AUTOMATIC LOADER IC

นาย จารึก มีเชื้อ 313604

นาย ชูชาติ จิตรน้อม 313606

นาย สัมฤทธิ์ บำรุงสุข 313630

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาอุตสาหกรรมศาสตรบัณฑิต

ภาควิชาเทคนิคอุตสาหกรรม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2533

หัวข้อเรื่อง AUTOMATIC LOADER IC

โดย นาย จารึก มีเชื้อ 313604
นาย ชูชาติ จิตรน้อม 313606
นาย สัมฤทธิ์ บำรุงสุข 313630

ภาควิชา เทคนิคอุตสาหกรรม
อาจารย์ที่ปรึกษา อาจารย์ กฤษดากร กลอมการ
ปีการศึกษา 2533

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง อนุมัติให้นำปริญญานิพนธ์นี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์

.....คณบดีคณะวิศวกรรมศาสตร์
( )

.....ประธานกรรมการ
( )

.....กรรมการ
( )

.....กรรมการ
( )

.....กรรมการ
( )

หัวข้อปริญญานิพนธ์	AUTOMATIC LOADER IC
โดย	นาย จารึก มีเชื้อ
	นาย ชูชาติ จิตรน้อม
	นาย สัมฤทธิ์ บำรุงสุข
ภาควิชา	เทคนิคอุตสาหกรรม
อาจารย์ที่ปรึกษา	อาจารย กฤษดากร กล่อมการ
ปีการศึกษา	2533

บทคัดย่อ

ปัจจุบันคอมพิวเตอร์ได้มีบทบาทในการควบคุมทางด้านอุตสาหกรรม เนื่องจากมีความแม่นยำถูกต้อง, รวดเร็ว สะดวกต่อการใช้งาน และจะต้องประหยัดในเชิงเศรษฐกิจ

โครงการนี้เป็นการสร้างเครื่องมือสำหรับใช้ในงานอุตสาหกรรมผลิต IC ซึ่งแบ่งออกเป็น 2 ส่วนคือ ส่วนรับ IC (input shuttle) และส่วนปล่อย IC (output shuttle) และแต่ละส่วนยังแบ่งตามโครงสร้างได้อีก 2 ส่วนคือ ส่วนที่เป็นอุปกรณ์อิเลคทรอนิคส์ (ระบบควบคุม) และส่วนที่เป็น MACHANIC

หลักการทำงาน อาศัยไมโครโปรเซสเซอร์ไปควบคุมการทำงานส่วนที่เป็น MACHANIC (ปิด, เปิด VALVE)

กิตติกรรมประกาศ

การทดลองโครงการนี้ได้สำเร็จลุล่วงไปด้วยดี ทางคณะผู้จัดทำขอขอบพระคุณบริษัทงานหวีอิเล็คทรอนิกส์ จำกัด ที่ได้ให้การสนับสนุนทั้งทางด้านเครื่องมือที่ใช้ในการทดลอง เงินทุน ค่าวัสดุอุปกรณ์ของโครงการนี้ ตลอดจนสถานที่ที่ใช้ในการทดลอง และขอขอบพระคุณอาจารย์ กฤษดากร กล่อมการ ผศ.ดร.กนก เจนจิระพงศ์เวช ที่ได้ให้คำแนะนำปรึกษาเพื่อเป็นวิทยาทานในการทดลองโครงการนี้

คณะผู้จัดทำ

คำนำ

ในการจัดทำ PROJECT ในหัวข้อเรื่อง AUTOMATIC LOADER IC นี้ เพื่อเป็นการตอบสนองนโยบายของรัฐบาล ในการที่จะทำให้ประเทศไทยเป็นประเทศที่มีความเจริญก้าวหน้าทางด้านอุตสาหกรรม หรือเป็นประเทศอุตสาหกรรมใหม่ (NIC) และในปัจจุบันอุตสาหกรรมการผลิตไอซีเป็นที่สนับสนุนจากรัฐบาล และทำให้มีเงินตราเข้าประเทศมากขึ้น แต่ยังขาดวิทยาการและเครื่องมือเครื่องใช้ในขบานการผลิตซึ่งจะให้ได้มาซึ่งผลิตภัณฑ์ต่าง ฉะนั้นในการทำ PROJECT นี้จะช่วยให้มีเครื่องมือเครื่องใช้ในอุตสาหกรรมการผลิตไอซี ซึ่งสามารถผลิตได้เองในประเทศไทยและจะได้เป็นแนวทางในการพัฒนาต่อไป

ในการจัดทำ PROJECT นี้อยู่ในความควบคุมและการให้คำปรึกษาของ ผศ.ดร.ณก เจนจิระพงศ์เวช และ อ.กฤษกาญ กล่อมการ ซึ่งคณะผู้จัดทำขอขอบพระคุณมา ณ โอกาสนี้ ด้วย

คณะผู้จัดทำ

สารบัญ

บทที่ ๑
บทนำ

ในปัจจุบันการพัฒนาทางด้านคอมพิวเตอร์ได้มีบทบาท สำคัญในวงการอุตสาหกรรมทั่วไป โดยในอุตสาหกรรมการผลิตไอซีนั้น จำเป็นต้องใช้เครื่องมือที่มีความเร็วสูง เที่ยงตรง และสะดวกต่อการใช้งาน โดยการทำขบวนการต่างๆเกี่ยวกับไอซี เช่น การพิมพ์เบอร์ไอซี การตัดขาไอซี และอื่นๆ ซึ่งเครื่องมือที่ใช้ในปัจจุบันมีไม่มากนักในประเทศไทย

ด้วยเหตุนี้การที่จะคิดค้นให้มีเครื่องมือที่ทำขบวนการต่างๆนั้นจึงจำเป็นอย่างยิ่ง ซึ่งเครื่อง AUTOMATIC LOADER IC ที่จัดทำนี้เป็นการใช้ไมโครโปรเซสเซอร์มาควบคุมร่วมกับระบบฮาร์ดแวร์ที่เกี่ยวข้องทำให้ได้เครื่องที่มีคุณสมบัติที่ดีด้วยกันหลายประการดังนี้

๑. สามารถทำงานที่เป็นขั้นตอนซ้ำๆกันได้ โดยมีความแม่นยำและรวดเร็วกว่าระบบทั่วๆไป

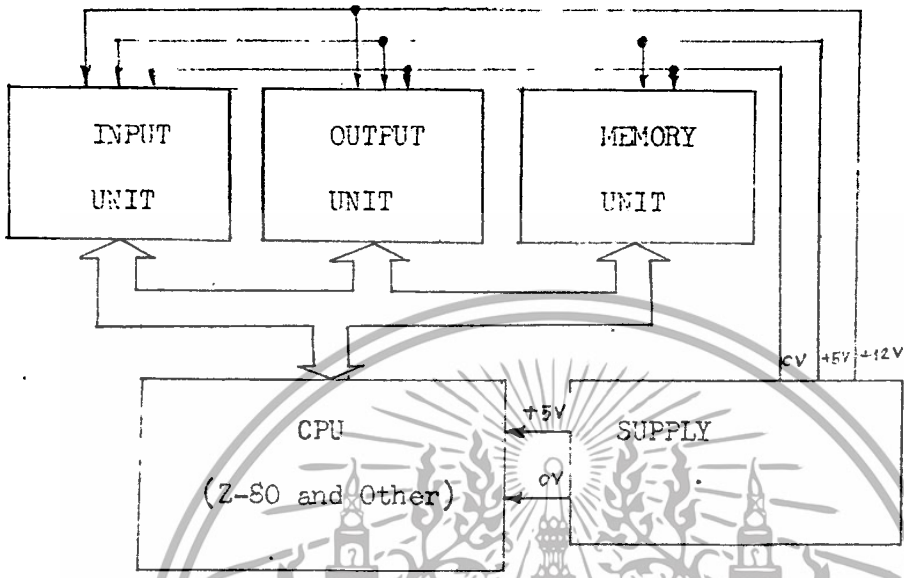
๒. สามารถต่อใช้งานกับเครื่องจักรที่มีอยู่เดิมซึ่งต้องใช้แรงงานคนได้เกือบทุกชนิด

๓. ควบคุมโดยสั่งให้เครื่องทำงานได้ง่าย และผู้ใช้สามารถเข้าใจได้ สะดวกและ รวดเร็ว

๔. สามารถลดต้นทุนการผลิตลง เพราะจะลดจำนวนผู้ปฏิบัติงานลงได้มาก และทำงานได้รวดเร็วกว่า

๕. มีระบบตรวจสอบ คุณสมบัติต่างๆที่คนไม่สามารถตรวจสอบได้ในเวลาอันรวดเร็ว เช่นการตรวจสอบว่า ไอซีที่ทำการสร้างขึ้นมานั้นมีการพิมพ์เบอร์กลับด้านหรือไม่ เป็นต้น

จากการที่เรานำเอาระบบ Micro-Processor มาใช้งานร่วมกับระบบทาง Machanic นั้นมีหลักการทำงานคร่าวๆดังนี้



รูปที่ ๑–๑ แสดงบล็อค ไดอะแกรม ของระบบ

จากรูปที่ ๑ – ๑ จะเป็น Block Diagram แสดงการทำงานของระบบ ซึ่งเป็นได้ทั้งส่วนปล่อยไอซี (OUTPUT SHUTTLE) และ ส่วนรับไอซี(INPUT SHUTTLE) โดยจะแตกต่างกันที่โปรแกรมในการควบคุม และ INPUT & OUTPUT UNIT บางส่วน

Central Processing Unit (CPU)

ซึ่งมีไมโครโปรเซสเซอร์ เป็นตัวควบคุมให้ระบบแมคคานิคปล่อยไอซีซึ่งบรรจุอยู่ในหลอด ให้ผ่านขบวนการทีละตัว (ในกรณีของ OUTPUT SHUTTLE) และจะตรวจสอบโดยใช้ โฟโต้เซนเซอร์ ว่าไอซีหมดหลอดหรือยัง ถ้ายังไม่หมดก็ให้ปล่อยไอซีตัวต่อไป แต่ถ้าหมดแล้วก็ให้นำหลอดเปล่าออกแล้วนำหลอดใหม่ซึ่งมีไอซีบรรจุอยู่เต็มมาใส่แทน และทำซ้ำอยู่เช่นนี้ต่อไป

ส่วนในการควบคุมระบบ แมคคานิคที่ทำหน้าที่บรรจุไอซีที่ผ่านขบวนการแล้วลงในหลอด (INPUT SHUTTLE) จะตรวจสอบว่า ไอซีที่ผ่านขบวนการมานั้น เต็มหลอดหรือยัง ถ้ายังก็ให้บรรจุต่อไปจนกว่าจะเต็ม แล้วนำหลอดที่ถูกบรรจุเต็มแล้วออก พร้อมกันนั้นก็นำหลอดเปล่ามารอรับไอซีแทน โดยทำซ้ำเช่นนี้ตลอด ทั้งนี้จะมีการตรวจสอบว่ามีหลอดเปล่าอยู่ในช่องรอรับหรือไม่ เพื่อป้องกันการผิดพลาดอันเกิดจากไอซีถูกบรรจุในขณะที่ยังไม่มีหลอด

โดยระบบทั้งหมดสามารถหยุดการทำงานหรือปล่อยหลอดออกได้ ขณะเกิดการผิดพลาดได้โดยใช้สวิทช์ควบคุม

บทที่ 2

นิยามไมโครโปรเซสเซอร์และไมโครคอมพิวเตอร์

โครงสร้างเบื้องต้น

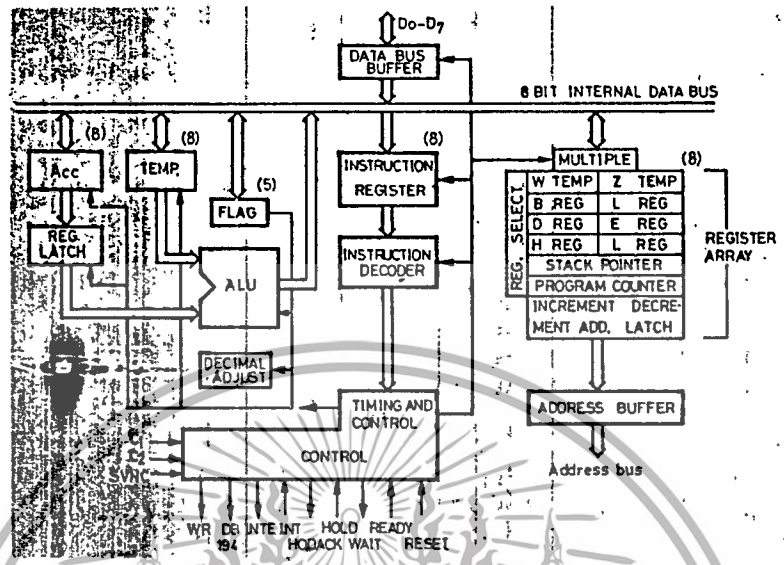
ดังได้กล่าวมาในบทที่แล้วถึงขบวนการต่าง ๆ ที่เกิดขึ้นในคอมพิวเตอร์เมื่อคอมพิวเตอร์ทำงานหน่วยต่าง ๆ ภายใน ซีพียู ซึ่งเป็นวงจรอิเลคทรอนิคส์มีลักษณะสมบัติของแต่ละหน่วยเฉพาะตัวและมีวงจรเชื่อมต่อกับหน่วยอื่น ๆ เพื่อทำงานร่วมกัน เป็นไมโครโปรเซสเซอร์ ลักษณะวงจรภายในและการเชื่อมต่อการควบคุมภายในเราถือเป็นโครงสร้าง (Architecture) ทางฮาร์ดแวร์ (hardware) ของไมโครโปรเซสเซอร์ตัวนั้น

ลักษณะวงจรและการควบคุมถือเป็นภายในของไมโครโปรเซสเซอร์ตัวหนึ่ง จะทำให้ไมโครโปรเซสเซอร์ตัวนั้นเข้าใจคำสั่งอยู่จำนวนหนึ่ง และลักษณะสมบัติของคำสั่งต่าง ๆ เราถือเป็นโครงสร้าง (Architecture) ทางซอฟท์แวร์ของมัน

โครงสร้างไมโครโปรเซสเซอร์

ลักษณะภายในของไมโครโปรเซสเซอร์ เมื่อเขียนให้เห็นละเอียดขึ้นเราจะเห็นหน่วยต่าง ๆ และการติดต่อหรือควบคุมภายใน ตลอดจน Registor ต่าง ๆ ดังรูปที่ 2.1

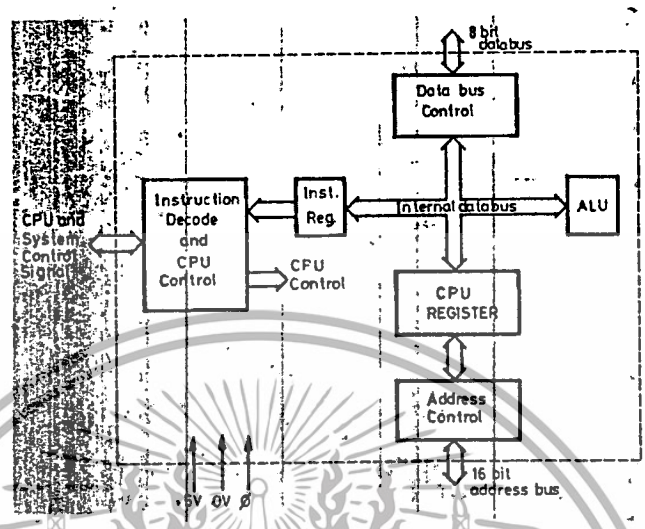
ตามรูปเป็นโครงสร้างภายในของไมโครโปรเซสเซอร์เบอร์ 8080 แสดงให้เห็นเป็นแผนภูมิแสดงหน่วยย่อย ๆ ต่าง ๆ ภายใน จะเห็นการเชื่อมโยงกันระหว่างหน่วยต่าง ๆ และ รีจิสเตอร์ เราเรียกเส้นที่เชื่อมโยงต่อกันนี้ว่า บัส (Bus) ซึ่งเราจะนิยามบัสได้ว่า บัส คือ เส้นทางเชื่อมโยงที่ใช้ในการส่งผ่านข้อมูลหรือสัญญาณควบคุมระหว่างหน่วยต่าง ๆ ตามรูปที่ 2.1 เส้นลูกศรหนา ๆ คือ บัส จะเห็นว่าเชื่อมต่อหน่วยต่าง ๆ ภายในและเชื่อมต่อกับระบบอื่นภายนอกก็คือ D_0-D_7 เรียกว่า บัสข้อมูล A_0-A_7 เป็น แอดเดรสบัส



