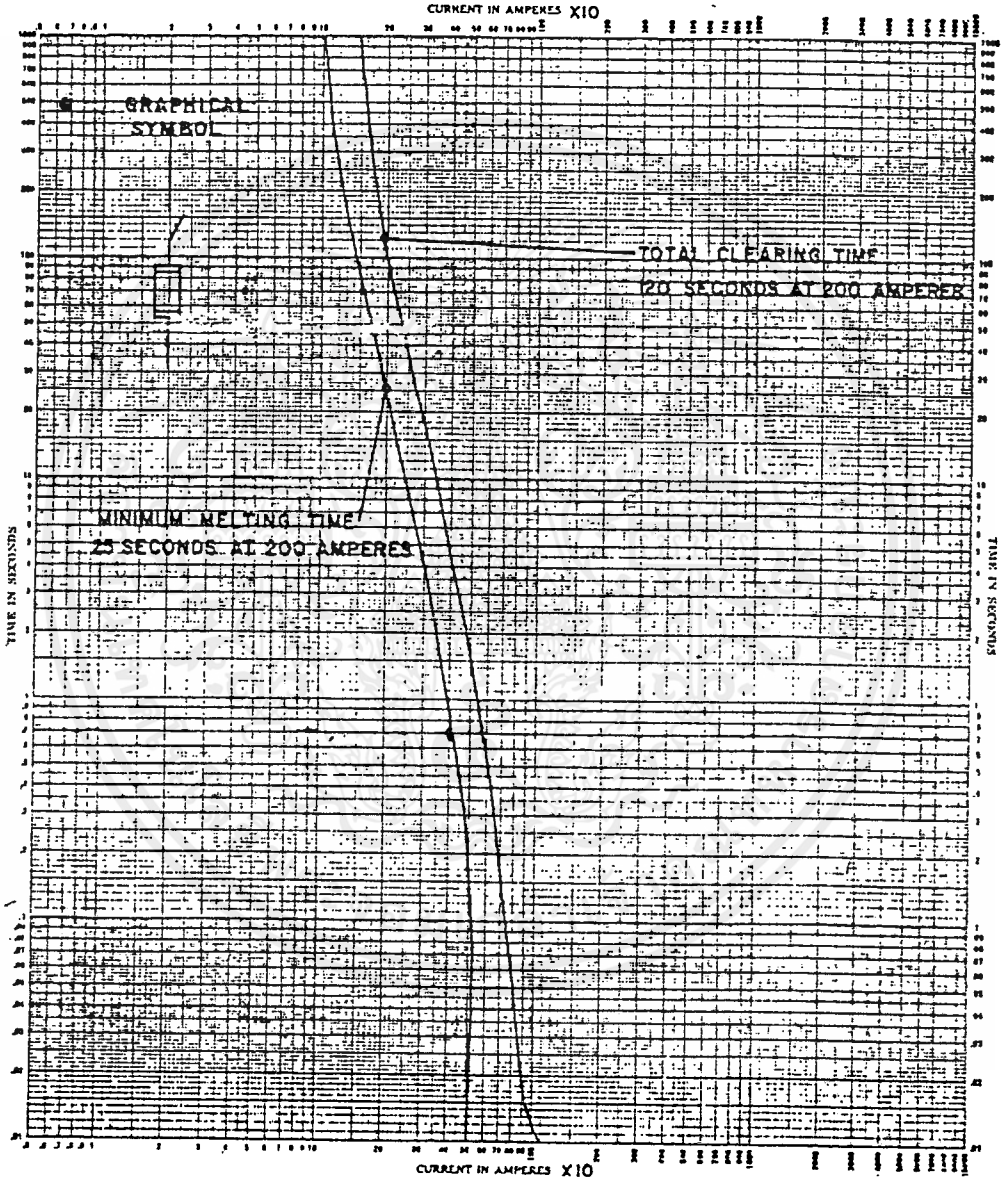
<u>ระบบไฟฟ้าที่มีแรงดันไฟฟ้าน้อยกว่า 600 V</u>

ระบบไฟฟ้าที่มีแรงดันไฟฟ้าน้อยกว่า 600 V สามารถจำแนกออกได้ดังนี้

<u>เซอร์กิตเบรกเกอร์</u>

สามารถจะจัดแบ่งได้เป็น 2 ชนิดใหญ่ๆ คือ

แผนภาพข้างล่าง จะบ่งบอกถึงเวลาในการเคลียร์กระแสที่ผ่านตัวฉน และค่าเวลาต่ำสุดในการหลอมตัวฉน



แต่กระแสไม่สูงมากเกินไปนัก นอกจากนี้ค่าการทนต่อกระแสลัดวงจรในขณะที่เปิดวงจรในขณะที่เปิดวงจรออกของ MCCB ก็จะมีค่าไม่สูงมากนัก กลไกทางของเซอร์กิตเบรกเกอร์แบบนี้จะบรรจุอยู่ในกล่องที่หล่อออกมาตามการใช้งาน และชนิดของเซอร์กิตเบรกเกอร์

2.1แบบอาศัยความร้อน (thermal trip action) โดยภายในเซอร์กิตเบรกเกอร์จะทำด้วยโลหะ 2 ชนิด ซึ่งขยายตัวเมื่อได้รับความร้อนไม่เท่ากัน เรียกว่า ไบแมตทอลลิกสทริบ (bimettallic strip)

2.2แบบอาศัยอำนาจแม่เหล็ก(magnetic trip action) แบบนี้อาศัยหลักการของการเกิดอำนาจแม่เหล็กในขดลวดที่พัน โดยมาจากหลักที่ว่าความเข้มของสนามแม่เหล็กจะเป็นสัดส่วนโดยตรงกับกระแสที่ไหลผ่าน

2.3แบบอาศัยความร้อนและอำนาจแม่เหล็ก (thermal-magnetic trip action) ส่วนมากเซอร์กิตเบรกเกอร์ทั่วๆไป จะเป็นลักษณะการทำงานแบบนี้ โดยเป็นการรวมการทำงานเปิดวงจรชนิดอาศัยความร้อน และ อำนาจแม่เหล็กเข้าด้วยกัน ทำให้เซอร์กิตเบรกเกอร์ชนิดนี้ สามารถป้องกันทั้งกรณีโหลดเกินและลัดวงจรได้

เซอร์กิตเบรกเกอร์บางชนิด โดยเฉพาะแบบที่มีกระแสพิกัดสูงๆมักจะมีอุปกรณ์พิเศษดังต่อไปนี้ คือ

1.อุปกรณ์เปิดวงจรแบบชนาน ใช้บังคับให้เซอร์กิตเบรกเกอร์ตัดวงจร จากที่ไกลหรือจากระบบควบคุมอื่น ซึ่งทำงานเมื่อมีการป้อนไฟขนาดแรงดันประมาณ 130-240 V

2.สวิตช์ช่วย เป็นหน้าสัมผัสที่ใช้ควบคุมการทำงานของระบบอื่น ให้สัมพันธ์กับการปิดและเปิดเซอร์กิตเบรกเกอร์

3.อุปกรณ์ป้องกันแรงดันไฟฟ้าตก วงจรนี้จะใช้กันมากในเซอร์กิตเบรกเกอร์ทั่วๆไป โดยคอยล์ขดลวดจะคอยล็อกกลไกในการทริบ โดยปกติคอยล์นี้จะถูกป้อนแรงดันไฟฟ้าตลอดเวลา ถ้าหากแรงดันคร่อมคอยล์ไม่มีหรือตกต่ำกว่าระดับแรงดันที่ตั้งไว้ เซอร์กิตเบรกเกอร์จะ "on" ไม่ได้และทริบทันที ฉะนั้นจึงกล่าวได้ว่าคอยล์นี้เป็นตัว

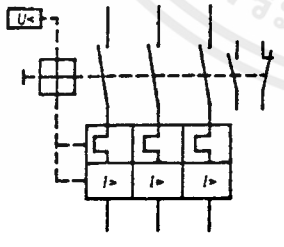
ป้องกันโหลดเกิน เพราะถ้าแรงดันตกลงมากๆ โหลดจะกินกระแสมาก ช่วงแรงดันที่ถือว่าเซอร์กิตเบรกเกอร์ต้องทริบตัดการจ่ายโหลดจะมีค่าประมาณ 20% หรือ 60% ของแรงดันในสาย

ปกติเซอร์กิตเบรกเกอร์แรงดันในสาย จะต้องมีค่าไม่ต่ำกว่า 80 % จึงจะสามารถ "on" ได้ วงจรทริบแบบอุปกรณ์ป้องกันแรงดันไฟฟ้าตกนี้ อาจจะใช้ไพสลับต่อโดยตรงเข้าคอยล์เลยก็ได้

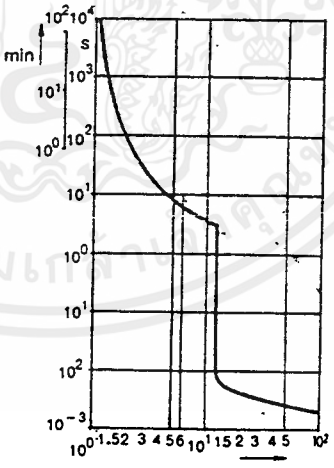
4.อุปกรณ์ป้องกันกระแสลงดิน จะทำงานร่วมกับอุปกรณ์ป้องกันการลัดวงจรลงดิน เพื่อให้เซอร์กิตเบรกเกอร์ตัดวงจรเมื่อเกิดการลัดวงจรลงดินเกิดขึ้น

5.สวิตช์เตือนภัย จะทำงานร่วมกับอุปกรณ์หรือวงจรอื่น เพื่อเตือนให้ทราบว่าเซอร์กิตเบรกเกอร์ตัดวงจรออกแล้ว

รูปข้างล่าง เป็นการแสดงคุณสมบัติของเซอร์กิตเบรกเกอร์ ที่มีการป้องกันโหลดเกินและการลัดวงจร



(ก) แผนภาพภายใน

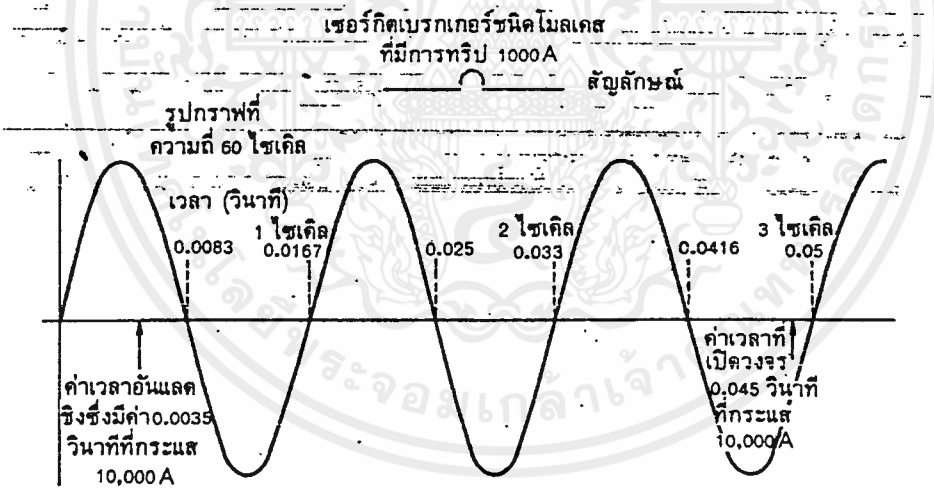


(ข) คุณสมบัติการเปิดวงจร

กระแสที่ทำให้เซอร์กิตเบรกเกอร์ทำงานหรือกระแสทริป ส่วนนี้จะหมายถึงค่าการทำงานของเซอร์กิตเบรกเกอร์ซึ่งจะสัมพันธ์กับกระแสลัดวงจรที่ไหลผ่านชุดคอยล์สนามแม่เหล็ก และสนามแม่เหล็กที่เกิดขึ้นในชุดคอยล์ ก็จะทำให้เปิดวงจรออกไป และค่านี้ปรับค่าได้

instantaneous opening เป็นค่าเวลาในการที่เซอร์กิตเบรกเกอร์จะเปิดวงจรโดยไม่มีการหน่วงเวลาเลย

อันแลตชิงไทม์ หมายถึงเมื่อมีกระแสผิดปกติเกิดขึ้น เซอร์กิตเบรกเกอร์พร้อมที่จะทำงาน จนถึงเวลาสิ้นสุดของอันแลตชิงไทม์นับ เซอร์กิตเบรกเกอร์ก็จะเริ่มทำงาน และเวลาต่อมานี้จะไม่สามารถ ที่จะหยุดการทำงานของเซอร์กิตเบรกเกอร์ได้ แต่ถ้าความผิดปกติกลับคืนสู่สภาวะปกติแล้ว ก็จะไม่มีการทำงานดังรูปข้างล่าง



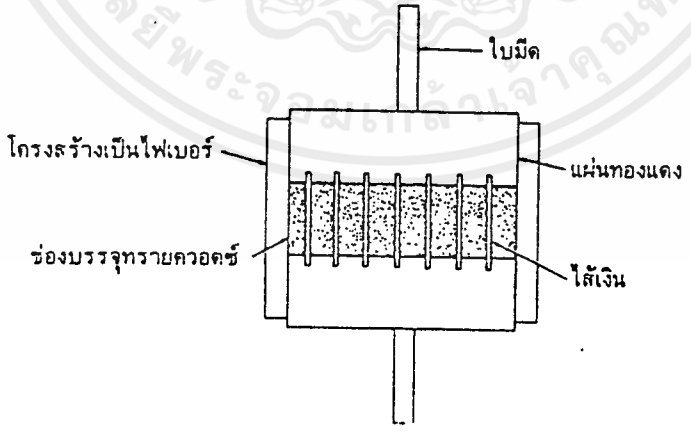
<u>ฟิวส์</u>

คืออุปกรณ์ป้องกันกระแสเกินชนิดหนึ่ง และนิยมนำไปใช้กันอย่างแพร่หลาย ซึ่งจะสามารถจำแนกออกได้เป็นลักษณะใหญ่ๆ ดังนี้

1.ฟิวส์ คืออุปกรณ์ในการป้องกันกระแสเกิน โดยจะตัดวงจรที่มีปัญหาออกจากระบบไฟฟ้าทั้งหมดได้ก็ต่อเมื่อมีกระแสไหลผ่านตัวมัน และกระแสที่ไหลผ่านนี้จะทำให้เกิดความร้อนขึ้นบนตัวฟิวส์ ซึ่งจะทำให้ฟิวส์ขาดและหลอมละลายได้

ในกรณีของการเกิดลัดวงจรขึ้นในระบบกระแสที่ไหลผ่านฟิวส์ จะทำให้ไส้ฟิวส์ขาดออกจากกันได้ ทั้งนี้เพราะกระแสที่เกิดขึ้นมีค่าสูงและใช้เวลาเพียงเล็กน้อย ในกรณีที่เป็นโหลดเกิน กระแสที่เกิดจากโหลดเกินจะไม่สามารถทำให้ไส้ฟิวส์ขาด หรือหลอมละลายได้ แต่ความร้อนที่เกิดจากกระแสโหลดเกิน ไหลผ่านตัวฟิวส์จะทำให้จุดรอยเชื่อมตรงบริเวณรอยต่อ เกิดการหลอมละลาย

ส่วนความเร็วในการเปิดวงจรนั้น เป็นสิ่งที่ต้องพิจารณาอย่างมาก ถ้าจะเปรียบเทียบความเร็วระหว่างฟิวส์มาตราฐานกับฟิวส์ชนิดจำกัดกระแสนั้น ปรากฏว่าฟิวส์แบบหลังจะสามารถเปิดวงจรได้ก่อนที่จะถึงจุดอันตราย แต่ฟิวส์มาตราฐานมีการใช้เวลาในการเปิดวงจรนานกว่า ซึ่งจะทำให้มีความร้อนและเกิดความดันขึ้นได้



1. เซอร์กิตเบรกเกอร์ชนิดอากาศ เซอร์กิตเบรกเกอร์ชนิดนี้โดยส่วนมากแล้วจะใช้สำหรับระบบเมน โดยจะเป็นตัวป้องกันสำหรับสายเมนทั้งหมดซึ่งมีส่วนประกอบดังนี้ คือ ชุดทางานโดยอาศัยทางกล หน้าสัมผัส อาร์กอินเตอร์รัพเตอร์ และอุปกรณ์เปิดวงจรเนื่องจากกระแสเกินที่บรรจุอยู่ภายใน โดย ต่ออนุกรมร่วมกับโหลดที่ไหลผ่านเซอร์กิตเบรกเกอร์ เซอร์กิตเบรกเกอร์ชนิดนี้จะมีขนาดการทนกระแส และการทนกระแสลัดวงจรได้สูง ส่วนอุปกรณ์เปิดวงจรที่บรรจุอยู่ภายใน หรือ อุปกรณ์ตัดกระแส ที่สร้างขึ้นมานั้นจะออกแบบมาเพื่อให้ตัดไฟออก ได้อย่างถูกต้องตามคุณลักษณะที่ต้องการ เช่น อาจจะให้เซอร์กิตเบรกเกอร์ชนิดนี้ทางานแบบหน่วงเวลานาน หน่วงเวลาสั้น และเปิดวงจรทันทีทันใดก็ได้ ทั้งนี้เพื่อให้สอดคล้องกับระบบที่เซอร์กิตเบรกเกอร์เข้าไปเกี่ยวข้อง

โดยส่วนมากชิ้นส่วนที่เป็นอุปกรณ์การทางานแบบทางกล จะมีอยู่หลายรูปแบบด้วยกันขึ้นอยู่กับผู้ผลิต ส่วนมือจับจะทาหน้าที่เป็นตัวมือจับเพื่อให้คนทาการเปิดหรือปิดวงจร (โดยเป็นการปิดหรือเปิดหน้าสัมผัส) ในการปิดหรือเปิดวงจรนี้ อาจจะใช้ชุดสปริงเข้ามาช่วยเพื่อให้หน้าสัมผัสในขณะที่จะเข้าหรือออกจากกันเป็นไปโดยรวดเร็ว ชุดทางกลจะประกอบไปด้วยขดลวดแม่เหล็ก(magnetic coil) พร้อมทั้งแกนลูกสูบ (plunger) ซึ่งจะเคลื่อนตัวเมื่อขดลวดได้รับพลังงานหรืออาจจะเป็นชนิดโซลีนอยด์ก็ได้ซึ่งจะทางานเพื่อให้หน้าสัมผัสปิดหรือเปิดออกจากกัน ในการปิดเปิดหน้าสัมผัสนั้นจะต้องทาโดยรวดเร็ว ซึ่งไม่ว่าจะเป็นชนิดโซลีนอยด์หรือแบบใด จะต้องอาศัยหลักการเก็บพลังงาน(stored energy) โดยจะสะสมพลังงานไว้ในรูปของสปริง ถ้าจะปิดหรือเปิดหน้าสัมผัส ก็จะนาพลังงานที่สะสมไว้นี้มาปิดหรือเปิดหน้าสัมผัสดังกล่าว

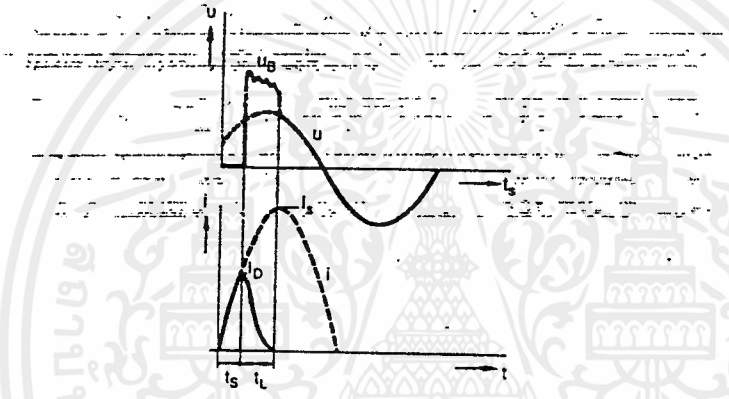
ส่วนอุปกรณ์ในการเปิดวงจรที่ต่ออนุกรมกับเซอร์กิตเบรกเกอร์นี้ ส่วนมากจะรับผลตอบสนองทางกลหลังจากที่ได้รับพลังงานไฟฟ้ามากระตุ้น ซึ่งทาให้สามารถที่จะปรับแต่งการทางานได้ นั่นคือปรับตาแหน่งการเคลื่อนตัวของหน่วยที่เปิดวงจร

2. เซอร์กิตเบรกเกอร์ชนิดโมลเคส(MCCB) เซอร์กิตเบรกเกอร์ชนิดนี้จะพบได้บ่อยในระบบการป้องกันของสายป้อนทุติยภูมิ หรือ วงจรย่อย หรือ ระบบเมนใหญ่

ข้อดี	ข้อเสีย
-ราคาถูกและความเร็วที่ใช้ในการทำงานดีเมื่อได้รับกระแสเกินที่มีปริมาณสูง ราคาจึงเป็นตัวตัดสินว่าควรใช้ฟิวส์ หรือ เซอร์กิตเบรกเกอร์ ทั้งนี้ขึ้นอยู่กับสภาพแวดล้อมและความชอบของแต่ละบุคคล	-ในวงจร 3 เฟส ถ้าฟิวส์ขาดเบเพียงเฟสเดียว จะทำให้เกิดความเสียหายแก่อุปกรณ์ไฟฟ้าที่ใช้ไฟ 3 เฟสได้ -เมื่อฟิวส์ขาด ถ้าการเปลี่ยนฟิวส์ตัวใหม่เกิดการเปลี่ยนไม่ได้ขนาดตามแบบเดิมจะทำให้การป้องกันลดลง -อุณหภูมิสภาวะแวดล้อม จะทำให้ฟิวส์มีการทำงานเปลี่ยนไปมากกว่าเซอร์กิตเบรกเกอร์ -ฟิวส์ จะไม่สามารถปรับค่ากระแสหรือทำให้ใช้งานได้ดังเดิมได้ หลังจากที่เกิดการผิดปกติ แต่เซอร์กิตเบรกเกอร์สามารถทำได้ -ฟิวส์ปรับคุณสมบัติของการทำงานไม่ได้จึงไม่อาจใช้ร่วมในการออกแบบระบบไฟฟ้าทั้งหมดได้

การทางานของฟิวส์ยังจะมีผลของ อุณหภูมิ สภาวะแวดล้อม เข้ามาเกี่ยวข้องด้วย โดยบกติการใช้งานควรอยู่ในช่วงระหว่าง -5 °C ถึง 45 °C

การกาจัดกระแส ในกรณีของกระแสลัดวงจรที่คาดว่า จะเกิดขึ้นและมีขนาดสูงไหลผ่านฟิวส์ ฟิวส์ที่ดีควรจะแตก ก่อนที่กระแสจะถึงค่าสูงสุดหรือค่า I_S กระแสที่มากที่สุดก่อนที่ฟิวส์จะขาด คือ I_D หรือกระแสที่ผ่านเข้าไปในฟิวส์สูงสุดก่อนที่ฟิวส์จะขาด นอกจากนี้จะสามารถแสดงถึงผลของการจากัดกระแสที่บรากฏขึ้น โดยแสดงไว้ในแผนภาพกระแสที่ผ่านเข้าไปในฟิวส์สูงสุด ก่อนที่ฟิวส์จะขาด



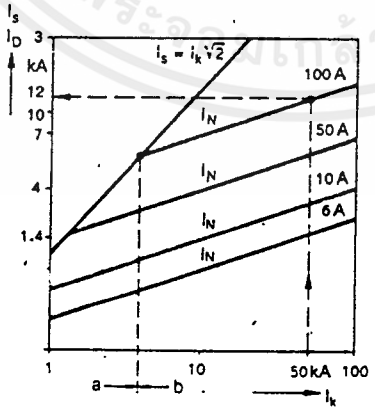
I_S หมายถึง กระแสลัดวงจรสูงสุด

I_D หมายถึง กระแสที่ผ่านเข้าไปในฟิวส์มากที่สุดก่อนที่ฟิวส์จะขาด หรือ let through current

t_s หมายถึง เวลาที่ใช้ในการหลอม

t_L หมายถึง เวลาที่ใช้ในการดับการหลอมหรือดับอาร์ก

u_B หมายถึง แรงดันไฟฟ้าที่อาร์ก



I_N หมายถึง พิกัดกระแสของฟิวส์

I_D หมายถึง กระแสที่ผ่านเข้าไปในฟิวส์มากที่สุดก่อนที่ฟิวส์จะขาด

a หมายถึง จำกัดกระแส

b หมายถึง ไม่จำกัดกระแส

การนำอุปกรณ์ป้องกันมาใช้งานและใช้งานร่วมกัน

<u>วัตถุประสงค์พื้นฐานในการนำอุปกรณ์ป้องกันมาใช้งาน</u>

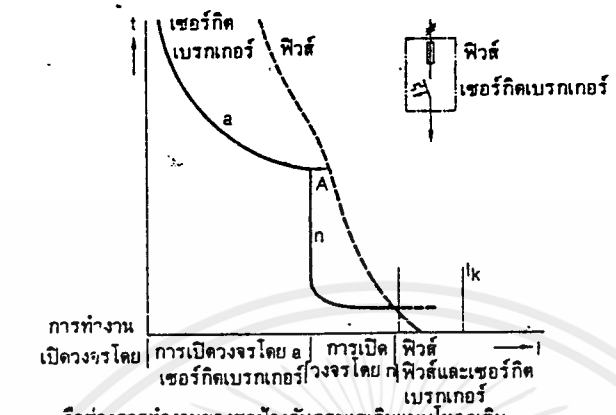
อาจกล่าวได้ว่าในระบบไฟฟ้ากำลังแบบอุดมคตินั้น จะเป็นระบบแบบมีแผนการทำงานร่วมกัน เพราะว่าจะต้องเลือกกันตั้งแต่อุปกรณ์ป้องกัน ไม่ว่าจะเป็นขนาดและการทำงานร่วมกันของอุปกรณ์ป้องกันภายในระบบไฟฟ้านั้น โดยการทำงานร่วมกันจะหมายถึงเมื่อกรณีที่เกิดการผิดปกติขึ้นในระบบไฟฟ้า อุปกรณ์ป้องกันตัวที่ใกล้จุดที่เกิดการผิดปกติที่สุดจะต้องทำงานได้อย่างมีประสิทธิภาพ และต้องทำงานก่อนอุปกรณ์ป้องกันตัวอื่นๆ

ในการออกแบบวงจรไฟฟ้านั้น จำเป็นที่จะต้องเลือกอุปกรณ์ป้องกันมาทำการป้องกันในกรณีที่เกิดการผิดปกติขึ้น ซึ่งจะขึ้นอยู่กับขนาดการทนต่อกระแสลัดวงจรและคุณสมบัติของเวลาและกระแสของอุปกรณ์ป้องกัน เช่น ฟิวส์จะมีคุณสมบัติคงที่ คือ จะสามารถปรับคุณสมบัติการทำงานได้ก็ต่อเมื่อเปลี่ยนขนาดและชนิดเท่านั้น หรือ กรณีของเซอร์กิตเบรกเกอร์ชนิดอากาศ จะมีชนิดการป้องกันกระแสเกิน ที่แตกต่างกันอยู่มากในการใช้งาน หรือ เซอร์กิตเบรกเกอร์ชนิดจมลเคสจะมีคุณสมบัติที่คงที่ ทำให้แทบจะไม่มีปัญหาในการเลือกใช้เลย

<u>อุปกรณ์สวิตช์เกียร์ที่มีฟิวส์เป็นองค์ประกอบ</u>

1.ฟิวส์ที่ใช้ร่วมกันกับเซอร์กิตเบรกเกอร์ ถ้ากระแสลัดวงจรที่เกิดขึ้นมีขนาดมากกว่าพิกัดของขนาดการทนได้ต่อกระแสลัดวงจร ของเซอร์กิตเบรกเกอร์ที่ติดตั้งไว้ในระบบไฟฟ้า จำเป็นที่จะต้องมีฟิวส์ป้องกันสำรอง ต่อเข้ากับเซอร์กิตเบรกเกอร์ทั้งนี้เพื่อเพิ่มพิกัดของขนาดการทนได้ต่อกระแสลัดวงจร

ในรูปหน้าถัดไป ฟิวส์จะทำการเปิดวงจรก่อนที่กระแสจะถึงพิกัดของขนาดการทนได้ ต่อกระแสลัดวงจรของเซอร์กิตเบรกเกอร์ สำหรับในช่วงการทำงานปกติไม่ว่ากระแสที่เกิดจากกรณีโหลดเกิน หรือ กรณีลัดวงจร เซอร์กิตเบรกเกอร์จะทำงานตัดวงจรออกทั้งหมด



a คือช่วงการทำงานของชุดป้องกันกระแสเกินแบบโหลดเกิน

n คือช่วงการทำงานของชุดป้องกันการลัดวงจร

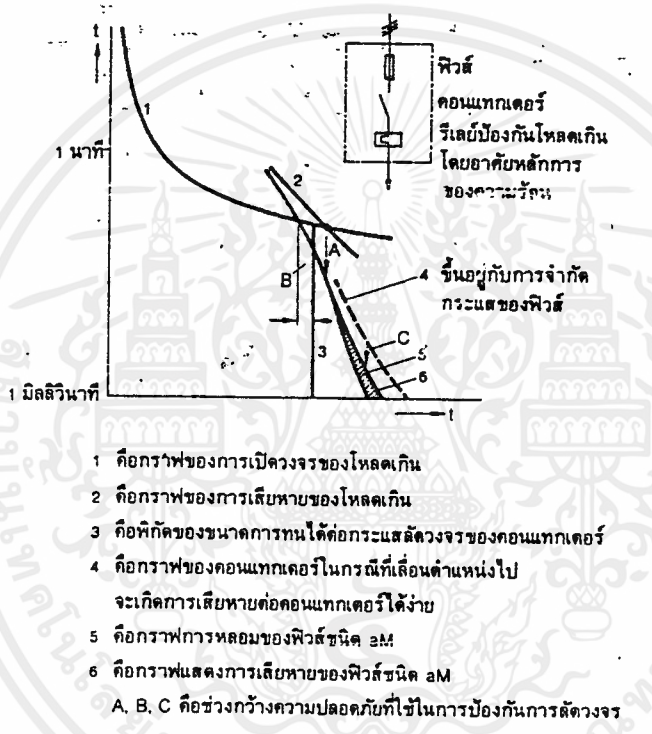
r.b.c. คือพิกัดของขนาดการทนได้ของกระแสลัดวงจร

I_k คือค่ากระแสลัดวงจรที่คาดว่าจะเกิดขึ้น (ค่าสูงสุด) ณ จุดที่เกิดการผิดปกติ

A คือระยะห่างของคุณสมบัติการป้องกันฟิวส์และเซอร์กิดเบรกเกอร์

2.ฟิวส์ คอนแทกเตอร์ และรีเลย์ป้องกันโหลดเกิน คอนแทกเตอร์ที่ใช้เพื่อจะใช้เป็นอุปกรณ์ในการสวิทชิงมอเตอร์ ส่วนรีเลย์โหลดเกิน จะมีไว้เพื่อเป็นตัวป้องกันมอเตอร์และสายตัวนำที่เข้าสู่มอเตอร์ และคอนแทกเตอร์ในกรณีที่เกิดการใช้งานเป็นแบบโหลดเกิน ส่วนฟิวส์จะใช้ป้องกันการลัดวงจรของคอนแทกเตอร์

ในการเลือกอุปกรณ์มาใช้งานร่วมกัน จะต้องระวังอย่างมาก ในกรณีที่จะหาอุปกรณ์เหล่านี้ทำงานอย่างเหมาะสม โดยพิจารณาจากรูปข้างล่าง



1 คือกราฟของการเปิดวงจรของโหลดเกิน
2 คือกราฟของการเสียหายของโหลดเกิน
3 คือพิกัดของขนาดการทนได้ต่อกระแสลัดวงจรของคอนแทกเตอร์
4 คือกราฟของคอนแทกเตอร์ในกรณีที่เลื่อนตำแหน่งไป
จะเกิดการเสียหายต่อคอนแทกเตอร์ได้ง่าย
5 คือกราฟการหลอมของฟิวส์ชนิด aM
6 คือกราฟแสดงการเสียหายของฟิวส์ชนิด aM
A, B, C คือช่วงกว้างความปลอดภัยที่ใช้ในการป้องกันการลัดวงจร

<u>การจัดลำดับการป้องกัน</u>

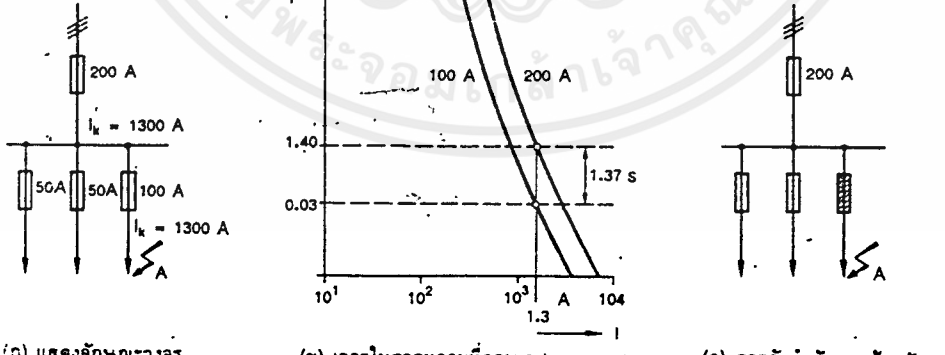
เมื่อเกิดการผิดปกติขึ้นในระบบไฟฟ้า อุปกรณ์ป้องกันกระแสเกินจะต้องตัดหรือเปิดวงจรส่วนที่ผิดปกตินั้นในเวลาอันรวดเร็ว และส่วนที่ถูกต้องนั้นจะต้องตัดออกน้อยที่สุดแต่ความปลอดภัยและความเชื่อถือ จะต้องมีมากที่สุด

การจัดลำดับการป้องกันในระบบไฟฟ้าแบบรากจะสามารถติดตั้งอุบกรณ์ป้องกันตามลำดับการไหลของกำลังไฟฟ้า อุบกรณ์ที่ใช้อาจเป็น

1. ฟิวส์เมนกับฟิวส์ย่อย
2. เมนเซอร์กิตเบรกเกอร์กับเซอร์กิตเบรกเกอร์ย่อย
3. เมนเซอร์กิตเบรกเกอร์กับฟิวส์ย่อย
4. ฟิวส์กับเซอร์กิตเบรกเกอร์ย่อย
5. ใช้สายป้อนที่ขนานกันหลายชุด พร้อมกับมีเซอร์กิตเบรกเกอร์ย่อย
6. ฟิวส์แรงสูงกับฟิวส์แรงต่ำย่อย

การจัดลำดับการทำงานในกรณีที่มีฟิวส์ต่ออนุกรมกัน

ในรูปข้างล่าง เป็นการอธิบายถึงระบบไฟฟ้าที่ประกอบด้วยสวนเข้า และสวนออกจากบัสบาร์ จะสังเกตเห็นว่ากระแสและพื้นที่หน้าตัดของวงจรต่างๆ อาจมีสามไม่เท่ากัน ดังนั้น ฟิวส์ที่ใช้จึงมีขนาดพิกัดต่างๆกัน แต่อย่างไรก็ตามในกรณีเกิดความผิดปกติขึ้นในระบบ กระแสลัดวงจรที่เกิดขึ้นจะไหลผ่านฟิวส์ทุกตัว

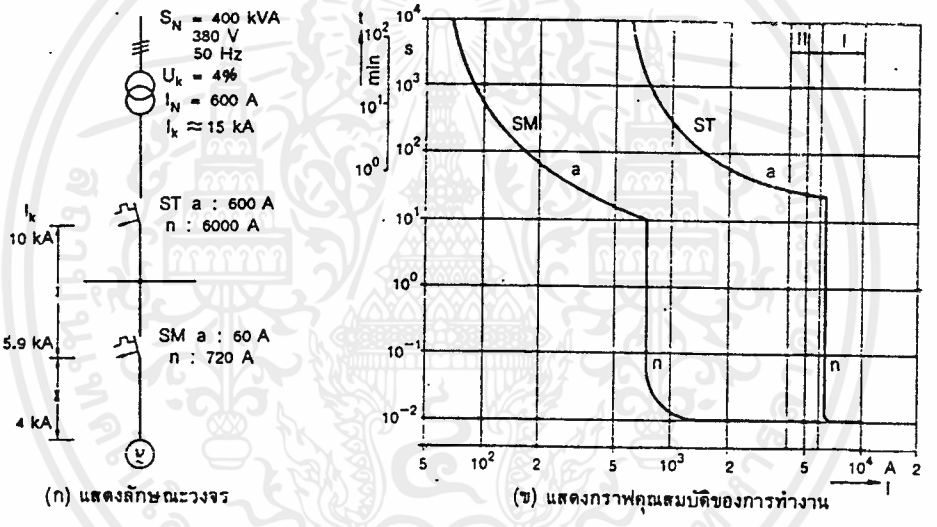


(ก) แสดงลักษณะวงจร (ข) เวลาในการหลอมที่กระแส I_k = 1300 A (ค) การจัดลำดับการป้องกันเมื่อเกิดการผิดปกติที่จด A

การจัดลำดับการทำงานในกรณีที่เซอร์กิตเบรกเกอร์ต่ออนุกรมกัน

จากรูปข้างล่าง จะเห็นว่ากระแสลัดวงจรจะเปลี่ยนแปลงไปตลอดเวลา เมื่อมีความยาวของระยะทางเกิดขึ้น ดังนั้นการปรับค่าเวลาเปิดวงจรหรือทริปของ n ของเซอร์กิตเบรกเกอร์ตัวใดๆ ในบริเวณโซนที่ครอบคลุมนั้น จะต้องมีค่ามากกว่าค่ากระแสที่มีค่าสูง อาจจะเกิดขึ้นในบริเวณโซนนั้น

ในรูปข้างล่างเป็นการเลือกใช้เซอร์กิตเบรกเกอร์ ที่เป็นแบบมีระบบการป้องกันหรือเปิดวงจร กรณีที่กระแสเกินเกิดขึ้นเป็นแบบอาศัยหลักการของแม่เหล็กไฟฟ้า



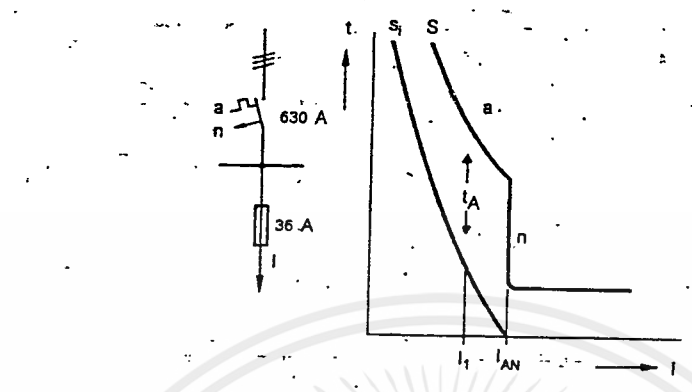
(ก) แสดงลักษณะวงจร (ข) แสดงกราฟคุณสมบัติของการทำงาน

SM คือเซอร์กิตเบรกเกอร์ป้องกันมอเตอร์ a คืออุปกรณ์ป้องกันโหลดเกิน

ST คือเซอร์กิตเบรกเกอร์ป้องกันหม้อแปลง n คืออุปกรณ์ป้องกันและเปิดวงจรทันทีทันใดเมื่อมีการลัดวงจรเกิดขึ้น