รูปที่ 2.1 ไมโครโปรเซสเซอร์ เบอร์ 8080

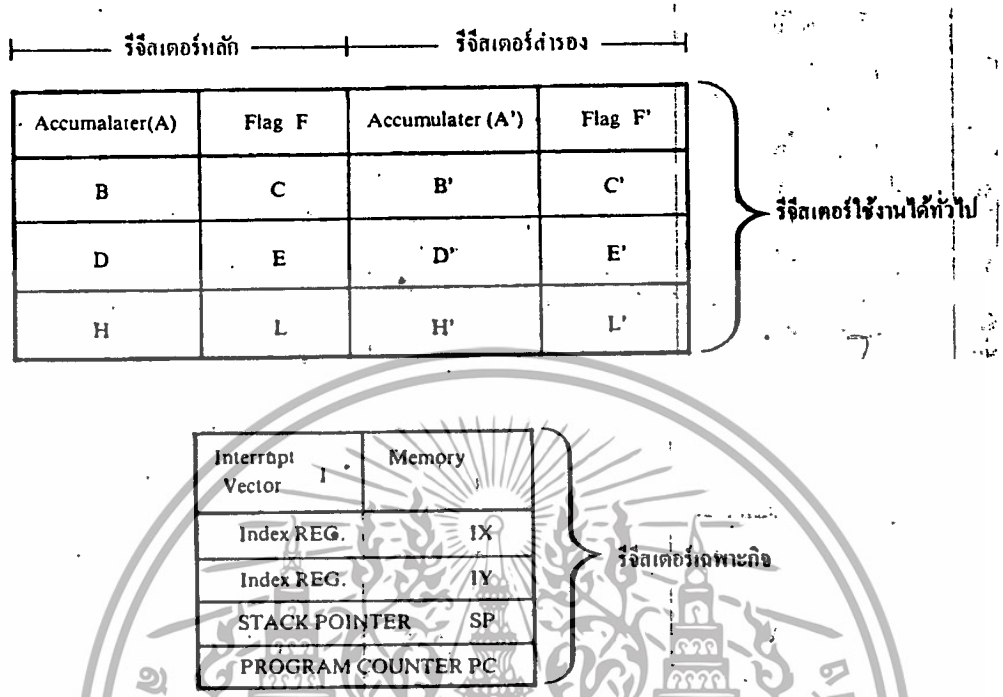
บัสสองทาง (Bidirectional bus) คือ บัสที่นำมาใช้ข้างต้นที่สามารถเป็นทางส่งผ่านข้อมูลหรือสัญญาณได้ทั้งสองทาง ตามรูป $D1_{oi}$-D_7 เป็นบัสสองทาง สังเกตง่าย ๆ ได้จากเส้นเชื่อมโยงที่มีลูกศรทั้งสองปลาย

ตามรูปจะเห็น รีจิสเตอร์ 2 ตัว ทำหน้าที่เก็บข้อมูลให้หน่วยคำนวณสำหรับทำฟังชั่นต่าง ๆ รีจิสเตอร์ที่เห็นจะมี IR (Instruction Register) 1 ตัว ทำหน้าที่เก็บคำสั่งที่อ่านมาจากหน่วยความจำภายนอก และรีจิสเตอร์ตัวอื่น ๆ คือ B, C, D, E, H, L และสแตคพอยเตอร์ โปรแกรมเคาน์เตอร์เป็นต้น ตัวเลขที่กำกับไว้ตรงมุมไดอะแกรมของรีจิสเตอร์ บอกถึงขนาดความกว้างของรีจิสเตอร์นั้น เช่น 8 คือ กว้างขนาด 8 บิท



รูปที่ 2.2 โครงสร้างภายในของซีพียู Z-80

ตัวอย่าง โครงสร้างของไมโครโปรเซสเซอร์อีกเบอร์หนึ่ง คือ Z-80 แสดงได้ดังรูปที่ 2.2 ตามรูป จะเห็นว่ามีส่วนคล้ายกับรูปที่ 2.1 อยู่มากกล่าวคือ มี IR (Instruction Register) มี ALU (Arithmetic and logical unit) บัสข้อมูล แอดเดรสบัส รีจิสเตอร์ (ในรูปนี้ไม่ได้บอกว่ามีเท่าใด) และตัวถอดรหัสคำสั่ง (Instruction decode) และวงจรควบคุมซีพียู (CPU control) ทำหน้าที่ถอดรหัสคำสั่งว่าให้ทำอะไรพร้อมกับส่งสัญญาณออกมาทางสายบัสควบคุมหน่วยต่าง ๆ ภายในให้ตอบสนองต่อคำสั่งนั้นๆ รูปที่ 2.2 เป็นไดอะแกรมทางด้านฮาร์ดแวร์ ส่วนในด้านซอฟท์แวร์ แล้ว เราจะเห็น Z-80 ตัวนี้เป็นดังนี้คือ



รูปที่ 2.3 เป็นโครงสร้างทางซอฟท์แวร์

จะเห็นว่าทางด้านซอฟท์แวร์เราจะมองเห็นว่า Z-80 มีรีจิสเตอร์อยู่จำนวนเท่าใด และมีคำสั่งใช้งานกี่คำสั่งและในรายละเอียดทางคำสั่งจะแสดงและอธิบายถึงการตอบสนองของ Z-80 ต่อสัญญาณภายนอก อย่างไรด้วยให้ดูรายละเอียดในหัวข้อ Z-80

<u>ขบวนการต่าง ๆ ของซีพียู</u>

เมื่อเริ่มรอบการทำงานของคำสั่ง ตัวเลขไบนารีในโปรแกรมเคาน์เตอร์ (program counter) หรือเรียกสั้น ๆว่า พีซี(PC) จะถือเป็นตำแหน่งในหน่วยความจำที่ซีพียูต้องการอ่านคำสั่งเอามาไว้ใน IR (Instruction Register) ตอนอ่านคำสั่งนี้เองตัวเลขที่ถือเป็นตำแหน่งหรือเรียกว่า แอดเดรส จะถูกส่งผ่านไปทางแอดเดรสบัส ซึ่งต่อไปถึงหน่วยความจำที่อยู่ภายนอกซีพียู หน่วยควบคุมจะส่งสัญญาณอื่น ๆ อีกเท่าที่จำเป็นไปยังหน่วยความจำ พร้อม ๆ กับส่งข้อมูลแอดเดรสออกไป เพื่อให้รู้ว่าซีพียูต้องการข้อมูลตรงตำแหน่งแอดเดรสนี้ หน่วยความจำภายนอกจะส่งข้อมูลของแอดเดรสนั้นมาให้ซีพียูผ่านทางบัสข้อมูล

2. ไซเคิลการเขียน (Write machine cycle) คือขบวนการเอาข้อมูลไปเก็บไว้ในหน่วยความจำ

3. ไซเคิลเอาท์พุท (Output machine cycle) คือขบวนการส่งข้อมูลไปยังอุปกรณ์ I/O

4. ไซเคิลอินพุท (Input machine cycle) คือขบวนการรับข้อมูลจากอุปกรณ์ I/O

ขบวนการทั้ง 4 ขบวนการนี้เกี่ยวข้องกับคำสั่งโดยตรง ยังมีขบวนการหลายแบบที่เป็นการตอบสนองต่อสัญญาณภายนอก และระบบภายนอกโดยไม่ขึ้นกับคำสั่ง เช่น

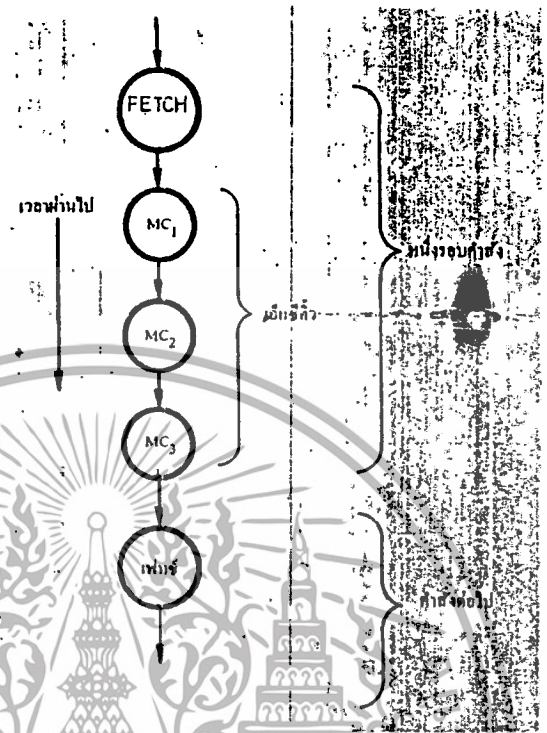
1. ไซเคิลการตอบสนองต่อการอินเตอร์รัพท์ (Interrupt acknowledge machine cycle) คือขบวนการตอบสนองสัญญาณภายนอกที่อินเตอร์รัพท์เข้ามา

2. ไซเคิลการตอบสนองต่อการใช้บัส (Bus acknowledge machine cycle) คือขบวนการตอบสนองต่อวงจรภายนอก ซึ่งต้องการใช้เส้นทางส่งข้อมูล (Bus)

3. ไซเคิลการเฟทช์ (Fetch machine cycle) คือขบวนการอ่านคำสั่งจากหน่วยความจำ รายละเอียดในการทำแต่ละแมชีนไซเคิล (machine cycle) ของไมโครโปรเซสเซอร์เบอร์ต่าง ๆ จะแตกต่างกันออกไป ขึ้นอยู่กับโครงสร้างทางวงจรฮาร์ดแวร์ (hard ware) ของไมโครโปรเซสเซอร์เบอร์นั้น ๆ

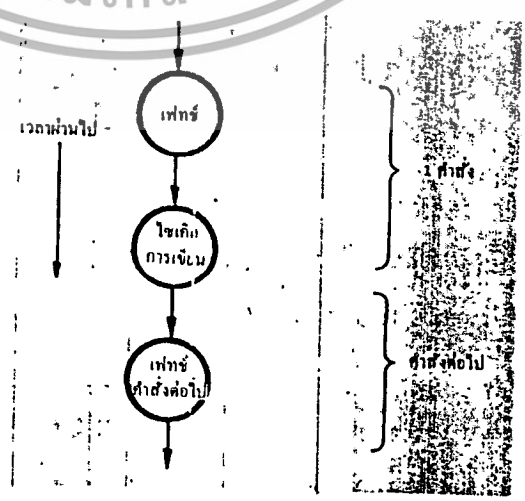
ขบวนการเฟทช์ ที่แท้จริงก็คือขบวนการอ่านข้อมูลจากหน่วยความจำนั่นเอง แต่ไม่เรียกเป็นไซเคิล การอ่าน (Read machine cycle) เพราะมีความแตกต่างที่ตอนซีพียูทำขบวนการเฟทช์ นอกจากจะอ่านข้อมูลมาแล้วยังมีการถอดระหัสและตัดสินใจภายในตัวซีพียูต่อไปอีก ซึ่งขบวนการอ่าน เป็นการอ่านมาเฉย ๆ สำหรับคำสั่งที่ต้องเอ็กซีคิ้วด้วยแมชีนไซเคิลหลาย ๆ อันจะเป็นดังนี้

รูปที่ 2.5

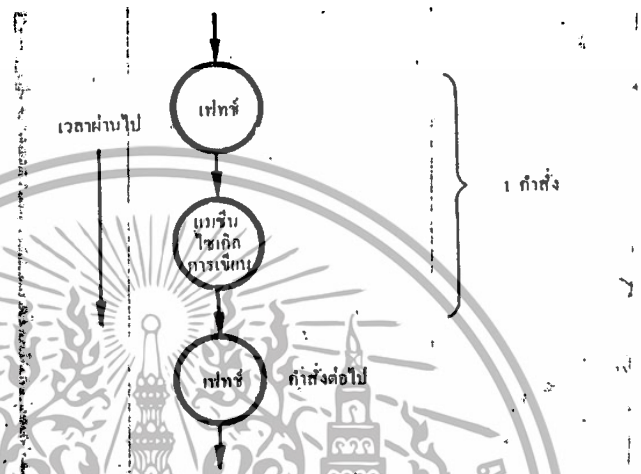


ตัวอย่าง คำสั่งให้อ่านข้อมูลจากหน่วยความจำมาไว้ในรีจิสเตอร์ตัวหนึ่งภายในซีพียูจะประกอบด้วยขั้นตอนดังรูปที่ 2.6

รูปที่ 2.6



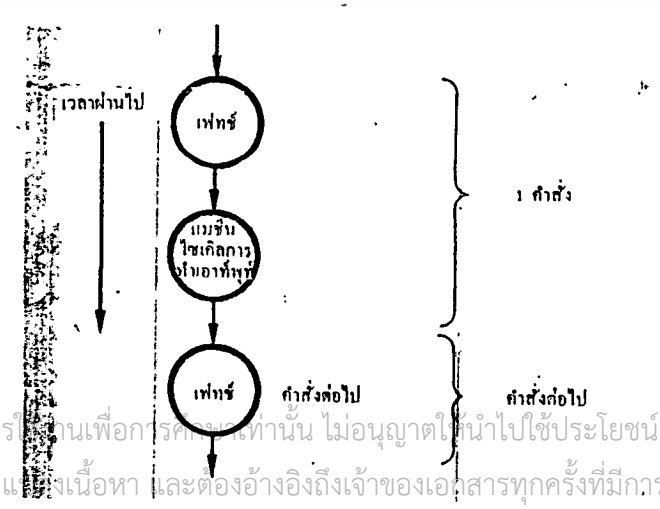
ตัวอย่าง คำสั่งให้เอาข้อมูลจากรีจีสเตอร์ตัวหนึ่งไปเก็บไว้ในหน่วยความจำ การทำคำสั่งนี้จะประกอบด้วยขั้นตอนต่อไปนี้



รูปที่ 2.7

ในกรณีที่คำสั่งใดที่ไม่เกี่ยวกับขบวนการติดต่อหน่วยภายนอกซีพียูเลย เช่น คำสั่งให้จ่ายข้อมูลจากรีจีสเตอร์ตัวหนึ่ง ภายในตัวซีพียูไปยังรีจีสเตอร์อีกตัวหนึ่งภายในซีพียูเช่นกัน ลักษณะเช่นนี้เมื่อซีพียูอ่านคำสั่งนี้มาถอดรหัสแล้ว ไม่ต้องทำขบวนการ Read หรือ Write หรืออื่น ๆ เลยเพราะการถ่ายข้อมูลเกิดภายใน

ตัวอย่าง (คำสั่ง I/O) คำสั่งให้เอาข้อมูลจากแอคคิวมูเลเตอร์ (A) ส่งไปยังอุปกรณ์ I/O ภายนอกการทำคำสั่งนี้จะประกอบด้วยขบวนการต่อไปนี้

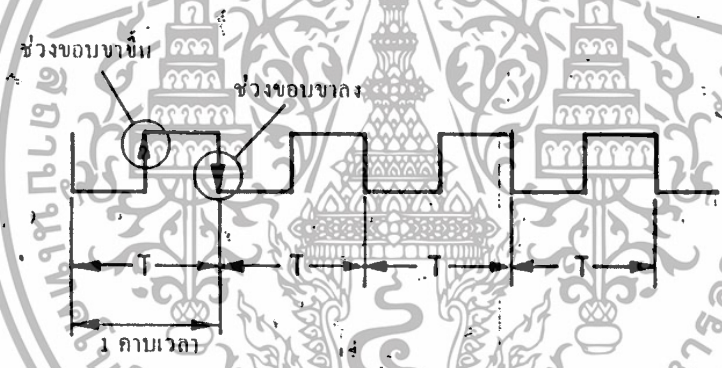


รูปที่ 2.8

ในทำนองเดียวกันคำสั่งให้เอาข้อมูลจากอุปกรณ์ I/O เข้ามายังแอคคิวมูเลเตอร์ ก็ต้องทำแมชีนไซเคิลอินพุทด้วย

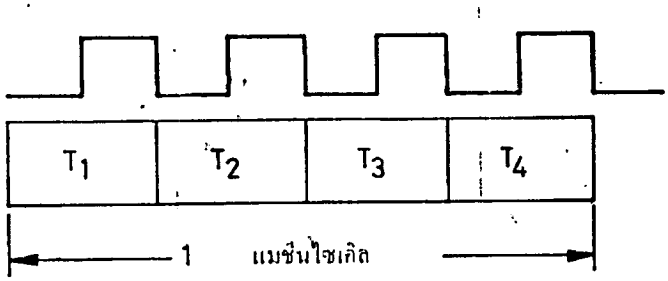
ให้สังเกตว่าทุกครั้งที่ซีพียูจะติดต่อกับระบบอื่น เช่น หน่วยความจำหรือ I/O ซีพียูจะต้องทำเป็นขบวนการเสมอ

เนื่องจากไมโครโปรเซสเซอร์ทำงานโดยอ้างอิงกับสัญญาณนาฬิกาเป็นหลัก การทำขบวนการต่าง ๆ ต้องใช้เวลา และขั้นตอนเป็นจังหวะก้าว ๆ ไป แมชีนไซเคิลใด ๆ จึงประกอบด้วยขั้นตอนย่อย ๆ ลงไปอีก ซึ่งต้องใช้เวลาโดยหน่วยควบคุมจะดำเนินการเปลี่ยนสถานะลอจิกภายในตลอดเวลาที่สัญญาณนาฬิกาแต่ละคาบผ่านไปรูปที่ 2.9 เป็นรูปสัญญาณนาฬิกา



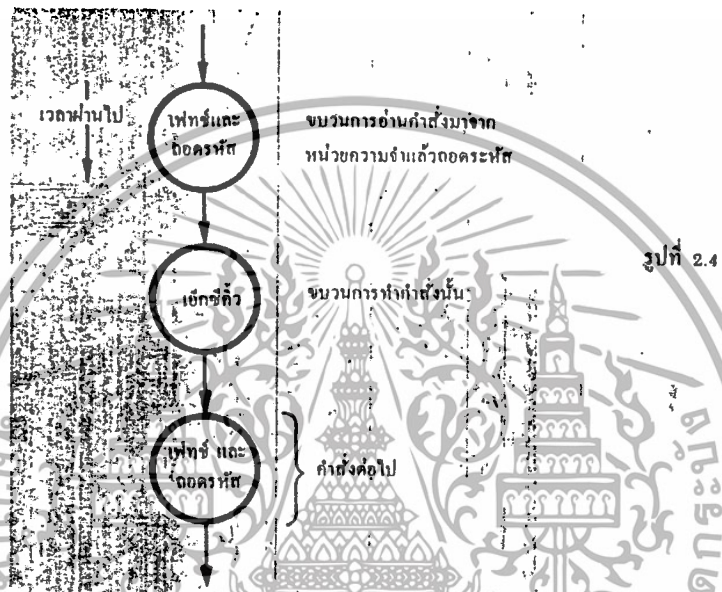
รูปที่ 2.9

ใน 1 แมชีนไซเคิลใด ๆ ใช้เวลาหลาย ๆ คาบจึงกระทำจบ ขบวนการแต่ละคาบเวลาเรียกว่า T_1, T_2, T_2, T_3 และ T_4 ตามรูปที่ 2.10



รูปที่ 2.10

ซีพียูจะนำมาเก็บไว้ใน IR (Instruction Register) แล้วจึงถอดระหัสและทำคำสั่งนั้น ลำดับขั้นตอนการทำงานเป็นดังรูปที่ 2.4



รูปที่ 2.4

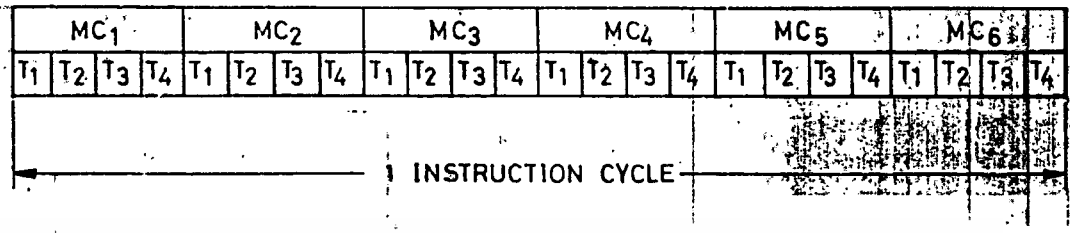
เริ่มขบวนการ เฟทช์ และ ถอดรหัส และ เอ็กซีคิว รวมกันเรียกว่า รอบของคำสั่ง ซึ่งเริ่มตั้งแต่ซีพียูอ่านคำสั่งจากหน่วยความจำแล้วถอดระหัสคำสั่งนั้นเสร็จแล้วก็ทำหรือเรียกว่า เอ็กซีคิว

การทำคำสั่งนั้น คือ การทำขบวนการย่อย ๆ หลาย ๆ ชนิดแบบต่อเนื่องกันจนจบสิ้น ขบวนการย่อย ๆ ในไมโครโปรเซสเซอร์เรียกว่า แมชีนไซเคิล (machine cycle) การทำคำสั่งแต่ละคำสั่งจะประกอบด้วยการทำแมชีนไซเคิลหลาย ๆ แบบที่แตกต่างกันหรือซ้ำกันก็ได้ขึ้นอยู่กับคำสั่งนั้น ๆ

แมชีนไซเคิล ต่าง ๆ เหล่านี้คือ

1. ไซเคิลการอ่าน (Read machine cycle) คือขบวนการอ่านข้อมูลจากหน่วยความจำ

1 คำสั่งซึ่งประกอบด้วย หลายแมชีนไซเคิล ดังนี้



ตามรูปจะเห็นว่า 1 รอบคำสั่งอาจประกอบด้วยแมชีนไซเคิลมากถึง 6 แมชีนไซเคิล แต่ละแมชีนไซเคิลประกอบด้วย T_1 ถึง T_4

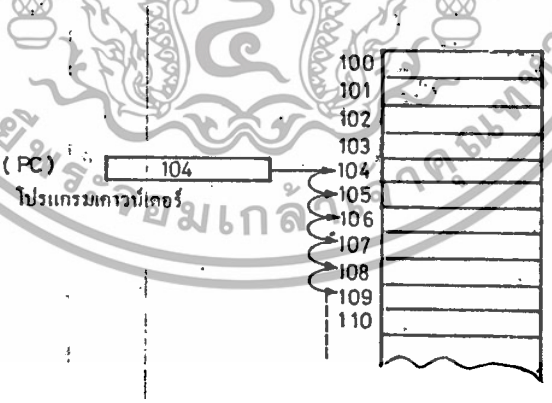
ตามรูปที่ 2.12 ซึ่งเป็นรูปสัญญาณนาฬิกา หน่วยควบคุมภายในจะอาศัยสัญญาณนาฬิกาเป็นตัวอ้างอิงโดยอาศัยการเปลี่ยนแปลงระดับของสัญญาณนาฬิกาจากระดับแรงดันหนึ่งไปยังอีกแรงดันหนึ่ง เรียกว่า edge หรือ สัน หรือ ขอบ ซึ่งมีอยู่ 2 สันคือ เปลี่ยนจากแรงดันสูงไปเป็นแรงดันต่ำ เราเรียกว่า ขอบลบ (negative edge) และเปลี่ยนจากแรงดันต่ำไปเป็นแรงดันสูงเราเรียกว่า ขอบบวก

ในแมชีนไซเคิลใด ๆ ก็ตาม การเปลี่ยนแปลงภายในจะอ้างอิงกับสันเสมอ เช่น ใน MC_2 ของคำสั่งหนึ่งคาบ T_1 วงจรควบคุมภายในซีพียูจะทำการควบคุมหรือเปลี่ยนแปลงสถานะทางลอจิกภายในแล้วยับไปอยู่ในคาบ T_2 การเปลี่ยนแปลงอาจจะเกิดจำเพาะภายในหรือแสดงเป็นสัญญาณออกมายังภายนอกซีพียูในรูปสัญญาณใด สัญญาณหนึ่งอย่างไรนั้น ขึ้นอยู่กับว่าซีพียูกำลังทำคำสั่งอะไรอยู่ ทางด้านวงจรฮาร์ดแวร์ (hardward) แล้วสัญญาณภายนอกอาจมายับยั้งไม่ให้ซีพียูดำเนินการอะไรต่อไป คือคงสถานะรออยู่เฉย ๆ ปล่อยให้หลายคาบเวลาผ่านไปก็ได้ สถานะที่ซีพียูรอค้างอยู่เราเรียกว่า T wait หรือ Tw Tw นี้เองช่วยให้วงจรภายนอกสามารถทำงานไปกับซีพียูได้โดยสามารถยับยั้งหรือให้ซีพียูรอให้วงจรภายนอกที่มีความเร็วต่ำกว่าทำงานทันกับซีพียูได้ รายละเอียดเรื่องแมชีนไซเคิลต่าง ๆ ของไมโครโปรเซสเซอร์ดูได้ในบทไมโครโปรเซสเซอร์ตัวอย่างโครงสร้างของ Z-80 ซึ่งมีกล่าวไว้โดยละเอียด

การทำงานของไมโครคอมพิวเตอร์

คำสั่งที่ถูกตระเตรียมมาก่อนหรือเรียกว่าโปรแกรมจะต้องถูกนำมาเก็บไว้ในหน่วยความจำ (ไม่ว่าจะด้วยวิธีใดก็ตาม) คำสั่งเหล่านั้นจะถูกเก็บไว้ในบริเวณใดบริเวณหนึ่งในหน่วยความจำ เมื่อจะให้คอมพิวเตอร์ทำโปรแกรมนั้นเราต้องบังคับให้ซีพียูไปเริ่มทำคำสั่งตั้งแต่จุดเริ่มต้นของโปรแกรมนั้น ทำได้โดยตั้งหรือจัดให้ (ไม่ว่าจะด้วยวิธีใดก็ตาม) ค่าของโปรแกรมเคาน์เตอร์ (PC) มีค่าเท่ากับแอดเดรส เริ่มต้นของคำสั่งที่เราจะให้ทำนั้นแล้วจึงให้ซีพียูเริ่มทำงาน

แอดเดรสในระบบหน่วยความจำเปรียบได้ง่าย ๆ เหมือนกับบ้านเลขที่ ที่เราใช้บอกตำแหน่งหรือแยกบ้านต่าง ๆ ออกจากกัน แต่แอดเดรสหรือบ้านเลขที่ไม่เกี่ยวกับสิ่งที่มีอยู่ในบ้านนั้น ในหน่วยความจำที่แอดเดรสต่าง ๆ ก็มีระหัสคำสั่งบรรจุอยู่ ซีพียูจะมีโปรแกรมเคาน์เตอร์ ซึ่งขนาดของมันต้องบรรจุตัวเลขไบนารีที่สามารถแทนแอดเดรสได้ทุกแอดเดรสในหน่วยความจำทั้งหมดได้ ทุกครั้งที่ซีพียูอ่านคำสั่ง 1 เวิร์ด จากหน่วยความจำไปแล้ว ค่าของโปรแกรมเคาน์เตอร์จะถูกเพิ่มขึ้นไปชี้ตำแหน่งถัดไปสำหรับการอ่านคำสั่งครั้งต่อไปเสมอ ดังรูปที่ 2.12

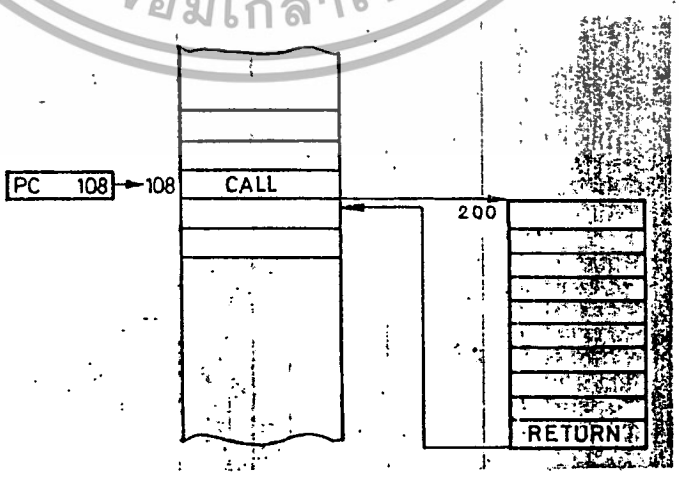


รูปที่ 2.12

ตัวอย่าง ตามรูป 2.12 ถ้า PC ถูกตั้งไว้ให้มีค่าเท่ากับ 104 เมื่อซีพียูเริ่มอ่านคำสั่ง ซีพียูจะอ่านคำสั่งจากตำแหน่ง 104 เสร็จแล้วตัว PC จะถูกเปลี่ยนค่าไปเป็น 105 เตรียมไว้สำหรับรอบคำสั่งต่อไป แล้วก็ทำคำสั่งที่ 104 ที่เพิ่งอ่านมานั้น จะเป็นอย่างนี้ไปเรื่อย ๆ จะเห็นว่าซีพียูจะทำคำสั่งจากตำแหน่งตัวเลขน้อยไปยังตัวเลขมาก

ในบางกรณีเราต้องการจะให้ละเว้นหรือข้ามบางคำสั่งไป จะมีคำสั่งประเภทหนึ่งเรียกว่าคำสั่ง JUMP หรือกระโดดข้ามไปยังจุดที่เราต้องการ แต่จะกระโดดไปไหนนั้นขึ้นอยู่กับการบอกตำแหน่งในคำสั่ง JUMP นั่นเอง

ในรูป 2.13 สมมติเราไม่ต้องการให้ซีพียูทำคำสั่ง 108 แต่ไปทำ 200 แทน คำสั่งใน 107 ต้องเป็น JUMP ไป 200 คำสั่งนี้จะทำให้ PC มีค่าเป็น 200 ทันทีเมื่อจบรอบคำสั่ง (Instruction Cycle) แทนที่จะเป็น 108 ทำให้เราเริ่มทำรอบคำสั่ง (Instruction cycle) ใหม่ ซีพียูจะไปเอาคำสั่งจาก 200 แทนในบางกรณีการกระโดดจากตำแหน่งหนึ่งไปที่ใหม่แล้ว ยังมีความจำเป็นต้องกลับมาตำแหน่งเดิมอีก ในกรณีนี้จะใช้คำสั่ง JUMP ธรรมดาไม่ได้ เพราะเมื่อเราต้องการกลับไปยังตำแหน่งเดิมเราต้องรู้ว่าเรากระโดดมาจากไหน จึงมีคำสั่งอีกประเภทหนึ่งซึ่งคล้าย JUMP แต่มีการเก็บค่าของ PC ไว้ยังที่ ๆ หนึ่งด้วย คำสั่งนั้นคือ CALL ตำแหน่งที่เก็บค่า PC ไว้เราเรียกว่า STACK เมื่อมีคำสั่ง CALL แล้วจะมีคำสั่งที่ใช้ควบคุมให้กระโดดกลับไปที่เดิมเราเรียกว่า RETURN ตามรูปแสดงความสัมพันธ์ระหว่าง CALL กับ RETURN

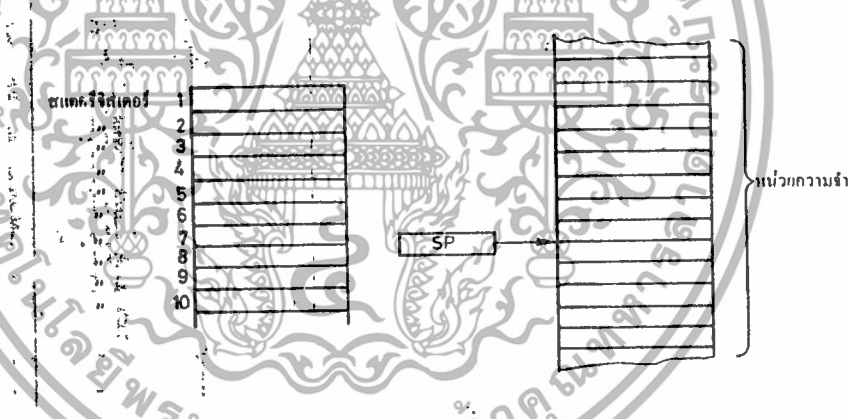


รูปที่ 2.13

สแตค (STACK) และสแตคพอยน์เตอร์ (STACK POINTER)

การทำคำสั่ง CALL ค่าของ PC จะถูกเก็บไว้ ณ.ที่หนึ่งในหน่วยความจำ ซึ่งเมื่อทำ RETURN ค่านี้จะถูกนำกลับมาใส่ใน PC อีก ที่ที่ PC ถูกเก็บไว้นี้เราเรียกว่าสแตค (STACK) ลักษณะของสแตค (STACK)มี 2 อย่างคือ ฮาร์ดแวร์สแตค (HARDWARE STACK) เป็นหน่วยความจำรีจิสเตอร์จำนวนหนึ่งที่มีไว้เฉพาะสำหรับเก็บค่า PC เมื่อทำ CALL ดังนั้นจำนวนเวิร์ดที่สามารถจะเก็บได้ก็คือจำนวนของรีจิสเตอร์นั่นเอง

ซอฟท์แวร์สแตค (SOFTWARE STACK) เป็น บริเวณหนึ่งในหน่วยความจำหลักที่ถูกกำหนดขึ้นให้ใช้เป็นที่เก็บ PC เมื่อทำคำสั่ง CALL ตำแหน่งนี้จะถูกกำหนดโดยค่าของสแตคพอยน์เตอร์ ซึ่งเป็นรีจิสเตอร์ตัวหนึ่ง ซึ่งโดยปกติก็มีขนาดกว้างของคำเท่ากับขนาดของ PC (ซึ่งแทนแอดเดรสได้ครบทุกตำแหน่งในหน่วยความจำ)



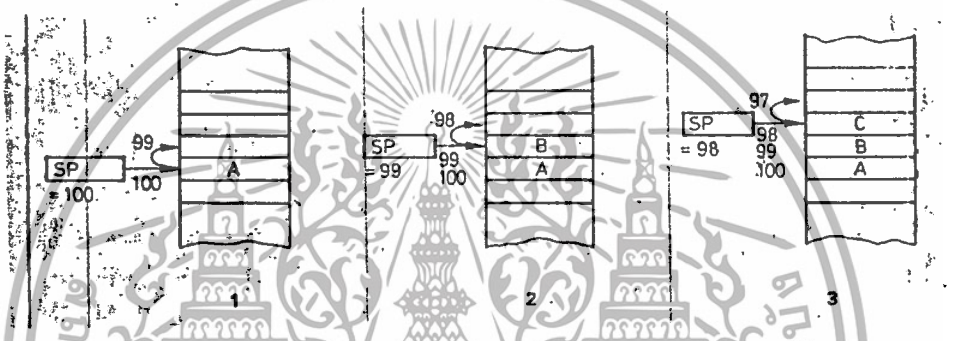
รูปที่ 2.14 ลักษณะของฮาร์ดแวร์สแตคจุได้ 10 คำ ลักษณะซอฟท์แวร์สแตคจุได้ คำ

ในความเป็นจริงแล้วสแตคไม่ได้มีไว้เก็บค่า PC เวลาซีพียูทำคำสั่ง CALL เท่านั้น แต่ยังมีคำสั่งที่สามารถให้เอาอะไรไปเก็บไว้ในสแตคก็ได้ คำสั่งนั้นคือ PUSH และก็ย่อมมีคำสั่งที่ควบคู่กันเป็นการให้นำสิ่งที่เก็บไว้ในสแตคกลับมา เรียกว่า POP

ลักษณะการเก็บข้อมูลในสแตคจะเป็นแบบเข้าทีหลังออกก่อน (Last in first out) กล่าวคือ ถูกเอาไปเก็บทีหลังจะถูกเอาออกมาก่อน ของใหม่จะซ้อนอยู่เหนือของที่เข้า