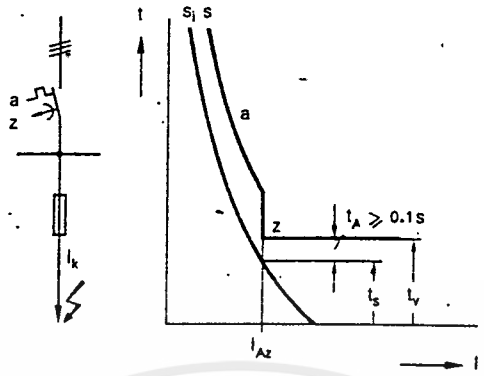
การจัดลำดับการทำงานในกรณีที่ประกอบด้วยเมนเซอร์กิตเบรกเกอร์ และฟิวส์ย่อย

ในรูปหน้าสัดไป จะพบว่าเซอร์กิตเบรกเกอร์ประกอบไปด้วย การทำงานแบบลัดวงจรเมื่อเกิดกรณีโหลดเกินและลัดวงจรขึ้น โดยจะทำงานตอบสนองแบบทันทีทันใดเมื่อมีกระแสเกินเกิดขึ้น



S_j คือฟิวส์
S คือเซอร์กิตเบรกเกอร์
a คือลุปกรณ์ป้องกันกระแสเกินกรณีโหลดเกิน
n คือลุปกรณ์ป้องกันกระแสเกินกรณีลัดวงจร
t_A คือช่วงกว้างของเวลาที่ปลอดภัย
I_{An} คือช่วงกระแสที่ทำให้ n เปิดวงจร

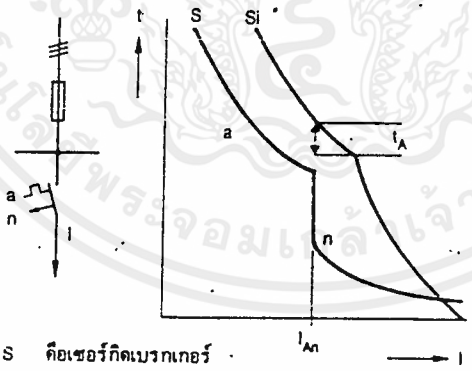
ถ้าในวงจรไฟฟ้านั้นเกิดการลัดวงจรขึ้นและกระแสลัดวงจรนี้ อาจมีค่าเท่ากับหรือ มากกว่าค่าบรับต้องการทางานตอบสนองแบบทันทีทันใดเมื่อมีกระแสเกินเกิดขึ้น ดังนั้นฟิวส์ก็จะทาหน้าที่จากัดกระแสลัดวงจรนี้ทันที โดยกระแสที่ผ่านเข้าไปในฟิวส์สูงสุดก่อนที่ฟิวส์จะขาด ที่โหลผ่านฟิวส์จะไม่มีค่าสูงถึงค่าทรบของเซอร์กิตเบรกเกอร์เลย ดังนั้นจึงกล่าวได้ว่าในกรณีเช่นนี้พิกัดของฟิวส์ จะต้องมีค่าต่ากว่าพิกัดของเซอร์กิตเบรกเกอร์มาก ดังรูปหน้าถัดไป



a คืออุปกรณ์ป้องกันโหลดเกิน
Z คืออุปกรณ์ป้องกันกระแสเกินและทำงานแบบหน่วงเวลาสั้น
t_A คือช่วงกว้างของเวลาที่ปลอดภัย
I_{Az} คือกระแสที่ตอบสนองต่อการทำงานของ z
t_S คือเวลาการหลอมของฟิวส์
t_V คือช่วงหน่วงเวลาของ z

การจัดสัดส่วนการทำงานระหว่างเมนฟิวส์และเซอร์กิตเบรกเกอร์ย่อย

ในบรรเดินนี้จะพิจารณาได้จากรูปข้างล่าง ซึ่งในช่วงการทำงานของเซอร์กิตเบรกเกอร์ในช่วงการทำงานเกิดวงจรเนื่องจากโหลดเกิน และการทำงานในกรณีลัดวงจร จะไม่ตัดกันกับคุณสมบัติของฟิวส์เลย



S คือเซอร์กิตเบรกเกอร์
S_i คือฟิวส์
a คืออุปกรณ์ป้องกันโหลดเกิน
n คืออุปกรณ์ป้องกันกระแสเกินกรณีลัดวงจร
t_A คือช่วงกว้างของเวลาที่ปลอดภัย
I_{An} คือกระแสตอบสนองต่อการทำงานของ n

แต่ในกรณีที่วงจรไฟฟ้าเกิดการลัดวงจร กระแสลัดวงจรที่ไหลผ่านวงจรจะทำให้ฟิวส์เกิดความร้อนขึ้น และเซอร์กิตเบรกเกอร์ก็จะเกิดการอาร์กขึ้นด้วย ในทางปฏิบัติจะนิยมให้คุณสมบัติของเวลาการหลอมละลายของฟิวส์ต่อกระแสของฟิวส์อยู่ห่างจากการเดินวงจรของเซอร์กิตเบรกเกอร์ ชนิดชุดการเดินวงจรอย่างทันทีทันใดเมื่อกระแสเกินไหลผ่านอยู่ 70 mS

การจัดลำดับการทำงานของอุปกรณ์ป้องกันแรงดันไฟฟ้าตก

ในกรณีที่เกิดการลัดวงจรขึ้น ระดับของแรงดันไฟฟ้า ณ จุดที่เกิดการลัดวงจรจะมีค่าที่ไม่เท่ากับปกติอาจจะเป็นศูนย์ก็ได้ ในทางปฏิบัติทั่วๆไป ถ้าเกิดการลัดวงจรขึ้น ผลที่จะตามมาก็คือการเกิดการอาร์ก ค่าแรงดันไฟฟ้าของการอาร์กนี้จะมีค่าประมาณ 30 V ถึง 70 V นอกจากนี้ค่าแรงดันไฟฟ้าของวงจรที่บรากฏบนตัวนำก็จะมีค่าลดน้อยลงไปเป็นส่วนๆไปเรื่อยๆ แรงดันไฟฟ้าที่ลดลงจะขึ้นอยู่กับความต้านทานของตัวนำและระยะทางไปยังจุดที่เกิดการผิดปกติ

เซอร์กิตเบรกเกอร์ ซึ่งติดตั้งไว้ป้องกันการลัดวงจรโดยตรงจะต้องเดินวงจรออก เวลาที่ใช้ในการเดินวงจรทั้งหมด จะสัมพันธ์และขึ้นอยู่กับขนาดและชนิดของเซอร์กิตเบรกเกอร์ โดยปกติจะมีค่าสูงสุดถึง 0.03 วินาที แต่ในกรณีที่เป็นเซอร์กิตเบรกเกอร์ชนิดจำกัดกระแสแล้ว จะมีค่าสูงสุด 0.01 วินาที

การออกแบบตัวควบคุม 8051

การแนะนำ ในบทนี้เป็นเรื่องขององค์ประกอบทางด้าน hardware สำหรับตัวควบคุม 8051 ซึ่งจะใช้เป็น hardware สำหรับตัวอย่างการประยุกต์ใช้งานในบทต่อๆไปสวน debugging programs ที่ให้ในบทนี้ จะใช้เป็น subroutines ของ pragram ในบทต่อไป

การออกแบบตัวควบคุม เริ่มจากการแยกแยะความต้องการ และความว่างเปล่าของกระดาษ หรือจอคอมพิวเตอร์ การพัฒนาของตัวควบคุมจะตามมาในขั้นตอนต่าง ๆ ดังนี้

1. การกำหนดรายละเอียด
2. ออกแบบระบบตัวควบคุมตามรายละเอียด
3. เขียน program ตรวจสอบการออกแบบ
4. เขียนและทดสอบ subroutines ที่ใช้บริการต่าง ๆ ไป

ขั้นตอนแรก ในการออกแบบมีความสำคัญมาก ถ้าชิ้นงานนั้นใช้เป็นแบบของผลผลิตจำนวนมาก (มากกว่า 1000 หน่วย)ดังนั้นจะเป็นภาระที่ต้องวิเคราะห์อย่างระมัดระวัง ความผิดพลาดหรือความรัดกุมในรายละเอียด จะเพิ่มมากขึ้นสำหรับส่วนของการลงทุนในอุตสาหกรรม เพื่อทำให้รายละเอียดลดลง โดยต่อไปผู้ออกแบบจะเขียนรายละเอียดที่กระชับในการประยุกต์ใช้งาน

ในความเป็นจริง ส่วนใหญ่จะไม่สามารถออกแบบตัวควบคุมได้ใหม่ หลังจากที่มันถูกพัฒนาจนไกลจากจุดเริ่มต้น เนื่องจากเราไม่สามารถเข้าใจที่มาของตัวควบคุมจากประสบการณ์ของผู้ออกแบบ รายละเอียดที่สมบูรณ์ จะต้องการ PORT I/O มากกว่า 1 PORT และต้องการหน่วยความจำมากกว่า 1 กิโลไบต์

Microcontroller Specification ตัวอย่างสำหรับการออกแบบตัวควบคุมมีดังนี้ การควบคุมเวลาและข้อมูลที่แน่นอนจำเป็นมากสำหรับตัวควบคุมที่ฉลาด ตัวควบคุมจะเป็นส่วนหนึ่งของ ระบบเครือข่ายคอมพิวเตอร์ที่ติดต่อกันทาง port อนุกรม ถ้าตัวควบคุมถูกผลิตออกมาเป็นจำนวนน้อย ตัวอย่างเช่น น้อยกว่า 1000 หน่วย ใน

งานใดงานหนึ่ง ราคาของของระบบจะต้องมีราคาต่ำ"

ตระกูลของ 8051 มีลักษณะดังนี้

1. ราคาถูก
2. มีผู้จำหน่ายมาก
3. เหมาะกับการใช้งานร่วมกับ NMOS และ CMOS
4. Software หาง่ายและราคาถูก
5. Compiler เป็นภาษาชั้นสูง

ลักษณะในข้อ 1-3 มีความสำคัญมากจาก ราคาที่เป็นมาตราฐาน ทำให้สามารถอาศัยSoftware ช่วยลดต้นทุน และทำให้ project สมบูรณ์ได้

ความต้องการในการเปลี่ยน program เพื่อใช้กับงานใดงานหนึ่ง มักจะต้องการ EPROM จากภายนอก ซึ่งต้องการใช้ Port 0 เป็นขา address A0-A7 และ port 2 เป็นขา address A8-A15 เพราะฉะนั้น port 0 และ 2 จะไม่เหมาะกับการใช้เป็น I/O Port

เนื่องจากความเป็นไปได้ที่ตัวควบคุมจะมีข้อมูลที่มากเกินกว่า RAM ภายในจะเก็บไว้ได้เพราะฉะนั้นอาจจำเป็นต้องใช้ RAM ภายนอก ซึ่งมีสัญญาณควบคุม การเขียน (WR, Activelow) bit ที่ 6 ของ port 3 และสัญญาณควบคุมการอ่าน (RD, Active low) bit ที่ 7ของ port 3 สำหรับหน่วยความจำภายนอกขนาด 64 K จะมีขาทั้งหมด 28 ขา

ตัวอย่างนี้ใช้ได้กับ EPROM ตั้งแต่ 8 K จนถึง 64 K ส่วน RAM ใช้ได้กับขนาด 8 K จนถึง 32 K ขนาดของหน่วยความจำที่แตกต่างกันสามารถเลือกได้จาก jumper เพื่อเพิ่มขา address ให้กับหน่วยความจำขนาดใหญ่และใช้ความต้านทาน pullup เพื่อสามารถใช้กับหน่วยความจำขนาดเล็กได้

ความต้องการในการส่งข้อมูลอนุกรมสามารถใช้ port อนุกรมภายในซึ่งมีสัญญาณที่ใช้ในการควบคุม bit ที่ 0 ของ port 3 เป็นสัญญาณ RXD และ bit ที่

1 ของ port 3 เป็นสัญญาณ TXD เพราะฉะนั้นเราสามารถใช้ port 1 ทั้งหมด และ port 3 pin 2-5 เป็น I/O หรือเป็น-interrupt จากภายนอก และ timing input ได้

การสูญเสีย ความสามารถของ I/O ที่เกิดขึ้นจากการนำมาใช้เป็น port functionเพื่อความถูกต้องของ program ข้อเสียนี้จะไม่มาก แต่อย่างก็ตามมี 2 วิธีในการ ขยาย I/Oคือจาก I/O ของคอมพิวเตอร์และ Memory map I/O

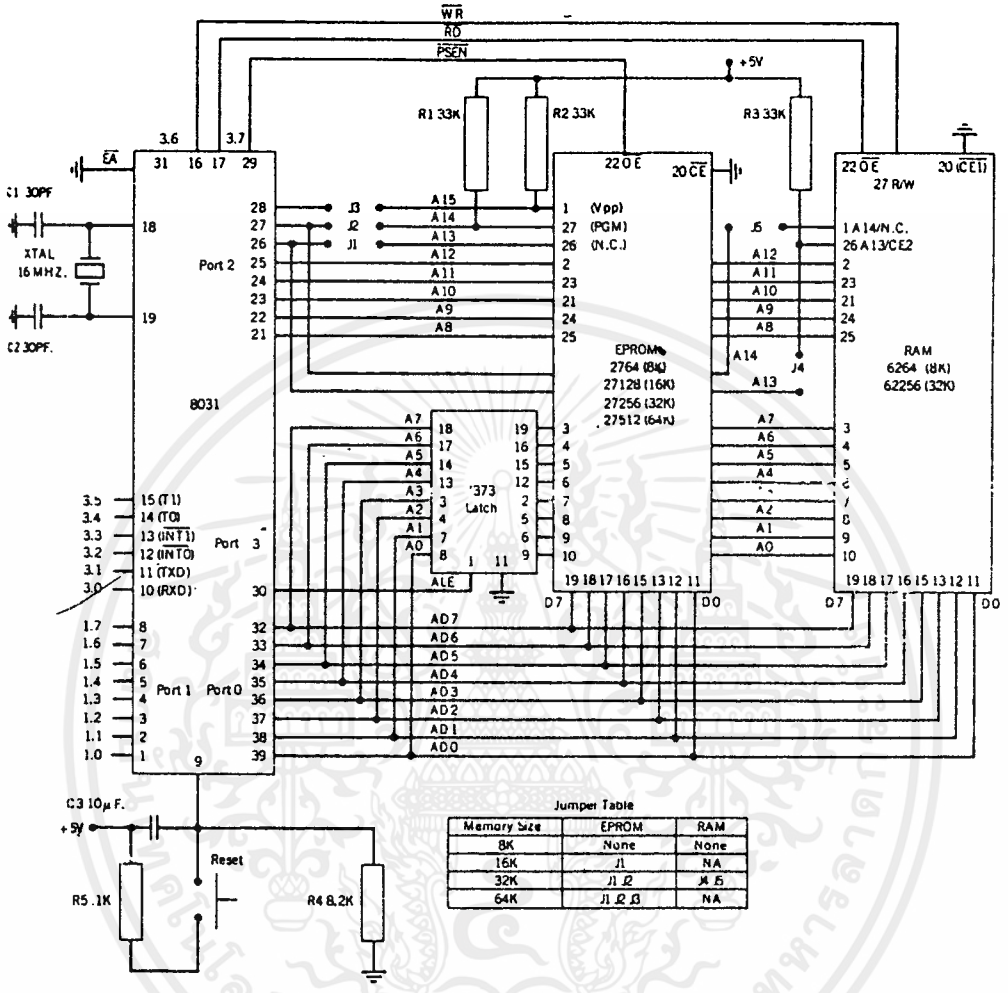
สุดท้ายนี้ เราใช้ crystal ความถี่ 16 MHz รายละเอียด โดยสมบูรณ์มีดังนี้

- 80c31-1 (ไม่มี ROM ภายใน) เป็น ตัวควบคุม
- EPROM ภายนอก ขนาด 64 KByte
- RAM ภายนอก ขนาด 32 KByte
- 8 port ที่ใช้งานทั่วไป
- 4 port ที่ใช้งานควบคุม progrom
- 1 full-duplex serial port
- clock ความถี่ 16 MHz

<u>Microcontroller Design</u> รูป 1 แสดงการออกแบบตัวควบคุมมีรายละเอียดดังนี้

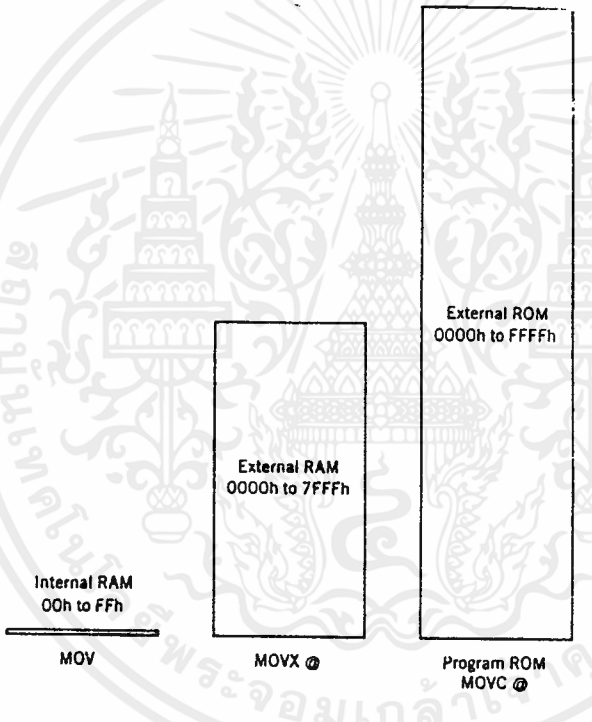
หน่วยความจำภายนอก และการถอดรหัสหน่วยความจำภายนอก

ใช้ port 0 ทำหน้าที่เป็น port ข้อมูล และเป็น port address byte ต่ำ ส่วนport 2 เป็น port address byte สูง ข้อมูล และ address byte ต่ำ ของ port 0 จะมี time multiplex ใช้ 373 ที่ต่อกับ port 0 เป็นตัว latch ค่า address byte ต่ำ โดยใช้สัญญาณ ALE จาก 8051เป็นสัญญาณในการ latch และ port 0 จะเป็นได้ทั้ง input และ output สำหรับอ่านและเขียนข้อมูล



address ของ RAM และ ROM จะต่างกันที่สัญญาณ PSEN (Active low) เมื่อ PSEN Active แสดงว่าเป็นการติดต่อกับ ROM และสัญญาณ WR

(Active low) หรือ RD (Active low) จะใช้สำหรับ RAM และผลจากขา address ที่แตกต่างกันของหน่วยความจำในแต่ละขนาด ทำให้เราต้องใช้ jumper และ ความต้านทาน pullup ดังนั้นผู้ใช้สามารถใช้ขนาดหน่วยความจำได้ตามต้องการ ตาราง jumper ในรูป 1 ใช้สำหรับ EPROM และ RAM รูป 2 แสดงความสัมพันธ์ของหน่วยความจำ ภายในและภายนอก

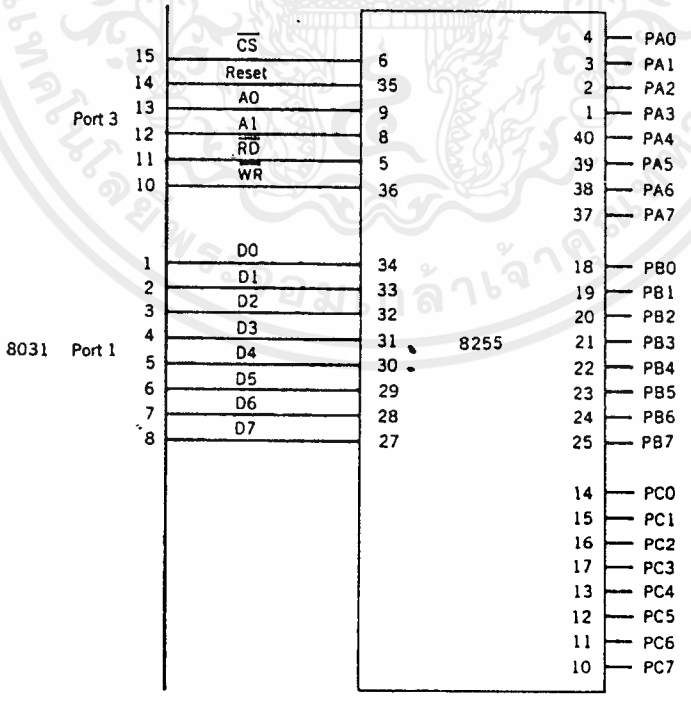


วงจร Reset และ Clock ใน 8051 ขา Reset จะ Active high เพราะฉะนั้นขา Reset จะต้องเป็น high ใน 2 machine cycle เมื่อเริ่มจ่ายไฟเลี้ยง และหลังจากนั้นขา Reset จะเป็น low โดยใช้วงจร RC เมื่อเริ่มจ่ายไฟเลี้ยงที่ขา Reset จะเป็น high จนกระทั่งตัวเก็บประจุ charge ไฟไว้ราว ๆ

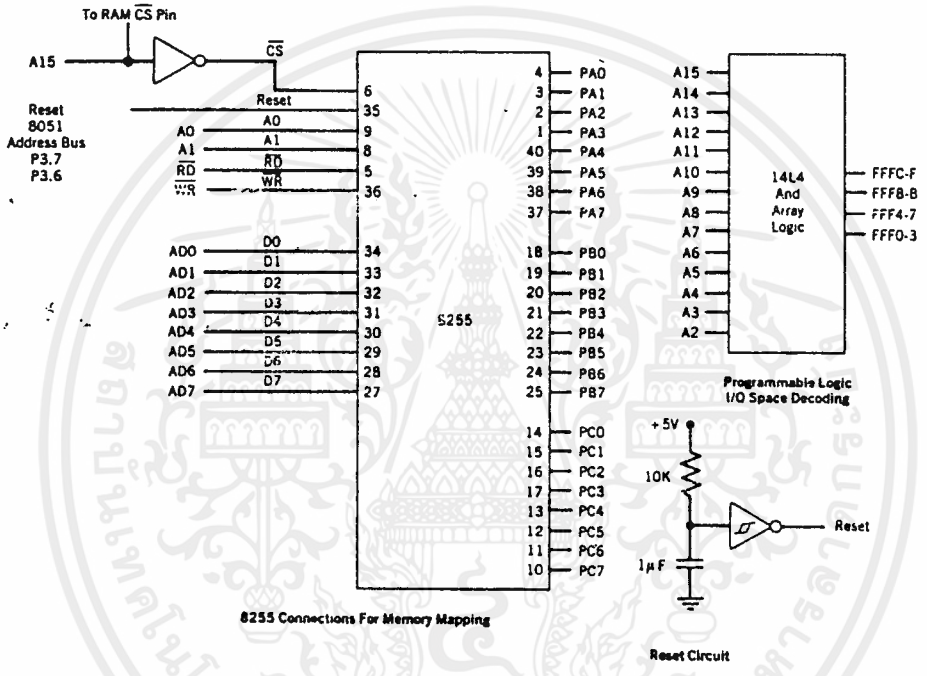
2.5 V จึงทำให้ขา Reset เป็น low และระบบจะเริ่มทำงาน ที่ขา Reset ยังมี Switch สำหรับ Reset ระบบเมื่อผู้ใช้ต้องการ

Expanding I/O Port 1 และ 3 สามารถใช้เป็นขาควบคุม และขาเข้าออกของข้อมูล data bus ใช้เชื่อมโยงกับวงจรภายนอกเพื่อขยาย I/O chip I/O ที่นิยมใช้เป็น port คือ 8255 ใน 8255 จะมี Register Cortrol word สามารถ Set โดย Software เพื่อกำหนดให้ port ของ 8255 ที่ชื่อว่า port A, B, C มีความสามารถเป็น input port,output port

รูป 3 แสดงวงจรที่เพิ่ม 8255 เข้าไปในระบบทำให้ขณะนี้มี portขนาด 8 bit อยู่ 3 port ปัญหาของการขยาย I/O คือความเร็วซึ่งเกิดจากการที่ต้องการใช้เวลาในการเขียน control bitใน port 1 และ 3 ก่อนที่ I/O จะทำงานได้ ส่วนจุดเด่นของการขยาย I/O port คือ ทำให้ 8051 สามารถใช้ติดต่อกับภายนอกได้ยาขึ้น port 1 และ 3



Memory-Mapped I/O chip ที่ใช้ขยาย port สามารถใช้พื้นที่ของหน่วยความจำที่ไม่ถูกใช้ในการออกแบบ แสดงไว้ในรูป 4 เราสามารถออกแบบได้ใน32K ด้านบน (8000H หรือ สูงกว่า) สำหรับการขยาย port โดย A15 เป็น high เนื่องจาก 32K ของ RAM (7FFFH หรือต่ำกว่า) A15 จะเป็น low ดังนั้นต้องการ inverter เพื่อถอดรหัสระหว่างพื้นที่ของหน่วยความจำ และ I/O



การออกแบบควรจะถอดรหัสให้มีพื้นที่ใช้สำหรับ RAM เป็นจำนวนมากและควรจะระบุตำแหน่งที่แน่นอนในการติดต่อกับ chip I/O รูป 4 แสดงการออกแบบที่รวม 3 memory -mapped port ที่ address FFF0H - FFF3H, FFF4H - FFF7H,FFF8H - FFFBH และ FFFCH - FFFFH โดย RAM สามารถใช้พื้นที่จาก 0000H - FFEFH ได้

Memory-Mapped I/O มีจุดเด่นที่ไม่ต้องใช้ port ของ 8051แต่ข้อเสียก็คือ ทำให้สูญเสีย พื้นที่ของ RAM เพื่อนำมาใช้เป็น address ของ I/O ลักษณะ

สวนาหญของ programจะเหมือนกับ port I/O แตกต่างกันที่ต้องใช้คาสั่ง Movx กับ Memory-mapped I/O

สาหรับการขยาย I/O ใน 2 แบบนั้น จะพิจารณาการเลือกใช้ที่ราคาของระบบซึ่งต้องดูจาก ความต้องการ I/O port มากน้อยเท่าใด และ ขนาดของหน่วยความจา

part Speed ข้อพิจารณาข้อหนึ่งที่มักจะไม่บรรลุในการออกแบบ คือส่วนของความถี่ที่จะใช้ในระบบ หน่วยความจาทั้งหมดจะใช้เวลาเข้าถึงข้อมูลเป็น ns (เวลาที่ใช้ในการอ่านหรือเขียนข้อมูลหลังจากกาหนด address แล้ว) เวลาที่ใช้นี้จะต้องมีค่ามากกว่าคาบเวลาในการออกแบบระบบ เวลาที่ใช้ในการอ่านหรือเขียนข้อมูลจาก RAM กับเวลาที่ใช้ในการอ่านข้อมูลจาก ROMซึ่งเวลาทั้งหมดจะเป็นตัวเลือกใช้ความถี่ของระบบ และในตัวอย่างของเรา EPROM มีเวลาที่มากที่สุดในการเข้าถึงข้อมูล คือ 150 ns และ RAM ใช้เวลา 400 ns ในส่วนอื่น เช่น 373 ซึ่งใช้เป็นตัว latch สามารถใช้ตระกูล LSTTL ถึงตระกูล HCMOSเพราะความเร็วของส่วนนี้จะห่างกับความเร็วของ 8051 มาก

ทดสอบการออกแบบ เมื่อ hardware ถูกประกอบแล้วมันจาเป็นต้องตรวจสอบการออกแบบ โดยการเขียน progran ขึ้นมาตรวจสอบการทางาน โดยเริ่มจาก program พื้นฐานทาการทดสอบในแต่ละส่วน

ทดสอบ Crystal ขั้นแรกต้องทดสอบให้มั่นใจได้ว่า วงจร Rest และ Crystal ทางานโดยทดสอบจากสัญญาณ ALE ดูความถี่ด้วย oscilloscope ซึ่งความถี่ของสัญญาณ ALE ต้องเป็น 1/6 ของ ความถี่ crystal ถัดไปก็ทดสอบการ Reset โดยกดปุ่ม Reset และตรวจสอบดู port ทั้งหมดซึ่งต้องอยู่ในสถานะ high (input)

<u>ทดสอบ ROM</u> เป็น program ทดสอบที่สาคัญที่สุดเพื่อแน่ใจว่าตัวควบคุมสามารถเข้า

ไบเอาข้อมูลจาก EPROM และ นำไบบฏิบัติได้ การ fetch สามารถทดสอบได้ด้วย การตรวจสอบในแต่ละ address ของ ROM โดยใช้ คำสั่งการกระทำซ้ำ การกระโดด ในแต่ละ address ในที่นี้จะทดสอบการกระโดดไบ address โดยเพิ่ม address ครั้งละ 2 เท่า เส้น address เพียงเส้นเดียวจะเป็น high นอกนั้นจะเป็น low การทดสอบจะทดสอบ ในแต่ละ address และตรวจสอบการ short กันระหว่าง 2 เส้น

ถ้าการทดสอบสำเร็จ program จะหยุด ที่ address สูงสุดที่เป็นไบได้ของ ROM ขาaddress สามารถทดสอบได้ด้วย logic probe ดูที่ address สูงสุดซึ่งจะบรากฏว่ามีค่าคงที่ ถ้าไม่คงที่ probe จะกระพริบ นั้นคือต้องแก้ไขระบบ

การทดสอบการทำงานโดยใช้ 373, EPROM ขนาด 64K, jump ที่จุด1-3 และทำการ Reset 8051 การทดสอบสามารถที่จะหยุดในทุก ๆ address โดยการกระโดดไบที่ address นั้น ตัวอย่างเช่นในขั้นตอนสุดท้ายของ progrom ทดสอบ ROM

```
ADDRESS     MNEMONIC        COMMENT

            .org 0000h      ;start at the bottom of ROM
begin:      ljmp add2       ;test address line A0 and A1
            .org 0004h      ;next jump at address 0004h (A2)
add2:       ljmp add3       ;test address line A2
            .org 0008h      ;next jump at address 0008h (A3)
add3:       ljmp add4       ;test address line A3
            .org 0010h      ;next jump at address 0010h (A4)
add4:       ljmp add5       ;test address line A4
            .org 0020h      ;next jump at address 0020h (A5)
```

```
add5:     ljmp add6       ;test address line A5
          .org 0040h      ;next jump at address 0040h (A6)
add6:     ljmp add7       ;test address line A6
          .org 0080h      ;next jump at address 0080h (A7)
add7:     ljmp add8       ;test address line A7
          .org 0100h      ;next jump at address 0100h (A8)
add8:     ljmp add9       ;test address line A8
          .org 0200h      ;next jump at address 0200h (A9)
add9:     ljmp add10      ;test address line A9
          .org 0400h      ;next jump at address 0400h (A10)
add10:    ljmp add11      ;test address line A10
          .org 0800h      ;next jump at address 0800h (A11)
add11:    ljmp add12      ;test address line A11
          .org 1000h      ;next jump at address 1000h (A12)

add12:    ljmp add13      ;test address line A12
          .org 2000h      ;next jump at address 2000h (A13)
add13:    ljmp add14      ;test address line A13
          .org 4000h      ;last jump at address 4000h (A14)
add14:    ljmp add15      ;test address line A14
          .org 8000h      ;test address line A15 and remain
                           here
add15:    ljmp add15      ;jump here in a loop
          .end            ;assembler use

;This address, A15, will remain latched while A2-A14 will
```

```
;remain low. A0 and A1 will vary as the bytes of the jump
;instruction are fetched
```

รูป 5 แสดงรายละเอียดของ program ที่ละ address

FIGURE 5 Assembled ROM Check Program

```
0000                    .org 0000h ;start at the bottom of ROM
0000 020004  begin:     ljmp add2  ;test address line A0 and A1
0004                    .org 0004h ;next jump at address 0004h(A2)
0004 020008  add2:      ljmp add3  ;test address line A2
0008                    .org 0008h ;next jump at address 0008h(A3)
0008 020010  add3:      ljmp add4  ;test address line A3
0010                    .org 0010h ;next jump at address 0010h (A4)
0010 020020  add4:      ljmp add5  ;test address line A4
0020                    .org 0020h ;next jump at address 0020h (A5)
0020 020040  add5:      ljmp add6  ;test address line A5
0040                    .org 0040h ;next jump at address 0040h (A6)
0040 020080  add6:      ljmp add7  ;test address line A6
0080                    .org 0080h ;next jump at address 0080h (A7)
0080 020100  add7:      ljmp add8  ;test address line A7
0100                    .org 0100h ;next jump at address 0100h (A8)
0100 020200  add8:      ljmp add9  ;test address line A8

0200                    .org 0200h ;next jump at address 0200h(A9)
0200 020400  add9:      ljmp add10 ;test address line A9
0400                    .org 0400h ;next jump at address 0400h(A10)
```

```
0400 020800   add10: ljmp add11 ;test address line A10
0800                 .org 0800h ;next jump at address 0800h(A11)
0800 021000   add11: ljmp add12 ;test address line A11
1000                 .org 1000h ;next jump at address 1000h(A12)
1000 022000   add12: ljmp add13 ;test address line A12
2000                 .org 2000h ;next jump at address 2000h(A13)
2000 024000   add13: ljmp add14 ;test address line A13
4000                 .org 4000h ;last jump at address 4000h(A14)
4000 028000   add14: ljmp add15 ;test address line A14
8000                 .org 8000h ;test address line A15 and remain
                                ;here
8000 028000   add15: ljmp add15 ;jump here in a loop
8003                 .end       ;assembler use
```

ทดสอบ RAM สามารถตรวจสอบจากความสามารถของตัวควบคุมในการ execute การทดสอบโดยทั่วไปจะเขียนค่าที่มีรูปแบบที่แน่นอนลงใน RAM ปกติมักใช้ค่า ที่มีรูปแบบเป็นเลข1 สลับกับ 0 เช่น 55h หรือ AAh

โปรแกรมต่อไปเราจะเขียนค่าที่กำหนดลงใน RAM ภายนอก และอ่านค่ากลับมาตรวจสอบในแต่ละ byte ที่เขียนลงไปถ้าการตรวจสอบล้มเหลว ดังนั้นความล้มเหลวนั้นอาจเกิดจากDPTR register ซึ่ง Port1 และ bit ที่ว่างของ Port 3 สามารถใช้เป็นตัวแสดงสถานะของ DPTR ได้

ในขณะนี้มี 14 bit ของ port ที่สามารถนำไปใช้ได้ (serial port ยังไม่ถูกใช้ดังนั้น bit 3.0 และ 3.1 จะเป็น อิสระ) และเราต้องการ 15 bit สำหรับ address ขนาด 32 K ในที่นี้ program จะใช้ทดสอบในช่วง 8 K ซึ่งใช้ 13bit ในการตรวจสอบ เพราะฉะนั้นจะมี 4 ส่วนให้ทดสอบ (32/8) และถ้าการทดสอบผ่าน bit ที่ 14 (port 3.5)จะเป็น 1 แต่ถ้าล้มเหลว bit ที่ 14

จะเป็น 0

จุดที่น่าสนใจการทดสอบนี้คือ จะไม่ตรวจสอบโดยการเขียนไบท์ address ตามบา address ที่ละบา อย่างการทดสอบ ROM ถ้าหากว่า address ทุกเส้นใช้งานได้การทดสอบจะทำงานได้ program ทดสอบ RAM มีดังนี้

```
ADDRESS  MNEMONIC              COMMENT
         .equ ramstart,0000h ;set RAM test start address
         .equ rmstphi,20h    ;set RAM test high stop address
         .equ pattern,55h    ;determine test pattern
         .equ good,20h       ;RAM good pattern P3.5 = 1
         .equ bad,0dfh       ;RAM bad pattern, P3.5 = 0
         .org 0000h          ;begin test program at 0000h
         mov p3,#0ffh        ;set port 3 high
         mov dptr,#ramstart ;initialize DPTR
test:    mov a,#pattern      ;set pattern byte
         movex @dptr,a       ;write byte to RAM
         inc dptr            ;point to next RAM byte
         mov a,#rmstphi      ;check to see if at stop address
         cjne a,dph,test       ;if not then loop until done
         move dptr,#ramstart  ;start read-back test
check:   movx a,@dptr          ;read byte from RAM
         cjne a,#pattern,fail ;test against what was written
         inc dptr           ;go to next byte if tested ok
         mov a,#rmstphi     ;check to see if all bytes tested
         cjne a,dph,check   ;if not then check again
         mov p3,#good       ;checked ok, set Port 3 to good
```

```
here:   sjmp here          ;stop here
fail:   mov p3,dph         ;test failed, get address
        anl p3,#bad        ;set p3.5 to zero
        mov p1,dpl         ;set Port 1 to low address byte
there:  sjmp there         ;stop there
        .end
```

ข้อสังเกตุ เปลี่ยน ramstart และ rmstphi เพื่อทดสอบส่วน 2000h – 3FFFh , 4000h –5FFFh และ 6000h – 7FFFh การทดสอบหน่วยความจำจะทดสอบเป็นส่วน ๆ คือ (20)00,(40)00,(60)00, และ (80)00hใน 8051ไม่มีคำสั่ง halt แต่สามารถใช้การกระโดดเป็น loop อยู่กับที่แทนได้

ขณะนี้เราทดสอบวงจรภายนอกที่ต่อกับ 8051 หมดแล้วและ subroutine ต่อไปนี้จะถูกใช้อีกในบทต่อไป

Timing Subroutine subroutine ที่ถูกเรียกใช้ โดย program หลักจะต้องเข้าใจง่ายและไม่ทำให้สภาพของ program หลัก เปลี่ยนไป โดยปกติแล้ว การใช้ subroutine ต้องไม่มีผลต่อข้อมูล ภายหลังจาก subroutine ทำงานเสร็จ

subroutine ต้องเก็บค่าในตำแหน่งต่าง ๆ ของระบบและสามารถนำค่าเหล่านั้นกลับมาอีกครั้งเมื่อ subroutine ทำงานเสร็จ การตกหล่นของข้อมูลที่เก็บจะมีผลทำให้ program หลัก เสียหายเพราะ program หลัก จะสมมุติว่า ทุก ๆ สิ่งเหมือนเดิมทั้งก่อนละหลังการเรียกใช้ subroutine

สุดท้ายจำเป็นต้องมีคำอธิบาย subroutine เพื่อสามารถใช้มันอย่างมีประสิทธิภาพ

Time Delays บางที subroutine ที่ใช้มากที่สุดก็คือ การหน่วงเวลาซึ่งอาจใช้softwareวนลูป หรือใช้ Timer นับ clock ภายในก็ได้

hardware Timer อาจใช้ได้ทั้งใน software หรือ hardware mode ใน software mode ตัว program จะตรวจสอบการเกิด overflow

ของ Timer จึงจะ กระโดดออกจาก loop เมื่อ overflow ถูก set ส่วน hardware mode จะใช้โครงสร้างของ 8051 รับ interrupt จาก Timer overflor และส่งมา interrupt program อีกที่หนึ่ง

วิธีการ interrupt ถูกใช้ในกรณีที่มีงานอื่นรออยู่ ทำให้สามารถทำงานอื่นต่อไปได้ ขณะที่อยู่ในช่วงเวลา delay แต่การใช้ Software ในการหน่วงเวลาและใช้ softwareร่วมกับ Timer ในการหน่วงเวลาตัวควบคุมจะไม่สามารถไปทำงานอื่นได้ขณะที่ทำการหน่วงเวลา

ถ้าใช้การ interrupt mode ตัว programจะต้องมีส่วนของ routine เพื่อให้ intetrupt vector เข้ามาทำงาน program ต้องมี interrupt control registor และวิธีนี้ ผู้ใช้จะต้องเขียน subroutine ขึ้นมาเอง

<u>การหน่วงเวลาโดยใช้ Software</u> คือ subroutine คือ "softime"โดยมีเวลาในการหน่วงตั้งแต่ 1-65535 ms โดยใช้ Register R7 ในการกำเนิดเวลาการหน่วงพื้นฐาน 1 ms ก่อนการเรียกใช้ ต้อง load ค่าเวลาที่ต้องการหน่วงลงใน Register A (LSB) และ RegisterB (MSB)

ค่าเวลาการหน่วงที่กล่าวไว้มาจากการทำงานในแต่ละ Machine cycle (12clockpulses) สำหรับ crystal ความถี่ 16 MHZ จะได้

$$\text{Cycle Time} = \frac{12 \text{ pulses}}{16{,}000{,}000 \text{ pulses/s}} = 0.75\ \mu s$$

การเปลี่ยนความถี่ของ crystal จะทำให้ค่าเวลาการหน่วงใน sub-routine เปลี่ยนไปด้วย

<u>Softime</u> จะทำการหน่วง จาก 1-65535 ms ตามค่าที่อยู่ใน Register A (LSB) และ B (MSB) ก่อนการใช้ Subroutine จะต้อง load ค่าลงใน Register A และ B ถ้า load ค่าศูนย์ลงใน Register ทั้งสองจะมีผลทำให้ Subroutine ส่งการทำงานกลับให้ programหลัก ทันที

ตัวเลขที่อยู่หลังcommaจะแสดงถึงจำนวนครั้งในการวน loop หน่วงเวลา

```
ADDRESS    MNEMONIC          COMMENT
           .equ delay,0ech   ;for 996 μs time delay = 222d
softime:   .org 0000h        ;set origin
           push 07h          ;save R7
           push acc          ;save A for A = B = 00 test
           orl a,b           ;will be 00 if both 00
           cjne a,#00h,ok    ;return if all 00
           pop acc           ;keep stack balanced
           sjmp done
ok:        pop acc           ;not all zeroes, proceed
timer:     mov r7,#delay     ;initialize R7, 1
onemil:    nop               ;tune the loop for 6 cycles, 1
           nop               ;this makes 2 cycles total, 1
           nop               ;3 cycles total, 1
           nop               ;4 cycles total, 1
           djnz r7,onemil    ;count R7 down; 6 cycles total, 2
;
;total delay is 6 cycles (4.5 μs) x 222d = 999d μs
;
           nop               ;tune subroutine .75 μs more
;
;total delay is 999.75μs,which is as close as possible for the
;frequency used (1000 μs = 4000/3 cycles)
;
           djnz acc,timer    ;count A and B down as one
```

```
        cjne a,b,bdown   ;A = 00, count B down until = 00

        sjmp done        ;if so then delay is done
bdown:  dec b            ;count B down and time again
        sjmp timer
done:   pop 07h          ;restore R7 to original value
        ret              ;return to calling routine
        .end
```

ข้อสังเกต Registor A ใน program จะถูกกำหนดให้ชื่อว่า "A" ซึ่งมันจะเป็น Registor ตัวเดียวกับ ACC ซึ่งใช้ในการบวกมวลผลและยังกำหนดชื่อให้ Registor R7 ซึ่งชื่อเหล่านี้จะใช้แทน เพื่ออ้างถึง Registor ที่กล่าวมาแล้ว

เมื่อกำหนดให้ A = B =00 แล้วในความเป็นจริง program เริ่มนับ A จาก 00...FFh...00 แล้วออก ถ้ามันถูกต้องการให้สามารถใช้เงื่อนไขนี้สำหรับA และ B ดังนั้นเงื่อนไขศูนย์สามารถทดสอบจากการใช้ค่า 0000 และดูการ setของ flag เมื่อค่า B ลดลงจาก 00 เป็น FFh

ตามความเป็นจริงแล้ว เวลาที่น้อยที่สุดในการหน่วงคือ 1 ms และเวลาที่ใช้ใน subraoutine เป็นเวลา 1014.75 µs หรือมีค่าผิดพลาด 1.5 %

การหน่วงเวลาโดยใช้ softrare ร่วมกับ Timer การหน่วงเวลาโดยใช้ Timer และมีส่วนsoftware คอยตรวจสอบ flag ซึ่งจะ set เมื่อ Timer ทำงานเสร็จ การใช้งานจะเหมือนกับการหน่วงเวลาโดยใช้ software คือต้องกำหนดค่าในการหน่วงใน Register A และ Bและมีช่วงเวลา 1-65535 ms

ความถี่ของ clock สำหรับ Timer คือความถี่ crystal หารด้วย 12 หรือ 1 machine cycle ซึ่งมีคาบเวลาเท่ากับ 0.75 ms สำหรับ Crystal 16 MHZ และการนับ 1ms กำหนดโดย

การนับสาหรับ 1000 μs = 1000/0.75 = 1333.33 (1333) การกาหนดครั้งนการนับที่ได้นี้ จะมีค่าผิดพลาดบ้าง ถ้าเวลานการหน่วงสาคัญมาก เราจะต้องเลือก crystal ความถี่ 12 MHz นการกาเนิด Time delay สาหรับใช้เป็น clock เพื่อหน่วงเวลา

Timer 0 จะถูกกาหนดให้นับ 1333 ลูก (0535h) จาก clock ภายนเพื่อกาเนิดเวลาหน่วง พื้นฐาน คือ 1 ms Register A และ B จะถูกนับลงเมื่อ T0 เกิด overflow เนื่องจาก Timer จะนับขึ้นเพราะฉะนั้น เราต้องหาค่า two complement ของ 0535h เพื่อกาหนดให้เป็นค่าเริ่มต้นนการนับของ Timer

Timer การหน่วงเวลาโดยใช้ Timer มีชื่อว่า "Timer" ใช้ Timer 0 และ Register A,B เพื่อกาเนิดเวลานการหน่วงระหว่าง 1-65535 ms โดยค่านใน Register A เป็น (LSB)และ B เป็น (MSB) และเมื่อ load ค่า 0000h จะมีผลให้การทางานกลับมาอยู่ที่ programหลักทันที

```
ADDRESS   MNEMONIC          COMMENT
          .equ onemshi,0fah ;2's complement of 535h = FACBh
          .equ onemslo,0cbh
          .org 0000h        ;set program origin
timer:    push t10          ;save timer 0 contents
          push th0
          cjne a,#00h,go    ;test for A = 00
          orl a,b           ;A = 00, test for B = 00
          jz done           ;A will be 00 if A = B = 00
          clr A             ;B is not 00, clear A
go:       anl tcon,#0cfh    ;clear timer 0 overflow and run
```

```
                                ;flags in TCON
            anl tmod,#0f0h      ;clear T0 part of TMOD,set T0 for
            orl tmod,#01h       ;timer operation, mode 1 (16 bit)
onems:      mov tl0,#onemslo    ;set T0 to count up from FACBh
            mov th0,#onemshi
            orl tcon,#10h       ;start timer 0
wait:       jbc tf0,dwnab       ;poll T0 overflow flag
            sjmp wait           ;loop until T0 overflow
dwnab:      anl tcon,#0efh      ;stop T0
            djnz acc,onems      ;count A down and loop until zero
            cjne a,b,bdown      if A = B = 00 then done, return
            sjmp done
bdown:      dec b               ;decrement B and count again
            sjmp onems
done:       pop th0             ;restore T0 contents
            pop tl0
            ret

            .end
```

<u>ข้อสังเกต</u> คำนวน T0 ไม่สามารถที่จะกำหนดให้มีความถูกต้องของเวลาได้อย่างแม่นยำเช่นนี้แล้ว จะไม่ต้องการเก็บ TCON และ TMOD bit สำหรับ T0 เพราะ T0 สามารถถูกใช้เพื่อเก็บข้อมูล โดยมันจะเป็นตัวเก็บเอง

program นี้ จริง ๆแล้วจะไม่ดีกว่า program softime เพราะทั้งคู่จะใช้เวลาของprocessor แต่ program timer เด่นกว่าเล็กน้อยในด้านความยืดหยุ่นของการเปลี่ยนแปลงค่าที่จะ load ให้ T0 ทำให้สามารถเปลี่ยนฐานเวลาได้

ง่าย

หนทางหนึ่งเพื่อให้ program ทำงานไปในขณะที่ Timer ยังทำงานไม่เสร็จก็คือการกำหนดเวลากลับไปตรวจ overflow เป็นช่วง ๆ ซึ่งถ้าช่วงเวลาที่กลับไปตรวจ overflow มีค่าน้อย เช่นนี้แล้วค่าผิดพลาดก็จะน้อยมาก

<u>การหน่วงเวลาโดยใช้ Hardware</u> ถ้าการหน่วงเวลานาน ๆ หรือ processorไม่สามารถสูญเสียเวลาแม้เป็นช่วงเวลาสั้น ๆ ดังนั้น จำเป็นต้องใช้ interrupt mode ดังนี้

1. การเกิด overflow จะไป interrupt processor ซึ่งแสดงว่า hardwareเรียก subroutine ที่มีตำแหน่งกำหนดไว้แน่นอนใน ROM มาทำงาน

2. สามารถจะกำหนด subroutine โดยไม่สนใจ เวลาที่ใช้ในการ delay (ถ้าไม่มี ,การทำงานจะส่งคืนให้กับที่ ๆ ถูก interrupt)

เวลาในการหน่วงกำหนดโดยผู้ใช้ program ที่จะเก็บค่าที่ต้องการหน่วงไว้ใน RAMภายนอกในตำแหน่งที่ชื่อว่า "savetime" และเรียก "startime" ซึ่งจะ set จังหวะในการนับ และ program หลักจะทำงานในขณะที่ยังเป็นช่วงการหน่วงเวลา

รูปแบบของ proguam ต้องมี rortines เก็บไว้ใน ROM เพื่อให้ program มาทำงานที่ตำแหน่งนี้ เมื่อเกิดการ interrupt ซึ่งผู้ใช้ต้องกำหนด routine นี้ก่อนจะมีการใช้การหน่วงเวลา

ในตัวอย่างนี้ มี 3 subroutine ที่ตำแหน่ง interrupt ของ ROM

1. Hardtime : Subroutine ถูกกำหนดโดยไม่คำนึงถึงเวลาในการหน่วงจะเสร็จสิ้นเมื่อใด (ถ้าเวลายังไม่สิ้นสุด ดังนั้น subroutine ก็จะคืนกลับไป program หลักของผู้ใช้ในตำแหน่งที่มันถูก interrupt โดย timer flag ถ้าเวลามากขึ้นมันจะเรียกผู้ใช้ program"Usertime")

2. Usertime : subroutime ถูกเขียนโดยผู้ใช้ตามความต้องการเวลาการหน่วง(สำหรับตัวอย่างนี้ เป็นsubroutime ง่าย ๆ คือการส่งกลับ)

3. Stoptime : subroutime นี้จะหยุด Timer

Note ตัว assembler นี้สามารถกำหนด 1 label ได้ทั้งอักษรตัวใหญ่และตัวเล็ก เช่นตัวอย่างHARDTIME, hardtime, และ HaRdTiMe จะมีความหมายเหมือนกัน

การหน่วงเวลาโดยใช้ hardware ในที่นี้จะใช้ Timer 1 สำหรับกำเนิดเวลาพื้นฐานเมื่อ Timer 1 เกิด overflow จะมีผลไป set flag และ program จะส่งไปที่ตำแหน่ง001Bh ใน program memory ถ้า bit ในinterrupt cortrol register IE และ IP ถูก set

ตัวอย่างนี้ผู้ใช้สามารถ set Timer 1 สำหรับการหน่วงเวลาตั้งแต่ 1-65535 msโดยการกำหนดไว้ใน RAM ภายนอก ในตำแหน่ง "savetime" (LSB) และ "savetime"+1(MSB) ส่วน Regidter A และ B จะไม่ถูกใช้ เนื่องจากเป็นความต้องการของ program ที่จะกัน Register 2 ตัวนี้ไว้

การหน่วงเวลาโดย hardware ถูกเรียกว่า "Hardtime" และเพื่อหลีกเลี่ยง ความสับสนใน subroutine กับ ส่วน program ของผู้ใช้ เพราะฉะนั้นในส่วนของผู้ใช้จะนำหน้าด้วยคำว่า "User"

Hardtime sudroutine "hardtime" นี้จะใช้ hardware เพียงอย่างเดียวในการกำหนดเวลาในการหน่วง เพื่อเริ่มการทำงาน IE.7 และ IE.3 (EA และ ET1) ต้องถูก set และ subroutine "startime" ถูกรวมโครงสร้าง 3 ส่วนถูกไว้เพื่อใช้สำหรับ Timer 1 ตำแหน่งที่ 001Bh คือ LJMP hardtime, ACALL usertime (กับ label "Userdly"), และ ACALL stoptime สิทธิพิเศษของ interrupt คือสามารถ set bit IP.3 (PT1)เป็น high (1) หรือ low (0) โดยรายละเอียดของ program แสดงไว้ข้างล่าง

```
ADDRESS    MNEMONIC                 COMMENT
           .equ savetime,0010h ;external RAM address for delay
userpgm:   .org 0000h          ;start user program
           sjmp userover       ;jump over interrupt address
```

```
            .org 001bh              ;interrupt location for TF1
            ljmp hardtime           ;jump to time delay subroutine
userdly:    acall usertime          ;called if delay is up
            acall stoptime          ;dissable timer interrupt
            reti                    ;return to main program
userover:   mov dptr,#savetime      ;point to delay address
            mov a,#01h             ;store desired delay, LSB first
            movx @dptr,a
            inc dptr                ;point tot next byte (MSB)
            mov a,#10h
            movx @dptr,a            ;desired delay now stored
            orl ie,#88h             ;enable T1 and all interrupts
            acall startime          ;start time delay
here:       sjmp here               ;loop to simulate user program
;
;the user program now continues while timer 1 runs until TF1
 =1
;the  interrupt  generated will vector to location 001Bh and
 execute
;a jump  to  hardtime  that  will decrement the contents for
 savetime
;until  the desired  time delay has been done; hardtime will
 return to
;the main program if the delay is not finished,or to userdly
 if the
;delay is up; userdly returns to call stoptime , which stops
```

```
 the timer
;and  returns to the RETI instruction for return to the main
 program
; .
startime: mov th1,#0fah   ;set T1 for a 1 ms delay
          mov tl1,#0cbh   ;(see TIMER example)
          anl tmod,#0fh   ;clear T1 part of TMOD
          orl tmod,#40h   ;set T1 to timer mode 1
          orl tcon,#40h   ;start timer 1
          ret             ;return to calling program
hardtime: push acc        ;save registers to be used
          push dph
          push dpl
          mov dptr,#savetime  ;get pointer to time delay
          movx a,@dptr        ;count delay number down to 0000
          dec a               ;low byte first
          cjne a,#00h,aff     ;check for 0000
          movx @dptr,a        ;save low byte = 00
          inc dptr            ;get high byte and look for 00
          movx a,@dptr
          jz done             ;done if low, high byte = 0
          sjmp sava           ;not 0,  delay again
aff:      cjne a,#0ffh,sava   ;if low byte = FF dec high
          movx @dptr,a          ;save low byte = FF
          inc dptr              ;point to high byte
          movx a,@dptr          ;count high byte down
```

```
            dec a
            sjmp sava           ;save the high byte
done:       pop dpl             ;finsihed, jump to userdly
            pop dph             ;restore all registers used
            pop acc
            ljmp userdly        ;continue at user delay
sava:       movx @dptr,a        ;delay not up, save byte
            pop dpl             ;restore saved registers
            pop dph
            pop acc
            acall startime      ;start T1 for next 1 ms
            reti                ;return to user program
;
; the user program "usertime" can now be written as needed;
  a return
;will be used to simulate the user routine
;
usertime: ret
;
;after the user program is done then "stoptime" will stop
 timer T1
;and return to the interrupted main program
;
stoptime: anl tcon,#0bfh            ;stop timer T1
          ret                       ;return to reti
         .end
```

ข้อสังเกต เวลาที่น้อยที่สุดในการหน่วงคือ 1 ms เพราะเป็นเวลาที่น้อยที่สุดใน cycle การinteruptถ้าที่ว่างของ stack ถูกจากัด เวลาทั้งหมดใน routime สามารถจะรวบรวมที่ interupt ตาแหน่ง 001Bh
คาสั่ง RETI ถูกใช้ในการส่งการทางานคืนให้ program หลัก หลังจากถูก interrupt ขณะที่
คาสั่ง RET ถูกใช้ในการส่งคืนจากการเรียก routine
ในที่นี้จะไม่มีการตรวจสอบค่าเริ่มแรก ของ 0000h

Look up Table สาหรับ 8051 มีหลายกรณีในการคานวณเพื่อเปลี่ยนเลขค่าหนึ่งไปเป็นอีกค่าหนึ่ง ตัวอย่างที่ว ๆ เป็นการเปลี่ยนตัวอักษร ASCII สาหรับเลข 0-9 ไปเป็นเลข birary (BCD) ซึ่ง ASCII 30h = 00d, 31h = 01d, จนกระทั่ง 39h จะเท่ากับ 09d

วิธีที่เข้าใจง่ายวิธีหนึ่งในการเปลี่ยนจาก ASCII ไปเป็น BCD คือการลบค่า ASCIIด้วย 30h จะได้เลข BCD ออกมา วิธีนี้ก็มีการใช้ตารางเก็บค่าของเลข BCD ไว้ใน ROM ตาม address ที่สัมพันธ์กับค่า ASCII ค่าที่บรรจุใน address จะถูกชี้ โดยตัวอักษร ASCII และตัวอักษร ASCII เรียกว่า "looked up" ซึ่งตัวมันจะใช้แทนเลข BCD

สาหรับในตัวอย่างนี้จะใช้ตัวอักษร ASCII 30h-39h เราสามารถสร้างจากprogramที่ตามมานี้ ที่ address จะแสดงค่าให้รู้ด้วยคาสั่ง .db

```
ADDRESS    MNEMONIC     COMMENT
           .org 1030h   ;start table at ROM location 1030h
           .db 00h      ;location 1030h contains 00 BCD
           .db 01h      ;location 1031h contains 01 BCD
           .db 02h      ;location 1032h contains 02 BCD
           .db 03h      ;location 1033h contains 03 BCD
```

```
.db 04h      ;location 1034h contains 04 BCD
.db 05h      ;location 1035h contains 05 BCD
.db 06h      ;location 1036h contains 06 BCD
.db 07h      ;location 1037h contains 07 BCD
.db 08h      ;location 1038h contains 08 BCD
.db 09h      ;location 1039h contains 09 BCD
```

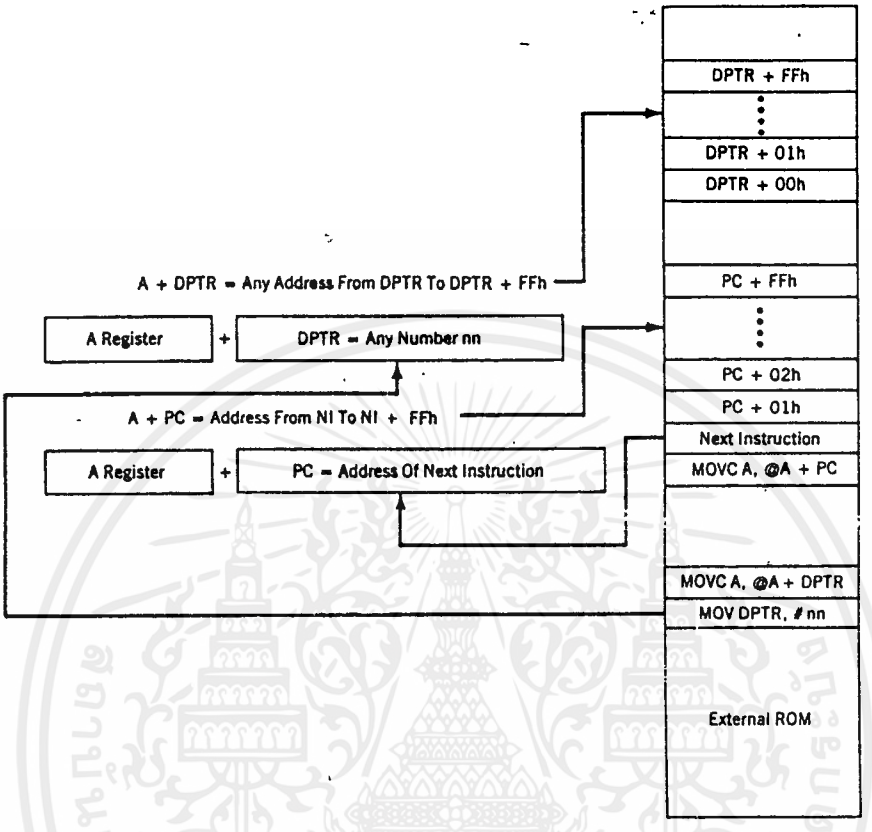
ในแต่ละ address byte ต่า จะเป็นตัวยักษร ASCII ซึ่งจะเก็บค่าที่เท่ากันของ BCD ไว้ ถ้า DPTR ถูก load ด้วยค่า 1000h และ A ถูก load ด้วยค่าที่กำหนดจาก ASCII ดังนั้น MOVC A,@ A + DPTR จะเคลื่อนที่ไบเท่ากับ BCD byte สำหรับ ค่าใน A

lookup table อาจใช้เพื่อหาค่าที่ยุ่งยาก เช่น การแปลงตรีโกณและ exponentialโดยlookup table ต้องการเพียงพื้นที่ใน ROM ซึ่งทำให้การแปลงค่าได้เร็วกว่าใช้การคำนวณ

8051 จะสร้าง Lookup table จากคำสั่งง่าย ๆ คือ MOVC A,@A + DPTR และ MOVC A,@A + PC จากทั้ง 2 กรณีต้องใช้ pointer หรือค่าที่คำนวณจาก pointer ซึ่งถูกเรียกว่า "offset" และ DPTR หรือ PC คือ "base" ของ address ในตัวอย่างมีฐานของaddress คือ 1000h และมี offset เป็นเลขจาก 30h-39h

การใช้ PC จะถูกใช้เฉพาะแห่งหรือใช้สำหรับตารางเล็ก ๆ แต่ DPTR จะถูกใช้ในตารางใหญ่ ๆ ที่รวบรวมจากรหัสต่าง ๆ ใน program

ในทั้ง 2 กรณี ต้องการที่จะหาข้อมูลใน address ของ ROM ซึ่ง address จะเท่ากับ base + offset รูป 6 แสดงการหา address ใน lookup table ที่คำนวณจาก Register ฐานทั้งสอง



ข้อจำกัดของ lookup table เกิดจากค่าที่แตกต่างกันได้เพียง 256 ค่า ซึ่ง มันจะเป็นตัวจำกัดขนาดของตาราง แต่ข้อจำกัดนี้สามารถใช้เทคนิคการเปลี่ยนฐาน address เมื่อมีการเพิ่มของ offset 256 byte แล้ว ดังนั้น offset A ที่เหมือนกันสามารถ เป็นตัวชี้ข้อมูลในตารางที่ต่าง address กันได้จากการเปลี่ยนฐาน address ตัวอย่าง เช่น การเปลี่ยนค่า DPTR จาก 1000h เป็น 1100h ในตารางเปลี่ยน ASCII เป็น BCD

ทั้ง PC และ DPRT ที่เป็นฐาน address จะเสนอตัวอย่างตามมาทีหลัง

<u>PC ทำหน้าที่เป็นฐาน</u> สมมุติว่า A เป็นตัวเลขระหว่าง 00h และ 0fh สำหรับค่ากลังสองของ A สามารถทำได้จากการ load ค่า A ลงใน B แล้วใช้คำสั่ง MUL AB เพื่อสร้างเป็น lookup table

ตารางไม่สามารถถูกวางไว้หลังคำสั่ง MOVC ต้องมีคำสั่งกระโดดอยู่ระหว่าง MOVC และตาราง หรือ program จะ fetch ข้อมูล byte แรกและทำการ execute มาเป็นรหัสจากไว้ว่าค่า PC ยังคงเป็นค่าของคำสั่งกระโดด (คำสั่งถัดจากคำสั่ง MOVE) เมื่อค่า address ของตารางถูกคำนวณ

PClook program "pclook" จะใช้ PC เป็นฐานและค่า offset คือ A หลังจากคำสั่งMOVC จะทำให้ Register A จะมีค่าเป็นกำลังสองของค่าเดิม

```
ADDRESS    MNEMONIC        COMMENT
           .org 0000h
pclook:    mov a,#0ah      ;find the square of 0Ah (64h)
           add a,#02h      ;adjust for two byte sjmp over
           movc a,@a+pc    ;get aquivalent data from table to A
           sjmp over       ;jump over the lookup table
;
;the lookup table is inserted here , at PC + 2. (PC = 0005h)
;
           .db 00h         ;begin table here, 00^2 = 00
           .db 01h         ;01^2 = 01d
           .db 04h         ;02^2 = 04d
           .db 09h         ;03^2 = 09d
           .db 10h         ;04^2 = 16d
           .db 19h         ;05^2 = 25d
           .db 24h         ;06^2 = 36d
           .db 31h         ;07^2 = 49d
           .db 40h         ;08^2 = 64d
           .db 51h         ;09^2 = 81d
```

```
        .db 64h         ;0A^2 = 100d
        .db 79h         ;0B^2 = 121d
        .db 90h         ;0C^2 = 144d
        .db 0a9h        ;0D^2 = 169d
        .db 0c4h        ;0E^2 = 196d
        .db 0e1h        ;0F^2 = 225d
over:   sjmp over       ;simulate rest of user program
        .end
```

รูป 7 แสดงรายละเอียดของ program ที่ละ address

<u>ข้อสังเกต</u> ค่าที่จะนำไปรวมกับค่า A มาจากค่าของ byte ในการใช้คำสั่ง SJMP ถ้าหากผลที่นำมาบวกมีค่ามากจะส่งผลให้เกิด overflow ขึ้นใน Register A เมื่อผลบวกเกินกว่า 255dนั้นคือการจากัดของค่าในตาราง

FIGURE 7 Lookup Table using the PC

```
0000            .org 0000h
0000 740A pclook: mov a,#0ah  ;find the square of 0Ah (64h)
0002 2402         add a,#02h  ;adjust for two byte sjmp over
0004 83           movc a,@a+pc ;get aquivalent data from
                               ;table to A
0005 8010         sjmp over   ;jump over the lookup table
0007       ;the lookup tabel is inserted here ,
           at PC + 2. (PC = 0005h)
0007 00    .db 00h            ;begin table here, 00^2 = 00
0008 01    .db 01h            ;01^2 = 01d
```

```
0009 04     .db 04h             ;02^2 = 04d
000A 09     .db 09h             ;03^2 = 09d
000B 10     .db 10h             ;04^2 = 16d
000C 19     .db 19h             ;05^2 = 25d
000D 24     .db 24h             ;06^2 = 36d
000E 31     .db 31h             ;07^2 = 49d
000F 40     .db 40h             ;08^2 = 64d
0010 51     .db 51h             ;09^2 = 81d
0011 64     .db 64h             ;0A^2 = 100d
0012 79     .db 79h             ;0B^2 = 121d
0013 90     .db 90h             ;0C^2 = 144d
0014 A9     .db 0a9h            ;0D^2 = 169d
0015 C4     .db 0c4h            ;0E^2 = 196d
0016 E1     .db 0e1h            ;0F^2 = 225d
0017 80F  over: sjmp over  ;simulate rest of user program
0019            .end
```

DPTR ทำหน้าที่เป็นฐาน ตัวอย่างต่อไปจะใช้ DPTR ในการสร้างตาราง การเลือกจากค่าของ A ที่ต้องเป็นค่าที่น้อยกว่า 10h ออกทางหาค่า A อยู่ระหว่าง 00h-FFh และ ค่ากำลังสองของเลขที่มากกว่า 0Fh ผลของมันต้องเก็บไว้ในขนาด 4 byte ซึ่งจะเก็บใน Register R0(LSB) และ R1 (MSB)

สองตารางจะถูกสร้างขึ้นมาในแต่ละส่วน คือส่วนของ LSB และส่วนของ MSB ซึ่ง ใช้DPTR เป็นตัวชี้ของทั้งสองตาราง การ set ค่าในตารางทั้ง2 จะไม่แสดงไว้ในตัวอย่างนี้ จะแสดงค่าของจริงร่างตารางเท่านั้น

Dplook program "dplook" จะใช้หาค่ากำลังสองของค่าใน Regidter A

ซึ่งผลลัพธ์แสดงเป็น R0 (LSB) และ R1(MSB) Register A จะเก็บค่าใน R1 เป็นค่าคราวซึ่งใช้เป็นตัวชี้ MSB byte

```
ADDRESS   MNEMONIC              COMMENT
          .equ lowbyte,0200h    ;base addresss of LSB table
          .equ hibyte,0300h     ;base address of MSB table
          .org 0000h
          dplook:mov a,#5ah     ;find the square of 5Ah (1FA4h)
          mov r1,a              ;store A for later use
          mov dptr,#lowbyte     ;set DPTR to base address of LSB
          movc a,@a+dptr        ;get LSB
          mov r0,a              ;store LSB in R0
          mov a,r1              ;recover A for pointing to MSB
          mov dptr,#hibyte      ;set DPTR to base address of MSB
          movc a,@a+dptr        ;get MSB
          mov r1,a              ;store MSB in R1
here:     sjmp here             ;simulate rest of user program
          .org lowbyte          ;place LSB table starting here
          .db 00h               ;00^2 = 0000
          .db 01h               ;01^2 = 0001
;place rest of table up to the LSB of 59^2 here
          .org lowbyte + 5ah        ;put LSB of 5A^2 here
          .db 0a4h                  ;LSB is A4h
;place rest of LSB table here
          .org hibyte           ;place MSB table starting here
```

```
            .db 00h                     ;00^2 = 0000
            .db 01h                     ;01^2 = 0001
;place rest of table up to the MSB of 59^2 here
            .org hibyte + 5ah           ;put MSB of 5A^2 here
            .db 1fh                     ;MSB is 1Fh
;place rest of MSB table here
            .end
```

<u>ข้อสังเกต</u> ไม่มีคำสั่งกระโดดไปเหนือตาราง เพราะบกติตารางจะวางไว้ท้าย program ไม่ต้องการการบรับค่า เพราะ DPTR จะเป็นค่าคงที่

รูป 8 แสดงรายละเอียดของ program ทีละ address

FIGURE 7.8 Lookup Table using the DPTR

```
0200    .equ lowbyte,0200h      ;base addresss of LSB table
0300    .equ hibyte,0300h       ;base address of MSB table
0000    .org 0000h
0000 745A  dplook: mov a,#5ah   ;find the square of 5Ah
                                ;(1FA4h)
0002 F9        mov r1,a         ;store A for later use
0003 900200    mov dptr,#lowbyte ;set DPTR to base address
                                ;of LSB
0006 93        movc a,@a+dptr   ;get LSB
0007 F8        mov r0,a         ;store LSB in R0
0008 E9        mov a,r1         ;recover A for pointing
```

```
                                    ;to MSB
0009 900300  mov dptr,#hibyte       ;set DPTR to base address
                                    ;of MSB
000C 93      movc a,@a+dptr         ;get MSB
000D F9      mov r1,a               ;store MSB in R1
000E 80FE  here:  sjmp here         ;simulate rest of user
                                    ;program
0200         .org lowbyte           ;place LSB table starting
                                    ;here
0200 00      .db 00h                ;00^2 = 0000
0201 01      .db 01h                :01^2 = 0001
0202       ;place rest of table up to the LSB of 59^2 here
025A         .org lowbyte + 5ah     ;put LSB of 5A^2 here
025A A4      .db 0a4h               ;LSB is A4h
025B       ;place rest of LSB table here
0300         .org hibyte            ;place MSB table starting
                                    ;here
0300 00      .db 00h                ;00^2 = 0000
0301 01      .db 01h                ;01^2 = 0001
0302       ;place rest of table up to the MSB of 59^2 here
035A         .org hibyte + 5ah      ;put MSB of 5A^2 here
035A 1F      .db 1fh                ;MSB is 1Fh
035B       ;place rest of MSB table here
035B                 .end
```

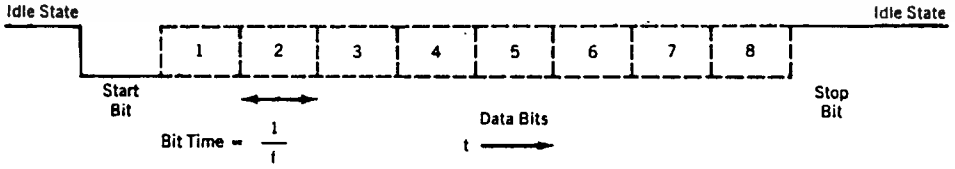
<u>การส่งผ่านข้อมูลแบบอนุกรม</u> ลักษณะการทำงานที่เกิดขึ้นพร้อม ๆ กัน ของระบบที่

เป็นเครือข่ายหรือ LAN ทำให้เกิดความซับซ้อน เช่นปัญหาในการแลกเปลี่ยนข้อมูลด้วย port I/O หรือการเชื่อมต่อกันของ processor หลาย ๆ ตัว ซึ่งก่อให้เกิดปัญหาในการจัดสรรอุปกรณ์ ที่ต้องใช้งานร่วมกัน และการส่งผ่านข้อมูล

จากความต้องการในการส่งหรือรับข้อมูลจากตำแหน่งอื่น ความยาวของระยะทางเป็นปัจจัยที่ทำให้ราคาของสายสูงขึ้น ดังนั้นการส่งข้อมูลแบบอนุกรมที่ทำการส่งข้อมูลที่ละ bit จึงทำให้ราคาของสายลดลงได้ แต่ถ้าความเร็วในการส่งข้อมูลเป็นปัจจัยที่สำคัญควรจะเลือกการส่งข้อมูลแบบขนานผ่านทางเส้นใยแสง ซึ่งจะมีราคาสูง

การใช้ chip ซึ่งทำหน้าที่ส่งข้อมูล และรับข้อมูล แบบอนุกรม อย่างแพร่หลาย บรากฏขึ้นในปี 1970 ซึ่ง chip นั้นเรียกว่า "univeral asynchronous reseicer transm-itters"หรือ UARTS แสดงให้เห็นว่าการสื่อสารแบบอนุกรมนิยมใช้กันมาก จนถึงปัจจุบันก็ยังคงใช้กันอยู่ อัตราการส่งข้อมูล 8bit จะถูกกำหนดไว้ในช่วง 300-19200 bit ต่อวินาที

การส่งข้อมูลแบบ asychronous จะใช้ start bit และ stop bit มาช่วย แสดงให้เห็นในรูป 9 แสดงการเตรียมพร้อมที่จะรับข้อมูลที่ส่งมาและบอกจุดจบของข้อมูล bit ที่นอกเหนือจากข้อมูลเหล่านี้ จะเป็นตัวประกอบทำให้การส่งข้อมูลช้าลง ซึ่งมีการสนับสนุน และ พัฒนาเป็นการส่งข้อมูลแบบ synchronous ซึ่งการส่ง และการรับ จะมีรูปแบบพิเศษ ที่จะเริ่มส่งข้อมูลและมีข้อมูลเป็นจำนวนมากตามมา และจะจบการส่งข้อมูลด้วยสัญญาณที่มีรูปแบบพิเศษเช่นกัน โดยปกติจะต้องมีการตรวจสอบตัวอักษรเหล่านั้นด้วย



ในแต่ละแบบจะมีจุดเด่น สาหรับการส่งข้อมูลที่สั้นหรือนาน ๆ ส่งข้อมูลที่ asynchro-nous mode จะดีที่สุด และสาหรับการส่งข้อมูลยาว ๆ synchronous mode จะเหมาะสมกว่า

ใน 8051 จะมี Serial port อยู่ภายใน ซึ่งสามารถกาหนดการทางานในแบบ asynchronousได้ทั้งหมด 4 รูปแบบ คือ mode 0-3 รูปแบบหนึ่งของทั้งหมดคือ mode 1เป็นมาตราฐาน สาหรับ UART mode และอีก 3 mode ซึ่งเป็นการติดต่อแบบ asynchronous ที่ง่าย จะถูกพัฒนาไว้ในที่นี้ สาน program ที่ซับซ้อน

<u>การส่งตัวอักษรโดยใช้ Time delay</u> บ่อยครั้งที่มีการส่งข้อมูลจากตัวควบคุมไปยังสาน output เช่น display หรือ printer แต่ละตัวอักษรที่ส่งไปจะใช้เวลาตั้งแต่ 33.3 - 0.5 ms ขึ้นอยู่กับอัตรา baud rate ที่เลือกใช้ ตัว program ต้องคอยจนกว่าอักษรที่ส่งไปก่อน ถูกรับแล้วจึงส่งอักษรตัวถัดไปเพื่อไม่ให้เกิดการสูญเสียข้อมูลระหว่างการส่ง วิธีที่ง่ายในการป้องกันการสูญหาย คือการหน่วงเวลาซึ่งต้องรู้เวลาในการส่งข้อมูลแต่ละตัว

<u>Sendchar</u> program นี้ถูกเรียกว่า "Sendchar"ซึ่งจะนาตัวอักษรใน Register A ส่งออกไปและมีการหน่วงเวลาช่วงหนึ่ง แล้วจึงส่งการทางานกลับ Timer1 ถูกใช้เพื่อกาหนด baud rate เท่ากับ 1200 เวลาการหน่วงสาหรับ 1 ตัวอักษรหรือ 10 bit คือ 1000/120 หรือ 8.4 ms โดย software ที่ใช้หน่วงเวลาถูกพัฒนาจาก Section 7.5 ที่ใช้ฐานเวลา 1 msเปลี่ยนเป็นฐานเวลา 0.1 ms โดยการกาหนด "delay" ใหม่ Timer 1 ต้องการกาเนิด baud rate ค่า 1200 โดยใช้ crystal ความถี่ 16 MHz ค่าที่ทาการ load ให้ Timer 1 คือ 256 - 16E6 / (16x12x1200) ซึ่งจะเท่ากับ 186.6 หรือเป็นจานวนเต็มคือ 187 ซึ่งค่านี้จะกาเนิด อัตราของ baud rate เท่ากับ 1208

```
ADDRESS   MNEMONIC          COMMENT
          .org 0000h
          .set delay,16h ;basic delay = 22d x 4.5 = 99 µs
          mov a,#'A'     ;for this example, send an A
          acall sendchar ;send it
here:     sjmp here      ;simulate rest for user program
sendchar: anl tcon,#0fh  ;after timer 1 configuration only
          orl tcon,#20h  ;set timer 1 for mode 2(auto reload)
          mov th1,#0bbh  ;set reload number to 187d(256 - 69)
          orl pcon,#80h  ;set SMOD bit to 1
          orl tcon,#40h  ;start timer 1 by setting TR1
          mov scon,#40h  ;set serial port to mode 1
          mov sbuf,a     ;load transmit register and wait
          mov a,#54h     ;delay for 8.4 ms (84d = 54h)
          acall softime  ;wait
          ret            ;character now sent
;
;softime will be simulated by a return instruction
softime:  ret
          .end                    ;assembler use only
```