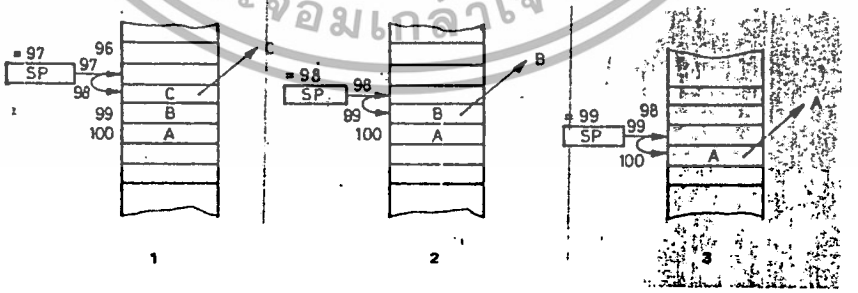
ไปก่อนเสมอไม่ว่าจะเป็นฮาร์ดแวร์สแตคหรือซอฟท์แวร์สแตค ทางฮาร์ดแวร์สแตคก็มีวงจรบังคับไว้เลย แต่ทางซอฟท์แวร์สแตคมีกลไกดังนี้คือ

ไม่ว่าซีพียูจะทำคำสั่ง CALL หรือ PUSH ก็ตาม สิ่งที่จะถูกเก็บเข้าสแตคจะถูกเก็บไว้ในตำแหน่ง ซึ่งชี้บอกด้วยค่าของสแตคพอยน์เตอร์ (SP) เมื่อสิ่งนั้นถูกเก็บไว้แล้วค่าของ SP จะถูกลดลงไป 1 ตำแหน่งเสมอ ดังนั้นถ้าทำ PUSH ซ้ำอีก ของใหม่จะไม่ถูกเก็บทับที่เดิม แต่จะอยู่ถัดไป ตามรูปแสดงการ PUSH 3 ครั้ง ติดต่อกัน



รูปที่ 2.15 แสดงกลไกการ PUSH ลงสแตค

คำสั่ง POP จะมีผลดังนี้คือ ค่าของ SP จะถูกเพิ่มขึ้นอีก 1 แล้วค่า (SP)+ 1 นี้เองจะชี้ไปยังตำแหน่งที่เก็บครั้งสุดท้ายเนื่องจากการทำ PUSH สิ่งที่ถูก PUSH อันสุดท้ายจะเอาออกมาก่อนดังรูปที่ 2.16



รูปที่ 2.16 แผนผังแสดงการ POP 3 ครั้งติดต่อกัน

โปรแกรมอินเตอร์รัพท์ (Program Interrupt)

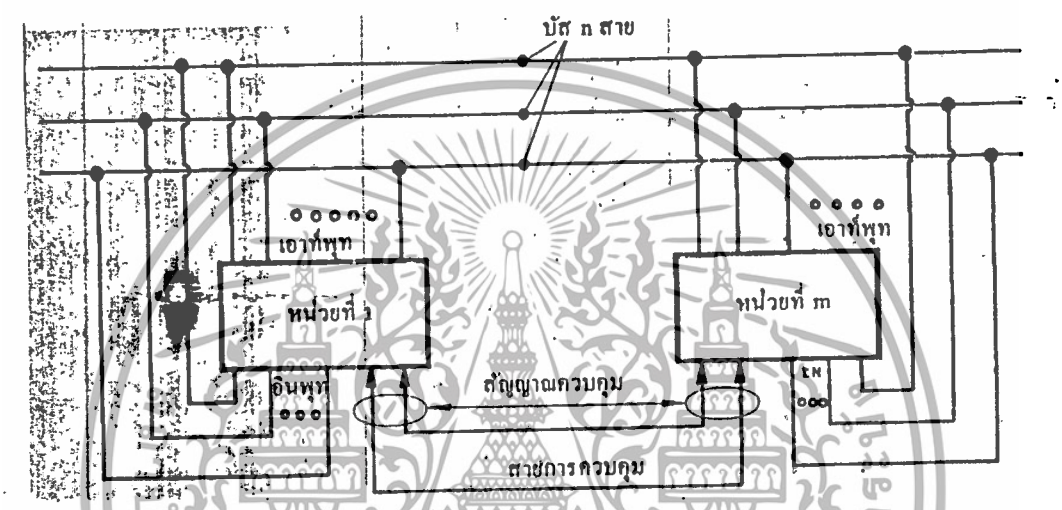
ในการใช้งาน ไมโครคอมพิวเตอร์จริง ๆ มีความจำเป็นจะต้องควบคุมและตอบสนองต่อสัญญาณที่มาจากภายนอก ไม่ว่าจะเป็นสัญญาณจากอุปกรณ์เอาท์พุทหรืออะไรก็ตาม ในขณะที่คอมพิวเตอร์กำลังทำโปรแกรมอยู่โปรแกรมหนึ่งนั้น วงจรฮาร์ดแวร์มันอาจได้รับสัญญาณจากภายนอกมาชัดจังหวะ ทำให้คอมพิวเตอร์ต้องเลิกทำโปรแกรมที่กำลังทำอยู่ชั่วคราว กระโดดข้ามไปทำโปรแกรมพิเศษที่ถูกเตรียมเพื่อเป็นการตอบรับสัญญาณอันนั้น เสร็จแล้วจึงกลับมาทำโปรแกรมเดิมต่อไป เราเรียกว่าโปรแกรมนี้ถูกชัดจังหวะ หรือถูกอินเตอร์รัพท์ (Interrupt) และโปรแกรมที่ถูกเตรียมไว้ให้คอมพิวเตอร์ทำเพื่อเป็นการตอบรับสัญญาณอินเตอร์รัพท์เราเรียกว่า โปรแกรมบริการอินเตอร์รัพท์ (Interrupt service routine) ลักษณะของสัญญาณที่สามารถทำให้เกิดการอินเตอร์รัพท์ และกลไกการตอบสนองต่อสัญญาณอินเตอร์รัพท์ของไมโครโปรเซสเซอร์แต่ละเบอร์ไม่เหมือนกัน ทั้งทางด้านวงจรและทางด้านโปรแกรม (Hardware และ software) แล้วแต่ว่าโครงสร้างของมันถูกออกแบบมาอย่างไร เช่น ไมโครโปรเซสเซอร์เบอร์ 8080 มีการตอบรับสัญญาณอินเตอร์รัพท์ได้แบบเดียว แต่เบอร์ Z-80 มีถึง 3 วิธี ซึ่งควบคุมได้ด้วยโปรแกรม ให้ดูรายละเอียดในเรื่อง Z-80 และรายละเอียดเรื่องอินเตอร์รัพท์

ระบบบัส

ก่อนจะถึงรายละเอียดเรื่อง ระบบความจำสิ่งที่ต้องทำความเข้าใจเสียก่อนคือ บัส (Bus) ดังนิยามมาแล้วว่า บัส คือ เส้นทางเชื่อมโยงที่ใช้ในการส่งผ่านข้อมูล หรือสัญญาณควบคุมระหว่างหน่วยต่าง ๆ ซึ่งสามารถใช้รวมกันได้ และข้อมูลที่ส่งผ่านบัสจะอยู่ในรูปของเวิร์ด (Word) ซึ่งมีหลาย ๆ บิท (bit) บัสอาจจะประกอบด้วยเส้นเชื่อมโยงแยกกันเป็นเส้น ๆ สำหรับแต่ละบิทในเวิร์ดหนึ่ง เราเรียกบัสแบบนี้ว่า บัสขนาน (word-parallel หรือ parallel bus) หรือบัสอาจจะประกอบด้วยเส้นเชื่อมโยงเพียงเส้นเดียวแต่ส่งข้อมูลทีละ 1 บิท ผลัดกันจนครบ 1 เวิร์ด บัสแบบนี้เราเรียกว่า บัสอนุกรม word-serial หรือ serial bus)

<u>บัสขนาน</u>

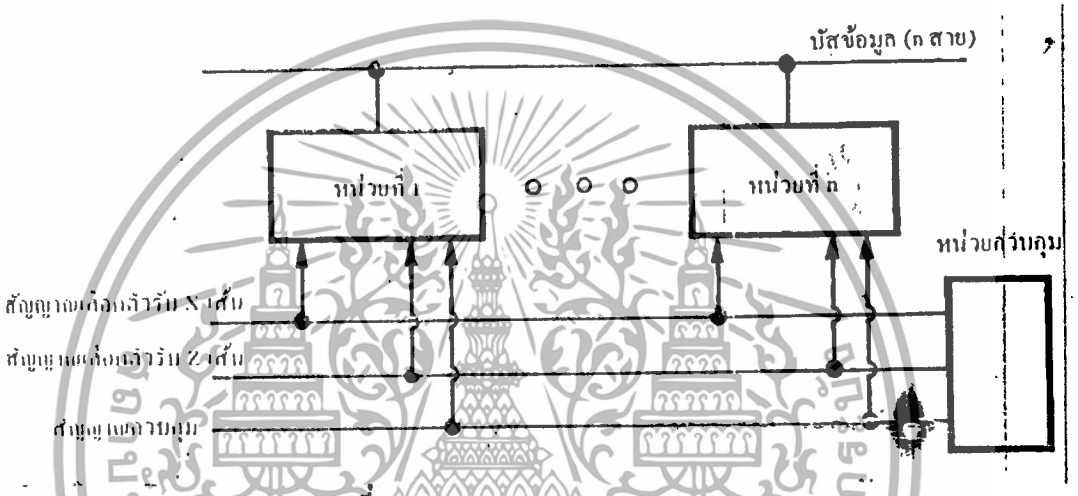
เนื่องจากบัสเป็นเส้นทางที่หลายหน่วยใช้ร่วมกันในการส่งและรับข้อมูล จึงต้องมีหน่วยควบคุมการใช้ในแต่ละเวลาโดยหน่วยควบคุมจะต้องกำหนดในเวลาหนึ่งว่าหน่วยไหนทำหน้าที่ส่งข้อมูล และหน่วยไหนรับข้อมูล ดังรูป 2.17



รูปที่ 2.17 แสดงบัสจำนวน n เส้น เชื่อมโยงระหว่างหน่วยที่ 1 ถึงหน่วยที่ m

ตามรูปที่ 2.17 หน่วยใด ๆ จะต่อสัญญาณออกจากหน่วยมายังบัส และต่อบัสเข้ากับจุดรับสัญญาณเข้าลักษณะของวงจรขับสัญญาณออกและรับสัญญาณเข้าต้องควบคุมได้โดยสัญญาณจากหน่วยควบคุม ไม่ให้สัญญาณจากบัสกลับมากวนภายในหน่วยที่ส่งขณะที่หน่วยกำลังส่งข้อมูลออก ถึงแม้จุดต่อสัญญาณออกกับเข้าเป็นจุดเดียวกันบนบัสก็ตาม หน่วยควบคุมจะอนุญาตให้หน่วยใดหน่วยหนึ่งเพียงหน่วยเดียวเท่านั้นที่ทำหน้าที่เป็นตัวส่งสัญญาณ ในขณะที่ตัวอื่นทั้งหมดเป็นตัวรับ (โดยรายละเอียดให้ไปอีกเล็กน้อย)

หน่วยควบคุมอาจส่งสัญญาณเลือกมาในลักษณะ bus ได้เช่นกัน โดย bus นี้เรียกว่า แอดเดรสบัส (address buses) ซึ่งต้องเชื่อมหน่วยต่าง ๆ ทุกหน่วยที่หน่วยควบคุมจะควบคุม โดยที่ตัวหน่วยที่ 1 ถึง n นั้นมีวงจรถอดระหัสจากแอดเดรสบัสเมื่อหน่วยควบคุมต้องการเลือกหน่วยใดหน่วยหนึ่งให้เป็นตัวส่งหน่วยควบคุม ต้องสั่งแอดเดรสออกมาทางแอดเดรสบัสพร้อมกับสัญญาณควบคุม หน่วยที่มีแอดเดรสตรงกับบัสข้อมูลจะทำหน้าที่เป็นตัวส่ง ตามรูปที่ 2.19

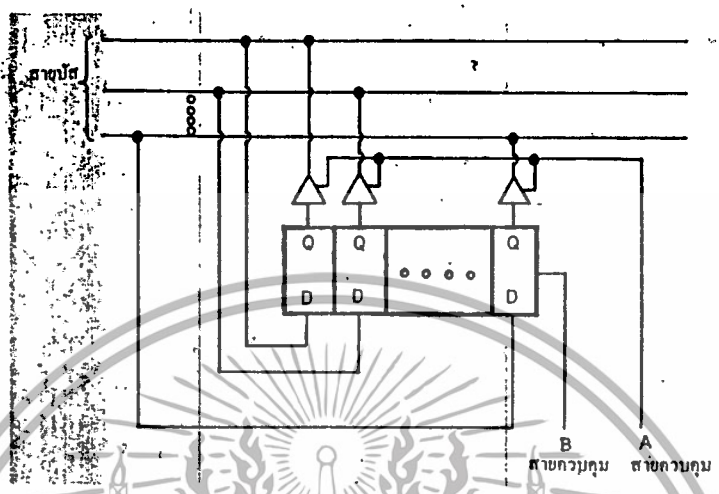


รูปที่ 2.19 แสดงแอดเดรสบัส

ซิงโครนัสและอะซิงโครนัสบัส (Synchronous and Asynchronous bus)

บัสที่เวลาในการส่งข้อมูลเกิดและดำเนินไปโดยอาศัยสัญญาณควบคุมที่มีระยะเวลาหรือกำหนดเวลาแน่นอน สัญญาณควบคุมอันนั้นคือ สัญญาณให้ส่งข้อมูลหรือรับข้อมูล (strobe Signal) โดยสัญญาณให้ส่งและรับจะส่งมาจากหน่วยควบคุมเป็นผู้กำหนดเวลามาให้ bus แบบนี้ เราเรียกว่า ซิงโครนัสบัส (Synchronous bus) ซิงโครนัสบัสเหมาะสมสำหรับระบบที่ไม่ยุ่งยากแต่ในกรณีที่ผู้ส่งกับผู้รับไม่ได้มีเพียงคู่เดียวแต่มีหลายหน่วย และความพร้อมเพรียงหรือความเร็วในการตอบรับต่อสัญญาณเลือกไม่เท่ากัน หน่วยควบคุมไม่สามารถจะประมาณหรือจะกำหนดเวลาที่แน่นอนว่าจะให้ส่งหรือรับเวลาใด การมีสัญญาณสโตรบควบคุมอย่างเดียวจะไม่เพียงพอ ในกรณีที่ผู้ส่งและผู้รับไม่สามารถจะกำหนดเวลาการส่งหรือความพร้อมของข้อมูลที่จะส่งให้แน่นอนได้ เพราะผู้ส่งหรือผู้รับอาจจะถูกบังคับด้วยสัญญาณอื่น ๆ จาก

จุดเชื่อมต่อกับบัสวงจรมักจะเป็นดังรูปที่ 2.18



รูปที่ 2.18 รายละเอียดหน่วยที่ 1 แสดงวงจรการรับและส่งและการควบคุม

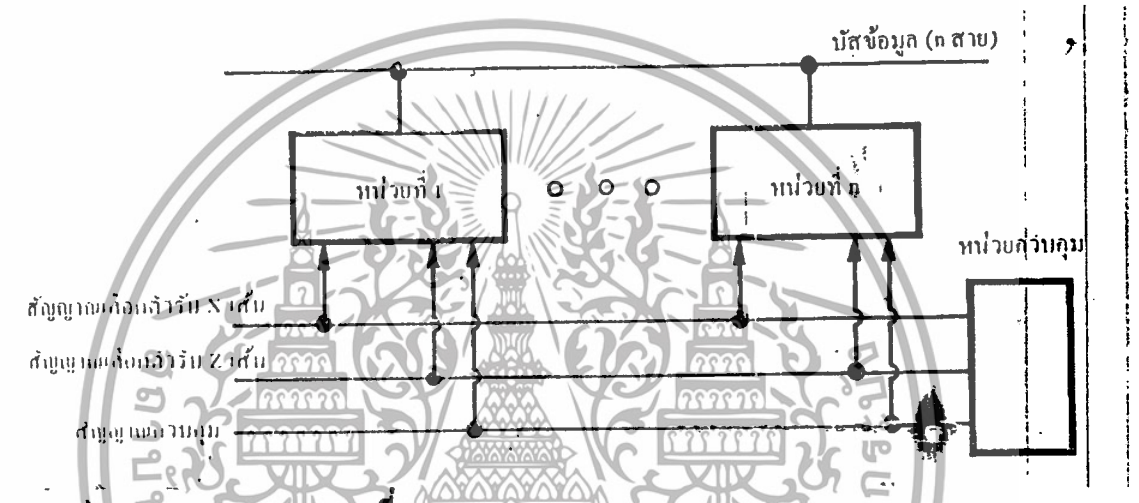
การติดต่อกันระหว่างหน่วยต่าง ๆ ผ่านทางบัสแบบนี้จะเป็นดังนี้ คือ สมมติหน่วยที่ 1 ส่งข้อมูลไปยังหน่วยที่ 5

(1) วงจรหน่วยควบคุมจะส่งสัญญาณควบคุมการปล่อยสัญญาณ (A) มายังหน่วยที่ 1 ข้อมูลจากรีจิสเตอร์ของ 1 จะไปออกทางบัส