<u>ข้อสังเกต</u> ถ้า Timer1 และ serial port ถูกใช้แตกต่างกันใน programดังนั้น การ push และ pop จะมีผลต่อ cortrol register แต่จากไว้ว่า T1 และ SBUF มีความสามารถเพียงถูกใช้สำหรับเป็น function ที่ทำเวลาเท่านั้นcrystal ความถี่ 16 MHZ ไม่เหมาะสมกับอัตรา baud rate มาตราฐาน คือ 300,1200 ,2400,4800,9600 หรือ 19200 โดยค่าผิดพลาดจาก crystal นี้แสดงไว้ข้าง

สางนี้

อัตรา	ค่าผิดพลาด (%)
300	0.08
1200	0.64
4800	2.12
9600	3.55
19200	8.51

ค่าผิดพลาดจะมากขึ้นเมื่อใช้กำเนิด baud rate ค่าสูงขึ้น เราสามารถใช้ crystalความที่ 11.059 MHZ เพื่อลดค่าผิดพลาดให้น้อยกว่า 0.002% ใต้

การส่งตัวอักษรโดยใช้ Polling ในการส่งข้อมูลแบบอนุกรม เมื่อการส่งอักษรตัวสุดท้ายเขียนลง SBUF จะมีการ set TI flag ดังนั้น polling routine จะต้องทำการ reset TI ก่อนที่จะคืนการทำงานให้ program ที่เรียกใช้ การคืนการทำงานโดยไม่ reset TI จะทำให้ไม่สามารถส่งข้อมูลได้ในครั้งต่อไป

เทคนิคนี้มีจุดเด่นที่ความง่ายของ program ใช้รหัสน้อย และ routine จะไม่สนใจอัตรา baud rate ที่ใช้ ในตัวอย่างนี้เราใช้ Timer 1 สร้าง baud rate ซึ่งการสร้างอัตรา Baud rate นี้ คล้ายกับตัวอย่างที่แล้ว

Xmit rubroutine "xmit" จะมีการ set TI llag ใน Register SCON เมื่อการเขียนลง SBUF เสร็จสิ้นลง

```
ADDRESS    MNEMONIC          COMMENT
           .org 0000h
           mov a,#'3'    ;send an ASCII 3 for this example
           acall xmit    ;send the character using xmit
here:      sjmp here     ;simulate remainder of user program
;
```

```
xmit:   mov sbuf,a      ;transmit the contents of A and wait
wait:   jnb scon.1,wait ;loop until TI = 1 (SBUF is empty)
        clr scon.1      ;reset TI to 0
        ret
        .end
```

ข้อสังเกต TI จะคงเป็นค่า 0 จนกว่า SBUF จะว่าง เมื่อมีการ reset 8051 หรือเมื่อเริ่มเปิดเครื่อง TI จะ set เป็น 0

การส่งตัวอักษรโดยใช้ Interrupt-Driven วิธีที่ 3 นี้เมื่อการส่งข้อมูลเสร็จสิ้นลงจะใช้โครงสร้างการ interrupt ของ 8051 interrupt vector จะอยู่ที่ address 0023h สำหรับทั้ง interrupt การส่ง TI และ interrupt การรับ RI เมื่อเกิด interrupt จาก serial port จะส่งผลให้ program ไปทำงานที่ตำแหน่ง 0023h ซึ่งเป็นทางเข้าของ routine ที่เขียนโดย programer

ผู้ใช้ program สามารถใช้ Subroutine โดยการ load ตัวอักษรส่งไปยัง SBUFและ enable interrupt ของ port อนุกรม ใน Register EI เมื่อ SBUF ว่าง TI จะถูก set มีผลให้เกิดการ interrupt ไปที่ address 0023h ถ้าไม่มีการ reset TI การinterrupt ก็จะเกิดขึ้นอีกดังนั้น sub-routine ที่ตำแหน่ง 0023h ซึ่งถูกเรียกว่า "Serial"จะต้องมีการ reset TI ก่อนที่จะคืนการทำงานให้กับตำแหน่งที่ถูก interrupt

program ที่ตามมานี้ใช้ทดสอบการส่งอักษรตัวเดียวสำหรับการส่งข้อความยาว ๆ ซึ่งใช้เทคนิค พื้นฐานจาก program นี้

SBUFR เป็น routine ส่งข้อมูลโดยใช้การ interrupt สำหรับการส่งตัวอักษรเพียงตัวเดียว ซึ่งมี interrupt ชี้ไปที่ตำแหน่ง 0023h ส่วนของ program ที่จะกระตุ้นการ interrupt routine แสดงไว้ข้างล่างนี้

```
ADDRESS   MNEMONIC        COMMENT
          .org 0000h
sbufr:    sjmp user       ;jump over interrupt vectors
          .org 0023h      ;put serial interrupt routine here
serial:   clr scon.1      ;clear TI
          reti            ;return to interrupted user program
;
user:     mov sbuf,#'X'   ;send an X in this example
          orl ie,#90h     ;enable serial interrupt
here:     sjmp here       ;simulate remainder of program
          .end
```

<u>ข้อสังเกต</u> ถ้า TIไม่ถูก clear ก่อนคำสั่ง RETI จะทำให้เกิดการ interrupt กลับไปตำแหน่ง 0023h อีก RETI ถูกใช้เพื่อคืนการทำงานให้กับส่วนที่ถูก interrupt แต่ไม่ได้เป็นการ clear interrupt bit

<u>การรับข้อมูลแบบอนุกรม</u> การส่งจากภายนอกมายัง 8051 อาจจะต้องการการรักษาเวลาที่แน่นอนในการส่งหรือไม่ต้องการก็ได้ข้อความสามารถที่จะส่งโดยกำหนดเวลาล่วงหน้าได้ มีสองวิธีที่ใช้เพื่อเตรียมพร้อมในการรับข้อมูลที่ส่งเข้ามา คือsoftware polling หรือ interrupt driven การส่งข้อมูลที่ส่งเข้ามาที่เวลาต่าง ๆ ต้องใช้อัตรา baud rate และ mode ในการส่งที่กำหนดล่วงหน้าไว้แล้ว โดยหน่วยรับของ serial port สามารถกำหนดอัตรา baud rate และ mode ที่ใช้ จาก program

ถ้า program รับผิดชอบสำหรับผู้พูดและผู้ฟังหน่วยอื่น ๆ การขัดแย้งจะตามมาเมื่อหน่วยที่ติดต่อกัน มีการแลกเปลี่ยนข้อมูลกันอย่างไม่เหมาะสม วิธีธรรมดาวิธี

หนึ่งที่ถูกใช้ทดสอบการติดต่อกันของ program จากผู้ใช้แต่ละคน จะใช้การจำลองให้ผู้ใช้แต่ละคนเป็นหน่วย ๆ หนึ่ง เมื่อทุกหน่วยถูกติดต่อกัน และเพื่อการทดสอบแล้วใช้ CRT เป็นสถานีบลายทางเพื่อความเข้าใจง่าย ซึ่งจะแสดงข้อมูลทั้งหมดที่ถูกติดต่อระหว่างสองระบบ

Polling สำหรับรับข้อมูล polling จะทดสอบว่ามีข้อมูลส่งมาหรือไม่จาก RI และจะเรียกใช้ subroutine เพื่อการรับข้อมูลเมื่อ RI ถูก set ใน routine ต้อง reset RI เพื่อไม่ให้เกิดการอ่านข้อมูลเดิมเข้ามาอีก

program จะวนลูบทดสอบ flagจนกว่าข้อมูลจะถูกรับหรือการทำงานของ programจะวนเป็นวงกลม เพื่อทดสอบ flag ในแต่ละวงจร การวนลูบเพื่อรับรองว่าข้อมูลที่อ่านเข้ามาถูกรับแล้ว อย่างไรก็ตามมีจอกาศน้อยมาก ที่สวนอื่นจะทำงานบรรลุผลในขณะที่กำลังคอยข้อมูล programลักษณะเป็นวงกลมเช่นนี้จะยอมให้ program ทำงานต่อไปได้ขณะที่รอข้อมูล

การส่งข้อมูลจะไม่ผิดพลาด โดย program ต้องสามารถทำงานเป็นวงจรที่สมบูรณ์ในช่วงเวลาที่ใช้สำหรับการรับอักษรตัวหนึ่งๆปัญหาด้านเวลาจะไม่มีในอักษรตัวแรกที่รับเข้ามาเนื่องจากมี buffer รองรับอยู่ แต่เมื่อมีการส่งตัวที่สองเข้ามาในขณะที่ตัวอักษรตัวแรกยังไม่ถูกอ่านถ้าอักษรตัวแรกอ่านเข้าไปก่อนที่ bit สุดท้ายของตัวอักษรตัวที่สองจะสมบูรณ์ ดังนั้นข้อมูลจะไม่สูญหาย และรวมไปถึงอักษรตัวต่อๆไปด้วย

เวลาที่ใช้กับอักษรตัวหนึ่ง คือจำนวนของ bit ในตัวอักษรหารด้วย baud rate สำหรับ mode 1 ตัวอักษรตัวหนึ่งจะมีทั้งหมด 10 bit คือ bit start, รหัส 8bit, stop และใช้อัตรา baud rate 1200 เป็นผลให้มีการติดต่อสื่อสาร 120 ตัวอักษรต่อวินาที หรือ เวลาที่ใช้ในตัวอักษรหนึ่ง ๆ เท่ากับ 8.33 ms โดยการเฉลี่ยให้คำสั่งหนึ่ง ๆใช้เวลา 18 ลูก ซึ่งเป็นเวลา 1.13 ms ดังนั้น program สามารถทำคำสั่งอื่น ๆ ได้ 7371 คำสั่ง ซึ่งค่าที่กล่าวมานี้จะเป็นที่พอใจสำหรับงานควบคุมง่ายๆ ถ้าเวลาถูกต้องการมากเราสามารถลด อัตรา baudrate ให้ต่ำเช่น 300 เพื่อให้ program มีเวลาทำงานอย่างอื่นมากขึ้น คือประมาณขนาด

29K byte ซึ่งมีค่าเกือบเป็นครึ่งหนึ่งของ ROM ใน 8051 ตัวเดียว

The polling program for the loop approach follows:

```
ADDRESS    MNEMONIC              COMMENT
here:      jnb scon.0,here       ;wait here until RI = 1
           clr scon.0            ;clear the RI bit
           acall getchar         ;getchar is some user routine
                                 ;which reads SBUF
```

[หน้าที่..]

The circular approach is very similar:

```
ADDRESS    MNEMONIC              COMMENT
           jnb scon.0,there      ;test for RI = 1, go on if not
           clr scon.0            ;clear the RI bit
           acall getchar         ;call user program getchar:
there:     sjmp there            ;reset of user program getchar
           ret                   ;simulate user routine
           .end
```

Interrupt-Driven Data Reception เมื่อข้อมูลจำนวนมาก ๆ ถูกส่งมา ทำให้การกำหนดเวลาในการรับอักษรตัวหนึ่ง ๆ ต้องมีความเที่ยงตรงมาก ๆ ทำให้ polling ไม่เหมาะสมกับการรับข้อมูลยาว ๆ แต่เหมาะกับการรับแบบ interrupt เพราะ Interrupt driven จะอนุญาตให้ program ทำงานอย่างอื่นขณะที่ข้อมูลยังไม่พร้อมและจะเป็นตัวคอยหยุด program เพื่อให้มารับข้อมูลเมื่อข้อมูลพร้อมแล้วจาก interrupt vector

Intdat การรับข้อมูลแบบ interrupt-driven จะมี interrupt vector ชี้ไปที่ subroutine ในตำแหน่ง 0023h

```
ADDRESS   MNEMONIC            COMMENT
          .org 0000h
intdat:   orl ie,#90h         ;enable serial and all interrupts
          sjmp over           ;jump over the interrupt locations
          .org 0023h          ;put serial interrupt program here
          jbc scon.1,xmit     ;if TI bit set, clear it and jump
          clr scon.0          ;must have been RI, clear it
          lcall recv          ;call receive subroutine
          reti                ;return to program where interrupted
xmit:     lcall trans         ;call transmit program
          reti                ;return to program where interrupted
over:     sjmp over
trans:    ret                 ;dummy transmit/receive routine
recv:     ret
```

ข้อสังเกต ถ้าทั้ง RI หรือ TI ถูก set routine นี้จะถูกใช้บริการในการติดต่อ หลังคำสั่ง RETI ซึ่งตามมาด้วยคำสั่ง LCALL RI จะยังคงถูก set เพื่อให้สามารถกลับไปตำแหน่ง 0023h เมื่อ routine การรับถูกเรียกใช้ถ้า subroutine การรับถูกเรียก จะใช้เวลามากกว่า เวลาในการรับอักษรตัวหนึ่ง ๆ ดังนั้น ข้อมูลจะสูญหาย

สรุป การออกแบบระบบ microprocessor ใช้ 8051 เป็นตัว cortroller รวมกับ External EPROM และ RAM ที่สามารถเลือกขนาดได้ ระบบที่ออกแบบมี

ดังนี้

RAM ภายนอก ขนาด 8K -32K byte

ROM ภายนอก ขนาด 8K -64K byte

I/O port 1-8 bit จาก port ของ 8051

port อื่น ๆ มี port 3.0 (RXD)

3.1 (TXD)

3.2 (INT0 ,Active low)

3.3 (INT1 ,ACtive low)

3.4 (T0)

3.5 (T1)

crystal ความถี่ 16 MHZ

สามารถใช้ crystal ความถี่อื่น ๆ กำเนิดจังหวะเวลาให้กับ Timer ได้

ต้องเขียน program ขึ้นมาเพื่อทดสอบการอ่านข้อมูลจาก ROMและทดสอบการเขียน และอ่านข้อมูลจาก RAM

Subroutine ที่ถูกพัฒนาโดยคำสั่งงาน 8051 เพื่อนำไปประยุกต์ใช้มีดังนี้

Time delays : software;timer, software polled;timer, interrupt driven

Lookup Tables : PC base, DPTR base

Serial data communications transmission : time delay, software polled , interrupt driven

Serial data communications reception : software polled ,interrupt driven

แอพพลิเคชั่น

บทนำ

ไมโครคอนโทรลเลอร์มีแนวโน้ม ที่จะใช้ในแอพพลิเคชั่นหลายอย่าง เหตุผลสำคัญ คือราคาไม่แพง ไมโครคอนโทรลเลอร์แบบใหม่ๆ มีซอฟแวร์ที่พัฒนามากขึ้น และช่วยโปรแกรมเมอร์ได้มาก แม้ว่าไมโครคอนโทรลเลอร์จะล้าสมัยกว่า CPU แต่สามารถใช้กับแอพพลิเคชั่นของ CPU 8 บิทได้มากกว่า

ต่อไปเราจะมาศึกษารายละเอียด ของโครงสร้างฮาร์ดแวร์และโปรแกรมที่ใช้ร่วมกัน โครงสร้างฮาร์ดแวร์ที่จะศึกษามีดังนี้

คีย์บอร์ด

displays

การวัดพัลส์

การแปลง A/D และ D/A

การอินเทอร์รัพจากแหล่งต่างๆ

คีย์บอร์ด

การอินเตอร์เฟสที่ใช้มาก ระหว่างมนุษย์กับคอมพิวเตอร์ (PC) คือคีย์บอร์ด ซึ่งใช้ปุ่มขึ้นลง โดยจะหาซื้อที่คอมพิวเตอร์ได้ออกแบบไว้ แอพพลิเคชั่นจะอยู่ตรงไหนก็ได้ ระหว่างคีย์บอร์ดกับคอมพิวเตอร์ ผู้ออกแบบจะต้องออกแบบ ปุ่มที่จำเป็นต้องใช้ในสภาพจริงๆ และควรมีการป้อนกลับแก่ผู้ใช้ เพื่อให้รู้ว่าเครื่องได้รับรู้แล้วโดยอาจใช้ แสง, เสียง beep, จอภาพแสดงผล เป็นต้น

ปัจจัยด้านมนุษย์

โปรแกรมในแอพพลิเคชั่นของคีย์บอร์ด ต้องป้องกันลักษณะที่อาจเกิดได้ดังนี้

กดคีย์มากกว่า 1 ปุ่มพร้อมกัน

กดคีย์ค้างไว้

กด–ปล่อย อย่างรวดเร็ว

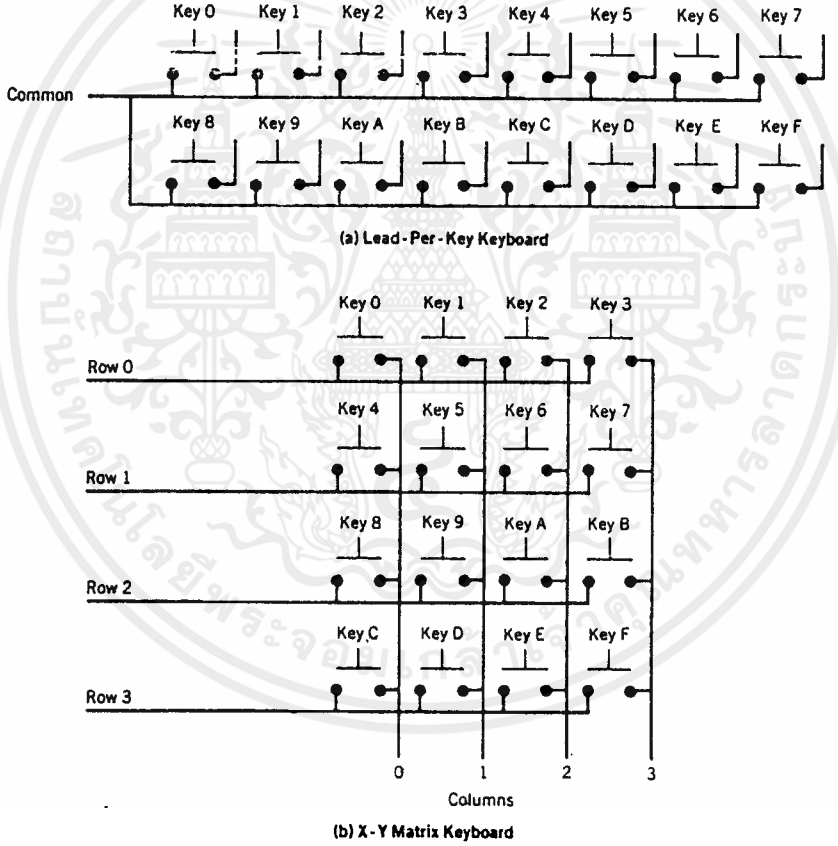
สถานการณ์ทั้งหมดนี้สามารถกำหนดได้ก่อน โดยวิธีการทางฮาร์ดแวร์ หรือซอฟแวร์ วิธีทางซอฟแวร์มีราคาถูก และจะเน้นในที่นี้

ปัจจัยด้านการคีย์

ลักษณะของคีย์ที่ครอบคลุมทั้งหมด ต้องมีความสามารถกระเด้ง (กดๆบสอยๆ) การสัมผัสของคีย์จะเปิดๆปิดๆในช่วง mS เมื่อคีย์ถูกกดและบสอย ลักษณะของพัลส์ที่รวดเร็วแบบนี้มนุษย์ไม่สามารถมองเห็น แต่พัลส์ก็ยังคงอยู่ได้นานในช่วง uS เราอาจทำให้คีย์ไม่กระเด้ง โดยใช้ RS flip-flop หรือซอฟแวร์ในการหน่วงเวลา

โครงสร้างของคีย์บอร์ด

คีย์บอร์ดที่มีขาย เป็นหนึ่งในสามของโครงสร้างที่สมมติเป็นขดลวดซึ่งเป็นการออกแบบ 16 คีย์ ดังรูปสาง โครงสร้าง lead-per-key ใช้เมื่อมีคีย์น้อยๆถูกใช้



เนื่องจากแต่ละคีย์สามารถต่อกับพอร์ทพิน จำนวนคีย์จึงไม่ควรเกิน 16 คีย์ โครงสร้างแบบนี้มีราคาถูกที่สุด

การต่อแมทริกซ์ X-Y ดังรูป 1 ที่ผ่านมาเป็นที่นิยมมากเมื่อจานวนคีย์มากกว่า 10 แมทริกซ์ มีประสิทธิภาพสูงสุดเมื่อจัดเป็นรูป square เพื่อว่าด้วนๆ N เส้นของ X และ N เส้นของ Y จะใช้คีย์ทั้งหมด N^2 คีย์ แมทริกซ์แบบนี้มีราคาต่าที่สุดสาหรับคีย์จานวนมากๆ

รหัสคีย์บอร์ด (coded keyboard) เกี่ยวข้องกับแอพพลิเคชั่นพวก เทรศัพท์ ซึ่งเกี่ยวข้องกับการสัมผัสเสียงสัญญาณ การเข้ารหัสสามารถมีคีย์ถูกกดหลายๆคีย์ได้ เพื่องายแก่การตรวจจับคุณภาพและความทนทานของคีย์เหล่านี้ถือว่ายอดเยี่ยม เนื่องจากจานวนคีย์ที่มาก หากคีย์แบบนี้มีไม่เกิน 16 คีย์ จะถือว่ามีราคาแพงที่สุด

<u>โปรแกรมสาหรับคีย์</u>

โปรแกรมที่ติดต่อกับมนุษย์โดยผ่านคีย์บอร์ด จะมีลักษณะดังนี้

การกระเด็ด(bounce): การหน่วงเวลาที่เรารู้ว่าเกินจากค่าที่กาหนด

คีย์หลายคีย์ : เฉพาะรูปแบบที่สร้างโดยคีย์ที่ valid ที่ถูกกดจึงจะยอมรับ

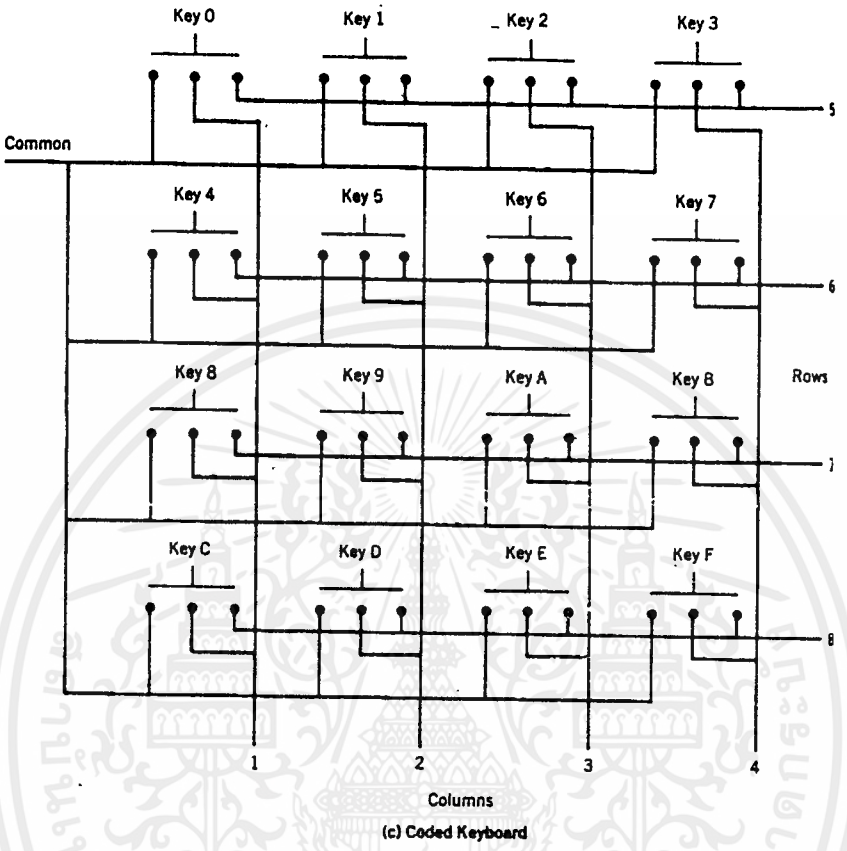
คีย์ที่เก็บค่า : รูปแบบคีย์ valid ที่ได้รับหลังจากการหน่วงการกระเด็ดถูกต้อง และจะไม่รับคีย์ใดๆจนกว่าจะถึงช่วงเวลาค่าหนึ่ง

การกดคีย์ด่วน : การออกแบบเพื่อให้คีย์ถูกสแกน ด้วยอัตราที่เร็วกว่า ที่มนุษย์ จะรู้สึกได้

เรื่องสุดท้ายที่เป็นจุดสาคัญ : คีย์บอร์ดควรถูกอ่านในรูปแบบลูบสแกน หรืออ่านเฉพาะเมื่อคีย์ถูกกด

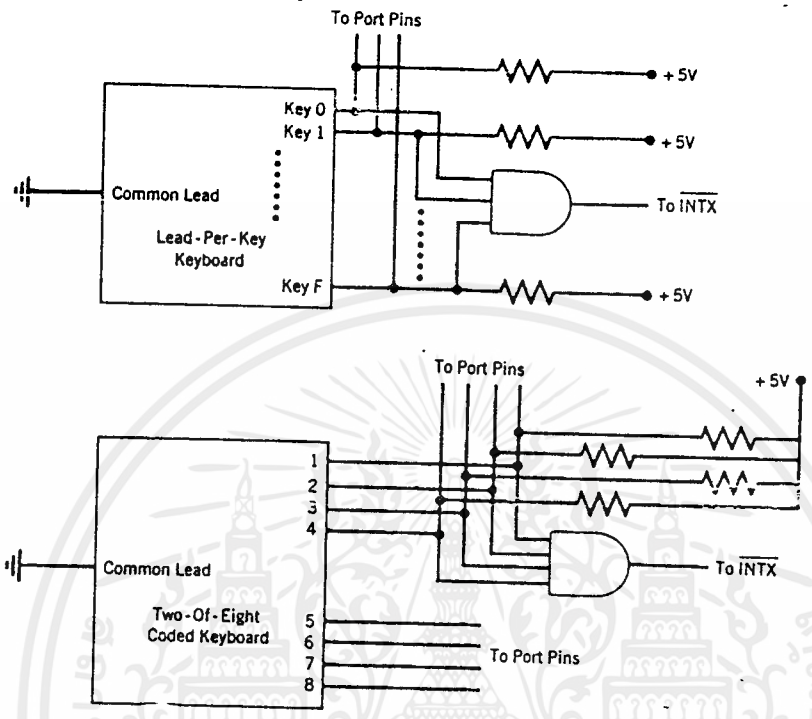
โดยทั่วไป คีย์บอร์ดจานานน้อย เช่น lead-per-key , coded สามารถต่อกราวน์ร่วมกัน และคีย์จะอ่านเป็นช่วงๆ ค่า low จากแต่ละเส้นสามารถทาให้แอดทีพ low ด้วยการ OR ดังรูป 2 และต่อเข้าเป็นเอาพุทภายนอกของขา INTX

คีย์บอร์ดแบบแมทริกซ์ถูกสแกน โดยนาจากแถว X ไสบเป็นลาดับ และตรวจจับจากคอลัมน์ Y ไสบเรื่อยๆ เพื่อให้ตรงกัน การสแกน X-Y สามารถทาโดยใช้วงจรสแกนคีย์บอร์ด หรือใช้พอร์ทของไมโครคอนโทรลเลอร์ภายใต้การควบคุมโปรแกรม วงจรสแกนจะเพิ่มต้นทุนของระบบ การใช้โปรแกรมจะใช้เวลาของโปรเซส



(c) Coded Keyboard

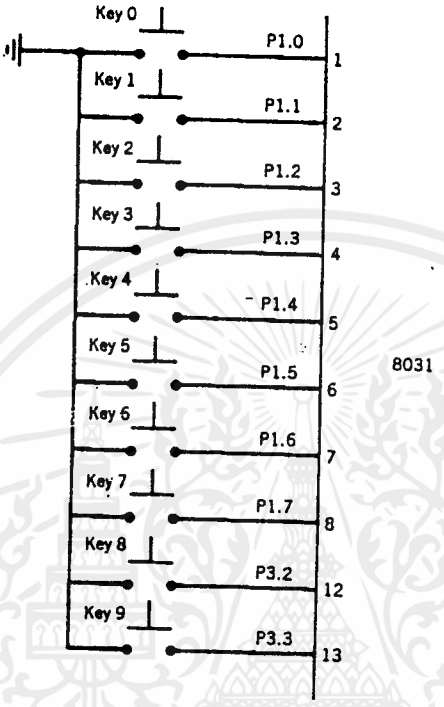
เซอร์ และความเป็นไปได้ที่จะตอบกลับผู้ใช้ อาจจะช้าถ้าโปรแกรมยุ่งยาก ควรยันทึก



ไว้ว่า คอมพิวเตอร์ใช้เวลาตอบในขณะที่คีย์ไม่ถูกกดนานเช่นนี้ เมื่อกำลังทำคำสั่งพิมพ์ ทางเลือกระหว่างการเพิ่มฮาร์ดแวร์สแกน หรือซอฟแวร์โปรแกรมถูกตัดสินใจโปรเซสเซอร์ยุ่งยากขนาดไหน และจำนวนข้อมูลที่ใส่โดยผู้ใช้

<u>โปรแกรมสแกนสำหรับคีย์บอร์ดจำนวนน้อย</u>

สันนิษฐานว่าคีย์บอร์ด lead-per-key ถูกอินเตอร์เฟสกับไมโครคอนโทรลเลอร์ คีย์บอร์ดมี 10 คีย์ (0-9) และเวลากระเด็น(bounce time) = 20 mS โปรแกรมมีการสแกนคีย์บอร์ดตลอดเวลา เพื่อรอคีย์ที่ถูกกดก่อนจะเรียกรูทีน คีย์ถูกต่อกับพอร์ท1 (0-7) และพอร์ท3.2 และ 3.3 (8-9) ดังรูป 3



Getkey

รูทีน getkey สแกนคีย์ขนาด 10 คีย์อย่างต่อเนื่องผ่านพอร์ท 0 และ 3 คีย์ถูกกระจัดกระจายทั้ง 2 ทิศทาง (กด-บอย) และใช้เวลา 50 mS ก่อนคีย์ใหม่จะถูกรับค่า หากรูปแบบคีย์ไม่ถูกต้องก็จะถูกบฏิเสธ

ในหน้าต่อไปเป็นโปรแกรม getkey

```
ADDRESS   MNEMONIC          COMMENT
          .equ bounce,14h   ;set debounce delay to 20d ms
          .equ next,32h     ;set interval between keys to
                            ;50d ms
          .equ newkey,70h   ;store accepted key in internal RAM
          .equ flag,00h     ;addressable bit 00 used as a flag
          .org 0000h
getkey:   mov p1,#0ffh      ;set ports 1 and 3 as inputs
          mov p3,#0ffh
scan:     acall keydown     ;keydown looks for any key(s) down
          jz scan           ;if A = 0 then no key(s) down: loop
          acall convert     ;convert returns flag set if not
                            ;valid
          jbc flag,scan     ;or A set to 00 to 09 for keys 0-9
          mov newkey,a      ;store key and wait for debounce
                            ;time
          mov a,#bounce     ;then check to see if same key
          acall softime     ;wait 20 ms
          acall keydown     ;see if a key is still down
          jz scan           ;if not down then must have been
                            ;noise
          acall convert     ;see if key is still valid and
                            ;matches
          jbc flag,scan     ;the original key found
          cjne a,newkey,scan ;check for equal
          acall vendit      ;call vending machine subroutine
wait:     acall keydown     ;now wait for all keys to go up
          jnz wait          ;wait until A = 00: keys all up
          mov a,#next       ;wait 50d ms and see if all
                            ;still up
          acall softime
          acall keydown     ;continue until keys are up
          jnz wait          ;loop until keys up for 50d ms
          sjmp scan         ;get next key
;
;"keydown" gets the contents of P1 and P3 pins, which are connected
;to the keys, and checks for any zero bits; no check is made to see
;if more than one bit is low
;
keydown:  mov r0,p1         ;get state of P1 keys to R0
          mov a,p3          ;get state of P3 keys to A
          orl a,#0f3h       ;make bits 0,1,and 4-7 a one
          anl a,r0          ;check for any one or more keys
                            ;down
          cpl a             ;A = FFh if all keys up, now 00
          ret               ;if A not 00 then at least one key
                            ;down
```

ADDRESS	MNEMONIC	COMMENT

Continued

```
;"convert" checks for more than one key down; if more than one key
;is down then addressable bit "flag" is set; if only one key is
;down then the one-of-ten bit pattern is converted to an
;equivalent 0-9 number in the A register and "flag" is reset
;valid patterns (a single 0 out of ten bits) are found by CJNE
;operations; A is counted up for each test to match the key number
;
convert:  clr flag                  ;assume that key hit is valid
          clr a                     ;A contains first possible key (00)
          mov r1,p1                 ;get P1 key pattern in R1
          mov r3,p3                 ;get P3 key pattern in R3
          orl 03h,#0f3h             ;make r3 bits 0,1 and 4-7 a one
          cjne r1,#0feh,one         ;search R1 for a legal pattern
          sjmp check3               ;check R3 for no key down
one:      inc a                     ;A contains next key possible (01)
          cjne r1,#0fdh,two         ;continue this for all valid
                                    ;patterns
          sjmp check3
two:      inc a                     ;A = 02
          cjne r1,#0fbh,three
          sjmp check3
three:    inc a                     ;A = 03
          cjne r1,#0f7h,four
          sjmp check3
four:     inc a                     ;A = 04
          cjne r1,#0efh,five
          sjmp check3
five:     inc a                     ;A = 05
          cjne r1,#0dfh,six
          sjmp check3
six:      inc a                     ;A = 06
          cjne r1,#0bfh,seven
          sjmp check3
seven:    inc a                     ;A = 07
          cjne r1,#7fh,eight
          sjmp check3
eight:    inc a                     ;A = 08
          cjne r3,#0fdh,nine        ;now look for a key in R3
          jnb p3.3 bad              ;check that key 9 is up
          sjmp good
nine:     inc a                     ;A = 09
          cjne r3,#0f7h,bad         ;redundant check
good:     ret
check3:   jnb p3.3,bad              ;if R1 has a low then P3 must be
                                     high
          jnb p3.4,bad
          sjmp good
```

```
ADDRESS   MNEMONIC      COMMENT
bad:      setb flag     ;signal an invalid key pattern
          ret
softime:  ret           ;simulate "softime" subroutine
vendit:   ret           ;simulate "vendit" subroutine
          .end
```

<u>รูบรแกรมอินเทอร์รัพสาหรับคีบอร์ดน้อยๆ</u>

ถ้าแอพพลิเคชั่นใวต่อเวลามากจนกระทั่ง การหน่วงเวลาและเวลากระจิดดไม่สามารถยอมรับได้ การอินเทอร์รัพบางแบบต้องใช้เพื่อว่า รูบรแกรมหลักสามารถทางานได้จิดยไม่ถูกขัดขวาง

การบระนีบระนอมในเรื่องนี้ ทาได้จิดยจพลล่งคีย์บอร์ดเหมือนกับลูบรูบรแกรมหลัก แต่การหน่วงเวลาทั้งหมดจะใช้ทามเมอร์ต่อรูบ เพื่อให้รูบรแกรมหลักไม่ต้องรอการหน่วงของซอฟแวร์ รูบรแกรม getkey สามารถเบลี่ยนแบลงเพื่อใช้ทามเมอร์สร้างการหน่วงร่วมกับเวลากระจิดด และการหน่วง "all-up"การทาทายด้วยการเข้าถึงจะมีรูบรแกรมที่การหน่วงกาลังถูกลดค่าเวลา ความจาที่การหน่วงกาลังทาอยู่สามารถจัดการจิดยใช้อิทแพลกมิฉะนั้นทามเมอร์ตัวหนึ่งจะถูกสร้างการหน่วงกระจิดดของการกดคีย์ และทามเมอร์อีกตัวจะสร้างการหน่วงของคีย์ที่บสอย การเข้าถึงด้วยแพลกจะถูกตรวจสอบจิดยตัวอย่างที่มีต่อรูบ

ลักษณะสาคัญของรูบรแกรม คือรูบรแกรมหลักจะตรวจสอบแพลกเพื่อหาว่ามีคีย์บอร์ดถูกกดหรือไม่ ถ้าแพลกเซ็ท รูบรแกรมจะค้นหาคีย์ที่เก็บในตาแหน่งของแรมและรีเซ็ทแพลก คีย์บอร์ดจะยังถูกจพลล่งจิดยรูบรแกรมหลัก แต่รูบรแกรมอินเทอร์รัพจะได้คีย์หลังจากจพลล่งแล้ว รูบรแกรม "Hardtime" จากบทที่แล้ว ใช้สาหรับการหน่วงเวลา ผู้ใช้คีย์บอร์ด อาจสังเกตพบความชักช้าของผลตอบสนอง ถ้ารูบรแกรมหลักใช้เวลาในลูบนานเกินรูบจนกระทั่งลาดับการเริ่มต้นของคีย์บอร์ดไม่ได้ถูกกระทาทุกๆ ควอเตอร์ที่ 2 และต่อๆ รูบ

Inkey

โปรแกรม Inkey ใช้ฮาร์ดแวร์ทามเมอร์ T1 สร้างการหน่วงเวลาตลอดเวลา สำหรับคีย์บอร์ดเริ่มต้นเมื่อตรวจพบว่าคีย์ถูกกด ฉะนั้นโปรแกรมจะทำงานต่อและตรวจสอบการกดคีย์ในลูปต่อไป คีย์ที่ถูกกดจะเริ่มต้นการหน่วงเวลากระโดดในทามเมอร์ T1 และเซ็ทแฟลกทามเมอร์ เพื่อสังเกตโปรแกรมอินเทอร์รัพของการใช้ทามเมอร์ โปรแกรมอินเทอร์รัพจะตรวจสอบว่าคีย์ยังคงถูกกด และเป็นคีย์ที่ถูกต้อง คีย์ที่ถูกต้องจะถูกเก็บไว้ และแฟลกถูกเซ็ทซึ่งอาจทดสอบโดยโปรแกรมหลัก ต่อจากนั้นโปรแกรมอินเทอร์รัพจะเริ่มต้นหน่วงเวลาบสอยคีย์และเซ็ทแฟลกทามเมอร์เพื่อแสดงสภาพนี้หลังจากหน่วงเวลาบสอยคีย์ โปรแกรมอินเทอร์รัพจะตรวจสอบคีย์ที่ไม่กดกทุกคีย์ การหน่วงเวลาจะเริ่มยกเว้นจากระดับทุกคีย์ถูกบสอย และอินเทอร์รัพของทามเมอร์จะหยุดชั่วขณะ

โปรแกรม Inkey แสดงในหน้าถัดไป

```
ADDRESS     MNEMONIC                COMMENT
            .equ newkey,70h         ;store any new key in RAM
            .equ flag,00h           ;addressable bit 00 used as a flag
            .equ newflg,01h         ;when newflg = 1 then there is
                                    ;a key
            .equ timflg,02h         ;timflg = 0 for debounce, 1 for
                                    ;delay
            .equ bounce,14h         ;set debounce delay to 20d ms
            .equ next,32h           ;set interval between keys to
                                    ;50d ms
            .equ savetime,0010h     ;external RAM address for delay
            .org 0000h
inkey:      .sjmp over              ;jump over interrupt locations
;
;when T1 times out it vectors here and jumps to "hardtime" for the
;desired delay. When the delay is up then the key program is called.
;
            .org 001bh              ;interrupt location for TF1
            ljmp hardtime           ;jump to time delay subroutine
userdly:    acall usertime          ;call usertime if delay done
            reti                    ;return to program when usertime
                                    ;done
;
;the main program begins here; the keyboard is scanned unless there
;is a new key to be processed, or T1 is counting, signifying that
;a key read is in progress
;
over:       mov p1,#0ffh            ;set ports 1 and 3 as inputs
            mov p3,#0ffh
            clr newflg              ;initialize all flags
            clr flag
            clr timflg
begin:      jbc newflg,key          ;check if a key is waiting and
                                    ;get it
            jb tcon.6,mainprog      ;if T1 is running then wait
            acall keydown           ;keydown looks for any key(s) down
            jz mainprog             ;if A = 0 then no key(s) down;
                                    ;go on
            acall convert           ;check for a valid key
            jz mainprog             ;go on with main program if not
                                    ;valid
            mov newkey,a            ;store key and start debounce timer
            clr timflg              ;signal interrupt program T1
                                    ;running
            mov dptr,#savetime      ;point to delay address
            mov a,#bounce           ;set 20 ms delay
            movx @dptr,a
            inc dptr                ;point to next byte
            mov a,#00h
            movx @dptr,a            ;desired delay now stored
```

```
ADDRESS     MNEMONIC            COMMENT
            orl ie,#88h         ;enable interrupts and T1 interrupt
            acall startime      ;start time delay go to mainprog
            sjmp mainprog
key:        mov a,newkey        ;get key and use in main program
mainprog:   sjmp begin          ;simulate main program and
                                ;loop back
;
;****************************** CONVERT ******************************
;"convert" checks for more than one key down; if more than one key
;is down then addressable bit "flag" is set; if only one key is
;down then the one-of-ten bit pattern is converted to an
;equivalent 0-9 number in the A register and "flag" is reset
;valid patterns (a single 0 out of ten bits) are found by CJNE
;operations; A is counted up for each test to match the key number
convert:    clr flag            ;assume that key hit is valid
            clr a               ;A contains first possible key (00)
            mov r1,p1           ;get P1 key pattern in R1
            mov r3,p3           ;get P3 key pattern in R3
            orl 03h,#0f3h       ;make R3 bits 0,1 and 4-7 a one
            cjne r1,#0feh,one   ;search R1 for a legal pattern
            sjmp check3         ;check R3 for no key down
one:        inc a               ;A contains next key possible (01)
            cjne r1,#0fdh,two   ;continue this for all valid
                                ;patterns
            sjmp check3
two:        inc a               ;A = 02
            cjne r1,#0fbh,three
            sjmp check3
three:      inc a               ;A = 03
            cjne r1,#0f7h,four
            sjmp check3
four:       inc a               ;A = 04
            cjne r1,#0efh,five
            sjmp check3
five:       inc a               ;A = 05
            cjne r1,#0dfh,six
            sjmp check3
six:        inc a               ;A = 06
            cjne r1,#0bfh,seven
            sjmp check3
seven:      inc a               ;A = 07
            cjne r1,#7fh,eight
            sjmp check3
eight:      inc a               ;A = 08
            cjne r3,#0fdh,nine  ;now look for a key in R3
            jnb p3.3 bad        ;check key 9 is up
            sjmp good
nine:       inc a               ;A = 09
            cjne r3,#0f7h,bad   ;redundant check
```

ADDRESS	MNEMONIC	COMMENT
Continued		

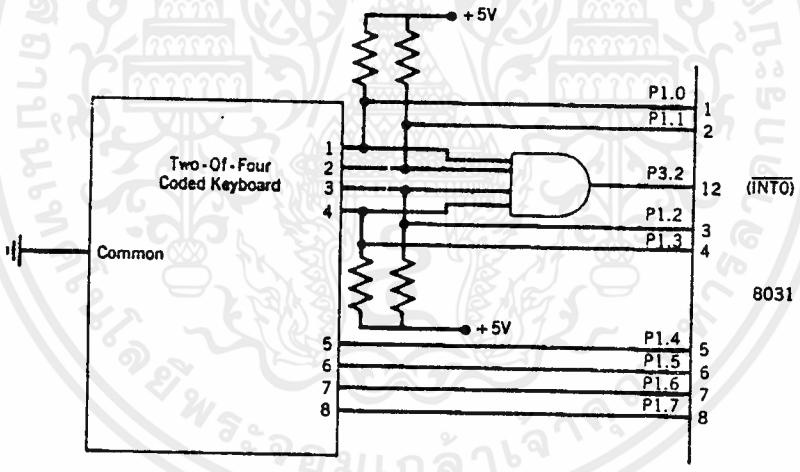
```
good:       ret
check3:     jnb p3.3,bad          ;if R1 has a low then P3 must
                                  ;be high
            jnb p3.4,bad
            sjmp good
bad:        setb flag             ;signal an invalid key pattern
            ret
;
; ****************************** HARDTIME ******************************
;"hardtime" will count the interrupts generated by T1 until the
;number placed in RAM location "savetime" is zero
;
hardtime:   push acc              ;save registers to be used
            push dph
            push dpl
            mov dptr,#savetime    ;get pointer to time delay number
            movx a,@dptr          ;count delay number down to 0000
            dec a                 ;low byte first
            cjne a,#00h,aff       ;check for 0000
            movx @dptr,a          ;save low byte = 00
            inc dptr              ;get high byte and look for 00
            movx a,@dptr
            jz done               ;done if low byte = high byte = 00
            sjmp sava             ;not 0000, reset T1 and delay again
aff:        cjne a,#0ffh,sava     ;if low byte is FF then dec high
            movx @dptr,a          ;save low byte = FF
            inc dptr              ;point to high byte
            movx a,@dptr          ;count high byte down
            dec a
            sjmp sava             ;save the high byte
done:       pop dpl               ;delay is finished
            pop dph               ;restore all registers used
            pop acc
            ljmp userdly          ;continue at user delay program
sava:       movx @dptr,a          ;delay is not up, save the byte
            pop dpl               ;restore saved registers
            pop dph
            pop acc
            acall startime        ;start T1 for next 1 ms delay
            reti                  ;return to place in user program
;
; ****************************** KEYDOWN ******************************
;"keydown" gets the contents of P1 and P3 pins that are connected
;to the keys and checks for any zero bits; no check is made to see
;if more than one bit is low
;
keydown:    mov r0,p1             ;get state of P1 keys to R0
```

```
ADDRESS     MNEMONIC              COMMENT
            mov a,p3              ;get state of P3 keys to A
            orl a,#0f3h           ;make bits 0,1,and 4-7 a one
            anl a,r0              ;check for any one or more
                                  ;keys down
            cpl a                 ;A = FFh if all keys up, now 00
            ret                   ;if A not 00 then one or more down
;
; ***************************** STARTIME *****************************
;"startime" initializes timer 1 and enables timing to begin
;
startime:   mov th1, #0fah        ;set T1 for a 1 ms delay
            mov tl1, #0cbh        ;(see "Timer" example in
                                  ;Chapter 7)
            anl tmod,#0fh         ;clear T1 part of TMOD
            orl tmod,#40h         ;set T1 to timer mode 1
            orl tcon,#40h         ;start timer 1
            ret                   ;return to calling program
;
; ***************************** STOPTIME *****************************
;"stoptime" disables T1
;
stoptime:   anl tcon,#0bfh        ;stop timer T1
            ret
;
; ***************************** USERTIME *****************************
;"usertime,"the user program called from the interrupt program after
;hardtime has timed out, will process the key and set the 50d ms
;delay if the key was valid
;
usertime:   acall stoptime        ;stop timer and determine T1 use
            jb timflg,keyup       ;if a delay then see if keys up
            acall keydown         ;see if a key is still down
            jz goback             ;if not down then must be noise
            acall convert         ;see if key is valid and matches
            jbc flag,goback       ;the original key found
            cjne a,newkey,goback
            setb newflg           ;set new key flag for main program
delay:      mov dptr,#savetime    ;point to delay address
            mov a,#next           ;set 50d ms delay
            movx @dptr,a
            inc dptr              ;point to next byte
            mov a,#00h
            movx @dptr,a          ;desired delay now stored
            orl ie,#88h           ;enable interrupts and T1 interrupt
            acall startime        ;start time delay
            setb timflg           ;set flag for delay condition
goback:     ret
```

```
ADDRESS     MNEMONIC                COMMENT
Continued
keyup:      acall keydown           ;see if keys are up after delay
            jnz delay               ;if not then delay again
            sjmp goback             ;return with T1 stopped
            .end
```

Codekey

ตัวอย่างของคีย์บอร์ดจำนวนน้อยที่เกิดอินเทอร์รัพที่สมบูรณ์ที่จะมีต่อไป ไม่ต้องการการกระทำของโปรแกรมจนกว่าคีย์จะถูกกด ฮาร์ดแวร์ต้องถูกเพิ่มเพื่อให้สภาพการเกิดอินเทอร์รัพที่สมบูรณ์ วงจรที่ใช้ เช่น รูป 4



Keyboard Code		
Key	Pins	Low
0	1	5
1	2	5
2	3	5
3	4	5
4	1	6
5	2	6
6	3	6
7	4	6
8	1	7
9	2	7

คีย์บอร์ดชนิด two-of-eight ซึ่งรหัสของคีย์มีดังนี้

KET	CODE(HEX)	KET	CODE(HEX)
0	EE	5	DD
1	ED	6	D8
2	EB	7	D7
3	E7	8	BE
4	DE	9	BD

การตรวจดูรหัสเผยาให้เห็น แต่ละบิบเบิลจะมีเพียง 1 บิทเท่านั้นที่เป็น 0 (low) สาหรับแต่ละคีย์ และมี 0 (low) 2 บิทใน 8 บิทสาหรับแต่ละคีย์ ถ้าแสดงมากกว่า 1 คีย์ จะต้องมีอย่างน้อย 3 บิทเป็น 0 ซึ่งสัญญาณที่ไม่เป็นตามเงื่อนไขเรื่องที่นิยมเช่นนี้สามารถมีได้ถึง 16 คีย์ที่สามารถเข้ารหัสได้ ซึ่งไม่เหมือนกับแบบ lead-per-key ซึ่งมีเพียง 4 สายเท่านั้นที่ต้อง OR กันเป็น 0 เพื่อสร้างการอินเทอร์รัพ

ฮาร์ดแวร์ช่วยตรวจจับเมื่อจานวนของคีย์ที่กดโดยใช้ AND เกทตรวจจับ เมื่อบิทบิบเบิลเป็น 0 การเปลี่ยนสถานะจาก 1 เป็น 0จะอินเทอร์รัพไมโครคอนโทรลเลอร์บนพอร์ท3.2 (INT0) โปรแกรมอินเทอร์รัพจะอ่านคีย์ที่ต่อกับพอร์ท1 และใช้ทามเมอร์ T0 สร้างเวลากระโดด และ T1 สาหรับหน่วงคีย์ที่บสอย เวลาหน่วงทั้งหมดที่เป็นได้ที่ 16 MHz สาหรับทามเมอร์ คือ 49.15 mS ซึ่งครอบคลุมเวลาหน่วงในตัวอย่างที่ผ่านมา

โปรแกรม Codekey ซึ่งถูกทาให้เกิดอินเทอร์รัพโดยการเปลี่ยนจาก 1 เป็น 0 ที่ INT0 ทามเมอร์ T0 และ T1 จะสร้าง debounce and delay time ในโหมดอินเทอร์รัพ อินพุทอินเทอร์รัพ INT0 จะไม่แอคทีฟจนกว่าทุกคีย์ถูกบสอยในช่วงการหน่วงของ T1 ตาราง lookup ถูกใช้เพื่อตรวจสอบว่ามีเพียง 1 คีย์ถูกกด

ต่อไปเป็นโปรแกรม Codekey

```
ADDRESS   MNEMONIC          COMMENT
          .equ newkey,70h   ;store a new key in RAM
          .equ base,400h    ;base of lookup table
          .equ newflg,00h   ;addressable bit 00 for new key flag
          .org 0000h
codekey:  sjmp over         ;jump over interrupt locations
          .org 0003h        ;this is the INT0 interrupt vector
          sjmp keyint
          .org 000bh        ;timer T0 interrupt vector
          sjmp tim0
          .org 001bh        ;timer T1 interrupt vector
          sjmp tim1
keyint:   mov tl0,#0d4h     ;set T0 for 20 ms delay
          mov th0,#97h      ;count from 97D4h to 0000
          setb tcon.4       ;start timer T0
          clr ie.0          ;disable INT0 interrupt
          reti              ;enable interrupt structure and
                            ;return
tim0:     push acc          ;save registers used
          push dpl

          push dph
          clr tcon.4        ;stop T0
          mov a,p1          ;get key pattern
          mov dptr,#base    ;set DPTR to point to lookup table
          movc a,@a+dptr    ;not valid = FFh
          cjne a,#0ffh,good
          pop dph
          pop dpl
          pop acc
          setb ie.0         ;enable INT0 interrupt
          reti              ;enable interrupt structure and
                            ;return
good:     mov newkey,a      ;store the newkey
          setb newflg       ;signal main program: new key present
          anl tl1,#00h      ;set T1 for maximum delay (49.1 ms)
          anl th1,#00h
          setb tcon.6       ;start timer T1
          pop dph           ;restore retgisters
          pop dpl
          pop acc
          reti              ;enable interrupt structure and
                            ;return
Tim1:     push acc          ;save A
          clr tcon.6        ;stop T1
          mov a,p1          ;see if keys up yet
          cjne a,#0ffh,wait ;all inputs will be high if all up
          setb ie.0         ;enable INT0 for next key
          pop acc
          reti
wait:     anl tl1,#00h      ;restart T1 and delay again
          anl th1,#00h
          setb tcon.6       ;start T1
          pop acc
          reti              ;return with interrupt enabled
over:     mov tcon,#01h     ;set INT0 for falling edge interrupt
          mov ie,#8bh       ;enable INT0, T0, and T1 interrupts
          mov tmod,#11h     ;choose timer operation; mode 1
simulate: jbc newflg,key    ;see if there is a new key and get it
          sjmp simulate     ;simulate rest of program
key:      mov a,newkey      ;get key and simulate rest of program
          sjmp simulate
          .org 04bdh        ;place lookup table here, keys 9
                            ;and 8
          .db 09h
          db 08h
```

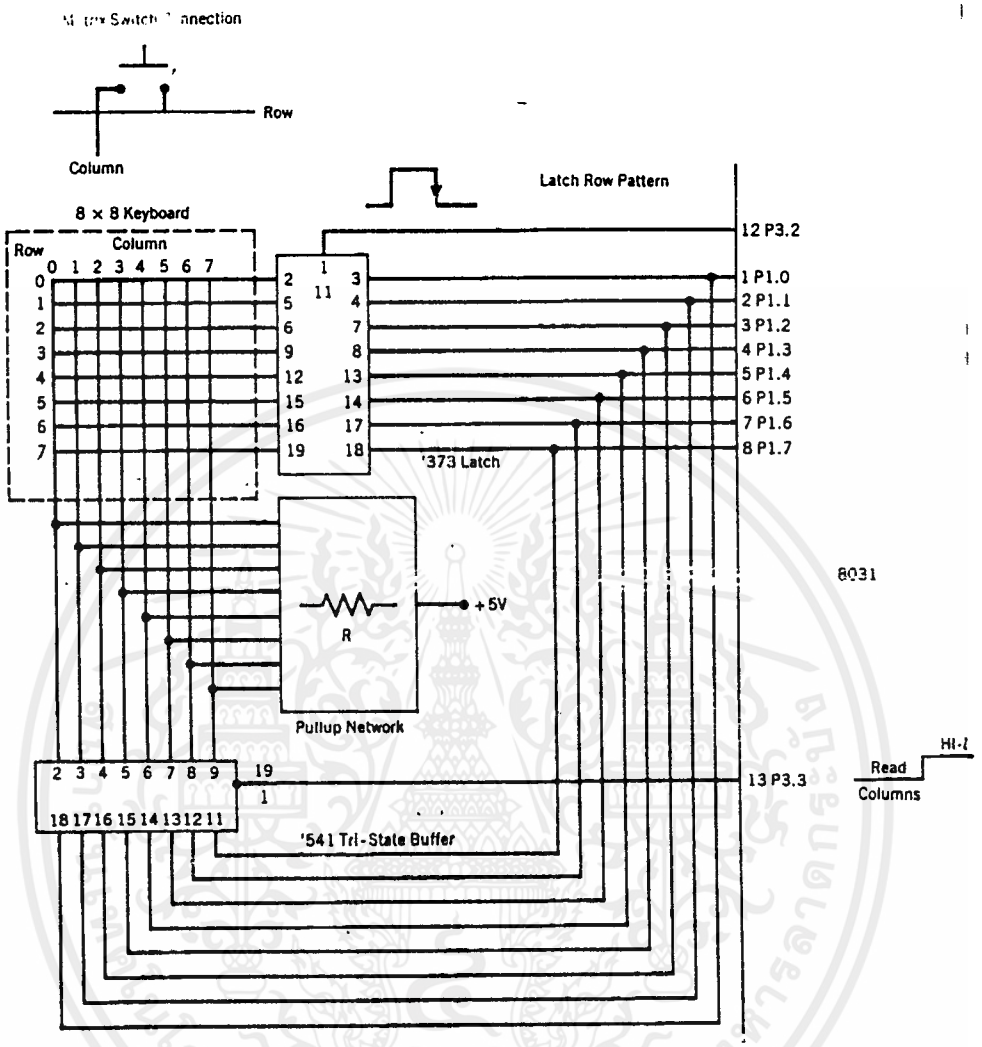
<u>โปรแกรมสำหรับคีย์บอร์ดแมทริกซ์ขนาดใหญ่</u>

คีย์บอร์ดขนาด 64 คีย์ ที่เรียงเป็น 8 แถว 8 คอลัมน์ จะอินเตอร์เฟสกับ 8051 ดังรูป5พอร์ท1 จะใช้กับแต่ละแถวโดยงบ แถวหนึ่ง ณ เวลาหนึ่งซึ่งใช้แลทช์ 8 บิทจะถูกสแตรบโดยพอร์ท 3.2 พอร์ท1 จะอ่านจากคอลัมน์ทั้ง 8 บิทโดยใช้บัฟเฟอร์ tri-state จากพอร์ท3.3 คีย์ที่แสดงจะมีรูปแบบของแถวกับคอลัมน์ที่ต่างกัน คีย์ที่ถูกกดหลายคีย์จะถูกบฏิเสธโดยรูปแบบที่ไม่สมมูลหรือความผิดพลาดที่ใช้กับไซเคิลที่สมบูรณ์ 3 ไซเคิล แต่ละแถวถูกสแกนในช่วงเวลา 1 mS หรือไซเคิลขนาด 8 mS สำหรับคีย์บอร์ดทั้งหมด คีย์ที่สมมูลต้องเห็นเหมือนกันทั้ง 3 ไซเคิล ต่อจากนั้นต้องไม่มีคีย์ใหนถูกกดก่อนที่จะรับค่าคีย์ใหม่ การหน่วง 1 mSระหว่างการสแกนถูกสร้างโดยทามเมอร์ T0 ในโหมดอินเทอร์รัพ

<u>Bigkey</u>

โปรแกรม Bigkey จะสแกนแมทริกซ์คีย์บอร์ดขนาด 8*8 โดยใช้ T0 สร้างการหน่วงเวลา 1 mS ในโหมดอินเทอร์รัพ แต่ละแถวถูกสแกนผ่านแลทช์ภายนอกซึ่งถูกขับโดยพอร์ท1 และถูกสโตรบโดยพอร์ท3.2 คอลัมน์ถูกอ่านผ่านบัฟเฟอร์ tri-state ภายใต้การควบคุมของพอร์ท3.3 คีย์ที่สมมูลจะผ่านโปรแกรมหลัก โดยการตั้งแฟลก "newflg" และเก็บคีย์ที่มีลักษณะตรงกันในตำแหน่ง "newrow" และ "newcol" โปรแกรมหลักจะรีเซ็ท "newflg" เมื่อคีย์ใหม่ถูกเพรซ์ R_4 ถูกใช้เป็นตัวนับไซเคิลที่จับคู่อย่างสมบูรณ์ และเพิ่มไซเคิลเวลา R_5 ใช้เก็บรูปแบบที่สแกนในแถวนอน: มี 1 บิทเท่านั้นที่เป็น 0

หน้าถัดไปเป็นโปรแกรม Bigkey



ADDRESS	MNEMONIC	COMMENT
	.equ newrow,70h	;store any valid key row address
	.equ newcol,71h	;store any valid key column address
	.equ newflg,C0h	;use addressable bit as a new
		;key flag

```
ADDRESS     MNEMONIC                COMMENT
            .equ upflg,01h          ;upflag signals start of key up
                                    ;delay
            .org 0000h
bigkey:     sjmp over               ;jump over T0 interrupt to main
                                    ;program
:
;The interrupt program begins here; T0 is reloaded to permit
.the next interrupt in 1 ms
:
             org 000bh              ;vector location for T0 overflow
                                    ;flag
            mov tl0,#0cbh           ;reload T0 for next interrupt
            mov th0,#0fah
            push acc                ;save A and the flags
            push psw
            mov p1,r5               ;get row scan pattern to port 1
            setb p3.2               ;generate a strobe to the latch
            clr p3.2
            mov p1,#0ffh            ;set P1 as an input port
            clr p3.3                ;read buffer and see if any
                                    ;key down
            mov a,p1                ;get column pattern
            setb p3.3               ;disable buffer
            jb upflg,upyet          ;if upflg = 1 then wait for
                                    ;keys up
            setb c                  ;set C to 1 and rotate A to find
                                    ;a low
            mov r3,#08h             ;8 rotates will restore A to
                                    ;original
look:       rrc a                   ;see if only one zero in A (valid)
            jnc test                ;if C = 0 then see if A = FFh
            djnz r3,look            ;go until C = 0 or rotate finished
            mov a,r5                ;check for a key down previous scan
            cjne a,newrow,goback
            mov newrow,#00h         ;if so then not repeated; zero
                                    ;newrow
            mov r4,#00h             ;if so then zero R4 and scan again
            sjmp goback             ;return to main program
test:       cjne a,#0ffh,bad        ;if A not all ones then invalid key
here:       rrc a                   ;good pattern; restore A
            djnz r3,here
            cjne r4,#00h,match      ;R4 counts pattern matches
newone:     mov newcol,a            ;first time seen; see if it recurs
            mov newrow,r5
            inc r4                  ;R4 contains key detected count
            sjmp goback
match:      push acc                ;save A and check R5 for a new row
            mov a,r5
```

ADDRESS	MNEMONIC	COMMENT
Continued		
	cjne a,newrow,unk	;if no match then this is a new key
	pop acc	;restore A and check for a new ;column
	cjne a,newcol,unkn	;if no match then this is a new key
	inc r4	;match: see if 24 ms have expired
	cjne r4,#04h,goback	;keep if seen for at least 3 cycles
good:	mov newrow,r5	;save new key row and column
	mov newcol,a	
	setb newflg	
	setb upflg	;set up flag for 3 cycles up
	mov r4,#00h	;reset R4 to count key up cycles
	sjmp goback	
unk:	pop acc	;restore new column pattern to A
unkn:	mov r4,#00h	;reset r4 to reflect a new key
	sjmp newone	;look for matches on next cycles
bad:	mov r4,#00h	;reset match counter
	sjmp goback	
upyet:	cjne a,#0ffh,notup	;look for A = FFh
	inc r4	;R4 now counts 3 cycle of up time
	cjne r4,#18h,goback	;look for 24d scans (3 cycles)
	clr upflg	;up time done, look for next key
notup:	mov r4,#00h	;reset R4
goback:	mov a,r5	;rotate R5 low bit to next row
	rl a	
	mov r5,a	
	pop psw	;restore PSW and A
	pop acc	
	reti	
;		
;the interrupt program finishes here and the main program begins;		
;the main program would normally get the new key row and column		
;patterns and convert these to a single byte number		
;		
over:	mov r5,#0feh	;initialize R5 for bottom row low
	mov tmod,#01h	;set T0 to mode 1
	mov tl0,#0cbh	;set T0 for a 1 ms delay
	mov th0,#0fah	;count 1333d @ .75 µs/count
	mov ie,#82h	;enable the T0 interrupt
	setb tcon.4	;start timer
	mov r4,#00h	;reset R4 for no valid key
	clr upflg	;reset key up flag
	clr newflg	;reset new key flag
main:	jbc newflg,simulate	;get key row and column addresses
	sjmp main	;simulate main program
simulate:	nop	;main program would get addresses ;here
	sjmp main	
	.end	

จอแสดงผล (Display)

ถ้าคีย์บอร์ดเป็นวิธีนิยมใช้อินเตอร์เฟส กับอินพุทมนุษย์มาก่อน ดังนั้นส่วนแสดงผลที่มองเห็น ก็เป็นความมากมายของเอาพุทมนุษย์ ส่วนแสดงผลอาจเป็นกลุ่มของลักษณะทั้ง 3 นี้

1.Single lights

2.Single characters

3.Intelligent lphanumeric

ส่วนแสดงผลแบบ single light เป็น incandescent และมี LED เป็นตัวบ่งชี้ซึ่งเป็นเลขฐานสองตัวหนึ่ง ซึ่งปิด-เปิดโดยโปรแกรม แบบ single-character จะรวมตัวเลขและอาร์เรย์ของตัวอักษร ซึ่งอาจใช้งานๆ เป็น 7-segment แสดงส่วนแสดงผลแบบ intelligent dot matrix ที่รับตัวอักษร ASCII 8 บิท และแปลงรหัส ASCII ให้เป็นแบบตัวอักษรที่สมนัยกัน ส่วนแสดงผลแบบ intelligent alphanumeric จะมีไมโครคอนโทรลเลอร์ภายในที่ถูกทำให้เหมาะสมที่สุดสำหรับแอพพลิเคชั่น จอแสดงภาพที่ไม่แพงถูกแสดงโดยหน้าต่าง LCD ที่มีหลายตัวอักษร ซึ่งกลายเป็นที่นิยม ส่วนแสดงผลที่แพงที่สุด คือ ข้อ CRT ASCII ซึ่งใช้อินเตอร์เฟสกับคอมพิวเตอร์ที่มีผู้ใช้หลายคน

ไฟเดี่ยวๆ(individual light) และจอแสดงอักษรเดี่ยวที่ง่ายต่อการใช้พอร์ทที่แสดงบิท หรือตัวอักษรจะสามารถอุปกรณ์ ข้อ ASCII ที่ดี โดยบกติเป็นอุปกรณ์อนุกรมม

สำหรับ 2 ตัวอย่างในบทนี้ 7-segmentและintelligent LCD display ต้องการโปรแกรมที่ยาว

จอภาพแสดงตัวเลขแบบ 7-segment

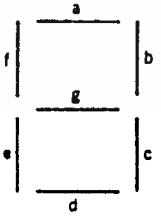
จอแสดงผลชนิด 7-segment โดยที่วงในประกอบด้วย LED เรียงกันเป็นเลข 8 และสายร่วมกัน 1 เส้น(อาโนด หรือ คาโธดก็ได้) และสายที่เป็นอิสระ 7 สาย สำหรับแต่ละหลอด LED รูป 6 แสดงรูปแบบ และวงจรสมมูลของตัวอย่าง ซึ่งเป็นสายร่วมคาโธด ถ้าต้องการจอแสดงผลมากกว่า 1 จอ ก็สามารถใช้ time-mult-

iplexed ตาของมนุษย์ไม่สามารถตรวจจับการกระพริบถ้าแต่ละจอแสดงผลกระพริบทุกๆ 10 mS เวลา 10 mS ถูกแบ่งโดยจำนวนจอแสดงผลที่ถูกใช้หาช่วงเวลาระหว่างการเปลี่ยนแปลงจอแสดงผล

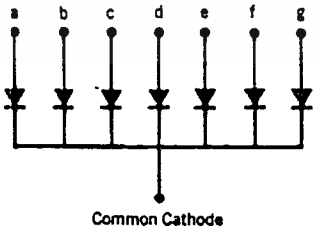
ตัวอย่างที่ทดสอบตรงนี้ ใช้จอแสดงผล 7-segment 4 ชุด ข้อมูลของแต่ละ segment เป็นเอาพุทของพอร์ท1 และ การเลือกคาธอดทางบนพอร์ท 3.2 ถึง 3.5 ที่แสดงในรูป 7 segment จะมีแสงก็ต่อเมื่อสาย segment เป็น highและคาธอดที่ต่อร่วมเป็น low

ทรานซิสเตอร์ต้องใช้จัดการกับกระแสไฟฟ้า ที่ต้องการโดย LED ที่ใช้กัน คือ 10 mA สำหรับ 1 segment และ 70 mA สำหรับ 1 คาธอดนี่เป็นค่ากระแสเฉลี่ย กระแสสูงสุดจะเป็น 4 เท่า เมื่อใช้กระแส 2.5 mS แต่ละจอแสดงผลก็จะแสดงผลออกมา(ไพติด)

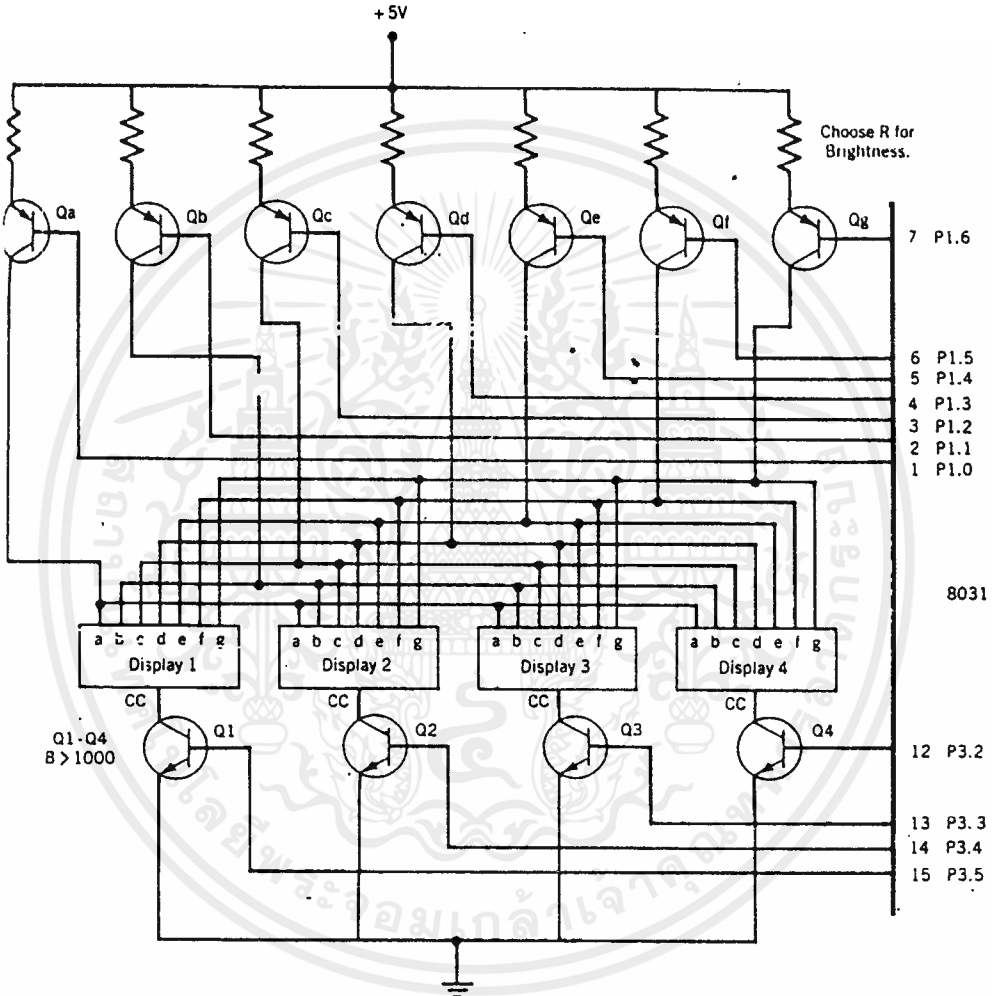
โปรแกรมที่ใช้เมื่อเกิดอินเทอร์รัพ โดย T0 ในลักษณะเดียวกับใช้โปรแกรม Bigkey โปรแกรมอินเทอร์รัพจะเป็น 1 ในตำแหน่งที่ 4 ของตัวอักษร 2 บิร์ท และค้นหารูปแบบ segment ของคาธอดที่จะถูกแลทช์กับพอร์ท1 และรูปแบบอาโนดถูกแลทช์กับพอร์ท โปรแกรมหลักใช้ตาราง lookup เพื่อแปลงจากตัวเลขฐาน 16 เป็นรูปแบบ segment สำหรับเลขนั้นๆ ในทางนี้โปรแกรมอินเทอร์รัพ จะแสดงเลขอะไรก็ตามโดยอัตโนมัติ ที่โปรแกรมหลักวางในตำแหน่งของตัวอักษร ในโปรแกรมหลักโหลดตำแหน่งของตัวอักษร และไม่เกี่ยวข้องกับข้อมูลว่าจะถูกแสดงอย่างไร



Segment Pattern



Segment Circuit



sunseg

โปรแกรม sunseg จะแสดงตัวอักษรที่พบในตำแหน่ง "ch1" ถึง "ch4"บน 7-segment คาโธดร่วม 4 ชุด พอร์ท1 เก็บรูปแบบ segment จากเบร์ทตำของ chx T0 สร้างช่วงเวลาหน่วง 2.5 mS ระหว่างตัวอักษรในหมดอินเทอร์รัพ โปรแกรมหลักใช้ตาราง lookup เพื่อแปลงจากเลขฐาน 16 เป็นรูปแบบที่สมนัยกัน R_0 ของแบงค์1 ใช้เป็นพอยเตอร์ของตัวอักษรที่แสดง

```
ADDRESS    MNEMONIC          COMMENT
           .equ ch1,50h      ;assign RAM character locations
           .equ ch2,52h      ;two bytes per character
           .equ ch3,54h
           .equ ch4,56h
           .org 0000h        ;jump over T0 interrupt location
svnseg:    mov sp,#0fh       ;get the stack above bank one
           sjmp over
;
;begin the interrupt-driven program at the T0 interrupt location
;
            org 000bh
           mov tl0,#0fbh     ;reload T0 for next interrupt
           mov th0,#0f2h
           setb psw.3        ;select bank one
           mov p1,@r0        ;place segment pattern on port 1
           inc r0            ;point to accompanying cathode pattern
           mov p3,@r0        ;place cathode patten on port 3
           inc r0            ;check for fourth character
           cjne r0,#58h,nxt
           mov r0,#ch1       ;if ch4 just displayed go to ch1
nxt:       clr psw.3         ;return to register bank 0
           reti              ;return to main program
;the main program loads sample characters and starts the T0
;interrupt.
           mov a,#00h        ;use an example sequence of 0, 1, 2, 3
           acall convert     ;convert to segment pattern and store
           mov ch1,a
           mov a,#01h
           acall convert
           mov ch2,a
           mov a,#02h
           acall convert
           mov ch3,a
           mov a,#03h
           acall convert
           mov ch4,a         ;last segment pattern stored
           setb psw.3        ;select register bank one
           mov r0,#ch1       ;set R0 to point to ch1 RAM location
           inc r0            ;now load anode pattern for ch1
           mov @r0,#20h      ;set anode for character 1 only high
           inc r0            ;point to next character and continue
           inc r0            ;load ch2 pattern
           mov @r0,#10h
           inc r0
           inc r0            ;load ch3 pattern
           mov @r0,#08h
           inc r0
           inc r0            ;load ch4 pattern
           mov @r0,#04h
           mov r0,#ch1       ;point to RAM address for ch1
           mov tl0,#0fbh     ;load T0 for first interrupt
           mov th0,#0f2h
```

```
ADDRESS     MNEMONIC            COMMENT
            mov tmod,#01h       ;set T0 to mode 1
            mov ie,#82h         ;enable T0 interrupt
            setb tcon.4         ;start timer
            clr psw.3           ;return to register bank 0
here:       sjmp here           ;loop and simulate rest of program
;
;convert uses the PC to point to the base of the 16-byte table
;
convert:    inc a               ;compensate for RET byte
            mov a,@pc+a         ;get byte
            ret                 ;return with segment pattern in A
            .db c0h             ;0
            .db f9h             ;1
            .db a4h             ;2
            .db b0h             ;3
            .db 99h             ;4
            .db 92h             ;5
            .db 82h             ;6
            .db f8h             ;7
            .db f0h             ;8
            .db 98h             ;9
            .db 88h             ;A
            .db 83h             ;b
            .db c6h             ;C
            .db b1h             ;d
            .db 86h             ;E
            .db 8eh             ;F
            .end
```

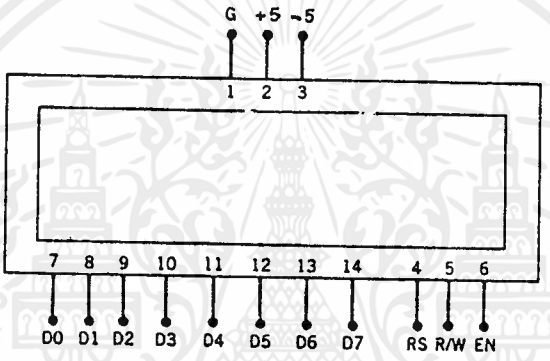
Intelligent LCD Display

ตอนนี้เราตรวจสอบ intelligent LCD display ของสองสาย 20ตัวยักษรต่อสายที่อินเตอร์เฟสกับ 8051 การ handshaking ของส่วนแสดงผล แสดงในรูป 7 และอินเตอร์เฟสกับ 8051 ในรูป 8

จอแสดงภาพประกอบด้วยรีจิสเตอร์ภายใน 2 ตัว ตัวหนึ่งสำหรับคำสั่ง(RS=0) และอีกตัวสำหรับตัวยักษรที่แสดง(RS=1) จอแสดงผลประกอบด้วยพื้นที่แรมที่โปรแกรมสำหรับผู้ใช้ ซึ่งสามารถถูกโปรแกรมเพื่อสร้างตัวยักษรที่ต้องการ ซึ่งถูกทำเป็นรูปแบบโดยใช้ dot matrix เพื่อแยกความแตกต่างระหว่างพื้นที่ข้อมูลทั้งสองนี้ บิตคำสั่งฐาน 16 ค่า 80 จะใช้เพื่อแสดงความหมายที่แอดเดรส 00H ของแรมของจอแสดงผลที่ถูกเลือก

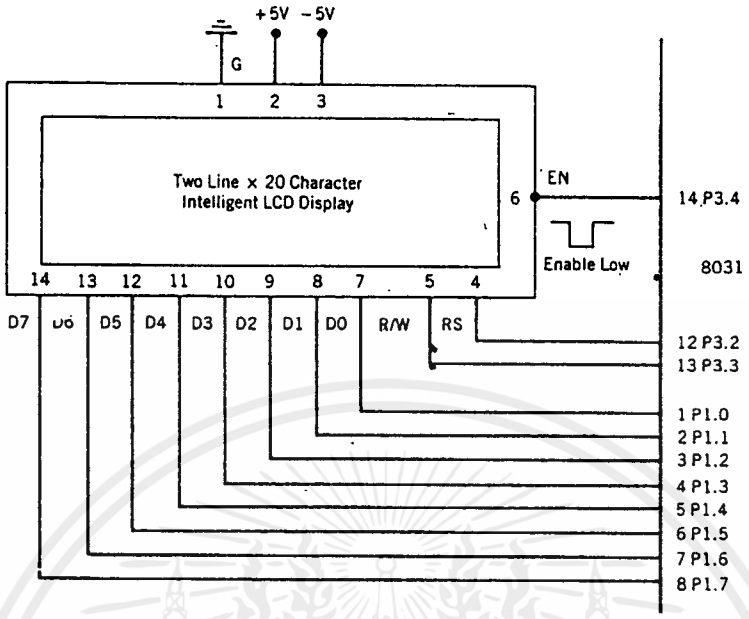
พอร์ท1 ถูกใช้เพื่อทาคาสั่ง หรือไบร์ทย้อมูล และพอร์ท3.2 ถึง3.4 ใช้เลือก รีจิสเตอร์และระดับที่เขียนหรืออ่าน