(2) วงจรหน่วยควบคุมจะส่งสัญญาณควบคุมการรับข้อมูลเข้า (B) ไปยังหน่วยที่ 5 ทำให้ข้อมูลบนบัสถูกเก็บเข้าไปไว้ในหน่วยที่ 5 ตัวอย่างบัสแบบนี้ในระบบไมโครคอมพิวเตอร์คือ แอดเดรสบัส (address bus) และบัสข้อมูล (data bus) และสัญญาณควบคุมบางอัน

<u>แอดเดรสบัส</u> ในหัวข้อที่แล้วเรื่องของบัสข้อมูล ซึ่งเชื่อมโยงระหว่างหน่วยต่าง ๆ จะถูกบังคับให้ทำหน้าที่เป็นตัวรับหรือตัวส่งตามสัญญาณควบคุมจากหน่วยควบคุม จะเห็นว่าหน่วยควบคุมต้องเลือกหน่วยต่าง ๆ ว่าอันไหนและว่าจะทำหน้าที่อะไร การเลือกหน่วยใดหน่วยหนึ่งนี้

หน่วยควบคุมอาจส่งสัญญาณเลือกมาในลักษณะ bus ได้เช่นกัน โดย bus นี้เรียกว่า แอดเดรสบัส (address buses) ซึ่งต้องเชื่อมหน่วยต่าง ๆ ทุกหน่วยที่หน่วยควบคุมจะควบคุม โดยที่ตัวหน่วยที่ 1 ถึง n นั้นมีวงจรถอดรหัสจากแอดเดรสบัสเมื่อหน่วยควบคุมต้องการเลือกหน่วยใดหน่วยหนึ่งให้เป็นตัวส่งหน่วยควบคุม ต้องสั่งแอดเดรสออกมาทางแอดเดรสบัสพร้อมกับสัญญาณควบคุม หน่วยที่มีแอดเดรสตรงกับบัสข้อมูลจะทำหน้าที่เป็นตัวส่ง ตามรูปที่ 2.19

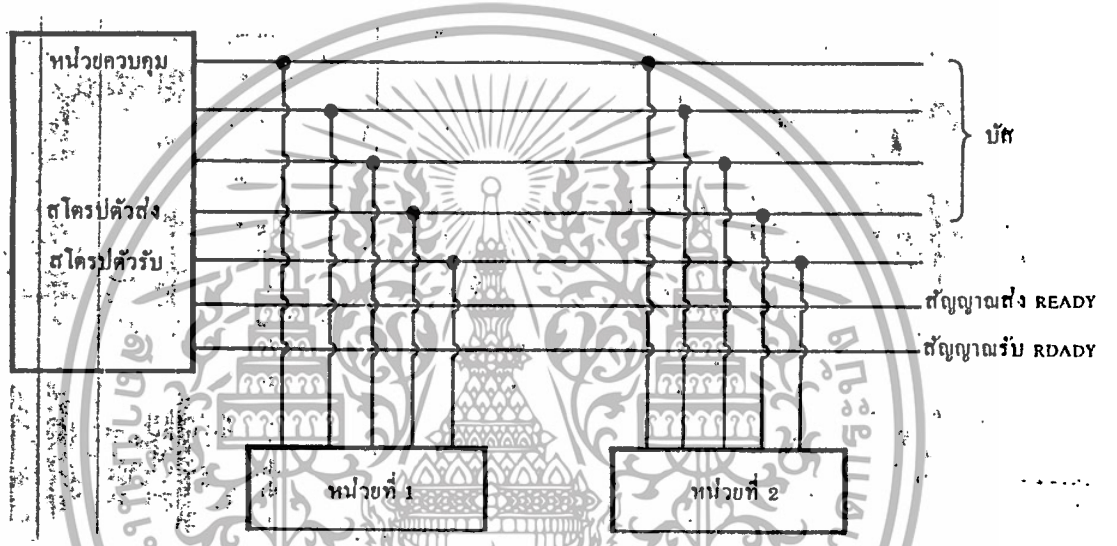


รูปที่ 2.19 แสดงแอดเดรสบัส

ซิงโครนัสและอะซิงโครนัสบัส (Synchronous and Asynchronous bus)

บัสที่เวลาในการส่งข้อมูลเกิดและดำเนินไปโดยอาศัยสัญญาณควบคุมที่มีระยะเวลาหรือกำหนดเวลาแน่นอน สัญญาณควบคุมอันนั้นคือ สัญญาณให้ส่งข้อมูลหรือรับข้อมูล (strobe Signal) โดยสัญญาณให้ส่งและรับจะส่งมาจากหน่วยควบคุมเป็นผู้กำหนดเวลามาให้ bus แบบนี้ เราเรียกว่า ซิงโครนัสบัส (Synchronous bus) ซิงโครนัสบัสเหมาะสมสำหรับระบบที่ไม่ยุ่งยากแต่ในกรณีที่ผู้ส่งกับผู้รับไม่ได้มีเพียงคู่เดียวแต่มีหลายหน่วย และความพร้อมเพรียงหรือความเร็วในการตอบรับต่อสัญญาณเลือกไม่เท่ากัน หน่วยควบคุมไม่สามารถจะประมาณหรือจะกำหนดเวลาที่แน่นอนว่าจะให้ส่งหรือรับเวลาใด การมีสัญญาณสโตรบควบคุมอย่างเดียวจะไม่เพียงพอ ในกรณีที่ผู้ส่งและผู้รับไม่สามารถจะกำหนดเวลาการส่งหรือความพร้อมของข้อมูลที่จะส่งให้แน่นอนได้ เพราะผู้ส่งหรือผู้รับอาจจะถูกบังคับด้วยสัญญาณอื่น ๆ จาก

ภายนอกหน่วย จำเป็นจะต้องมีสัญญาณเพิ่มเติมต่อระหว่างผู้ส่งและรับเพื่อแสดงสถานะของความพร้อมเหรียง เราเรียกสัญญาณที่เพิ่มเติมมานี้ว่า สายสเตตัส (Status line) หน่วยควบคุมไม่ใช่จะควบคุมสัญญาณสโตรบเท่านั้น แต่ยังต้องตรวจสอบสถานะหรือ Status ของผู้รับและผู้ส่งด้วยว่าพร้อมจึงจะกำหนดสโตรบออกไป บัสที่ทำงานในลักษณะเช่นนี้เราเรียกว่า อะซิงโครนัสบัส (Asynchronous bus) ตามรูปที่ 2.20



รูปที่ 2.20 แสดง Status line อีก 2 เส้น ในระบบอะซิงโครนัสบัส คือ Sender Ready และ Reciever Ready

ในกรณีของระบบในรูปที่ 2.20 เมื่อทำงานแบบ Synchronous สัญญาณแสดงสถานะอีกเส้น คือ Sebder Ready แสดงสถานะของผู้ถูกกำหนดให้เป็นผู้ส่ง และ Receiver Ready แสดงสถานะของผู้ถูกกำหนดให้เป็นผู้รับ หน่วยควบคุมจะส่งสัญญาณสโตรบ ซึ่งเป็นสัญญาณแสดงการกระทำการส่งหรือรับออกไปต่อเมื่อตัวส่งและตัวรับอยู่ในสถานะที่พร้อมที่จะส่งและพร้อมที่จะรับเท่านั้น การทำงานจะมีความสัมพันธ์กันดังต่อไปนี้คือ

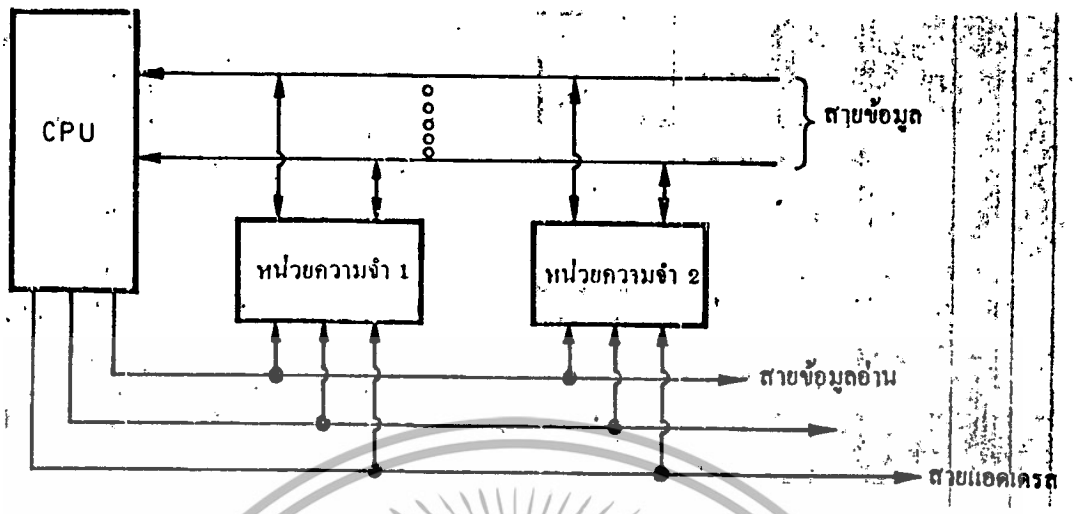
1. หน่วยควบคุมเลือกหรือกำหนดผู้ส่งและผู้รับ
2. รอจนสถานะของผู้รับและผู้ส่งพร้อม จึงส่งสัญญาณสโตรบ

ในรายละเอียดจะเป็นดังนี้ เมื่อผู้ส่งมีข้อมูลพร้อมจะส่งมันจะส่งสัญญาณ Sender Ready ถ้าผู้รับพร้อมจะรับมันจะต้องส่ง Receiver Ready เมื่อพร้อมทั้งคู่หน่วยควบคุมจะกระทำการส่งทันที โดยส่งสัญญาณ strobe ออกไป เมื่อผู้รับรับข้อมูลไปแล้วมันจะไม่ส่งสัญญาณ Recevier Ready ออกมา (เพราะมันอาจต้องเอาข้อมูลที่ได้ไปส่งต่อให้ระบบภายนอกอีกต่อก็ได้) ในขณะเดียวกัน ผู้ส่งก็จะไม่ส่ง Sender Ready ออกมาอีกต่อไป (เพราะมันอาจต้องเสียเวลาเตรียมข้อมูลที่จะส่งตัวต่อไปอีก) เมื่อผู้ส่งพร้อมจะส่งตัวต่อไปอีกมันจึงจะส่ง Sender Ready ออกมาอีกและรอจนกว่าจะมี Receiver Ready หน่วยควบคุมจึงจะกระทำการส่งข้อมูลการทำงานที่มีการตรวจสถานะระหว่างสองระบบที่ติดต่อกัน ลักษณะนี้เราเรียกว่า hand shaking ให้ดูตัวอย่างอะซิงโครนัสบัส ของ Z-80

ระบบบัสในไมโครคอมพิวเตอร์

ในระบบไมโครคอมพิวเตอร์ การส่งผ่านข้อมูลส่วนมากจะเป็นระหว่าง ตัวไมโครโปรเซสเซอร์กับหน่วยภายนอกทั้งหมด ผ่านบัส ไมโครคอมพิวเตอร์จะมีบัสต่าง ๆ ดังนี้ บัสข้อมูล แอดเดรสบัส บัสควบคุม

ไมโครโปรเซสเซอร์จะเป็นตัวควบคุมการส่งข้อมูลจากตัวมันเองไปยังหน่วยภายนอกหรือรับข้อมูลจากหน่วยภายนอก เพื่อเอามาประมวลภายในตัวมันผ่านบัสข้อมูล การเลือกว่าจะส่งหรือรับข้อมูลจากหน่วยไหน ไมโครโปรเซสเซอร์ต้องส่งสัญญาณเลือกออกมาทางแอดเดรสบัส พร้อมทั้งสัญญาณควบคุมทางบัสควบคุม เพื่อบังคับว่าจะอ่านข้อมูลเข้ามาหรือจะส่งข้อมูลออกไปจากตัวมัน ระบบภายนอกจะตอบรับต่อสัญญาณควบคุมนั้น แต่ก็ไม่ใช่ว่าไมโครโปรเซสเซอร์เท่านั้นที่เป็นตัวบังคับครองบัสทั้งหมด หน่วยภายนอกสามารถหยุดยั้งการใช้บัส ของไมโครโปรเซสเซอร์และดำเนินการรับส่งข้อมูลระหว่างหน่วยภายนอกเอง ผ่านบัสได้เป็นกรณีพิเศษเหมือนกัน เช่น ข้อมูลจากหน่วยความจำสำรองขนาดใหญ่ สามารถส่งมายังหน่วยความจำหลักได้โดยไม่ผ่านไมโครโปรเซสเซอร์เลยก็ได้ โดยใช้ขบวนการเรียกว่า ขบวนการ DMA (Direct Memory Access)



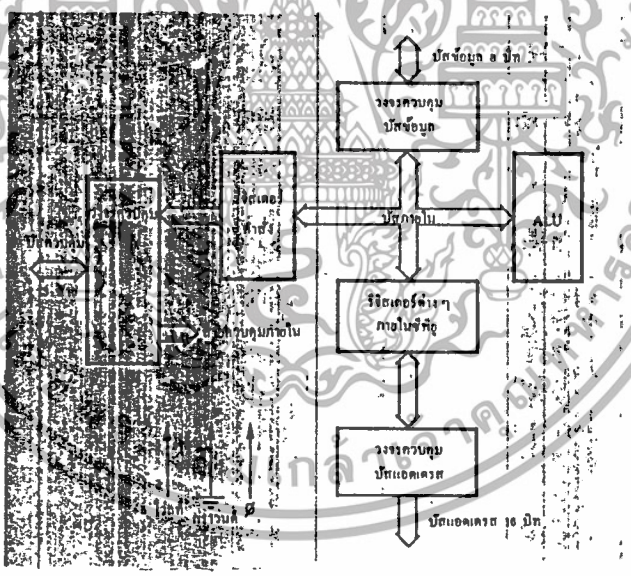
รูปที่ 2.21 การติดต่อของซีพียูกับระบบอื่น ๆ ผ่าน bidirectional bus (Data bus)

บทที่ 3

สถาปัตยกรรมของ Z-80

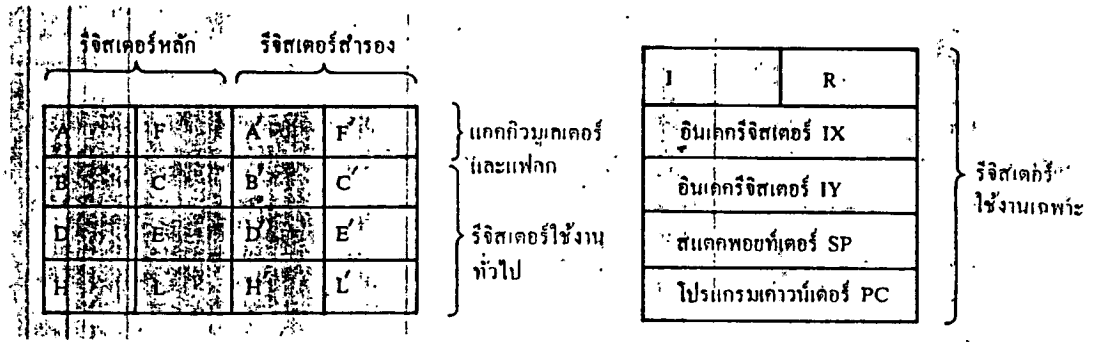
โครงสร้างของซีพียู Z-80

โครงสร้างของซีพียู Z-80 มีโครงสร้างที่พัฒนามาจาก 8080 ดังนั้นในแง่โครงสร้างพื้นฐานจะเหมือนกับซีพียูของ 8080 แต่เนื่องจาก Z-80 มีการพัฒนามามากขึ้นทางซอฟท์แวร์และฮาร์ดแวร์จึงทำให้มีรายละเอียดแตกต่างเพิ่มเติมอีกหลายประการด้วยกัน บล็อกไดอะแกรมรูปที่ 3.1 เป็นไดอะแกรมแสดงให้เห็นโครงสร้างของ Z-80 โดยโครงสร้างของซีพียูนี้จะบรรจุลงในแอลเอสไอขนาด 40 ขา



รูปที่ 3.1 บล็อกไดอะแกรมซีพียู Z-80

โครงสร้างภายในของ Z-80 ซีพียูประกอบด้วยรีจิสเตอร์ภายในที่สามารถเขียนและอ่านได้ถึง 208 บิท โดยแยกเป็นกลุ่มของรีจิสเตอร์ขนาด 8 บิท 18 รีจิสเตอร์ และรีจิสเตอร์ขนาด 16 บิท อีก 4 รีจิสเตอร์ โดยมีชุดรีจิสเตอร์แสดงได้ดังรูปที่ 3.2



รูปที่ 3.2 แสดงรีจิสเตอร์ต่าง ๆ ที่มีอยู่ใน Z-80

รีจิสเตอร์หลักที่ใช้งานทั่วไป

รีจิสเตอร์ในกลุ่มแรกคือ A,F,B,C,D,E,H,L เป็นรีจิสเตอร์ขนาด 8 บิท ที่ใช้งานทั่วไป โดยรีจิสเตอร์เหล่านี้สามารถประกอบรวมกันเป็นคู่รีจิสเตอร์ได้ คือ AF, BC, DE และ HL โดยคู่รีจิสเตอร์เหล่านี้จะได้รับการใช้งานในลักษณะของรีจิสเตอร์ขนาด 16 บิท การกระทำภายในซีพียูอาจจะอาศัยเพียงรีจิสเตอร์เดียวหรือกระทำเป็นคู่รีจิสเตอร์ได้ โดยที่ A คือ แอคคิวมูเลเตอร์ F คือ แฟลก แฟลกของ Z-80 จะมีด้วยกันทั้งหมด 6 ตัว จึงใช้เพียง 6 บิท แต่ Z-80 อาศัยการเพิ่มบิทขึ้นอีก 2 บิท และกลายเป็นรีจิสเตอร์ F รีจิสเตอร์ F นี้สามารถได้รับการเซท รีเซทการกระทำตามคำสั่งทางคณิตศาสตร์ หรือลอจิกได้ และเราสามารถใช้ F เหมือนรีจิสเตอร์หนึ่ง ซึ่งเมื่อรวมกันกับ A แล้ว จะกลายเป็นรีจิสเตอร์ขนาด 16 บิทได้

กลุ่มรีจิสเตอร์สำรอง

เป็นกลุ่มรีจิสเตอร์ที่สามารถเก็บข้อมูลได้ โดยเป็นตัวเก็บข้อมูลที่มาจากรีจิสเตอร์หลัก รีจิสเตอร์ชุดนี้จึงมีด้วยกัน 8 ตัว คือ A',F',B',C',D',E',H',L' รีจิสเตอร์เหล่านี้เป็นรีจิสเตอร์ที่ใช้ในการเก็บข้อมูลชั่วคราว ในการที่ต้องการใช้รีจิสเตอร์หลักทำงานอย่างอื่นก่อน ดังนั้นรีจิสเตอร์กลุ่มนี้จึงไม่สามารถกระทำทางคณิตศาสตร์และลอจิกได้

กลุ่มรีจิสเตอร์ที่ใช้งานเฉพาะอย่าง

โปรแกรมเคาว์เตอร์ (PC-Program counter) โปรแกรมเคาว์เตอร์เป็นรีจิสเตอร์ขนาด 16 บิท ที่เป็นตัวกำหนดตำแหน่งของโปรแกรมในขณะสภาวะการกระทำการเฟทช์ โดยขณะทำการเฟทช์ค่าที่อยู่ในโปรแกรมเคาว์เตอร์จะไปปรากฏอยู่ที่แอดเดรสบัส เพื่อชี้ไปยังตำแหน่งในหน่วยความจำให้ซีพียูอ่านคำสั่งมาตีความหมาย ค่าที่อยู่ในโปรแกรมเคาว์เตอร์จะเพิ่มค่าขึ้นได้อย่างอัตโนมัติหลังการกระทำการเฟทช์ แต่ถ้าหากซีพียูกระทำคำสั่งให้ข้ามไปยังตำแหน่งอื่น (Jump) ค่าแอดเดรสที่จะกระโดดข้ามนั้นจะโหลดเข้ามายังโปรแกรมเคาว์เตอร์ได้อย่างอัตโนมัติ

สแตคพอยท์เตอร์ (SP=Stack pointer) เป็นรีจิสเตอร์ที่มีขนาด 16 บิทที่ใช้สำหรับชี้ไปยังแอดเดรสชั้นบนสุดของสแตคที่อยู่ใน RAM โดยส่วนของสแตคมีลักษณะโครงสร้างเป็นหน่วยความจำเป็นแบบเก็บที่หลังเรียกออกได้ก่อน ข้อมูลในสแตคอาจได้รับการพุชหรือพอพมาจากข้อมูลรีจิสเตอร์ภายในซีพียู ลักษณะของสแตคในที่นี้ยังเป็นส่วนช่วยในการกระทำอินเตอร์รัพท์ และการเรียกโปรแกรมย่อย กล่าวคือในการอินเตอร์รัพท์ค่าของโปรแกรมเคาว์เตอร์จะได้รับการเก็บรักษาไว้ในชั้นสแตค ครั้นเมื่อโปรแกรมกลับจากอินเตอร์รัพท์ไปกระทำยังโปรแกรมหลักก็จะนำค่าจากสแตคกลับเข้ามายังโปรแกรมเคาว์เตอร์ใหม่ ในทำนองเดียวกัน การกระโดดไปกระทำยังโปรแกรมย่อย ก็เช่นเดียวกัน ดังนั้นการกระทำในรูปของอินเตอร์รัพท์หรือโปรแกรมย่อย สามารถซ้อนกันได้ไม่มีสิ้นสุด

อินเดครีจิสเตอร์ (IX,IY-index register) ซีพียู Z-80 มีอินเดครีจิสเตอร์ขนาน 16 บิท 2 ตัว แต่ละตัวใช้ประโยชน์หลักในการทำหน้าที่เป็นตัวเก็บแอดเดรสฐาน (base address) เพื่อทำหน้าที่อ้างแอดเดรสแบบอินเดคแอดเดรสซิ่ง (index addressing) ในโหมดของอินเดคแอดเดรสซิ่งมีข้อมูลที่อยู่ในอินเดรีจิสเตอร์นี้จะรวมกับข้อมูลที่ติดมากับคำสั่งอีก 8 บิท เพื่อเป็นตัวกำหนดแอดเดรสให้กับคำสั่งข้อมูลที่ติดมากับคำสั่งนี้เราเรียกว่า ดิสเพลสเมนท์ (displacement) ซึ่งจะเก็บในรูปของตัวเลข คอมพลีเมนท์

อินเตอร์รัพท์เพจแอดเดรสรีจิสเตอร์ (I-Interrupt page address register) การอินเตอร์รัพท์ของ Z-80 มีหลายโหมด และโหมดหนึ่งที่ทำให้การอินเตอร์

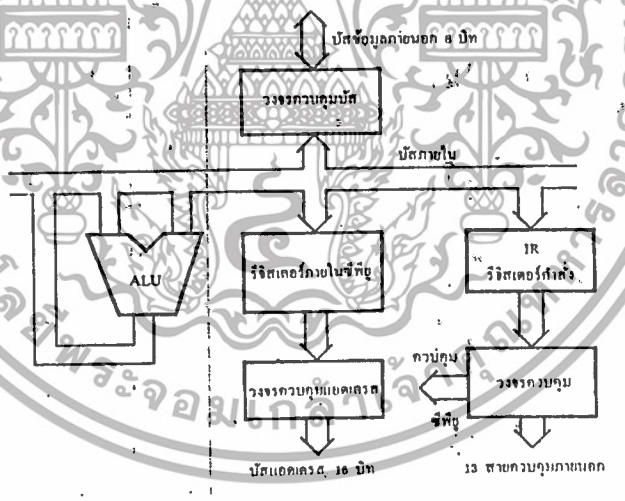
รัพท์ของ Z-80 มีหลายโหมด และโหมดหนึ่งที่ทำให้การอินเตอร์รัพท์ของ Z-80 มีประสิทธิภาพสูง กล่าวคือ เมื่อเกิดการอินเตอร์รัพท์ในโหมดนี้ขึ้นมันสามารถอ้างแอดเดรสโดยทางอ้อมไปกระทำโปรแกรมในที่ใดก็ได้ในหน่วยความจำ โดยอาศัยค่าในรีจิสเตอร์ I รวมกับค่าที่ส่งมาจากอุปกรณ์เพอริเฟอรัลอีก 8 บิท ชี้ไปยังค่าในหน่วยความจำ เพื่อนำค่านั้นมาโหลดเข้าในโปรแกรมเคาว์เตอร์เพื่อกระทำต่อไป ด้วยวิธีการนี้เราจึงสามารถรกระโดดเข้าไปทำที่ส่วนใดก็ได้ในหน่วยความจำ

รีจิสเตอร์รีเฟรชหน่วยความจำ (R-memory reflesh register) การต่อซีพียูกับหน่วยความจำนั้น โดยปกติจะต่อกับหน่วยความจำชนิดสแตติคได้โดยง่าย แต่อย่างไรก็ดีชนิดไดนามิคที่ทองการรีเฟรชมีราคาถูกกว่า มีความหนาแน่นสูงกว่า Z-80 ให้ข้อดีกว่าประการหนึ่งคือ มันสามารถให้การรีเฟรช หน่วยความจำได้อย่างอัตโนมัติ โดยค่าใน R รีจิสเตอร์จะเพิ่มค่าขึ้นอีก 1 ทุกครั้งที่มีการกระทำการเฟทช์คำสั่ง และข้อมูลในรีจิสเตอร์ R นี้จะส่งออกไปยังแอดเดรสบัสในส่วนบิทที่มีนัยสำคัญต่ำกว่าจังหวะของการส่งนี้จะเป็นจังหวะเดียวกันกับที่ซีพียูส่งสัญญาณรีเฟรชออกมา ผู้โปรแกรมสามารถกำหนดค่าให้กับรีจิสเตอร์ R นี้ได้ แต่ค่าในรีจิสเตอร์นี้จะเรียกใช้โดยผู้โปรแกรมทางคำสั่งโดยตรงไม่ได้

แอคคิวมูเลเตอร์ (accumulator) และแฟลก (flage) ซีพียูจะมีรีจิสเตอร์ที่ใช้เป็นหลักในการเป็นตัวโอเปอร์แรนด์สำหรับกระทำทางคณิตศาสตร์และลอจิก โดยรีจิสเตอร์หลักนี้จะมีเพียง 8 บิท เรียกว่า "แอคคิวมูเลเตอร์ (accumulator) การกระทำในส่วนของหน่วยคณิตศาสตร์และลอกจิกย่อมเกิดเงื่อนไขได้หลายอย่างที่จะต้องแสดงสถานะภาพของเงื่อนไขเหล่านั้น เช่น เงื่อนไขผลลัพธ์เป็นศูนย์ ผลลัพธ์เป็นบวกหรือลบ มีตัวทดหรือตัวขอยืมในการกระทำทางคณิตศาสตร์ แสดงเงื่อนไขพาริตี้คู่หรือคี่ ฯลฯ สิ่งเหล่านี้จะให้ผลลัพธ์แสดงสถานะได้ด้วยแฟลก (flag) แฟลกเป็นรีจิสเตอร์ขนาด 8 บิท ซึ่งสามารถรวมกับแอคคิวมูเลเตอร์เป็นรีจิสเตอร์ขนาด 16 บิทได้ ผู้โปรแกรมยังสามารถใช้คำสั่งในการเคลื่อนย้ายข้อมูลจากแอคคิวมูเลเตอร์ A และแฟลก F ไปเก็บไว้ใน A' และ F' ได้ เพื่อทำให้การใช้งานของ A และ F มีประสิทธิภาพดียิ่งขึ้น

หน่วยคำนวณทางคณิตศาสตร์และลอจิก (ALU-arithmetic and logic unit) การประมวลผลที่สำคัญของซีพียูของคอมพิวเตอร์ยังขึ้นอยู่กับหน่วยคำนวณทางคณิตศาสตร์และลอจิก (ALU) ส่วน ALU นี้จะนำข้อมูลซึ่งอาจจะมาจากภายนอกซีพียูหรือภายในซีพียูก็ได้มาประมวลผล การประมวลผลในส่วน ALU ที่สำคัญจะประกอบด้วย

การบวก (add)	การเลื่อนบิททางซ้ายหรือขวา
การลบ (substract)	การเพิ่มค่า (increment)
ลอจิก AND	การลดค่า (decrement)
ลอจิก OR	การเซทบิท (set bit)
ลอจิก EX-OR	การรีเซทบิท (reset bit)
เปรียบเทียบ (compare)	การทดสอบบิท (test bit)

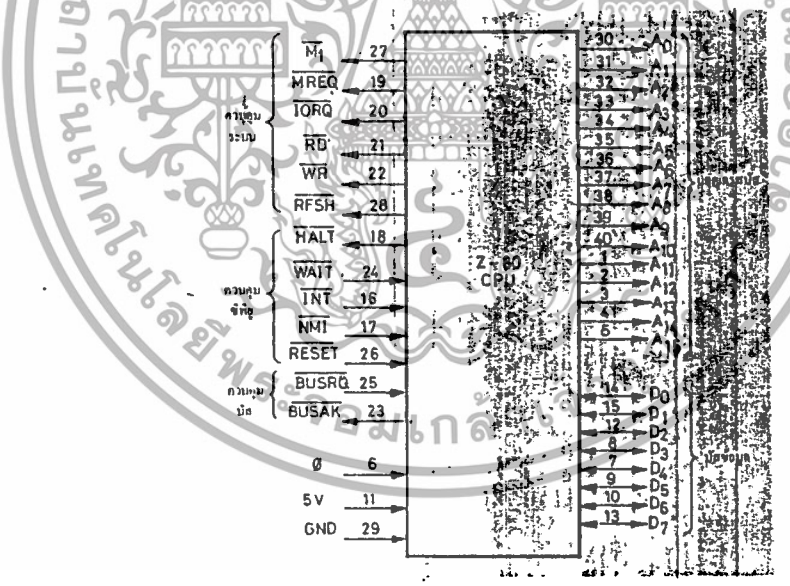


รูปที่ 3.3 แสดงการทำงานของ ALU ภายใน

<u>รีจิสเตอร์คำสั่งและส่วนควบคุม</u> (Instruction register and control) ในกระบวนการทำการเฟทช์ ซีพียูจะอ่านคำสั่งจากหน่วยความจำที่เป็นส่วนของโปรแกรมโดยรอคำสั่งนั้น มาเก็บไว้ใน IR เพื่อทำการถอดรหัสคำสั่งและส่งสัญญาณควบคุมการทำงานภายในซีพียู หรือควบคุมการทำงานของระบบสัญญาณควบคุมเหล่านี้จะออกมาในจังหวะต่าง ๆ กัน เพื่อใช้ควบคุมระบบในการทำงานต่อไป

<u>การจัดขาของ Z-80</u>

Z-80 ซีพียูเป็นไอซีไมโครโปรเซสเซอร์ที่มีขาเพียง 40 ขา โดยหลักการแล้ว Z-80 เป็นซีพียูได้โดยสมบูรณ์ กล่าวคือ Z-80 ไม่ต้องประกอบกับอุปกรณ์ประกอบอื่นที่จะแยกการทำงานเพื่อรวมเป็น ซีพียู ส่วนของสัญญาณจะประกอบด้วยบัสแอดเดรส บัสข้อมูล และสัญญาณควบคุม การจัดวางขาแสดงได้ดังรูปที่ 3.4



รูปที่ 3.4 ลักษณะของขาไอซี Z-80 ซีพียู

รายละเอียดของขาต่าง ๆ แสดงได้ดังนี้

A_0-A_{15} บัสแอดเดรส สัญญาณที่ออกมาจากขาไอซีเหล่านี้จะให้แอคทีฟขณะ high โดยขาเหล่านี้เป็นเอาท์พุทแบบไตรสเทท บัสแอดเดรสมีด้วยกันทั้งหมด 16 สาย เพื่อให้ซีพียูติดต่อกับหน่วยความจำได้มากถึง 2^{16} = 64K ไบท์ นอกจากนี้ส่วนของแอดเดรสยังเป็นตัวกำหนดเบอร์พอร์ทของอุปกรณ์อินพุท-เอาท์พุท โดยขณะที่ซีพียูกระทำคำสั่งเกี่ยวกับอินพุทหรือเอาท์พุท ค่าของแอดเดรสบัสใน 8 บิทล่าง (A_0-A_7) จะแสดงค่าเบอร์พอร์ท ดังนั้นเราจึงมีอุปกรณ์อินพุทหรือเอาท์พุทได้ทั้งหมด 2^8 = 256 พอร์ท และในขณะช่วงเวลารีเฟรช โดยเมื่อสัญญาณรีเฟรชปรากฏขึ้นที่ขารีเฟรช ($\overline{RFSH}$) ค่าในแอดเดรสบัส A_0-A_7 จะแสดงค่าแอดเดรสของหน่วยความจำที่จะได้รับการกระทำการรีเฟรช

D_0-D_7 บัสข้อมูล (data bus) เป็นลักษณะบัสแบบสองทิศทาง Z-80 ซีพียูมีบัสข้อมูล 8 เส้น บัสข้อมูลเป็นเส้นทางผ่านของข้อมูลระหว่างซีพียูกับหน่วยความจำ ซีพียูกับอุปกรณ์อินพุท-เอาท์พุท หรือการติดต่อระหว่างอุปกรณ์อินพุท-เอาท์พุทกับหน่วยความจำ

$\overline{M}_1$ (machine cycle one) มีลักษณะเป็นแอคทีฟที่ลอจิก "0" $\overline{M}_1$ เป็นส่วนที่จะบอกให้ทราบว่าขณะนี้ซีพียูกำลังอยู่ในสภาวะเฟทช์ ในขณะที่ซีพียูเฟทช์คำสั่งที่มีออพโค้ดสองไบท์ ส่วนของ $\overline{M}_1$ จะสร้างขึ้นขณะเฟทช์ในแต่ละไบท์ ลักษณะของคำสั่งที่มีออพโค้ด สองไบท์จะขึ้นต้นด้วย CBH, DDH, EDH, FDH (H ต่อท้ายหมายถึง ตัวเลขฐานสิบหก) นอกจากนี้ $\overline{M}_1$ ยังสร้างสัญญาณร่วมกับ $\overline{IORQ}$ เพื่อบอกสถานะการตอบรับการอินเตอร์รัพท์

$\overline{MREQ}$ (memory request) เป็นลักษณะไตรสเทท ให้ลอจิกแอคทีฟที่ "0" เป็นสายสัญญาณที่บอกให้ทราบว่า ซีพียูต้องการเขียนหรืออ่านหน่วยความจำตามแอดเดรสที่ปรากฏอยู่ในแอดเดรสบัส