จอแสดงผลใช้จำนวนเวลาต่างๆกัน เพื่อให้ฟังก์ชันในรูป 8 ทาเสร็จสิ้น LCD บิท 7 ถูกติดตามว่ามีลอจิกเป็น 1 เพื่อให้มั่นใจว่าจอแสดงภาพจะไม่ถูกเขียนทับ จอแสดงผล LCD ที่ซับซ้อนกว่าเล็กน้อย คือ (4lines*40lines) ที่ถูกใช้เมื่อไม่นานมานี้ในการวินิจฉัยทางการแพทย์ เพื่อให้บรแกรมที่คุ้นเคยมากๆทำงาน



Intelligent LCD Display

BIT	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Function
	0	0	0	0	0	0	0	0	0	1	Clear LCD and memory, home cursor
	0	0	0	0	0	0	0	0	1	0	Clear and home cursor only
	0	0	0	0	0	0	0	1	I/O	S	Screen action as display character written S = 1/0: Shift screen/cursor I/O = 1/0: Cursor R/L, screen L/R
	0	0	0	0	0	0	1	D	C	B	D = 1/0: Screen on/off C = 1/0: Cursor on/off B = 1/0: Cursor Blink/Noblink
	0	0	0	0	0	1	S/C	R/L	0	0	S/C = 1/0: Screen/Cursor R/L = 1/0: Shift one space R/L
	0	0	0	0	1	DL	N	F	0	0	DL = 1/0: 8/4 Bits per character N = 1/0; 2/1 Rows of characters F = 1/0; 5X10/5X7 Dots/Character
	0	0	0	1	Character address						Write to character RAM Address after this
	0	0	1	Display data address							Write to display RAM Address after this
	0	1	BF	Current address							BF = 1/0: Busy/Notbusy
	1	0		Character byte							Write byte to last RAM chosen
	1	1		Character byte							Read byte from last RAM chosen



LCdisp

โปรแกรม Lcdisp ส่งข่าวสาร "hello" แก่ intelligent LCD display แสดงในรูป 8 พอร์ท1 ให้เป็นข้อมูล พอร์ท3.2 เลือกรีจิสเตอร์เป็นคำสั่ง0 หรือข้อมูล 1 พอร์ท3.3 เป็นการอ่าน 0 หรือ เขียน 1 และ พอร์ท3.4 สร้างการสะดวบแอคทีฟ low

โปรแกรม Lcdisp แสดงอยู่ด้านบน

การวัดพัลส์

เซนเซอร์ถูกใช้สำหรับแอพพลิเคชั่น ที่ควบคุมทั้งทางอุตสาหกรรมและพาณิชย์ บ่อยครั้งที่สร้างพัลส์ที่มีข้อมูลเกี่ยวกับการตรวจสอบปริมาณ ความถี่เอาพุทของเซนเซอร์บนิดต่างๆใช้ duty cycle คงที่ แต่พัลส์ของความถี่ที่เปลี่ยนแปลงได้เป็นตัวชี้การเปลี่ยนแปลงในตัวแปรที่สามารถวัดได้ เป็นแบบที่ธรรมดาที่สุด การเปลี่ยนระยะความกว้างพัลส์ มีผลในความถี่ที่คงที่

ความถี่ที่วัด

ทามเมอร์ T0 และ T1 ใช้วัดความถี่ภายนอก โดยทามเมอร์ตัวหนึ่งเป็นเคาน์เตอร์ และใช้อีกตัวสร้างช่วงเวลา timing ในการนับครั้งแรก ความถี่ของขบวนพัลส์ที่นับเป็น

unknown freq. = counter/timer

สาหรับตัวอย่าง ถ้าเคาน์เตอร์นับ 200 พัลส์ ในเวลา 0.1S ที่สร้างโดยทามเมอร์ ความถี่คือ

UF = 200/0.1 = 2000Hz

พัลส์อินพุทต้องเปลี่ยนจาก 1 เป็น 0 ใน 2 แมชชีนไซเคิลสุดท้ายหรือ f/24 ถูกนับข้อจากัดบนการเปลี่ยนแปลงพัลส์นี้ จะให้ความถี่ 667 Hz ซึ่งใช้คริสตัล 16Mz

ความถี่ต่าสุดที่สามารถนับ ถูกจากัดโดยระยะเวลาที่สร้างซึ่งสามารถนานเกินกว่าการใช้แรมทั้งหมดนับ timer rolloversมีข้อจากัดที่ไม่สามารถทาได้บนความถี่ที่ต่าที่สุดที่ถูกนับ

โดยทั่วไปเซนเซอร์ที่เปลี่ยนความถี่ได้ ส่วนมากสร้างสัญญาณที่อยู่ในช่วง 0 ถึง 667 Hz บ่อยครั้งสัญญาณอยู่ในช่วง 1,000 ถึง 10,000Hz

ตัวอย่างที่ใช้เซนเซอร์วัดโวลเตจ dc จาก 0 ถึง 5 โวลต์ ที่ 0 โวลต์เอาพุทเซนเซอร์เป็น 1,000Hz และที่พิกัด 5โวลต์ เอาพุทเป็น 10,000Hz และเราต้องการวัดโวลเตจให้ถูกต้อง 0.01 โวลต์มากที่สุด หรือ 10Hz ช่วงเวลา 1S สร้างความถี่ที่นับได้อย่างแน่นอน คือเข้าใกล้ 1 Hz

<u>การวัดความกว้างพัลส์</u>

ในทางทฤษฎี ถ้ารู้ว่าพัลส์อินพุทเป็น square wave ความถี่ของพัลส์สามารถวัดได้โดยหาเวลาที่คลื่นเป็น high (T_h) ความถี่จะเป็น

UF = 1/(T_h*2)

ถ้า T_h เป็น 200 uS ดังตัวอย่างนี้ จะได้ UF = 2500 Hz ความแน่นอนในการวัดจะเป็นคลื่นอินพุทที่มี duty cycle 50%

ทามเมอร์x อาจถูกใช้เพื่อจับคลื่นภายในถูกนับ เฉพาะเมื่อขา INTX เป็น high โดยเซ็ทบิทที่ GATEx ใน TMOD ความแน่นอนในการวัดจะใกล้เคียงกับ 1/2 ของคาบของทามเมอร์ หรือ 0.375 uS สาหรับคริสตัล 16 MHz ความแน่นอนนี้หาได้ก็ต่อเมื่อ เริ่มต้นวัดคลื่นอินพุทที่ low และหยุดเมื่ออินพุทต่อไปเป็น low ความกว้างของพัลส์ที่มากกว่าความสามารถของเคาน์เตอร์ คือ 49.152 mS สามารถวัด

โดยนับจากเวอร์โพล์ของแพลกทามเมอร์ และเพิ่มข้อมูลในเคาน์เตอร์

เพื่อจะวัดความแน่นอน T0 จะใช้นับคล็อกภายใน เมื่อ INT0 เป็น high การวัดจะไม่เริ่มจนกว่า INT0 เปลี่ยนจาก high เป็น low ใช้เวลาอย่างน้อย 100 uS เพื่อเริ่ม T0 การวัดทำในขณะที่ INT0 เป็น high และหยุดเมื่อ INT0 เป็น low อีกครั้ง ขบวนการนี้สามารถถูกอินเทอร์รัพโดยใช้แพลกอินเทอร์รัพกับINT0 แพลก IE0 สามารถเซ็ทเมื่อ INT0 เปลี่ยนจาก high เป็น low เพื่อบอกให้โปรแกรมเริ่มนับความกว้างพัลส์ และต่อไปก็หยุด

D/A and A/D conversions

การแปลงระหว่าง อนาล็อก กับดิ จิตอล ต้องใช้วงจร IC ที่ออกแบบให้อินเตอร์เฟสกับคอมพิวเตอร์ ตัวแปลงที่ดีควรมีลักษณะดังนี้

บัสข้อมูลขนาน: tri-state 8บิท

บัสควบคุม:สามารถเลือก อ่าน/เขียน ข้อมูล ,ทำงาน/ว่าง

นักออกแบบจะต้องใช้ตัวแปลงเป็นตำแหน่งความจำของแรม ต่อกับบัสความจำ หรือว่าเป็นอุปกรณ์ I/O ต่อกับพอร์ท หลังจากเลือกแล้ว คำสั่งของโปรแกรม จะถูกจำกัดลง การแปงตำแหน่งความจำเป็นข้อจำกัดที่มากที่สุด นักออกแบบควรใช้แอดเดรสของแรม 32K ด้านบนสำหรับตัวแปลง เป็นการขยายความสามารถของพอร์ทโดยใช้แผนผังความจำ

D/A conversions

ตัวแปลง D/A ชนิด ชนิด R-2R เป็นพื้นฐานที่ใช้กันอยู่ ถูกต่อกับพอร์ท1 และ 3 ดังรูป 9 พอร์ท1 จะแปลงเบร์ทดิจิตอลเป็นวาลเตจอนาล็อก พอร์ท3 ควบคุมขบวนการแปลง D/A ตัวแปลงมี 3 ลักษณะ

$V_{out} = -V_{ref} * (\text{byte in}/100H)$

$V_{ref} = +10$ หรือ $-10V$

conversion time : 5uS

control sequence : CS then WR

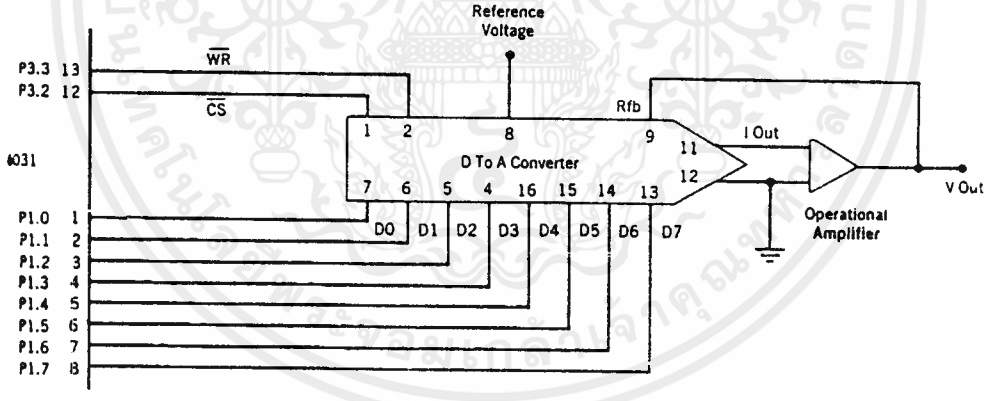
สำหรับตัวอย่างคลื่นซายน์ 100 Hz ที่ถูกสร้างสามารถโปรแกรมความถี่ได้

V_{ref} เลือกเท่ากับ -10โวลต์ และคลื่นจะแกว่งจาก +9.96 โวลต์ กับ 0 โวลต์ รอบจุดกลาง 4.48 โวลต์ โปรแกรมใช้ตาราง lookup สร้างขนาดของแต่ละจุดของคลื่นซายน์ ช่วงเวลาที่ตัวแปลงนาโนร์ทจากตารางเป็นตัวกำหนดความถี่ของคลื่น

เวลาของการแปลงจากัดความถี่สูงสุด ที่สามารถสร้างโดยใช้จุดตัวอย่าง S ในตัวอย่างคาบเวลาที่น้อยที่สุดเป็น

$$T_{min} = S * 5uS \quad , \quad f_{max} = 200,000/S$$

แนวคิดที่ออกแบบเป็นความถี่สูง กับผลลัพธ์ที่ high สำหรับคลื่น 1000 Hz S ควรเป็น 200d ในความเป็นจริง เราไม่สามารถใช้ตัวอย่างมากมายเช่นนี้ โปรแกรมไม่สามารถเพทช์ข้อมูล แลทช์ข้อมูลในพอร์ท 1 และสวิตรบพอร์ท 3.3 ใน 5 uS การตรวจตราโปรแกรมจะแสดงว่า เวลาที่ใช้สำหรับจุดของคลื่นเดียว 1 จุด เป็น 6 uS และเวลาสำหรับคลื่นต่อๆ ไปใช้ 2.25 uS S จะเป็น 166d ที่ความถี่ 1001.75 Hz



A/D conversion

การแปลง A/D ที่ง่ายที่สุด คือแบบ flash ซึ่งทำให้การแปลงที่เป็นอาร์เรย์ของคอมพาราเตอร์ภายใน การแปลงที่เร็วที่สุดน้อยกว่า 1 uSดังนั้นคอนเวอร์เตอร์สามารถถูกบอกให้เริ่มต้น และค่าดิจิตอลที่ตรงกับอนาล็อกอินพุทจะถูกจำนวน 1 หรือ 2 คำสั่งต่อมา คอนเวอร์เตอร์แบบรีจิสเตอร์ที่ใกล้เคียงอย่างฉลาดจะไม่ lagอย่าง

ไรก็ตามเวลาของการแปลงอยู่ในช่วง 2-4 uS สำหรับ 8 บิท

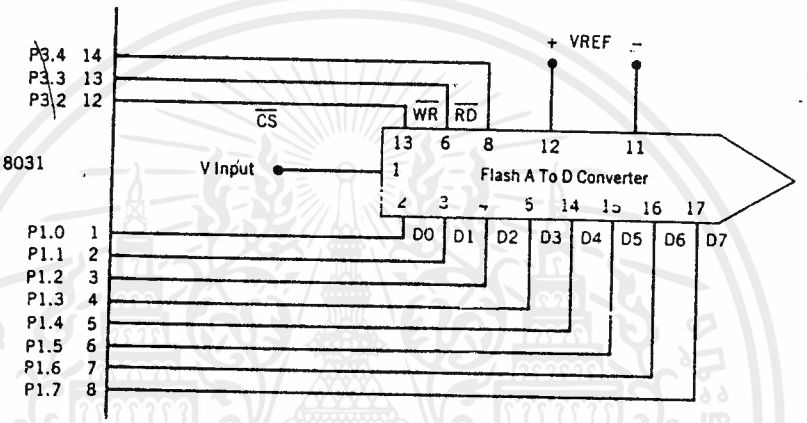
ลักษณะคอนเวอร์เตอร์แบบ flash เป็นดังนี้

Data:$V_{in} = V_{ref}$,data = 00h ; $V_{in} = V_{ref}$,data = FFh

conversion time: 1uS

control sequence: CS then WR then RD

วงจรตัวอย่างแสดงในรูป 10



พอร์ท1 ใช้อ่านค่าของเบร์ทอนาล็อกเวลเตจอินพุท และพอร์ท 3 ควบคุมการแปลงจะเริ่มโดยพัลส์ เส้นเขียนเป็น low และข้อมูลถูกอ่านโดยทำให้เส้นอ่านเป็น low

<u>การอินเทอร์รัพหลายชั้น</u>

8051 มีอินเทอร์รัพจากภายนอก 2 ชั้น คือที่ขา INT0 และ INT1 ซึ่งพอเพียงสำหรับระบบเล็กๆ แต่ความต้องการอาจเพิ่มมากกว่าอินเทอร์รัพ 2 ขานี้ มีอยู่หลายลักษณะที่ขยายจำนวนชุดอินเทอร์รัพ ซึ่งขึ้นอยู่กับลักษณะดังนี้

การต่อแหล่งอินเทอร์รัพเป็นส่วนรวม

กำหนดแหล่งอินเทอร์รัพที่ใช้ซอฟแวร์

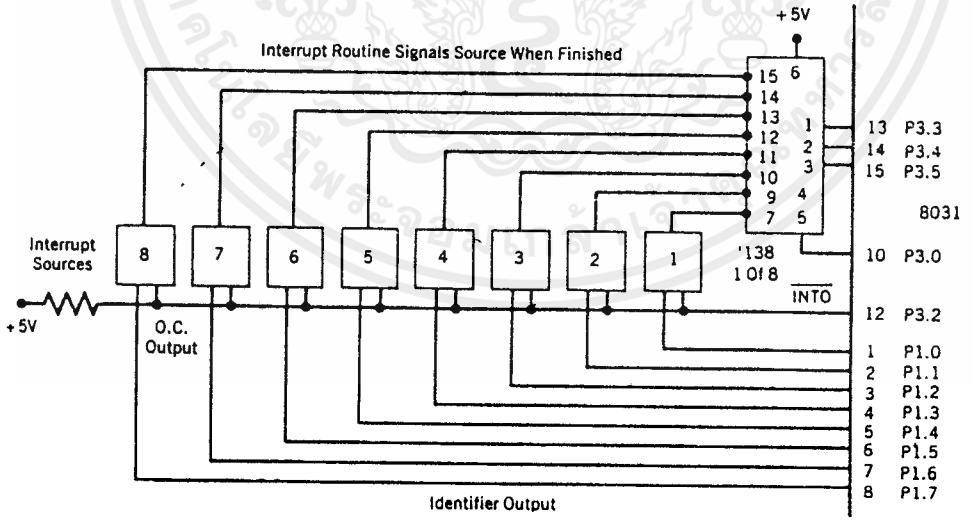
เพราะว่าอินเทอร์รัพภายนอกจะแอคทีฟ low การต่อจากแหล่งอินเทอร์รัพไปยังขา INTX ต้องใช้ open-collector หรืออุปกรณ์ tri-state

ตัวอย่างการเพิ่ม INT0 จาก 1 เป็น 8 ชุด ด้วยรูป 11 แต่ละแหล่งจะแอค

ที่พ low เมื่อต้องการอินเทอร์รัพ ขาของพอร์ทหนึ่งที่สมนัยจะเต้ แหล่งอินเทอร์รัพที่กำหนดมา โปรแกรมอินเทอร์รัพจะไปจัดการกับอินเทอร์รัพ แหล่งอินเทอร์รัพต้องรับรู้เพื่อว่า สายอินเทอร์รัพของแหล่งสามารถกลับเป็น high พอร์ท 3 ขา 3.3, 3.4 ,และ 3.5 ให้ผ่าน 3-to-8 decoder สัญญาณย้อนกลับจะกลับไปยังแหล่งอินเทอร์รัพที่ถูกต้อง ตัวถอดรหัสใช้โดยขา 3.0

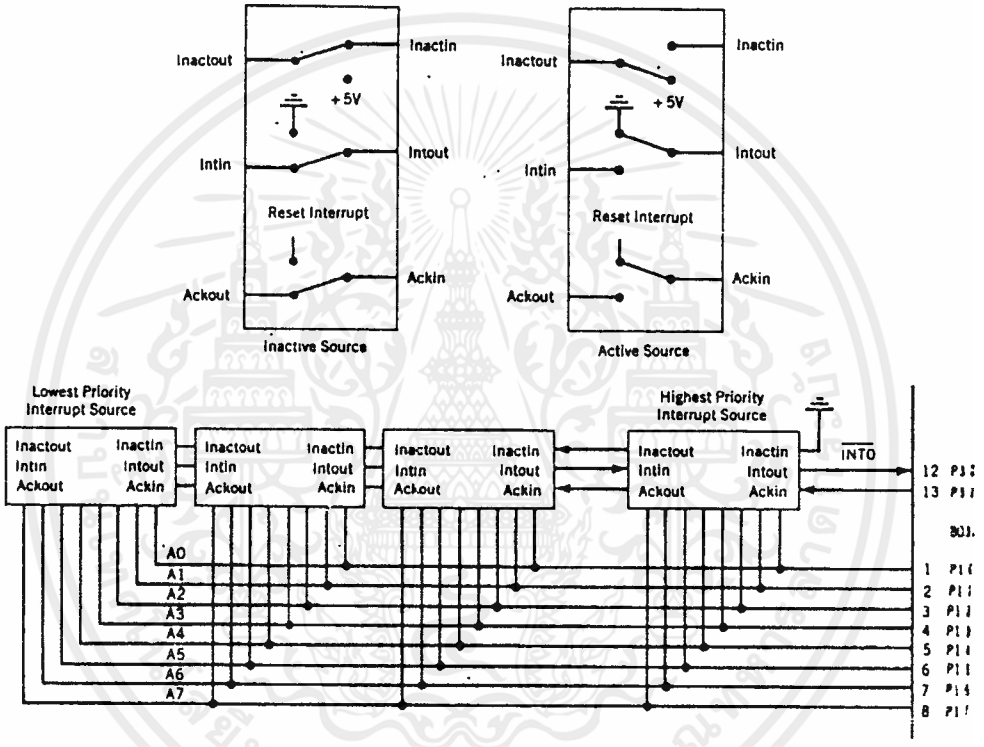
การอินเทอร์รัพหลายๆตัวพร้อมกัน จะจัดการโดยโปรแกรมที่ซับซ้อน ถ้าไม่มีความต้องการอินเทอร์รัพใดๆอย่างเร่งด่วน แล้วก็สามารถจัดการที่ขาพอร์ท1ในการสแกนค่า low

ระบบ priority ธรรมดา สามารถสร้างเมื่อแหล่งอินเทอร์รัพที่สำคัญมากสุดตรวจสอบเป็นลำดับแรก และโปรแกรมอินเทอร์รัพจะทำงานจนกว่าจะเสร็จระบบ priority ที่ละเอียดเกี่ยวข้องกับลำดับการมาก่อนของแต่ละแหล่ง ระบบที่ละเอียดจะรู้การอินเทอร์รัพโดยทันที ดังนั้นการรีเซ็ทสายอินเทอร์รัพและเริ่มจัดจัดการโปรแกรมอินเทอร์รัพของแหล่ง อินเทอร์รัพตัวใหม่จากแหล่งที่สำคัญกว่า จะทำให้โปรแกรมอินเทอร์รัพที่ทำอยู่ก่อนหยุด และทำตัวอินเทอร์รัพใหม่



วงจรฮาร์แวร์สำหรับอินเทอร์รัพหลายตัว

ผลของปัญหาเรื่องอินเทอร์รัพที่มีมากขึ้น จะเน้นการใช้จำนวนวงจรภายนอกที่น้อยที่สุด ที่จะจัดการกับอินเทอร์รัพหลายๆตัว ที่เกิดพร้อมกัน วิธีทางฮาร์ดแวร์สามารถถูกขยายเป็นแหล่งอินเทอร์รัพ 256 แหล่ง ดังรูป 12 วงจรนี้เป็นที่นิยมมาก



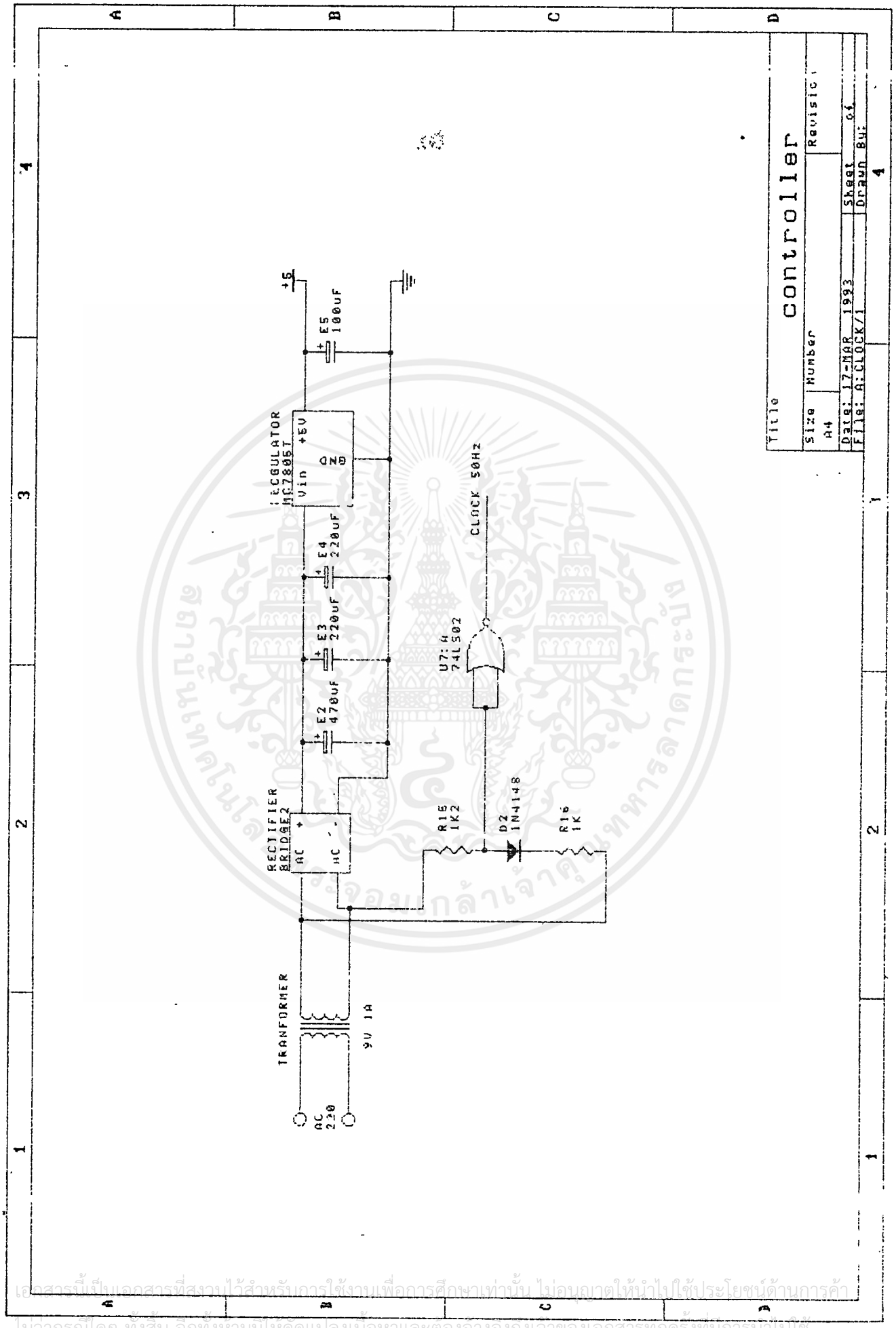
ความคิดทั้งหมดในการออกแบบมีดังนี้

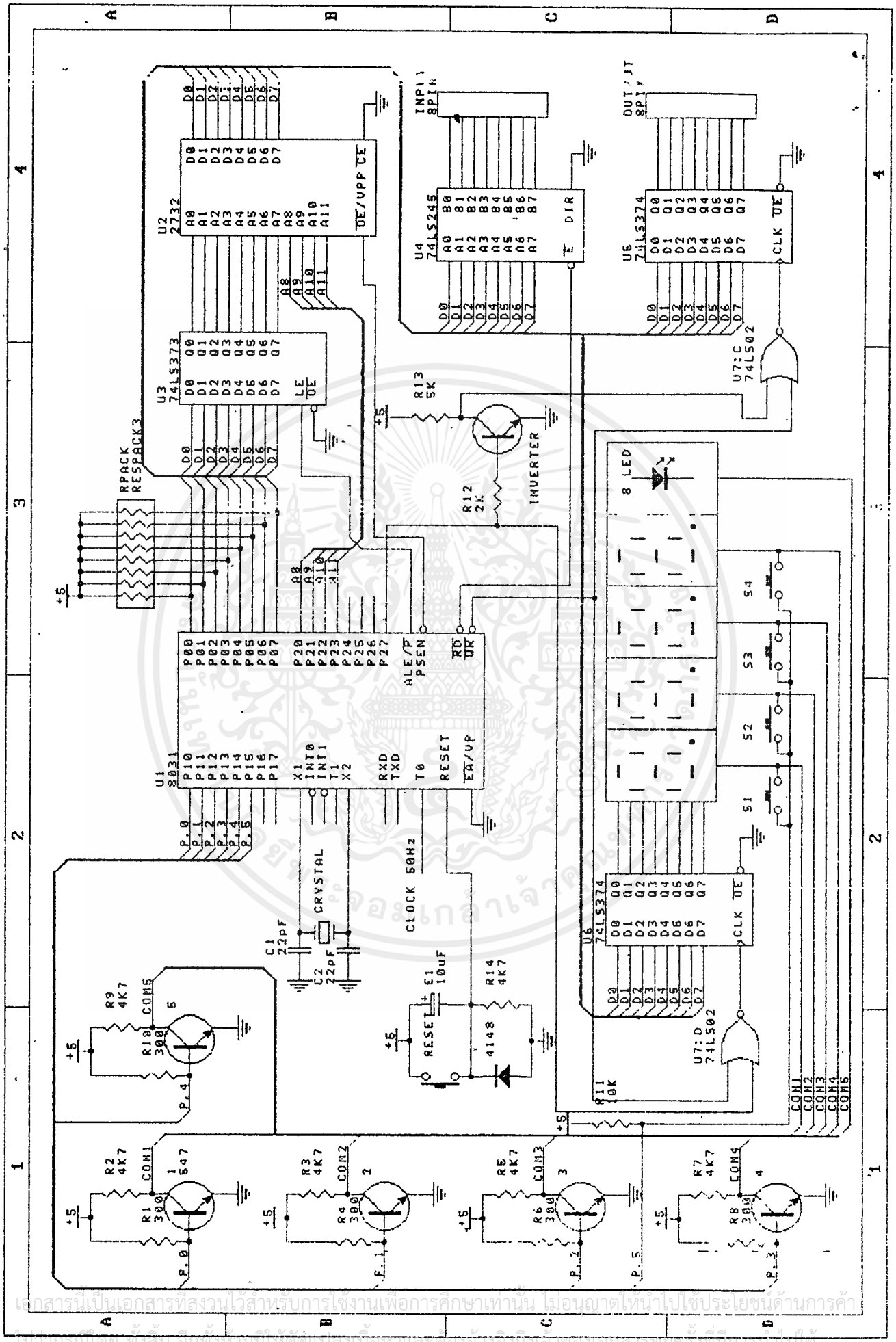
1.แหล่งอินเทอร์รัพที่สำคัญที่สุด ถูกต่อกันในลักษณะลูกโซ่ กับแหล่งที่สำคัญน้อยกว่า แหล่งที่สำคัญน้อยกว่าจะอยู่ข้างหลัง

2.แหล่งอินเทอร์รัพสามารถหยุดสัญญาณทั้งหมดที่อยู่ ที่อยู่ข้างหลังทุกแหล่ง

3.รูทีนของอินเทอร์รัพสร้างสัญญาณ ACKIN แก่แหล่งจาก 8051 ที่จุดสิ้นสุดของรูทีน ต่อจากนั้นแหล่งอินเทอร์รัพจะเคลื่อนย้าย INOUT และแอดเดรสของแหล่ง

ขนาด 8 บิท เมื่อรับรู้การอินเทอร์รัพ แหล่งอินเทอร์รัพต้องนำสาย INOUT เป็น high อย่างน้อยที่สุด 1 แมชชีนไซเคิล อินเทอร์รัพสามารถจากการเปลี่ยนจากhigh เป็น low บน INTO ได้





ทดสอบ hardware controller

program test_port1 ใช้ทดสอบการทำงานของ 8031 ร่วมกับ ROM และตัว latch address เป็นการทดสอบการ fetch คำสั่งจาก program memory เข้ามาทำงาน program และผลการทำงานแสดงไว้ในรูป A

program test_output ทดสอบการ fetch คำสั่งจาก program memory เพื่อส่ง output ไปแสดงผลที่ port output [74LS374 (U5)] โดยใช้คำสั่ง MOVX @DPTR,A เมื่อ address A15 = "1" program และผลการทำงานแสดงไว้ในรูป B

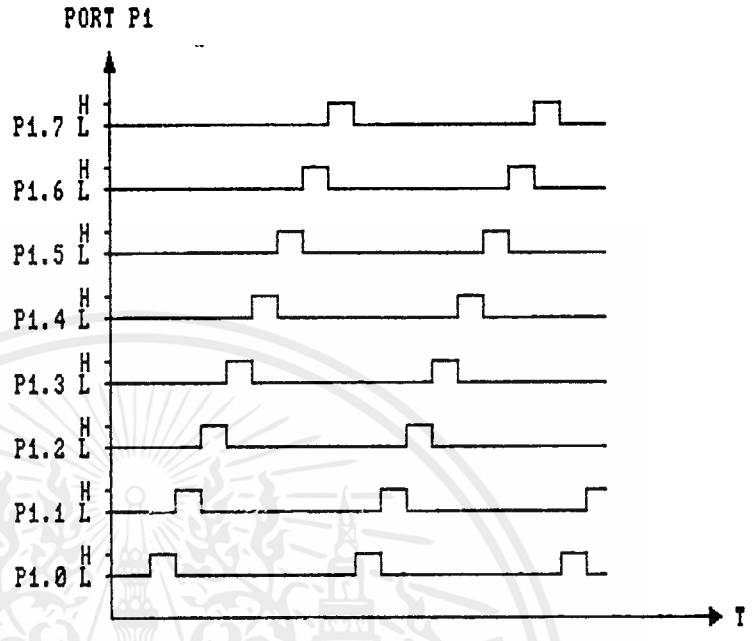
program test_input ใช้ทดสอบการ fetch คำสั่งจาก program memory เพื่อรับ input จาก port input (74LS245) โดยใช้คำสั่ง MOVX A,@DPTR และ แสดงผลการรับที่ port output [74LS374 (U5)] โดยใช้คำสั่ง MOVX @DPTR,A เมื่อ address A15 = "1" program และผลการทำงานแสดงไว้ในรูป C

program test_display ทดสอบการ fetch คำสั่งจาก program memory เพื่อส่ง output ไปแสดงผลที่ seven segment ผ่านตัว latch 74LS374 (U6) โดยใช้คำสั่ง MOVX @DPTR,A เมื่อ address A15 = "0" รวมทั้งทดสอบการ drive transistor ซึ่งทำหน้าที่เป็นขา common cathode program และผลการทำงานแสดงไว้ในรูป D

program test_key ใช้ทดสอบการ fetch คำสั่งจาก program memory เพื่อ ทดสอบการกด key โดยสภาวะปกติ (ไม่มีการกด key) P1.5 = "1" และ แสดงผลการรับที่ port output [74LS374 (U5)] โดยใช้คำสั่ง MOVX @DPTR,A เมื่อ address A15 = "1" program และผลการทำงานแสดงไว้ในรูป E

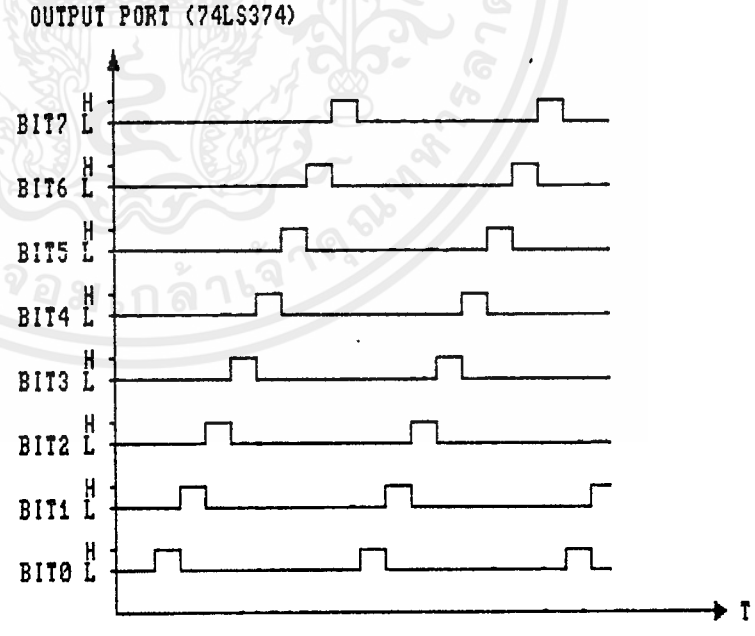
ผลการทดสอบ hardware controller
ได้ผลตามการออกแบบวงจร

```
TEST_PORT1: MOV     A,#01H
LOOP:       MOV     P1,A
            ACALL   DELAY
            RL      A
            SJMP    LOOP
```



FIGTURE A

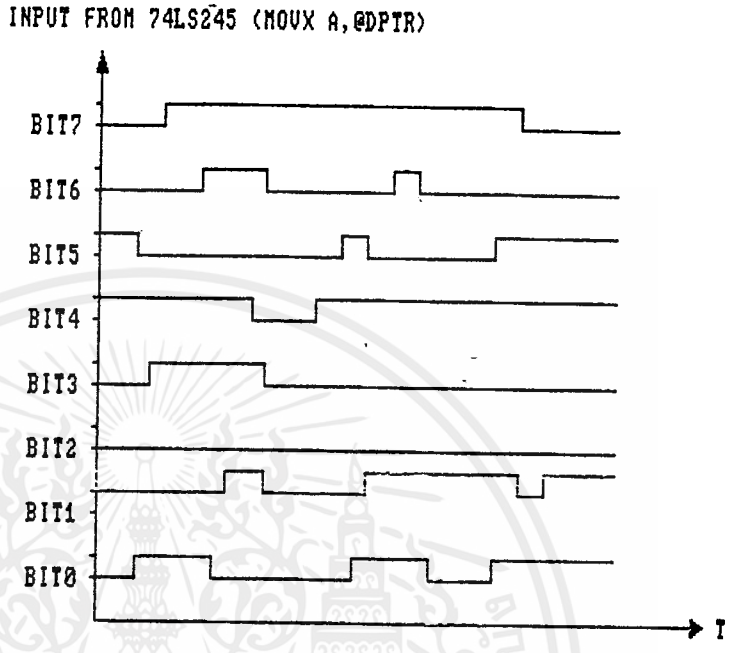
```
T_OUTPUT:   MOV     A,#01H
            MOV     DPH,#80H
LOOP:       MOVX    @DPTR,A
            ACALL   DELAY
            RL      A
            SJMP    LOOP
```

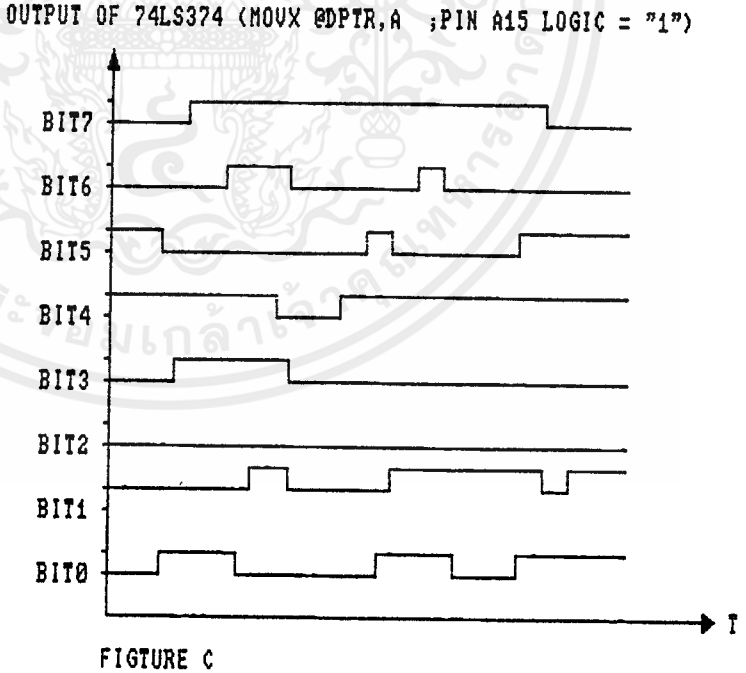


FIGTURE B

```
DELAY:      MOV     R6,#0FFH
FIRST:      MOV     R7,#0FFH
SECOND:     DJNZ    R7,SECOND
            DJNZ    R6,FIRST
            RET
```

```
T_INPUT:    MOV     DPH,#80H
LOOP:       MOVX    A,@DPTR
            NOP
            MOVX    @DPTR,A
            SJMP    LOOP
```