$\overline{IORQ}$ (input output request) เป็นเอาท์พุทลักษณะไตรสเตทให้ลอจิกแอคตีฟที่ "0" เป็นสัญญาณที่บอกให้ทราบว่า ซีพียูต้องการติดต่อกับอุปกรณ์อินพุท-เอาท์พุท โดยแอดเดรสบัส 8 บิทล่างจะให้แสดงค่าเบอร์พอร์ท ส่วนบัสข้อมูลจะแสดงข้อมูลที่จะมีการส่งถ่ายระหว่างซีพียู กับ I/O นอกจากนี้ $\overline{IORQ}$ ถ้าเกิดขึ้นพร้อมกับสัญญาณ $\overline{M_1}$ เป็นตัวบอกถึงสถานที่ซีพียูกำลังตอบสนองผลการอินเตอร์รัพท์ โดยขณะนี้ส่วนของบัสข้อมูลจะมีการส่งผ่านเข้ามาด้วยค่าของอินเตอร์รัพท์เวคเตอร์

$\overline{RD}$ (memory read) เป็นเอาท์พุทที่ไตรสเตทและแอคตีฟขณะลอจิก "0" $\overline{RD}$ เป็นตัวบอกว่า ขณะนี้ซีพียูต้องการอ่านข้อมูลจากหน่วยความจำหรืออุปกรณ์

$\overline{WR}$ (memory write) เป็นขาเอาท์พุทแบบไตรสเตท และแอคตีฟขณะลอจิก "0" $\overline{WR}$ เป็นสัญญาณบอกว่าซีพียูต้องการเขียนข้อมูลโดยจะเขียนข้อมูลในตำแหน่งที่แอดเดรสบัสกำหนดขึ้นอาจเป็นหน่วยความจำหรืออุปกรณ์ I/O ก็ได้

$\overline{RFSH}$ (reffesh) เป็นขาเอาท์พุทแอคตีฟขณะลอจิก "0" $\overline{RFSH}$ เป็นสัญญาณที่จะบอกให้ทราบว่าสัญญาณในแอดเดรสบัส ในส่วน A_0-A_6 เป็นแอดเดรสที่จะใช้ในการรีเฟรชหน่วยความจำชนิดไดนามิกส์ ส่วนบิท A_7 จะเป็น "0" ส่วนบิท A_{15}-A_8 จะแสดงค่าของรีจิสเตอร์ I

$\overline{HALT}$ (halt state) เป็นเอาท์พุทที่แอคตีฟด้วยลอจิก "0" สัญญาณ $\overline{HALT}$ จะแสดงเมื่อซีพียูได้กระทำคำสั่ง HALT และจะหยุดรอจนกว่าจะมีการอินเตอร์รัพท์หรือรีเซท ขณะที่อยู่ในช่วง HALT ซีพียูจะเสมือนกำลังกระทำคำสั่ง NOP (no operation) เพื่อให้เกิดไซเคิลในการทำงานเพื่อส่งสัญญาณไปกระทำการรีเฟรชหน่วยความจำชนิดไดนามิกส์

$\overline{WAIT}$ (wait) เป็นขาอินพุท จะแอคตีฟด้วยลอจิก "0" $\overline{WAIT}$ เป็นตัวกำหนดแสดงเพื่อบอกซีพียู ให้ซีพียูหยุดรอ ในกรณีที่อุปกรณ์อินพุท-เอาท์พุท หรือหน่วยความจำไม่สามารถรับหรือส่งข้อมูลได้ทัน $\overline{WAIT}$ จะเป็นตัวทำให้ซีพียู ซิงค์ได้พอดีกับอุปกรณ์อินพุทเอาท์พุทที่ทำงานด้วยความเร็วช้า ๆ

$\overline{INT}$ (interrupt request) เป็นขาอินพุท แอคตีฟด้วยลอจิก "0" $\overline{INT}$ เป็นสัญญาณที่สร้างขึ้นมาจากอุปกรณ์อินพุทเอาท์พุท เพื่อต้องการที่จะอินเตอร์รัพท์ซีพียู ซีพียูจะทำการตรวจสอบสัญญาณนี้ทุก ๆ ครั้งที่จบการกระทำแต่ละคำสั่ง การตอบสนองของตัวการอินเตอร์รัพท์ สามารถควบคุมได้ด้วยซอฟท์แวร์ ด้วยการเซทค่าอินเตอร์รัพท์ฟลิบฟลอบ (IFF) การตอบสนองอินเตอร์รัพท์จะเกิดได้ยังต้องให้ $\overline{BUSRQ}$ ไม่แอคตีฟ เมื่อซีพียูตอบสนองต่อการอินเตอร์รัพท์ ซีพียูจะสร้างสัญญาณด้วยการสร้างสัญญาณ $\overline{IORQ}$ ระหว่างช่วงเวลา $\overline{M_1}$ การตอบสนองต่อการอินเตอร์รัพท์มีแยกแยะได้ 3 แบบ ซึ่งจะอธิบายในรายละเอียดต่อไป

$\overline{NMI}$ (nonmaskable interrupt) เป็นขาอินพุท ที่จะทริกบอกซีพียูในขณะขอบพัลซ์ขาลง การอินเตอร์รัพท์ด้วยวิธีนี้ ซีพียูจะให้ความสำคัญสูงกว่า $\overline{INT}$ กล่าวคือมันจะตอบสนองและกระทำทันทีด้วยการเริ่มเอ็กซีคิว คำสั่งในตำแหน่ง 0066H โดยอัตโนมัติ การกระโดดไปกระทำในกรณีนี้ ซีพียูจะเก็บค่าโปรแกรมเคาน์เตอร์เดิมไว้ในสแตค เพื่อจะได้กลับไปทำงานเดิม เมื่อเสร็จสิ้นการอินเตอร์รัพท์ได้

$\overline{RESET}$ (reset) เป็นขาอินพุทที่แอคตีฟด้วยลอจิก "0" การรีเซทในกรณีนี้จะมีผลดังนี้

1. ค่าของ PC มีค่าเป็น "0"
2. IFF จะได้รับการ Disable
3. รีจิสเตอร์ I จะมีค่า 00H
4. รีจิสเตอร์ R จะมีค่า 00H
5. จะมีการเซทอินเตอร์รัพท์โหมดมาอยู่ที่โหมด 0

ระหว่างการรีเซทสายแอดเดรสบัสและบัสข้อมูลจะได้รับการกระทำให้มีค่าอินพีแดนซ์สูง เพื่อแยกออกจากซีพียูส่วนสายสัญญาณควบคุมจะได้รับการทำให้เป็นสัญญาณที่ไม่แอคตีฟ การรีเฟรชจะไม่เกิดขึ้น

$\overline{BUSRQ}$ (bus request) เป็นขาอินพุทที่แอคตีฟด้วยลอจิก "0" $\overline{BUSRQ}$ เป็นสัญญาณที่ส่งบอกกับซีพียู เพื่อต้องการให้ซีพียูควบคุมบัส กล่าวคือต้องการให้ซีพียูทำให้บัสแอดเดรสและบัสข้อมูลอยู่ในสถานะอิมพีแดนซ์สูง คือต้องการแยกซีพียูออกจากบัสนั่นเอง

$\overline{\text{BUSAK}}$ (bus acknowledge) เป็นขาเอาท์พุท แอคตีฟชณะลอจิก "0" $\overline{\text{BUSAK}}$ เป็นสัญญาณตอบจากซีพียูว่าซีพียูได้แยกตัวเองออกจากแอดเดรสบัสและบัสข้อมูลเรียบร้อยแล้ว

ϕ (clock) สัญญาณนาฬิกาที่จะป้อนเข้าระบบ

ไดอะแกรมเวลาของซีพียู

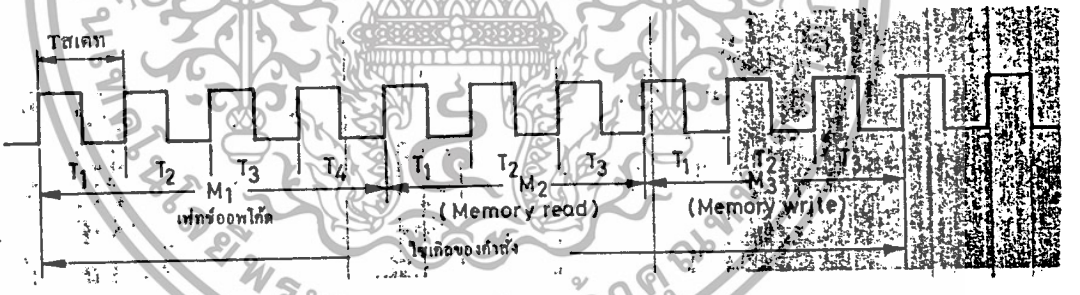
Z-80 ซีพียูจะทำงานในลักษณะพื้นฐาน ที่สำคัญประกอบด้วย

การเขียน-อ่านหน่วยความจำ

การเขียน-อ่านอุปกรณ์อินพุท-เอาท์พุท

การตอบสนองต่อการรีเฟรชรีเจ็ท

ลักษณะการทำงานจะสัมพันธ์กับสัญญาณนาฬิกา ขอให้พิจารณาจากรูปที่ 3.5



รูปที่ 3.5

จากที่กล่าวมาแล้วในการทำงานของไมโครคอมพิวเตอร์ในแต่ละไซเคิลของคำสั่งประกอบด้วยส่วนการทำงานสองจังหวะที่สำคัญคือ สภาวะเฟทช์ (Fetch) และสภาวะเอ็กซีคิว ส่วนของสภาวะเฟทช์จะประกอบด้วยสเตทย่อย ๆ หลายสเตท แต่ละสเตทจะซิงโครไนซ์กับสัญญาณนาฬิกา

ไซเคิลการทำงานที่แสดงในรูปที่ 3.5 จะเห็นได้ว่าในหนึ่งไซเคิลของคำสั่งจะประกอบด้วยหลายแมชีนไซเคิล (M ไซเคิล) กล่าวคือ M_1 M_2 M_3.....M_5 นั่นเอง ในกรณีของ Z-80 จำนวนแมชีนไซเคิลของแต่ละคำสั่งไม่เท่ากัน มากบ้างน้อยบ้าง ที่น้อยที่สุดใช้ 1 แมชีนไซเคิล ส่วนที่มากที่สุดใช้ 5 แมชีนไซเคิล คำสั่งที่มีสภาวะการทำงานแต่เพียงภายในซีพียู โดยเมื่อเฟทช์คำสั่งจากหน่วยความจำมาถอดความแล้ว ก็สามารถทำได้เลย เช่น LDr_1,r_2 เป็นต้นจะใช้เพียง 1 แมชีนไซเคิล

เนื่องจากในแต่ละคำสั่งจะประกอบด้วยแมชีนไซเคิลหลายแบบด้วยกัน แต่แมชีนไซเคิลที่ใช้ทั้งหมดสามารถสรุปออกมาเป็นแบบอย่างได้เป็น

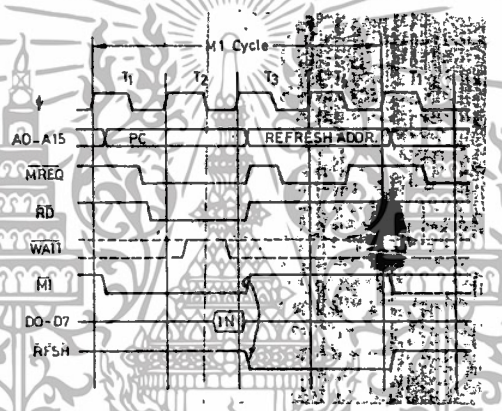
1. ไซเคิลการเฟทช์ (M_1)
2. ไซเคิลการเขียนหรืออ่านหน่วยความจำ
3. ไซเคิลการรับส่งข้อมูลอินพุทหรือเอาท์พุท
4. ไซเคิลการต้องการใช้บัส หรือการตอบสนองการใช้บัส
5. ไซเคิลการอินเตอร์รัพท์ และการตอบสนองต่อการอินเตอร์รัพท์
6. ไซเคิลการรับอินเตอร์รัพท์แบบนอนมาสค์เคเบิ้ล
7. ไซเคิลการออกจากคำสั่ง (HALT)

การทำงานในแต่ละไซเคิลจะตอบสนองได้ตามลักษณะของคำสั่งที่กระทำนั่นเอง คราวนี้ลองมาพิจารณารายละเอียดของแต่ละไซเคิลดูรายละเอียดต่อไป

<u>ไซเคิลการเฟทช์ออพโค้ด</u> (OCF-opcode fetch)

ในสภาวะ M_1 นี้ เป็นภาวะเริ่มแรกของทุกคำสั่ง ในสภาวะ M_1 นี้ ขอให้สังเกตจากระบบไดอะแกรมเวลาจะเห็นสภาวะการทำงานของซีพียูได้ดี กล่าวคือ เมื่อเริ่มเข้าสู่สภาวะเฟทช์ ข้อมูลจากโปรแกรมเคาน์เตอร์จะถ่ายลงไปยังแอดเดรสบัสและหลังจากครึ่งไซเคิลของสัญญาณนาฬิกา ซีพียูก็จะส่งสัญญาณ $\overline{MREQ}$ ซึ่งเป็นสัญญาณบอกว่าซีพียูต้องการติดต่อกับหน่วยความจำ และขณะเดียวกันก็ส่งสัญญาณ $\overline{RD}$ บอกหน่วยความจำว่าต้องการหรืออ่านข้อมูลในหน่วยความจำตามแอดเดรสที่ต้องการ ครั้นเมื่อถึงขาลงของ T_2 ซีพียูจะ

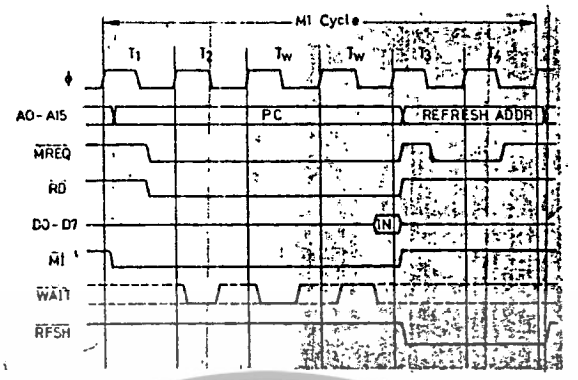
ตรวจสอบสัญญาณที่ทางขา $\overline{WAIT}$ ว่าเป็น "0" หรือไม่ ถ้าไม่ก็จะทำงานต่อแบบปกติ แต่ถ้าขา $\overline{WAIT}$ เป็น "0" ซีพียูจะเพิ่มสภาวะการรอ (WAIT state) เกิดขึ้น การอ่านข้อมูลจากหน่วยความจำในที่นี้ ถ้านำเอาสัญญาณ $\overline{MREQ}$ และ $\overline{RD}$ เป็นตัวอ่าน ข้อมูลที่บัสข้อมูลก็จะมีข้อมูลที่อ่านได้ในขณะที่ $\overline{MERQ}$ OR $\overline{RD}$ เป็น "0" ทั้งคู่ แต่ซีพียูจะรับการเปิดบัสนี้เข้าใน IR เพียงชั่วขณะสั้นเท่านั้น (ดูรูป) เมื่อข้อมูลจากบัสเข้า IR เรียบร้อย แล้วก็จบสภาวะการเฟทช์ สัญญาณที่ขา $\overline{M}_1$ ก็จะเป็น "1" บอกว่าเฟทช์เสร็จแล้ว แต่อย่างไรก็ตามมีสภาวะ $\overline{M1}$ ไซเคิลยังไม่หมดและซีพียูจะส่งสัญญาณทำการรีเฟรชหน่วยความจำไดนามิกส์นั่นคือ ในแอดเดรสบัสจะมีข้อมูลของแอดเดรสที่ต้องการรีเฟรชและขา $\overline{RFSH}$ ก็จะบอกสภาวะว่าต้องการรีเฟรชนั่นเอง



รูปที่ 3.6 ไซเคิลการเฟทช์ (M_1 Cycle)

M_1 ไซเคิลที่มี WAIT STATES

รูปที่ 3.7 เป็นสภาวะของ M_1 ไซเคิลที่มีการ $\overline{WAIT}$ กล่าวคือทุก ๆ ขาลงของ T_2 จะมีการตรวจสอบที่ขา $\overline{WAIT}$ ว่าเป็น "0" หรือไม่ สมมุติว่าเป็น "0" ซีพียูจะสร้างสเตทการ $\overline{WAIT}$ อีก 1 ไซเคิล โดยที่ค่าสัญญาณอื่น ๆ ไว้คงเดิม และทุก ๆ ขาลงของของ T_w ก็จะตรวจสอบขา $\overline{WAIT}$ อีกเช่นกัน ถ้าขา $\overline{WAIT}$ เป็น "0" ก็จะสร้าง T_w ต่อไปอีกและเป็นเช่นนี้เรื่อยไปไม่สิ้นสุด ในขณะที่ซีพียูสร้างสัญญาณ T_w สัญญาณขาต่าง ๆ จะยังคงค่าเดิมไว้หมด สัญญาณต่าง ๆ เหล่านี้ได้แก่ A_0-A_{15}, $\overline{MREQ}$, $\overline{RD}$, $\overline{M}_1$ และ $\overline{RFSH}$ ครั้นเมื่อขาลงของ T_w ซีพียูตรวจสอบสัญญาณ $\overline{WAIT}$ พบลอจิก "1" มันจะเริ่มการอ่านข้อมูลจากบัสข้อมูลและต่อท้ายด้วย T_3 และ T_4 ต่อไป