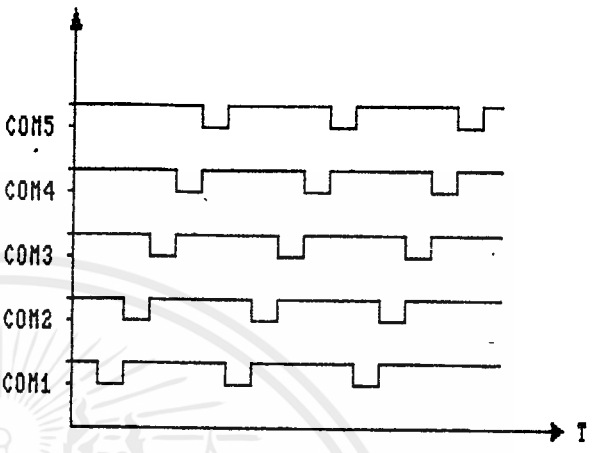




FIGTURE C

```
T_DISPLAY:  MOV     DPH,#00H
            MOV     A,#0FFH
            MOVX    @DPTR,A
            MOV     R0,#01H
LOOP:       MOV     P1,R0
            ACALL   DELAY
            MOV     A,R0
            RL      A
            MOV     R0,A
            CJNE    A,#20H,LOOP
            MOV     R0,#01H
            SJMP    LOOP
```

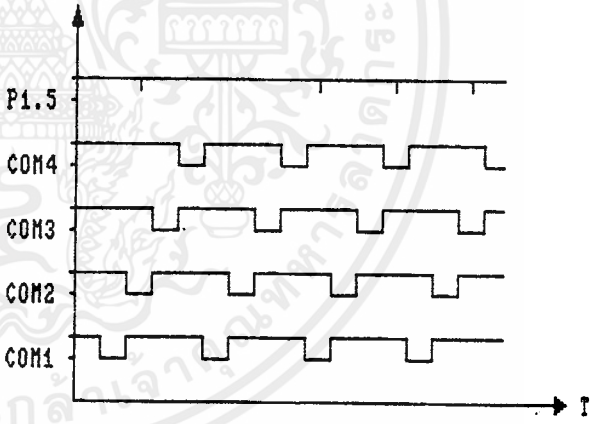
COMMON PIN OF SEVEN SEGMENT



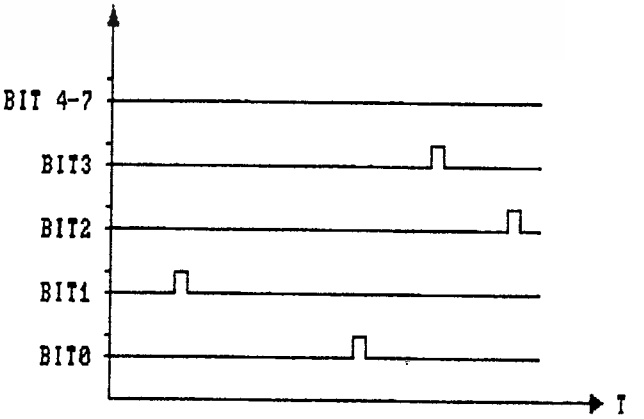
FIGTURE D

```
T_KEY:      MOV     R0,#01H
LOOP:       MOV     A,#20H
            ORL     A,R0
            MOV     P1,A
            NOP
            MOV     A,P1
            JNZ     KEY_IN
            MOV     A,R0
            RL      A
            MOV     R0,A
            CJNE    A,#10H,LOOP
            MOV     R0,#01H
            SJMP    LOOP
KEY_IN:     MOVX    @DPTR,A
            ACALL   DELAY
            CLR     A
            MOVX    @DPTR,A
            SJMP    LOOP
```

P1.5="0" SAME KEY_IN



OUTPUT PORT (74LS374 WHEN A15="1")



FIGTURE E

INTERNAL RAM

VARIABLE VALUE

		CODE USER	THO VALUE	TLO VALUE
11)	12H	XXH	1DH XXH	28H XXH
10)	11H	XXH	1CH XXH	27H XXH
9)	10H	XXH	1BH XXH	26H XXH
8)	0FH	XXH	1AH XXH	25H XXH
7)	0EH	XXH	19H XXH	24H XXH
6)	0DH	XXH	18H XXH	23H XXH
5)	0CH	XXH	17H XXH	22H XXH
4)	0BH	XXH	16H XXH	21H XXH
3)	0AH	XXH	15H XXH	20H XXH
2)	09H	XXH	14H XXH	1FH XXH
1)	08H	XXH	13H XXH	1EH XXH

TABLE OF USER

INTERNAL RAM

VARIABLE VALUE

2EH	เก็บค่าสถานะ output signal
2DH	แสดงค่า fall ซึ่งเป็นค่าที่ปัจจุบันที่สุด
2CH	byte ที่ใช้ตรวจสอบเงื่อนไข delay from cut
2BH	byte ที่ใช้ตรวจสอบเงื่อนไข delay read (I,V)
29H	address แรกที่ว่างของตารางเวลา

MSB LSB

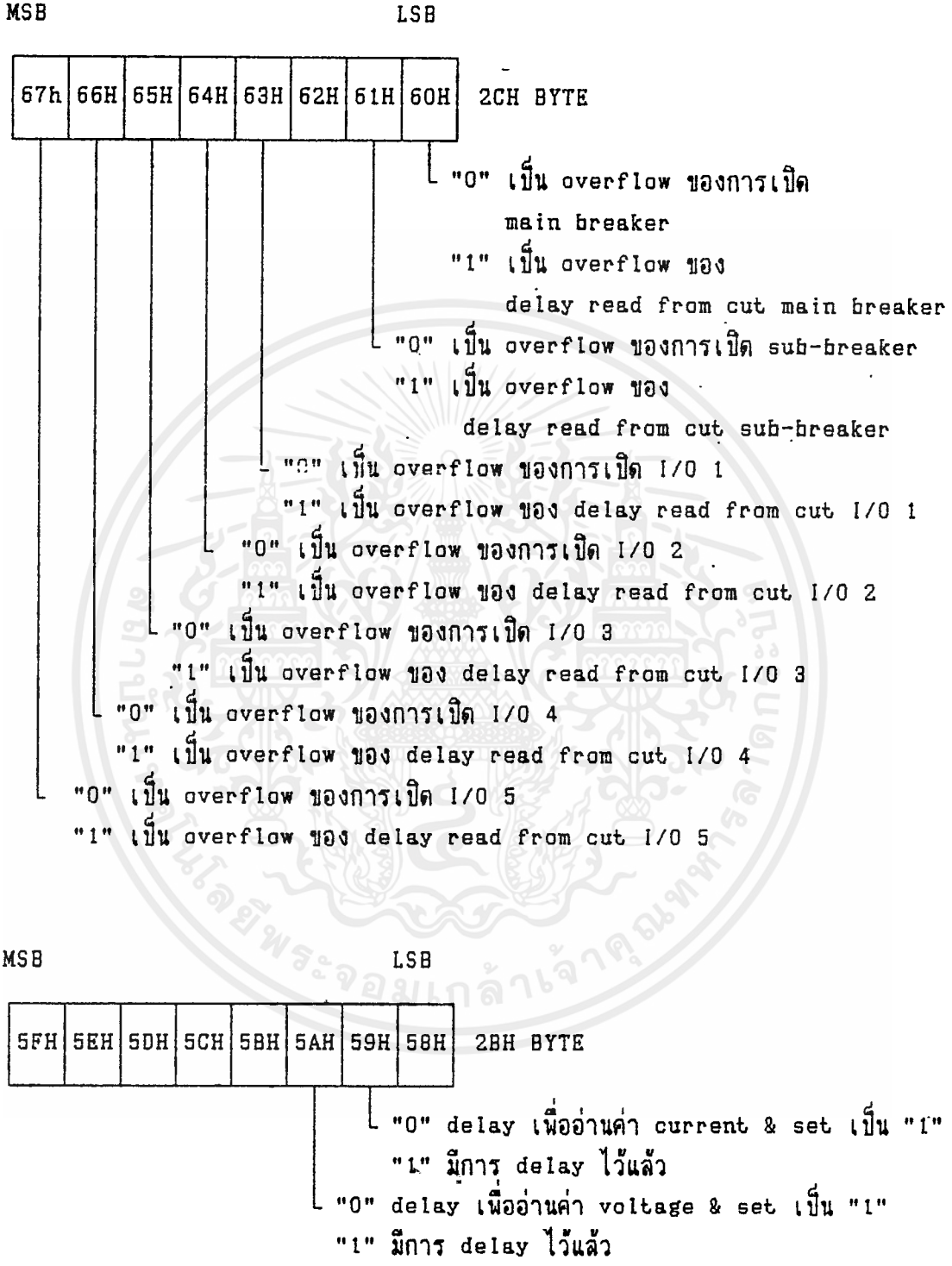
77H	76H	75H	74H	73H	72H	71H	70H	2EH BYTE
I/O 5	I/O 4	I/O 3	I/O 2	I/O 1		V&I	E	

output signal for

MSB LSB

6FH	6EH	6DH	6CH	6BH	6AH	69H	68H	2DH BYTE
I/O 5	I/O 4	I/O 3	I/O 2	I/O 1	V	I	E	

fall ที่ปัจจุบันที่สุดของ



INTERNAL RAM

VARIABLE VALUE

delay read from cut when THO = FFH

TLO VALUE

I/O 5	35H	01H
I/O 4	34H	01H
I/O 3	33H	01H
I/O 2	32H	01H
I/O 1	31H	01H
I,V	30H	F6H
E	2FH	F1H

value of delay open

THO VALUE		TLO VALUE	
3CH	FFH	43H	01H
3BH	FFH	42H	01H
3AH	FFH	41H	01H
39H	FFH	40H	01H
38H	FFH	3FH	01H
37H	B9H	3EH	B0H
36H	FEH	3DH	0CH

44H	F6H	value of delay read I
45H	F6H	value of delay read V

ค่าหน่วงเวลาต่างๆ set ค่าเริ่มแรกโดย load จาก ROM
สามารถเปลี่ยนแปลงค่าได้ตามความต้องการของผู้ใช้

INTERNAL RAM

VARIABLE VALUE

4AH	buffer ของ seven segment ตัวที่ 1
4BH	buffer ของ seven segment ตัวที่ 2
4CH	buffer ของ seven segment ตัวที่ 3
4DH	buffer ของ seven segment ตัวที่ 4

LED SEVEN SEGMENT 4 DIGIT

EXTERNAL ROM

CONSTANT VALUE

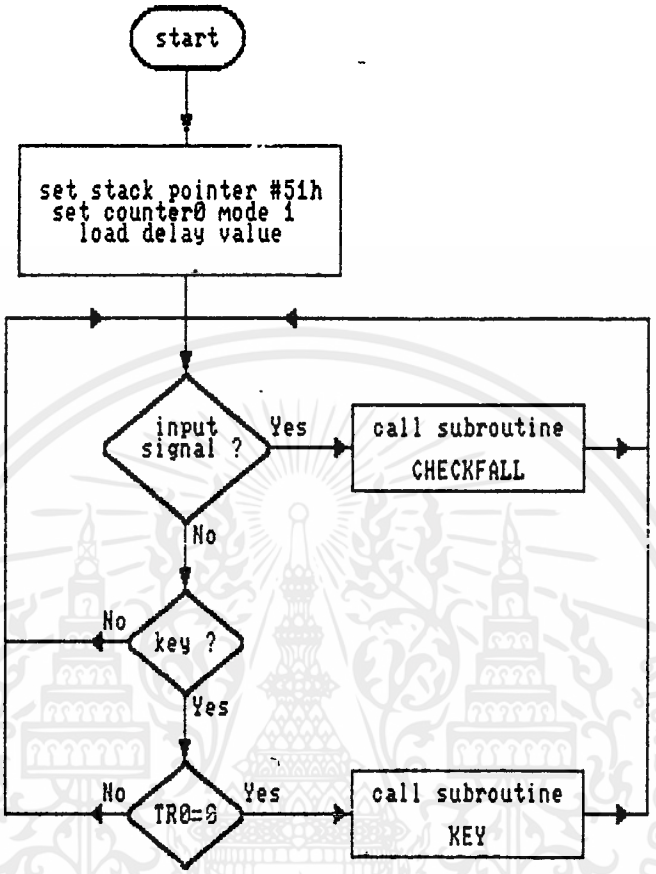
delay read from cut when TH0 = FFH

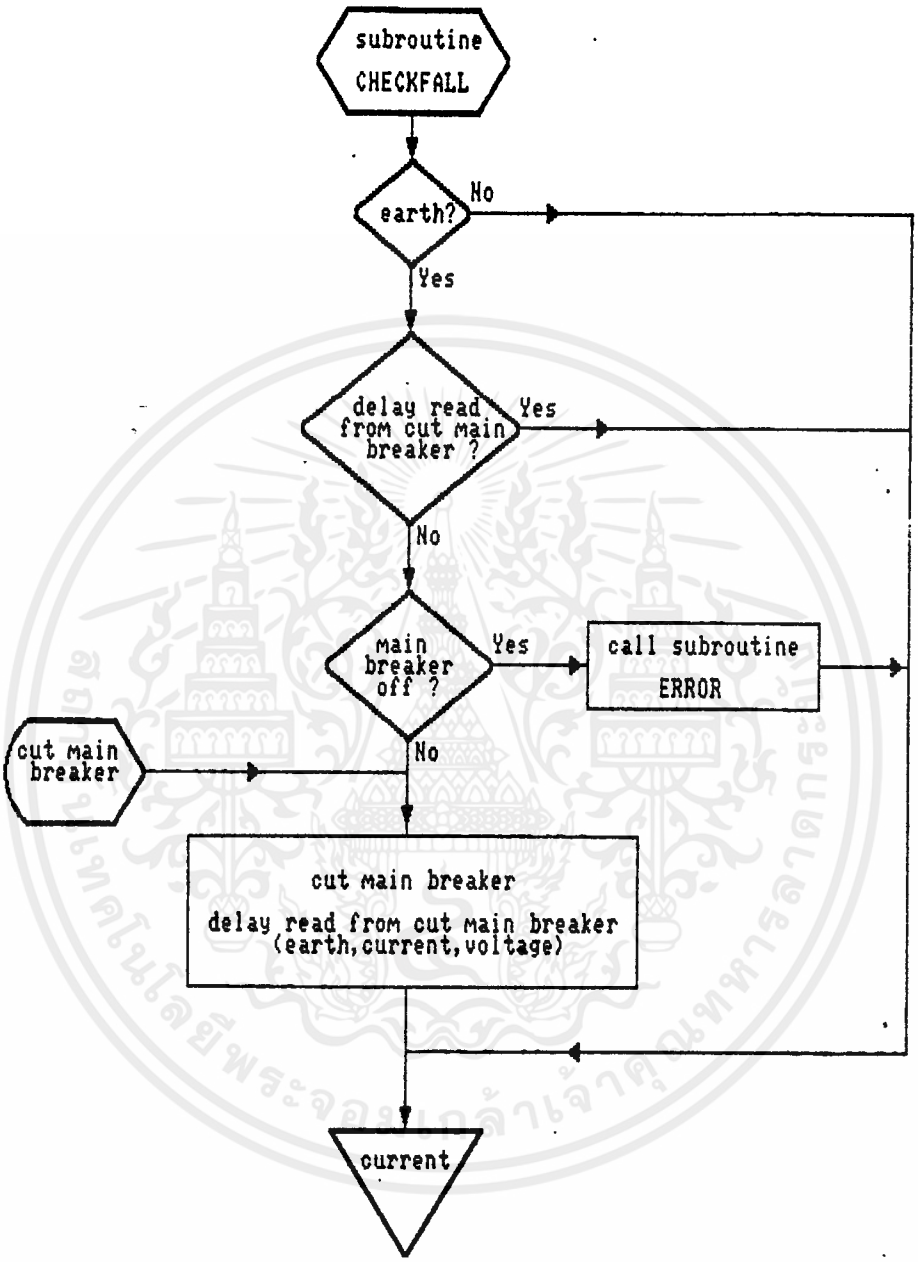
		TL0 VALUE
I/O 5	35H	01H
I/O 4	34H	01H
I/O 3	33H	01H
I/O 2	32H	01H
I/O 1	31H	01H
I,V	30H	F6H
E	2FH	F1H

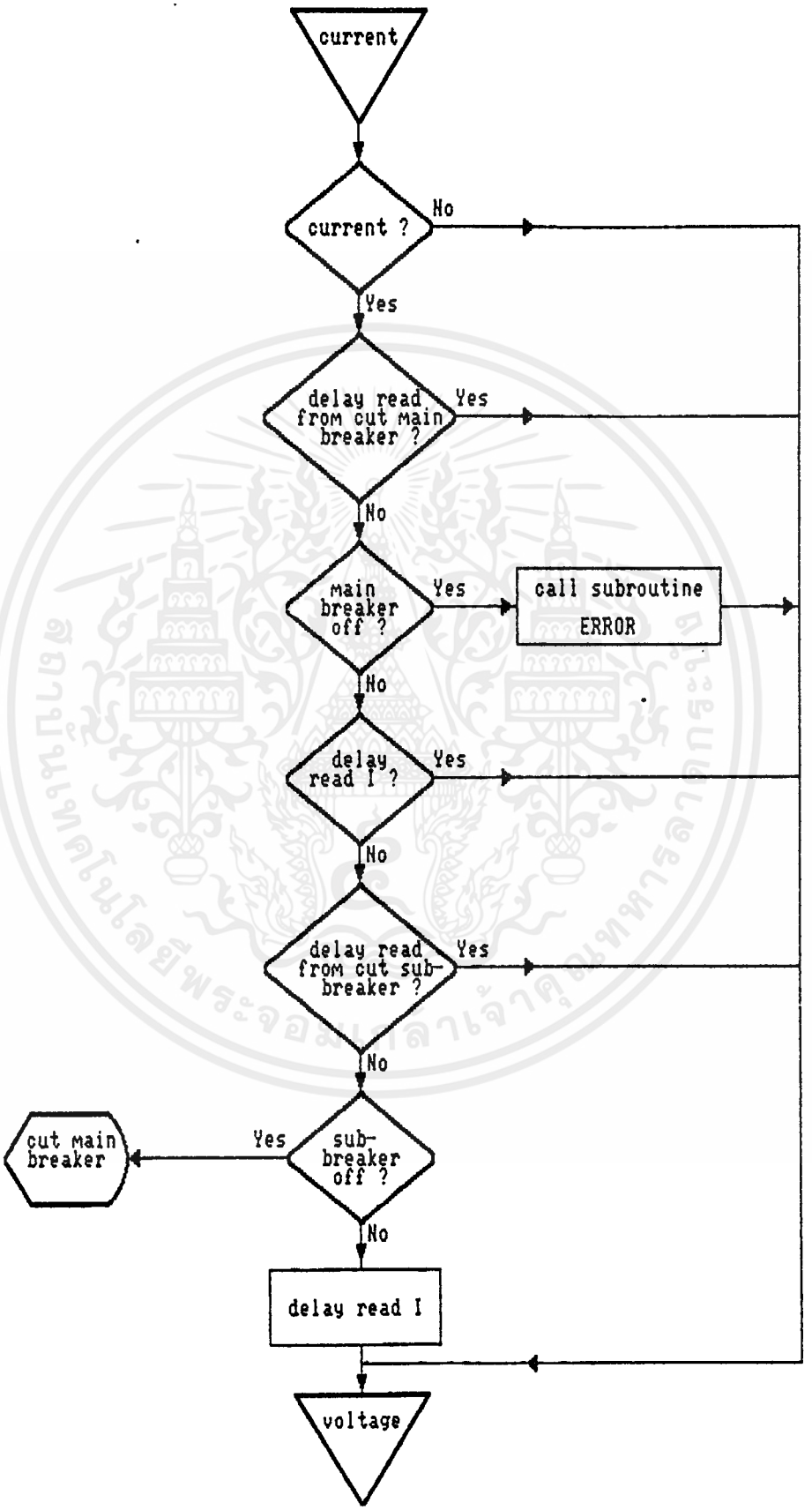
value of delay open

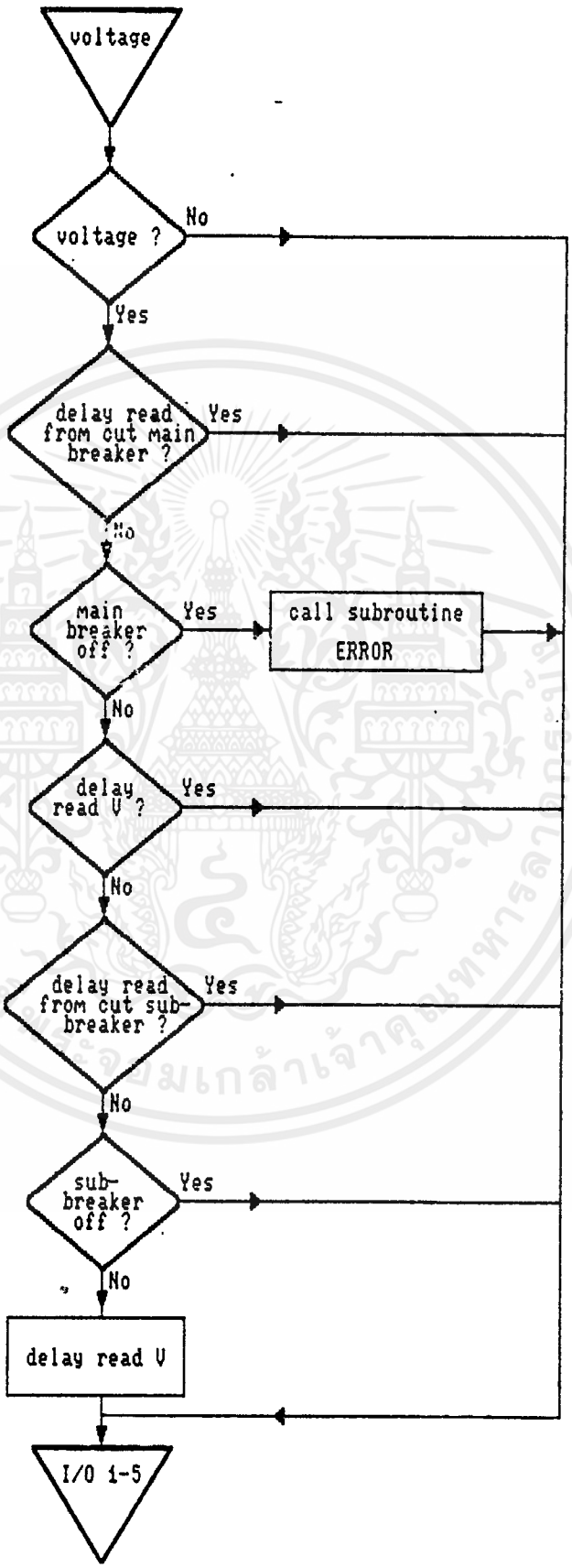
	TH0 VALUE		TL0 VALUE
3CH	FFH	43H	01H
3BH	FFH	42H	01H
3AH	FFH	41H	01H
39H	FFH	40H	01H
38H	FFH	3FH	01H
37H	B9H	3EH	B0H
36H	FEH	3DH	0CH

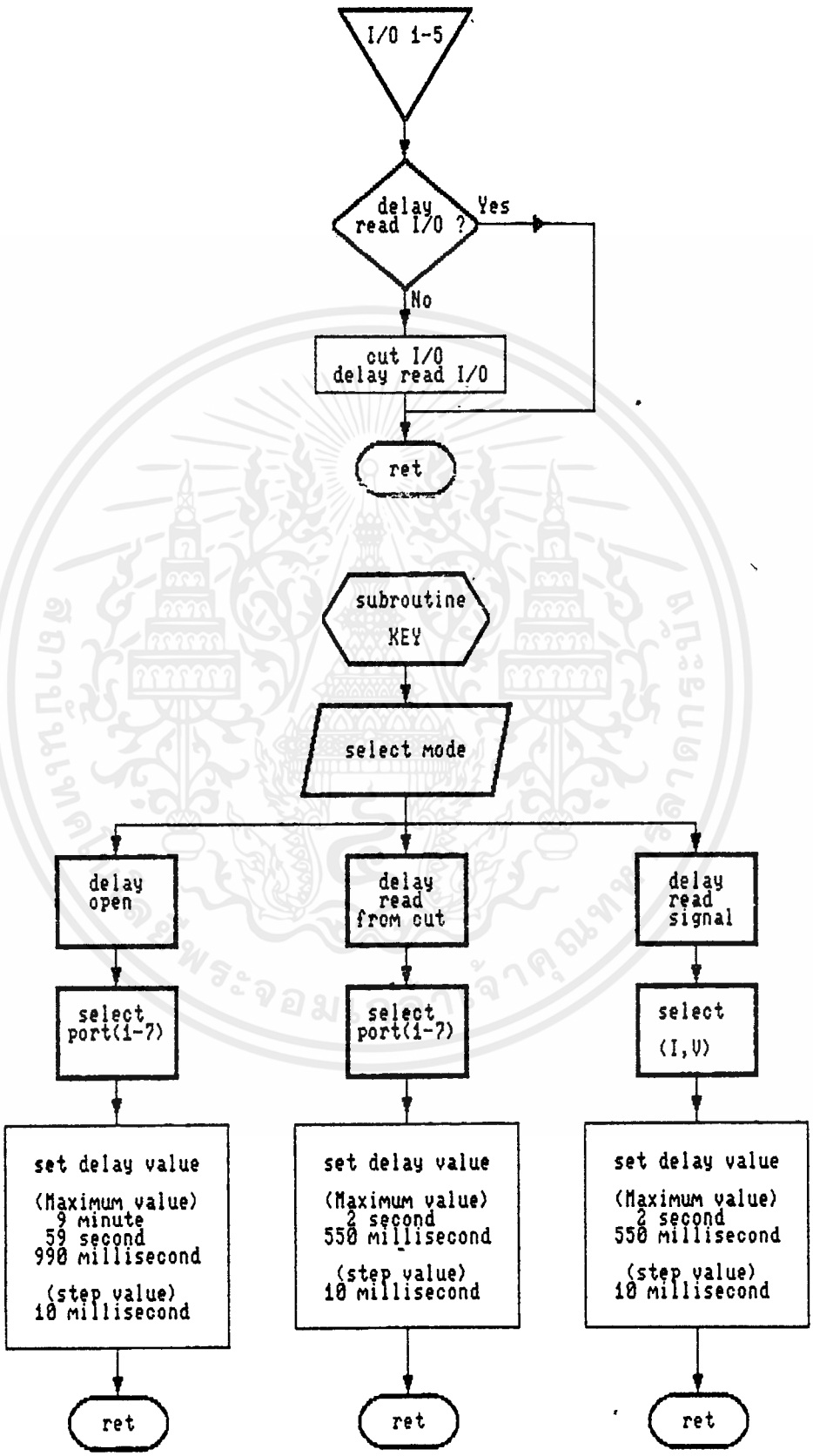
44H	F6H	value of delay read I
45H	F6H	value of delay read V

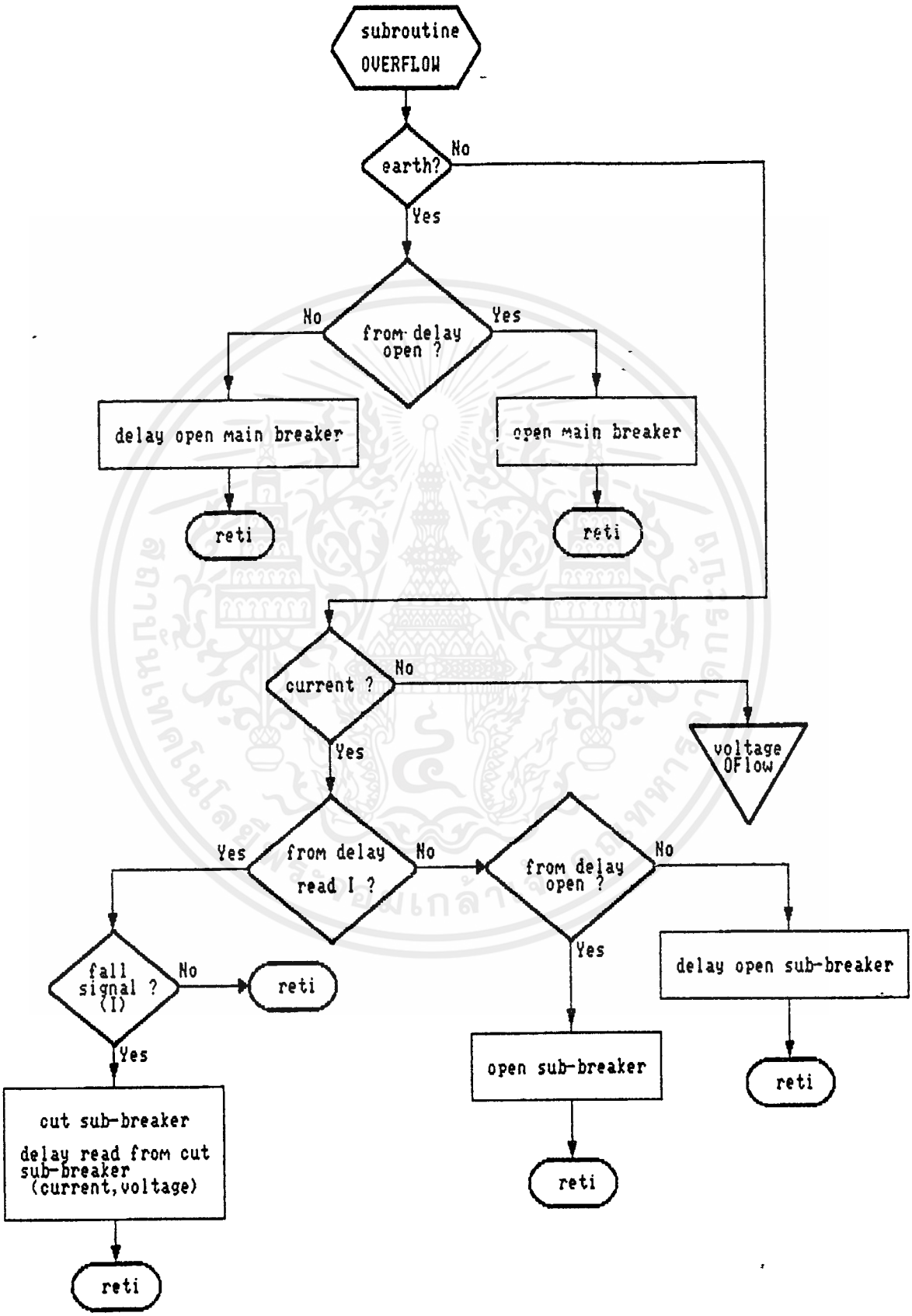


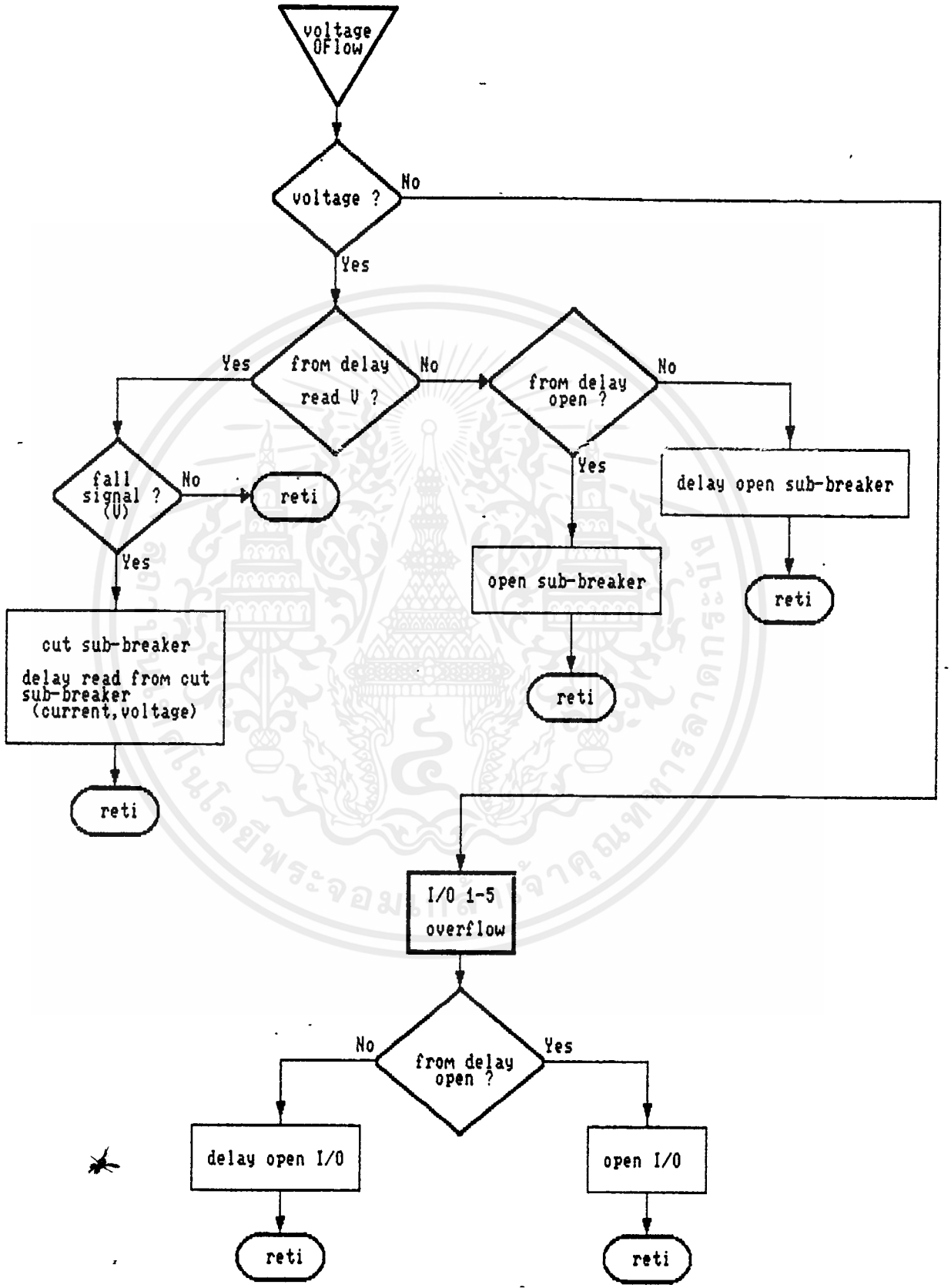


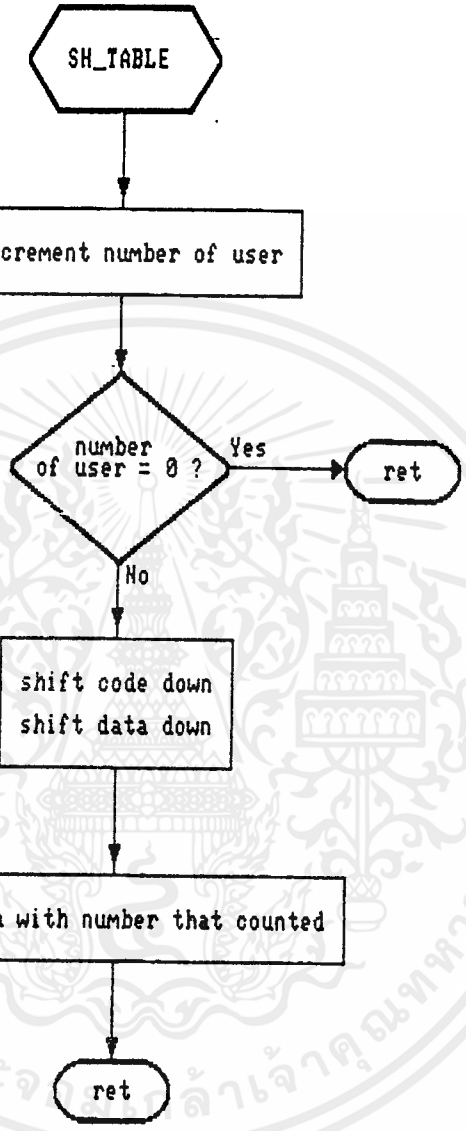


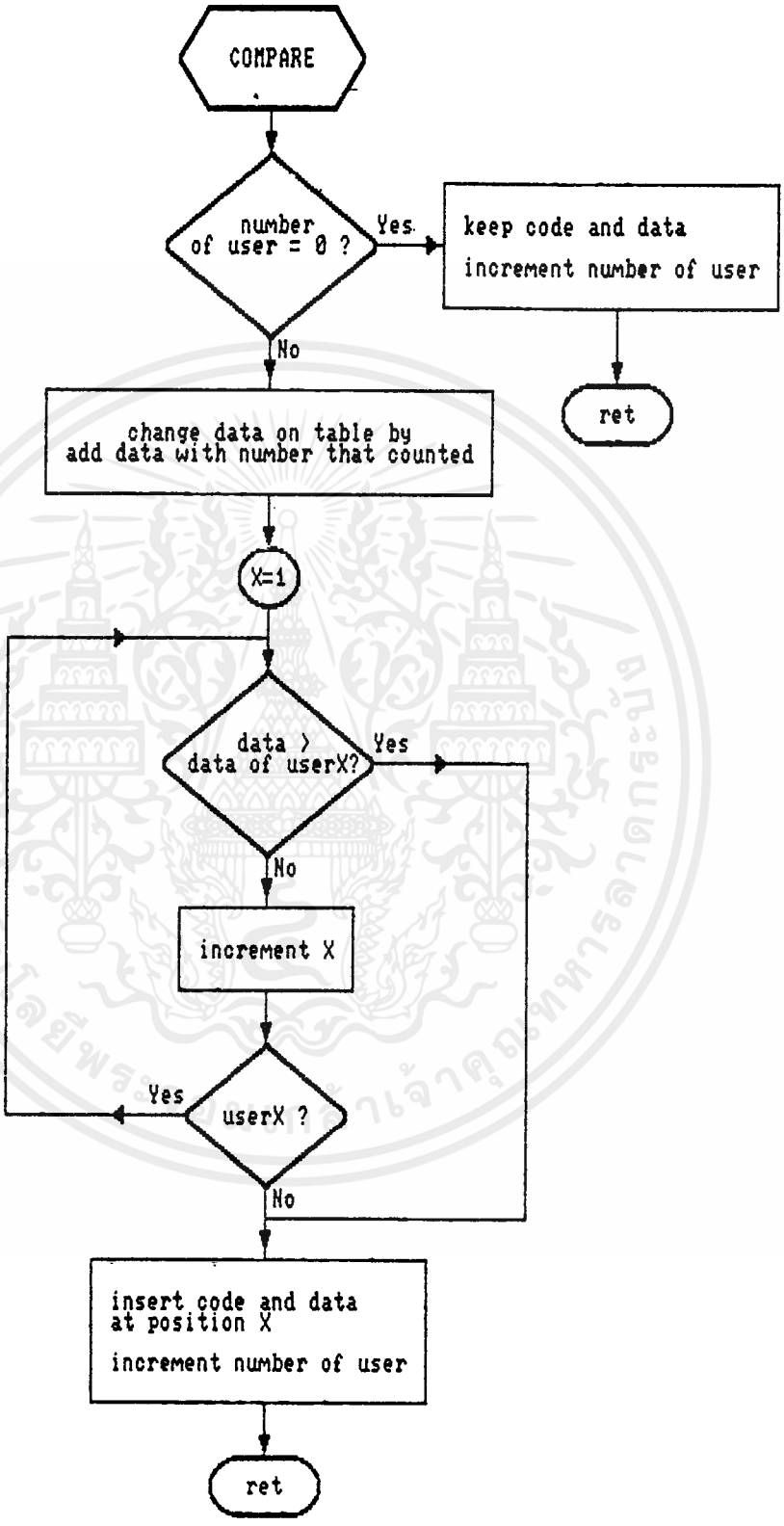












PROGRAM MEMORY

```
BASE        EQU     0E890H
            ORG     0000H
            AJMP    BEGIN
            ORG     000BH
            AJMP    OVERFLOW
            ORG     002FH
            DB      0F1H,0F6H,01H,01H,01H,01H,01H
            DB      0FEH,0B9H,0FFH,0FFH,0FFH,0FFH,0FFH
            DB      0CH,0B0H,01H,01H,01H,01H,01H
            DB      0F6H,0F6H,00H,00H,01H,01H
            DB      00H,00H,00H,00H,01H
            ORG     004FH
            DB      3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
            DB      77H,7CH,39H,5EH,79H,71H,00H
            DB      0BFH,86H,0DBH,0CFH,0E6H,0EDH
            DB      0FDH,87H,0FFH,0EFH,6DH,79H,0B8H
            DB      40H,73H,38H,0B9H,54H,44H,0D4H,6DH
            DB      00H,54H,44H,0EDH,0EDH,54H,44H,6DH
            DB      5EH,79H,0E6H,30H,3EH
            DB      5EH,71H,0B9H
            DB      37H,07H,04H,58H,50H,5CH,39H,5CH,54H,46H
            DB      50H,5CH,54H,06H,06H,18H,50H
            DB      00H,00H,00H,00H,31H
            ORG     009CH
BEGIN:      MOV     SP,#51H
            CLR     A
            MOV     DPH,#80H
            MOVX    @DPTR,A
            MOV     DPH,A
            MOVX    @DPTR,A
            MOV     R7,A
```

```
                MOV     TCON,#05H  ;TRANSMITTION EXTERNAL INTERRUPT
                MOV     TMOD,#25H  ;COUNTER 0 MODE 1 & TIMER 1 MODE 2
                MOV     IE,#97H    ;DISABLE TIMER 1 OVERFLOW
                MOV     IP,#0FEH   ;EXTERNAL INTERRUPT 0 LOW PRIORITY
                MOV     SCON,#0E0H ;SERIE MODE 3 & SM2=1
                MOV     PCON,#80H  ;SET SMOD
                MOV     R0,#29H
                MOV     @R0,#08H
CLEAR:          INC     R0                      ;
                MOV     @R0,A                   ; clear ค่าใน RAM
                CJNE    R0,#2EH,CLEAR           ; address 08H-2EH
SETVALUE:       INC     R0                      ;
                MOV     A,R0                    ;
                MOVC    A,@A+DPTR               ; load ค่า delay
                MOV     @R0,A                   ; จาก ROM
                CJNE    R0,#4EH,SETVALUE        ;
                MOV     DPL,#4FH
                MOV     A,#10H
                MOV     R1,#4AH
BLANK:          MOV     @R1,A                   ;
                INC     R1                      ; clear ค่าใน buffer
                CJNE    R1,#4FH,BLANK           ; ของ seven segment
                MOV     R6,#3FH
DELAY_SHOW:     ACALL   SCAN                    ; scan seven segment
                DJNZ    R6,DELAY_SHOW
```

```
SHOW1:      MOV     R5,#36H          ;
SHOW2:      MOV     A,R5             ;
            ACALL   CHANGE           ;
            MOV     R6,#20H          ;
            MOV     4EH,2DH          ;
SHOW3:      ACALL   SCAN             ;  แสดงการเริ่มการทางาน
            CJNE    R2,#00H,OUT_SHOW ;
            DJNZ    R6,SHOW3         ;
            INC     R5               ;
            CJNE    R5,#4EH,SHOW2    ;
            SJMP    SHOW1            ;
OUT_SHOW:   SJMP    WORK             ;

CHANGE:     MOV     R1,#4DH          ;
CHANGE1:    XCH     A,@R1            ;  subroutine ทาหน้าที่
            DEC     R1               ;  shift ค่าใน buffer
            CJNE    R1,#49H,CHANGE1  ;  ของ seven segment
            RET                      ;

K_DELAY:    MOV     R4,#02H          ;
D_R4:       MOV     R0,#0FFH         ;
D_FIRST:    MOV     R1,#0FFH         ;  subroutine ทาหน้าที่
D_SECOND:   DJNZ    R1,D_SECOND      ;  delay สาหรับ key
            DJNZ    R0,D_FIRST       ;
            DJNZ    R4,D_R4          ;
            RET                      ;
```

```
WORK:       MOV     DPH,#80H        ; เริ่มเข้าสู่ main program
            ACALL   K_DELAY
            MOV     R3,#01H
SCANFALL:   MOVX    A,@R0           ; read fall
            CPL     A
            MOV     R2,A
            MOV     A,2EH
            MOV     @DPTR,A
            MOV     R0,#10H
DELAYREAD:  DJNZ    R0,DELAYREAD
            MOVX    A,@R0           ; read fall again
            CPL     A
            ANL     A,R2            ; check real fall
            MOV     2DH,A           ; เก็บ fall ครั้งสาสุด
            MOV     4EH,A
            JNZ     FALL            ; jump ถ้ามี fall
            MOV     A,P1
            ANL     A,#0F0H
            MOV     P1,A
            CLR     A
            MOV     DPH,A
            CJNE    R3,#10H,NO_FLAG
            MOV     A,4EH
NO_FLAG:    MOVX    @DPTR,A
            MOV     DPH,#80H
            MOV     A,P1
            ORL     A,R3
            ORL     A,#20H
            MOV     P1,A
            MOV     A,R3
            RL      A
```

```
                CJNE    A,#10H,NO_CH
                MOV     R3,#01H
                SJMP    CON_KEY
NO_CH:          MOV     R3,A
CON_KEY:        MOV     A,P1
                JB      P1.5,NO_KEY          ; P1.5 = 0 มีการกด key
                MOV     A,29H
                CJNE    A,#08H,NO_KEY
                MOV     R7,#00H
                MOV     DPH,#00H
                ACALL   K_DELAY
                ACALL   KEY                  ; call subroutine key
                ACALL   K_DELAY
                MOV     DPH,#80H
                MOV     R3,#01H
NO_KEY:         SJMP    SCANFALL
FALL:           ACALL   CHECKFALL            ; call checkfall
                MOV     R3,#01H
                SJMP    SCANFALL
```

```
KEY:            MOV     A,#1BH
                ACALL   SET
                MOV     R0,A
SELECT:         ACALL   SCAN
                CJNE    R2,#00H,DELAY_OPEN    ; select mode
                SJMP    SELECT
DELAY_OPEN:     CJNE    R2,#01H,DELAY_WORK
                SJMP    D_OPEN                ; set delay open
DELAY_WORK:     CJNE    R2,#02H,DELAY_READ
                AJMP    D_WORK                ; set delay read หลังจาก
                                                ส่ง output signal
DELAY_READ:     CJNE    R2,#04H,SELECT
                AJMP    D_READ                ; set delay read I,V
```

```
D_OPEN:     MOV     R1,#35H
            MOV     A,#1FH
            ACALL   PLC                 ; select port
            CJNE    R1,#35H,CON_OPEN
            RET
CON_OPEN:   MOV     A,#22H
            ACALL   SET
            MOV     R0,A
MIN_SEC:    ACALL   SCAN
            CJNE    R2,#01H,MIN_SEC     ; แสดงให้รู้ว่าตั้งค่าเวลาน
                                          หน่วย นาที กับ วินาที
            ACALL   SETZERO
SCAN2:      ACALL   SCAN
            CJNE    R2,#00H,OK2
            SJMP    SCAN2
OK2:        CJNE    R2,#01H,INC2
            SJMP    PAST2               ; jump เมื่อกด key 1 (OK)
INC2:       CJNE    R2,#02H,DEC2        ;
            INC     @R0                 ;
            CJNE    R0,#4BH,INC21       ; การเพิ่มค่าเวลา
            CJNE    @R0,#1BH,SCAN2      ;
            MOV     @R0,#11H            ;
            SJMP    SCAN2               ;
INC21:      CJNE    R0,#4CH,INC22       ;
            CJNE    @R0,#06H,SCAN2      ;
            MOV     @R0,#00H            ;
INC22:      CJNE    @R0,#0AH,SCAN2      ;
            MOV     @R0,#00H            ;
            SJMP    SCAN2               ;
DEC2:       CJNE    R2,#04H,SHIFT2      ;
            DEC     @R0                 ;
```

```
                CJNE    R0,#4BH,DEC21       ;
                CJNE    @R0,#10H,SCAN2      ;
                MOV     @R0,#1AH            ;
                SJMP    SCAN2               ;
DEC21:          CJNE    @R0,#0FFH,SCAN2     ; - การลดค่าเวลา
                CJNE    R0,#4CH,DEC22       ;
                MOV     @R0,#05H            ;
                SJMP    SCAN2               ;
DEC22:          MOV     @R0,#09H            ;
                SJMP    SCAN2               ;
SHIFT2:         DEC     R0                  ;
                CJNE    R0,#4AH,SCAN2       ; - shift digit
                MOV     R0,#4DH             ;
                SJMP    SCAN2               ;
PAST2:          MOV     A,4BH               ; เริ่มแปลงค่าเวลาเป็นค่า HEX
                CLR     C
                SUBB    A,#11H
                MOV     B,#3CH
                MUL     AB
                MOV     R6,A
                MOV     R5,B
                MOV     A,4CH
                MOV     B,#0AH
                MUL     AB
                ADD     A,4DH
                ADD     A,R6
                MOV     R6,A
                JNC     NOC1
                INC     R5
NOC1:           MOV     A,R5
                MOV     B,#64H
```

```
        MUL     AB
        MOV     B,#0FFH
        MOV     R3,A
        MUL     AB
        ADD     A,R3
        JNC     NOC2
        INC     B
NOC2:   XCH     A,R6
        MOV     R5,B
        MOV     B,#64H
        MUL     AB
        JNC     NOC3
        INC     B
NOC3:   ADD     A,R6
        MOV     R6,A
        MOV     A,B
        ADDC    A,R5
        MOV     R5,A            ; ได้ค่าเวลาเป็นค่า HEX
        MOV     A,#26H
        ACALL   SET
        MOV     R0,A
MS:     ACALL   SCAN            ; แสดงเพื่อรู้ว่าตั้งค่าเวลาเป็น
                                หน่วย 10 มิลลิวินาที
        CJNE    R2,#01H,MS
        MOV     R0,#4AH
        MOV     @R0,#10H
        CLR     A
ZERO:   INC     R0
        MOV     @R0,A
        CJNE    R0,#4DH,ZERO
        DEC     R0
```

```
SCAN3:      ACALL   SCAN
            CJNE    R2,#00H,OK3
            SJMP    SCAN3
OK3:        CJNE    R2,#01H,INC3
            SJMP    PAST3           ; jump เมื่อกด key 1 (OK)
INC3:       CJNE    R2,#02H,DEC3
            INC     @R0             ; การเพิ่มค่าเวลา
            CJNE    @R0,#0AH,SCAN3
            MOV     @R0,#00H
            SJMP    SCAN3
DEC3:       CJNE    R2,#04H.SHIFT3
            DEC     @R0             ; การลดค่าเวลา
            CJNE    @R0,#0FFH,SCAN3
            MOV     @R0,#09H
            SJMP    SCAN3
SHIFT3:     DEC     R0              ; shift digit
            CJNE    R0,#4AH,SCAN3
            MOV     R0,#4CH
            SJMP    SCAN3
PAST3:      MOV     A,4BH           ; เริ่มแปลงค่าเวลาเป็นค่า HEX
            MOV     B,#0AH
            MUL     AB
            ADD     A,4CH
            ADD     A,R6
            CPL     A
            MOV     R6,A
            CLR     A
            ADDC    A,R5
            CPL     A
            MOV     R5,A
            MOV     A,#01H
```

```
                ADD     A,R6
                MOV     R6,A
                CLR     A
                ADDC    A,R5
                MOV     R5,A            ; ได้ค่าเวลาเป็นค่า HEX
                ACALL   CH_R5
                ACALL   SETACALL        ; show value
                CJNE    R2,#01H,FAIL3   ; check ความพอใจ
                MOV     A,#07H
                ADD     A,R1
                MOV     R0,A
                CLR     IE.7
                MOV     @R1,05H
                MOV     @R0,06H
                SETB    IE.7
FAIL3:          RET


D_WORK:         MOV     R1,#2EH
                MOV     A,#33H
                ACALL   PLC             ; select port
                CJNE    R1,#2EH,CON_WORK
                RET
CON_WORK:       ACALL   H255            ; call subroutine for
                                          set value
                RET
```

```
D_READ:     MOV     A,#2EH
            ACALL   SET
SCAN5:      ACALL   SCAN
            CJNE    R2,#00H,OK5
            SJMP    SCAN5               ; jump เมื่อได้แล้ว
OK5:        CJNE    R2,#01H,INC5        ;
            MOV     A,#13H              ;
            ADD     A,@R0               ;
            MOV     R1,A                ;
            SJMP    PAST5               ;
INC5:       CJNE    R2,#02H,DEC5        ;
            INC     @R0                 ;
            CJNE    @R0,#33H,SCAN5      ;
            MOV     @R0,#31H            ;
            SJMP    SCAN5               ; select I or V
DEC5:       CJNE    R2,#04H,READ5       ;
            DEC     @R0                 ;
            CJNE    @R0,#30H,SCAN5      ;
            MOV     @R0,#32H            ;
            SJMP    SCAN5               ;
READ5:      PUSH    01H                 ; ดูค่าเดิม
            MOV     A,@R0
            MOV     R5,A
            ADD     A,#13H
            MOV     R1,A
            MOV     A,@R1
            MOV     R6,A
            POP     01H
            ACALL   CH_B
            ACALL   CH_R6
            MOV     R0,#00H
```

```
READ51:     ACALL   SCAN
            CJNE    R2,#00H,READ52
            SJMP    READ51
READ52:     CJNE    R2,#01H,READ53
            RET                         ; return เมื่อพอใจค่าเดิม
READ53:     MOV     A,#2EH
            ACALL   R_L
            MOV     A,R5
            MOV     @R0,A
            SJMP    SCAN5
PAST5:      ACALL   H255                ; call subroutine for
                                          set value
            RET
```

```
H255:       MOV     A,#2AH          ; subroutine set value
            ACALL   SET
            MOV     R0,A
SEC_MS:     ACALL   SCAN            ; แสดงเพื่อรู้ว่าตั้งค่าเวลาน
                                      หน่วย วินาที กับ เดซิวินาที
            CJNE    R2,#01H,SEC_MS
REFAIL4:    ACALL   SETZERO
            MOV     A,#05H
SCAN4:      ACALL   SCAN
            CJNE    R2,#00H,OK4
            SJMP    SCAN4
OK4:        CJNE    R2,#01H,INC4
            MOV     A,4BH           ;
            CLR     C               ;
            SUBB    A,#11H          ;
            MOV     B,#64H          ;
            MUL     AB              ;
            MOV     R6,A            ;
            MOV     A,4CH           ; - แปลงค่าเวลาเป็นค่า HEX
            MOV     B,#0AH          ;
            MUL     AB              ;
            ADD     A,4DH           ;
            ADD     A,R6            ;
            CPL     A               ;
            INC     A               ;
            MOV     R6,A            ;
            ACALL   CH_B
            ACALL   SETACALL
            CJNE    R2,#01H,FAIL4   ; check ความพอใจ
            CLR     IE.7
            MOV     @R1,06H
```

```
                SETB    IE.7
FAIL4:          RET
INC4:           CJNE    R2,#02H,DEC4        ; increment value
                ACALL   O4INC
                AJMP    SCAN4
DEC4:           CJNE    R2,#04H,SHIFT4      ; decrement value
                ACALL   O4DEC
                AJMP    SCAN4
SHIFT4:         DEC     R0                  ; shift digit
                CJNE    R0,#4AH,OUT_S
                MOV     R0,#4DH
OUT_S:          AJMP    SCAN4
O4INC:          INC     @R0
                CJNE    R0,#4BH,INC41
                CJNE    @R0,#14H,INC42
                MOV     @R0,#11H
                RET
INC42:          CJNE    @R0,#13H,INC4_1
                CJNE    A,4CH,INC43
                CJNE    A,4DH,INC44
INC44:          JNC     INC4_1
                MOV     4DH,A
INC4_1:         RET
INC43:          JNC     INC4_1
                MOV     4CH,A
                MOV     4DH,A
                RET
INC41:          CJNE    R0,#4CH,INC45
                MOV     R5,4BH
                CJNE    R5,#13H,INC46
                CJNE    @R0,#06H,INC47
```

```
                MOV     @R0,#00H
                RET
INC47:          CJNE    @R0,#05H,INC4_2
                CJNE    A,4DH,INC49
INC49:          JNC     INC4_2
                MOV     4DH,A
INC4_2:         RET
INC46:          CJNE    @R0,#0AH,INC4_2
                MOV     @R0,#00H
                RET
INC45:          MOV     R5,4BH
                CJNE    R5,#13H,INC48
                CJNE    A,4CH,INC48
                CJNE    @R0,#06H,INC4_3
                MOV     @R0,#00H
INC4_3:         RET
INC48:          CJNE    @R0,#0AH,INC4_3
                MOV     @R0,#00H
                RET
04DEC:          DEC     @R0
                CJNE    R0,#4BH,DEC41
                CJNE    @R0,#10H,DEC4_1
                MOV     @R0,#13H
                CJNE    A,4CH,DEC42
                CJNE    A,4DH,DEC43
DEC43:          JNC     DEC4_1
                MOV     4DH,A
DEC4_1:         RET
DEC42:          JNC     DEC4_1
                MOV     4CH,A
                MOV     4DH,A
```

```
                RET
DEC41:          CJNE    R0,#4CH,DEC44
                MOV     R5,4BH
                CJNE    R5,#13H,DEC45
                CJNE    @R0,#0FFH,DEC4_2
                MOV     @R0,A
                CJNE    A,4DH,DEC46
DEC46:          JNC     DEC4_2
                MOV     4DH,A
DEC4_2:         RET
DEC45:          CJNE    @R0,#0FFH,DEC4_2
                MOV     @R0,#09H
                RET
DEC44:          MOV     R5,4BH
                CJNE    R5,#13H,DEC47
                CJNE    A,4CH,DEC47
                CJNE    @R0,#0FFH,DEC4_3
                MOV     @R0,A
DEC4_3:         RET
DEC47:          CJNE    @R0,#0FFH,DEC4_3
                MOV     @R0,#09H
                RET
```

```
PLC:        ACALL   SET             ; subroutine สำหรับ
                                      select port
            MOV     @R0,#01H
SCAN1:      ACALL   SCAN
            CJNE    R2,#00H,OK1
            SJMP    SCAN1
OK1:        CJNE    R2,#01H,INC1
            MOV     A,R1
            ADD     A,@R0
            MOV     R1,A
            RET                     ; return แบบไม่พอใจค่าเดิม
INC1:       CJNE    R2,#02H,DEC1
            INC     @R0
            CJNE    @R0,#08H,SCAN1
            MOV     @R0,#01H
            SJMP    SCAN1
DEC1:       CJNE    R2,#04H,READ1
            DEC     @R0
            CJNE    @R0,#00H,SCAN1
            MOV     @R0,#07H
            SJMP    SCAN1
READ1:      PUSH    01H             ; ดูค่าเดิม
            CJNE    R1,#35H,READ11
            MOV     B,#1FH
            MOV     A,@R0
            PUSH    0E0H
            ADD     A,R1
            MOV     R1,A
            MOV     A,@R1
            MOV     R5,A
            MOV     A,R1
```

```
                ADD     A,#07H
                MOV     R1,A
                MOV     A,@R1
                MOV     R6,A
                ACALL   CH_R5
                SJMP    READ12
READ11:         MOV     A,@R0
                MOV     B,#33H
                PUSH    0E0H
                ADD     A,R1
                MOV     R1,A
                MOV     A,@R1
                MOV     R6,A
                ACALL   CH_B
READ12:         ACALL   CH_R6
                MOV     R0,#00H
READ13:         ACALL   SCAN
                CJNE    R2,#00H,READ14
                SJMP    READ13
READ14:         CJNE    R2,#01H,READ15
                POP     0E0H
                POP     01H
                RET                             ; return แบบพอใจค่าเดิม
READ15:         MOV     A,B
                ACALL   R_L
                POP     0E0H
                POP     01H
                MOV     @R0,A
                AJMP    SCAN1
```

```
CH_B:       MOV     A,#10H          ;┐
            MOV     4AH,A           ;├ reset ค่า digit ที่
            MOV     4BH,A           ;│ 1,2
            RET                     ;┘

CH_R6:      MOV     A,R6            ;┐
            SWAP    A               ;│
            ANL     A,#0FH          ;│
            MOV     4CH,A           ;├ set ค่า digit ที่
            MOV     A,R6            ;│ 3,4 ตามค่า R6
            ANL     A,#0FH          ;│
            MOV     4DH,A           ;│
            RET                     ;┘

CH_R5:      MOV     A,R5            ;┐
            SWAP    A               ;│
            ANL     A,#0FH          ;│
            MOV     4AH,A           ;├ set ค่า digit ที่
            MOV     A,R5            ;│ 1,2 ตามค่า R5
            ANL     A,#0FH          ;│
            MOV     4BH,A           ;│
            RET                     ;┘

SET:        MOV     R0,#4AH
LOOPSET:    MOV     @R0,A
            CJNE    R0,#4DH,CONSET
            RET
CONSET:     INC     R0
            INC     A
            SJMP    LOOPSET
```

```
SETZERO:        MOV     4AH,#10H
                MOV     4BH,#11H
                MOV     4CH,#00H
                MOV     R0,#4DH
                MOV     @R0,#00H
                RET

ERROR_FAIL:     MOV     4AH,#10H
                MOV     4BH,#39H
                MOV     4CH,#33H
                MOV     4DH,#33H
                MOV     R0,#00H
                ACALL   EMIT
                RET

R_L:            MOV     R0,#4AH
LOOPR_L:        MOV     @R0,A
                INC     R0
                INC     A
                CJNE    R0,#4DH,LOOPR_L
                RET

EMIT:           ACALL   SCAN
                CJNE    R2,#00H,EMIT1
                SJMP    EMIT
EMIT1:          RET
```

```
SURE:       MOV     4AH,#05H
            MOV     4BH,#32H
            MOV     4CH,#4BH
            MOV     4DH,#0EH
QUES3:      ACALL   SCAN
            CJNE    R2,#00H,QUES31
            SJMP    QUES3
QUES31:     RET

SETACALL:   ACALL   CH_R6
            MOV     R0,#00H
            ACALL   EMIT
            ACALL   SURE
            RET

SCAN:       CJNE    R2,#01H,NOOK        ; subroutine scan seven
                                          segment
OK:         ACALL   SUBSCAN
            CJNE    R2,#01H,CONK
            SJMP    OK
CONK:       RET
NOOK:       CJNE    R2,#00H,ELSE
            ACALL   SUBSCAN
            RET
ELSE:       PUSH    05H
            MOV     R5,#20H
DELAY:      ACALL   SUBSCAN
            DJNZ    R5,DELAY
            POP     05H
            RET
```

```
                MOV     R2,A
DD:             XCH     A,@R1
                ADD     A,R4
                XCH     A,R6
                XCH     A,@R0
                ADDC    A,R2
                XCH     A,R6
                DEC     R0
                DEC     R1
                CJNE    R0,#12H,DD
                MOV     TH0,13H
                MOV     TL0,1EH
                RET

FUNCTION:       CJNE    R3,#01H,OCURRENT
                JBC     60H,END_DELAY_E
                CLR     70H
                SJMP    OUT_SIGNAL
END_DELAY_E:    JB      58H,LONG_E
                MOV     R6,36H
                MOV     R7,3DH
                ACALL   COMPARE
LONG_E:         RET
OCURRENT:       CJNE    R3,#02H,OVOLTAGE
                JBC     59H,END_DELAY_RI
                JBC     61H,END_DELAY_M
                JB      62H,OUTM
OPEN_M:         JB      6AH,CON_CUTM
                JB      69H,CON_CUTM
                CLR     71H
OUT_SIGNAL:     MOV     A,2EH
```

```
        JB      P1.5,NOKEY
        MOV     R4,#10H
CHECK:  DJNZ    R4,CHECK
        MOV     A,P1
        JB      P1.5,NOKEY
        PUSH    03H
        POP     02H
NOKEY:  MOV     R4,#0C0H
LFALL:  PUSH    0F0H
        MUL     AB
        POP     0F0H
        DJNZ    R4,LFALL
        MOV     A,R3
        RL      A
        INC     R1
        CJNE    A,#20H,SHIFT
        POP     0E0H
        POP     01H
        RET
```

```
CHECKFALL:      MOV     P1,#0F0H
                MOV     R2,A            ;KEEP ERROR IN R2
                MOV     A,#01H
NEXT:           MOV     R3,A            ;KEEP CONDITION CHECK
                ANL     A,R2
                JNZ     EARTH
                SJMP    CHECKNEXT
CONCHECK:       POP     03H
                POP     02H
CHECKNEXT:      MOV     A,R3
                RL      A
                CJNE    A,#01H,NEXT
                RET
EARTH:          PUSH    02H
                PUSH    03H
                CJNE    R3,#01H,CURRENT
                JB      60H,CONCHECK
                JNB     70H,NO_ERROR_E
                ACALL   ERROR
                AJMP    CONCHECK
NO_ERROR_E:     SETB    70H
                SETB    60H
                MOV     A,2EH
                MOVX    @DPTR,A
                MOV     R6,#0FFH
                MOV     R7,2FH
                MOV     R3,#01H
                ACALL   COMPARE0
                AJMP    CONCHECK
CURRENT:        CJNE    R3,#02H,VOLTAGE
                JB      60H,OUTI
```

```
                JNB     70H,NO_ERROR_I
                ACALL   ERROR
OUTI:           AJMP    CONCHECK
NO_ERROR_I:     JB      59H,OUTI
                JB      61H,OUTI
                JB      71H,NO_ERROR_E
                SETB    59H
                MOV     R6,#0FFH
                MOV     R7,44H
                MOV     R3,#02H
                ACALL   COMPARE0
                AJMP    CONCHECK
VOLTAGE:        CJNE    R3,#04H,OTHER
                JB      60H,OUTV
                JNB     70H,NO_ERROR_V
                ACALL   ERROR
OUTV:           AJMP    CONCHECK
NO_ERROR_V:     JB      5AH,OUTV
                JB      61H,OUTV
                JB      71H,OUTV
                SETB    5AH
                MOV     R6,#0FFH
                MOV     R7,45H
                ACALL   COMPARE0
                AJMP    CONCHECK
OTHER:          MOV     A,R3
                ANL     A,2CH
                JZ      OTHER1
                AJMP    CONCHECK
OTHER1:         MOV     A,R3
                ANL     A,2EH
```

```
                JZ      NORMAL_O
                ACALL   ERROR
                AJMP    CONCHECK
NORMAL_O:       MOV     A,R3
                ORL     A,2CH
                MOV     2CH,A
                MOV     A,R3
                ORL     A,2EH
                MOV     2EH,A
                MOVX    @DPTR,A
                MOV     R6,#0FFH
                CJNE    R3,#20H,NO_IN3
                MOV     R7,33H
                SJMP    END_SET_R3
NO_IN3:         JC      IN1_IN2
                CJNE    R3,#40H,NO_IN4
                MOV     R7,34H
                SJMP    END_SET_R3
NO_IN4:         MOV     R7,35H
                SJMP    END_SET_R3
IN1_IN2:        CJNE    R3,#10H,NO_IN2
                MOV     R7,32H
                SJMP    END_SET_R3
NO_IN2:         MOV     R7,31H
END_SET_R3:     ACALL   COMPARE0
                AJMP    CONCHECK
```

```
OVERFLOW:   CLR     TR0
            PUSH    0E0H
            PUSH    00H
            PUSH    01H
            PUSH    02H
            PUSH    03H
            PUSH    04H
            PUSH    06H
            PUSH    0D0H
            MOV     R3,#01H
KEEP:       PUSH    08H
            ACALL   SH_TABLE
            MOV     A,13H
            ORL     A,1EH
            JNZ     NO_ZERO
            INC     R3
            SJMP    KEEP
NO_ZERO:    DEC     R3
            CJNE    R3,#00H,CON_GET
            POP     03H
            ACALL   FUNCTION
            SJMP    COMPLETE_SH
CON_GET:    POP     0E0H
            PUSH    03H
            MOV     R3,A
            ACALL   FUNCTION
            POP     03H
            SJMP    NO_ZERO
COMPLETE_SH: POP    0D0H
            POP     06H
            POP     04H
```

```
                POP     03H
                POP     02H
                POP     01H
                POP     00H
                MOV     A,29H
                CJNE    A,#08H,COUNT
                POP     0E0H
                RETI
COUNT:          POP     0E0H
                SETB    TR0
                RETI

SH_TABLE:       DEC     29H             ;DEC EMPTY BYTE
                MOV     A,29H
                CJNE    A,#08H,USER
                RET
USER:           MOV     R1,A
                ADD     A,#0BH
                MOV     R0,A            ;POINTER TO HIGH BYTE
CD:             XCH     A,@R1
                DEC     R1
                CJNE    R1,#07H,CD
                MOV     A,#0BH
                ADD     A,R0
                MOV     R1,A            ;POINTER TO LOW BYTE
                CLR     A
                CLR     C
                SUBB    A,1EH
                MOV     R4,A
                CLR     A
                SUBB    A,13H
```

```
SUBSCAN:    PUSH    01H
            PUSH    0E0H
            MOV     R2,#00H
            MOV     R1,#4AH
            MOV     A,#01H
SHIFT:      MOV     R3,A
            MOV     A,R1
            XRL     A,R0
            JNZ     NOR0
            INC     R7
            MOV     R4,#00H
            CJNE    R7,#11H,CONR7
            MOV     R7,#00H
CONR7:      MOV     A,#08H
            SUBB    A,R7
            JNC     DARK
NOR0:       MOV     A,@R1
            CJNE    R1,#4EH,GET
            SJMP    LED
GET:        MOVC    A,@A+DPTR
LED:        MOV     R4,A
DARK:       MOV     A,P1
            ANL     A,#0E0H
            MOV     P1,A
            MOV     A,R4
            MOVX    @DPTR,A
            MOV     A,P1
            ORL     A,R3
            ORL     A,#20H
            MOV     P1,A
            MOV     A,P1
```

```
                MOVX    @DPTR,A
OUTM:           RET
END_DELAY_RI:   JB      69H,CUTM
                RET
CUTM:           SETB    71H
                SETB    61H
                MOV     A,2EH
                MOVX    @DPTR,A
CON_CUTM:       MOV     R6,#0FFH
                MOV     R7,30H
                ACALL   COMPARE
                RET
END_DELAY_M:    JB      62H,LONG_I
                MOV     R6,37H
                MOV     R7,3EH
                ACALL   COMPARE
LONG_I:         RET
OVOLTAGE:       CJNE    R3,#04H,OOTHER
                JBC     5AH,END_DELAY_RV
                JBC     61H,END_DELAY_M
                JB      62H,OUTOV
                SJMP    OPEN_M
END_DELAY_RV:   JB      62H,OUTOV
                JB      6AH,CUTM
OUTOV:          RET
OOTHER:         MOV     A,R3
                ANL     A,2CH
                JZ      OOTHER1
                CPL     A
                ANL     A,2CH
                MOV     2CH,A
```

```
                CJNE    R3,#20H,NO_OV3
                MOV     R6,3AH
                MOV     R7,41H
                SJMP    END_SET_R23
NO_OV3:         JC      OV1_OV2
                CJNE    R3,#40H,NO_OV4
                MOV     R6,3BH
                MOV     R7,42H
                SJMP    END_SET_R23
NO_OV4:         MOV     R6,3CH
                MOV     R7,43H
                SJMP    END_SET_R23
OV1_OV2:        CJNE    R3,#10H,NO_OV2
                MOV     R6,39H
                MOV     R7,40H
                SJMP    END_SET_R23
NO_OV2:         MOV     R6,38H
                MOV     R7,3FH
END_SET_R23:    ACALL   COMPARE
                RET
OOTHER1:        MOV     A,R3
                CPL     A
                ANL     A,2EH
                MOV     2EH,A
                MOVX    @DPTR,A
                RET
```

```
COMPARE0:   CLR     TR0
            ACALL   COMPARE
            SETB    TR0
            RET

COMPARE:    MOV     A,R6
            ORL     A,R7
            JNZ     CON_COM
            ACALL   FUNCTION
            RET
CON_COM:    MOV     R1,#08H          ;INSERT ADDRESS
            MOV     A,29H
            CJNE    A,#08H,NOFIRST
            MOV     TH0,R6
            MOV     TL0,R7
            MOV     08H,R3
            MOV     13H,R6
            MOV     1EH,R7
            INC     29H
            RET
NOFIRST:    MOV     A,TL0
            CLR     C
            SUBB    A,1EH
            MOV     R5,A
            MOV     A,TH0
            SUBB    A,13H
            MOV     R4,A
            MOV     A,29H
            DEC     A
            ADD     A,#0BH
            MOV     R0,A
```

```
                ADD     A,#0BH
                MOV     R1,A
CC:             MOV     A,@R1
                ADD     A,R5
                MOV     @R1,A
                MOV     A,@R0
                ADDC    A,R4
                MOV     @R0,A
                DEC     R0
                DEC     R1
                CJNE    R0,#12H,CC
                MOV     A,29H
                ADD     A,#0BH
                MOV     R4,A
                MOV     A,#13H
                MOV     R0,A
                ADD     A,#0BH
                MOV     R1,A            ;R0 = POINTER TO LOW BYTE
                SJMP    SUBCOMP
TABLE:          INC     R1
                INC     R0
                MOV     A,R4
                XRL     A,R0
                JZ      A_SHIFT
SUBCOMP:        MOV     A,R7            ;REGISTER A KEEP LOW BYTE
                CLR     C
                SUBB    A,@R1
                MOV     A,R6            ;REGISTER A KEEP HIGH BYTE
                SUBB    A,@R0
                JC      TABLE
A_SHIFT:        MOV     A,R1
```

```
                CLR     C
                SUBB    A,#16H
                MOV     R1,A
                CJNE    R1,#08H,NO_CHANGE
                MOV     TH0,R6
                MOV     TL0,R7
NO_CHANGE:      MOV     A,R3
                MOV     R0,29H
                ACALL   SHIFTUP
                MOV     A,R1
                ADD     A,#0BH
                MOV     R1,A            ;R1 = FIRST BYTE
                MOV     A,29H
                ADD     A,#0BH
                MOV     R0,A
                MOV     A,R6
                ACALL   SHIFTUP
                MOV     A,R1
                ADD     A,#0BH
                MOV     R1,A
                MOV     A,29H
                ADD     A,#16H
                MOV     R0,A
                MOV     A,R7
                ACALL   SHIFTUP
                INC     29H
                RET
```

```
SHIFTUP:    PUSH    01H
SH_NEXT:    PUSH    0E0H            ;KEEP INSERT VALUE
            MOV     A,R0
            XRL     A,R1
            JZ      LAST            ;CHECK LAST SHIFT
            POP     0E0H
            XCH     A,@R1
            INC     R1
            SJMP    SH_NEXT
LAST:       POP     0E0H
            XCH     A,@R1
            POP     01H
            RET

ERROR:      MOV     DPH,#00H
            MOV     A,R3
            MOVX    @DPTR,A
            MOV     R2,#0FFH
DELAY_E1:   MOV     R3,#0FFH
DELAY_E2:   DJNZ    R3,DELAY_E2
            DJNZ    R2,DELAY_E1
            CLR     A
            MOVX    @DPTR,A
            MOV     DPH,#80H
            RET
            END
```

ทดสอบ program controller

complier program ด้วย software SXA51 และทดสอบ step การทำงานด้วย software S51

ผลการทดสอบ program controller

เนื่องจาก program ประกอบด้วย 2 ส่วน คือ ส่วนการ key ค่า delay กับส่วนของ I/O ได้ทำการทดลองทีละส่วน ในแต่ละส่วนได้ผลเป็นไปตามจุดประสงค์ เมื่อนำทั้ง 2 ส่วนมาเชื่อมโยงกัน จะพบปัญหา คาดว่าเกิดจาก stack overflow ตอนเกิด overflow ของ timer ในขณะที่ program ทำงานอยู่ในส่วนการ key ค่า delay

เพื่อตัดปัญหาดังกล่าว การ key ค่า delay จึงมีข้อจำกัด คือ ไม่สามารถ key ค่าdelay ได้ในขณะที่มีการใช้ timer และมีข้อจำกัดของการรับ fault จาก port input คือ ไม่สามารถ ตรวจจับ fault ได้ในขณะที่มีการใช้ key

สรุปและวิจารณ์ผลการทดลอง

เนื่องจากแนวความคิดในการออกแบบ hardware controller เน้นที่การออกแบบวงจรให้มีความสามารถตามความต้องการโดยมีราคาที่ต่ำ (ไม่ฟุ่มเฟือยในการใช้งาน IC) ทำให้การพัฒนา program มีข้อจำกัดจาก stack overflow เพราะว่า ส่วนหนึ่งของ internal ram ถูกใช้เป็น buffer ของข้อมูล เนื้อที่ในการ push pop มีน้อยลง ผลคือการควบคุม stack เมื่อเกิด overflow ทำได้ยาก ทำให้ความสามารถของ program controller มีข้อจำกัด คือ ไม่สามารถ key ค่า delay ได้ในขณะที่มีการใช้ timer และมีข้อจำกัดของการรับ fault จาก port input คือ ไม่สามารถ ตรวจจับ fault ได้ในขณะที่มีการใช้ key

บรรณานุกรม

1.รศ.ศุลี บรรจงจิตร,"หลักการและเทคนิคการออกแบบระบบไฟฟ้ากำลัง",บริษัท ซีเอ็ด ยูเคชั่น จำกัด

2.Kenneth J. Ayala,"The 8051 Microcontroller",West publishing company

3.Intel,"MCS-51 data book",1980

4.ซีเอ็ด ยูเคชั่น,"คู่มือไอซี ตระกูลทีทีแอล"