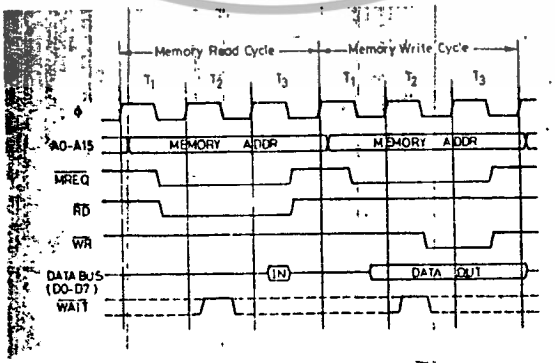


รูปที่ 3.7 ไซเคิลการเฟทช์ที่มี WAIT

ประโยชน์ของการใช้ T_w ก็ด้วยเหตุผลในการยืดความกว้างของสัญญาณต่าง ๆ เพื่อซิงโครนัสหรือให้ทำงานร่วมกับอุปกรณ์ภายนอกที่ความเร็วช้าได้ เช่น ถ้าต้องการอ่านหน่วยความจำที่มีความเร็วต่ำ เราจำเป็นต้องยืดสัญญาณ $\overline{RD}$ และ $\overline{MERQ}$ ออกเพื่อให้ส่งไปควบคุมหน่วยความจำได้ เราก็สร้างสัญญาณ $\overline{WAIT}$ ซีพียูนั่นเอง

ไซเคิลการอ่านหน่วยความจำ (MR-memory read cycle และไซเคิลการเขียนหน่วยความจำ (MW-memory write cycle)

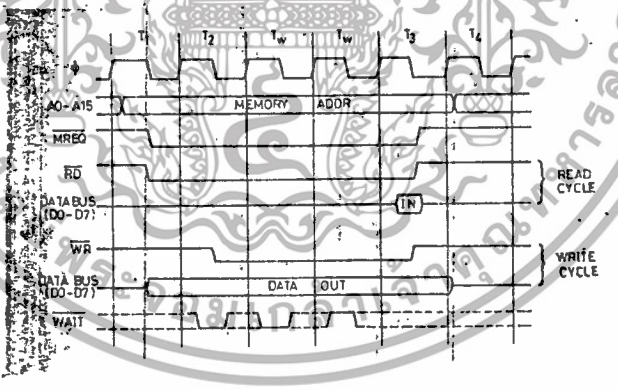
รูปที่ 3.8 แสดงไดอะแกรมของการอ่านและเขียนหน่วยความจำในการอ่านหรือการเขียนหน่วยความจำนี้ ซีพียูจะใช้เวลาทั้งสิ้น 3 ไซเคิลสัญญาณนาฬิกา คือ T_1, T_2, T_3



รูปที่ 3.8 ไดอะแกรมเวลาการอ่านและการเขียนหน่วยความจำ

ในการอ่านหน่วยความจำนั้นซีพียูจะส่งแอดเดรสที่ต้องการอ่านเข้าสู่แอดเดรสบัส A_0-A_{15} ก่อนในตอนแรกของแมชีนไซเคิลนี้ หลังจากนั้นต่อมาก็ส่งสัญญาณ $\overline{MREQ}$ และ $\overline{RD}$ ออกมาและทุกครั้งของขาลงในช่วงจังหวะ T_2 ซีพียูจะตรวจสอบขา $\overline{WAIT}$ ว่าเป็นลอจิก "0" หรือไม่ ถ้าไม่ ซีพียูก็จะทำงานตามปกติ แต่ถ้าเป็น "0" ก็จะสร้าง T_w เพิ่มขึ้น ช่วงจังหวะที่แอดเดรสบัสและสัญญาณ $\overline{MREQ}$ + $\overline{RD}$ ไปยังหน่วยความจำจะมีข้อมูลออกจากหน่วยความจำลงมาในบัสข้อมูลแล้ว แต่ซีพียูยังไม่อ่านจังหวะการอ่านของซีพียูจะเกิดขึ้นในจังหวะขาลงของ T_3 ช่วงการอ่านของซีพียูจะใช้เวลาสั้น ๆ เพื่อป้องกันการอ่านข้อมูลผิดพลาดคือ ผิดตำแหน่งแอดเดรสหรืออ่านได้หลายแอดเดรสนั่นเอง

ในทำนองเดียวกันในจังหวะของ $\overline{MW}$ ก็เช่นกันประกอบด้วย T_1 T_2 และ T_3 (กรณีไม่มี T_w) ในที่นี้ซีพียูจะสร้างสัญญาณ $\overline{MREQ}$ กับ $\overline{WR}$ ให้แอดดีฟ อย่างไรก็ตามซีพียูจะส่งข้อมูลออกไปรอไว้ที่บัสข้อมูลก่อนและยาวนานกว่า พัลซ์การเขียน ($\overline{WR}$) การเขียนข้อมูล ($\overline{WR}$) จะใช้เวลาเพียงสั้น ๆ ในจังหวะที่ข้อมูลการอยู่ที่บัสเรียบร้อยแล้ว

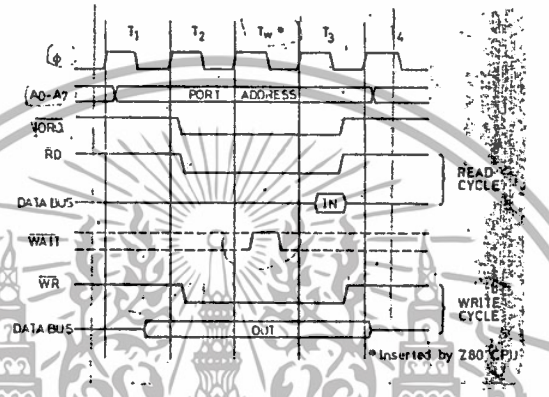


รูปที่ 3.9 ไดอะแกรมเวลาการอ่านและเขียนข้อมูลที่มี T_w

ในกรณีที่ต้องการหน่วงเวลาการทำงานในแมชีนไซเคิลเหล่านี้ สามารถทำได้ด้วยการสร้างขา $\overline{WAIT}$ ให้มีค่าเป็น "0" ในขณะขาลงของ T_2 ซึ่งซีพียูจะสร้าง T_w ขึ้นตราบเท่าที่ขา $\overline{WAIT}$ ยังคงเป็น "0" อยู่

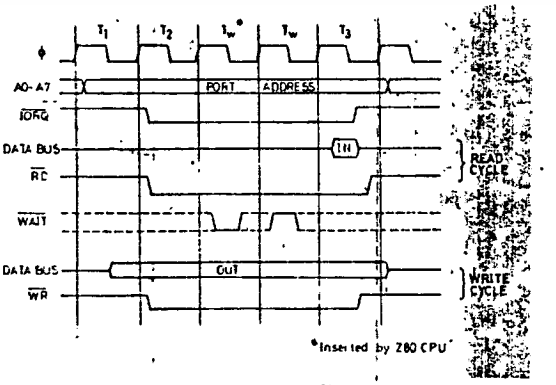
<u>แมชีนไซเคิลของการเขียนเอาท์พุทและอ่านอินพุท</u> (PW-port write and PR-port read)

ในการเขียนเอาท์พุทและอ่านอินพุทก็คล้ายคลึงกับเขียนและอ่านหน่วยความจำนั่นเอง โดยการเปลี่ยนสัญญาณ $\overline{MREQ}$ เป็น $\overline{IORQ}$ เท่านั้น ไดอะแกรมการเขียนเอาท์พุทและอ่านอินพุทแสดงให้เห็นดังรูปที่ 3.10



รูปที่ 3.10 การเขียนเอาท์พุทและอ่านอินพุท

ข้อแตกต่างที่เห็นได้ชัดเมื่อเทียบกับการเขียนและอ่านหน่วยความจำอีกข้อหนึ่ง ส่วนของแอดเดรสนั้นจะแสดงเบอร์พอร์ทในส่วนของ A_0-A_7 เท่านั้น และในการทำงานแบบปกติซีพียูจะสร้าง T_w แทรกเข้ามา 1 ลูก โดยอัตโนมัติแมจะไม่มีการ $\overline{WAIT}$ ทั้งนี้เพราะช่วงจังหวะ $\overline{IORQ}$ นั้นซีพียูสร้างในขณะ T_2 และซีพียูจะไม่สามารถตรวจสอบจังหวะ $\overline{WAIT}$ ในขณะขาลงของ T_2 ได้การตรวจสอบว่า $\overline{WAIT}$ จะเกิดขึ้นในจังหวะของ T_w แทนนั่นเอง และถ้าหากมีการ $\overline{WAIT}$ ซีพียูก็จะสร้างสัญญาณ T_w ต่อมาอีกเช่นเดียวกับที่กล่าวแล้ว

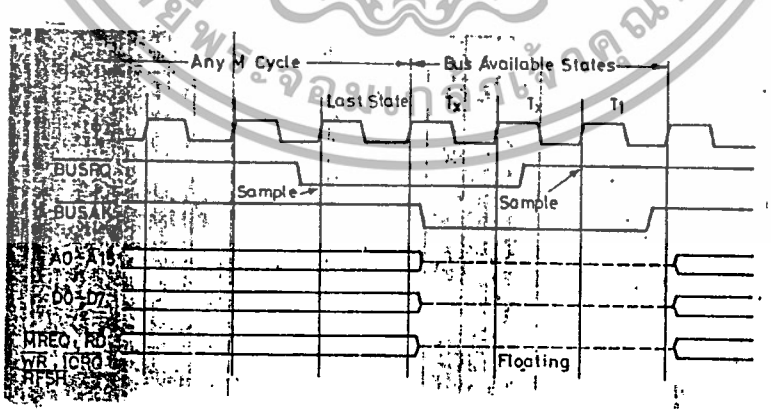


รูปที่ 3.11 การเขียนเอาท์พุทและอ่านอินพุทขณะมีการ WAIT

ไซเคิลการรับสัญญาณอินเตอร์รัพท์และการตอบสนองต่อสัญญาณอินเตอร์รัพท์ (Interrupt

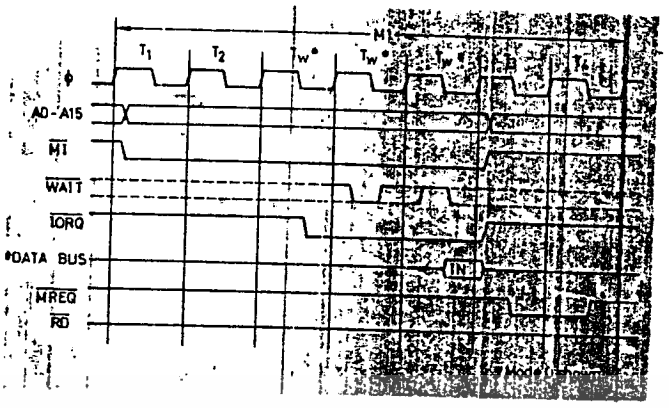
ไซเคิลการขอใช้บัสและการตอบสนองการให้ใช้บัส (BUS REQUEST and ACKNOWLEDGE CYCLE)

รูปที่ 3.12 เป็นไดอะแกรมเวลาของการขอใช้บัสและการตอบสนองต่อการใช้บัส การใช้บัสนั้นจะใช้สัญญาณที่ส่งเข้ามาขอทางขา $\overline{\text{BUSRQ}}$ ซีพียูจะทำการตรวจสอบขา $\overline{\text{BUSRQ}}$ ในขณะสัญญาณนาฬิกากำลังจะเปลี่ยนจาก "0" ไปเป็น "1" ในขณะสัญญาณสเตทสุดท้ายของทุก ๆ M ไซเคิล ถ้า $\overline{\text{BUSRQ}}$ แอคตีฟ ซีพียูจะทำการเซทแอดเดรสบัส และบัสข้อมูลให้อยู่ในสถานะไตรสเตท ในสัญญาณนาฬิกาลูกต่อมา และซีพียูก็จะส่งสัญญาณตอบรับการขอใช้บัส ($\overline{\text{BUSAK}}$) เมื่อซีพียูควบคุมไตรสเตทแล้ว อุปกรณ์ภายนอกก็จะใช้บัสได้ ซึ่งอาจจะเป็นกรรมวิธีการ DMA (direct memory access) หรือการติดต่อระหว่างอุปกรณ์ I/O กับหน่วยความจำโดยตรงได้ กรณีเช่นนี้ทำให้การถ่ายข้อมูลเป็นไปได้เร็วกว่าใช้ซีพียูควบคุมโดยตรง การถ่ายเทข้อมูลในกรณีนี้มักใช้สำหรับการถ่ายข้อมูลเป็นจำนวนมาก ขณะนี้การขอใช้บัสนี้ ซีพียูจะตรวจสอบขา $\overline{\text{BUSRQ}}$ ทุก ๆ สเตทเช่นกัน และถ้า $\overline{\text{BUSRQ}}$ เป็น "1" เมื่อใดในสัญญาณนาฬิกาสเตทต่อไปก็จะส่ง $\overline{\text{BUSAK}}$ = "1" เพื่อบอกการเลิกใช้บัส และการควบคุมไตรสเตทก็หมดสิ้นภาระกิจ



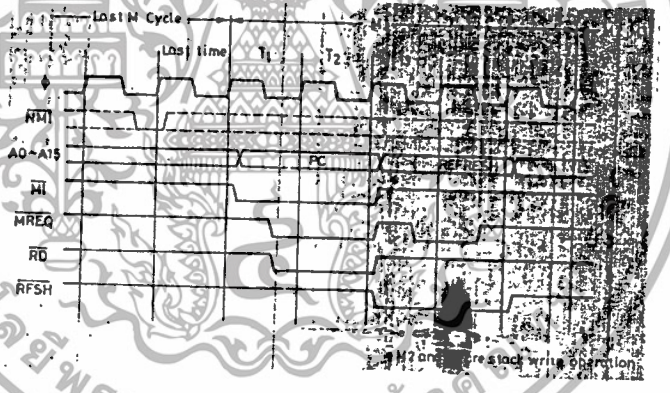
รูปที่ 3-12 ไดอะแกรมการขอใช้บัสและการตอบสนองต่อการใช้บัส

ในขณะการใช้บัสในกรณีนี้ซีพียูไม่สามารถตอบสนองต่อการอินเตอร์รัพท์ได้ ไม่ว่าจะเป็นการอินเตอร์รัพท์แบบ $\overline{\text{NMI}}$ หรือ $\overline{\text{INT}}$ ก็ตาม



รูปที่ 3.13 ไดอะแกรมการรับสัญญาณอินเตอร์รัพท์และการตอบสนองต่ออินเตอร์รัพท์

<u>การตอบสนองต่อการอินเตอร์รัพท์แบบนอนมาสเคเบิ้ลอินเตอร์รัพท์</u> (Nonmaskable interrupt response)



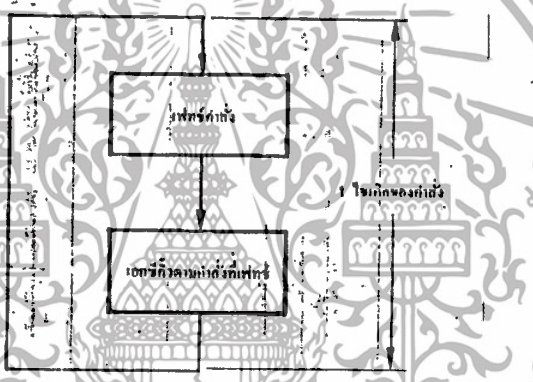
รูปที่ 3.14 ไดอะแกรมการตอบสนองต่อการอินเตอร์รัพท์แบบนอนมาสเคเบิ้ล

รูปที่ 3.14 เป็นไดอะแกรมเวลาของการตอบสนองต่อการอินเตอร์รัพท์แบบนอนมาสเคเบิ้ล เมื่อถึงพัลซ์ของสัญญาณนาฬิกาลูกสุดท้ายก่อนการกระทำการจบคำสั่งแต่ละคำสั่ง ซีพียูจะทำการตรวจสอบขา $\overline{\text{NMI}}$ และถ้า $\overline{\text{NMI}}$ มีคว แอคทีฟ (=0) ซีพียูก็จะแลทช์ไว้แล้วตอบสนองต่อการอินเตอร์รัพท์ทันที การอินเตอร์รัพท์ด้วยวิธีนี้ถือว่าซีพียูให้ความสำคัญสูงสุด และขบวนการทางซอฟท์แวร์ก็ไม่สามารถที่จะป้องกันไม่ให้ซีพียูตอบสนองได้ การตอบสนองจะเกิด

ขึ้นทันที ดังนั้นการอินเตอร์รัพท์ด้วยวิธีนี้จึงนำมาใช้ในกรณีที่สำคัญเป็นพิเศษ เช่น ใช้อินเตอร์รัพท์เมื่อมีแรงดันไฟตก เป็นต้น การอินเตอร์รัพท์วิธีนี้ซีพียูจะส่งข้อมูลเดิมที่อยู่ในโปรแกรมเคาว์เตอร์ไปเก็บไว้ที่สแตค แล้วเซทค่าโปรแกรมเคาว์เตอร์เป็นค่า 0066H เพื่อกระทำในส่วนของโปรแกรมย่อยที่ตำแหน่ง 0066

การทำงานของซีพียูในแต่ละคำสั่ง

ในการทำงานแต่ละคำสั่งของซีพียู ซีพียูจะมีไซเคิลการทำงานที่แน่นอนคือ การเฟทช์ และการเอกซีคิ้วสลับกันไปตามไดอะแกรมรูปที่ 3.15



รูปที่ 3.15 ไซเคิลการทำงานในแต่ละคำสั่ง

สภาวะการกระทำการเฟทช์ หมายถึง การที่ซีพียูส่งข้อมูลของโปรแกรมเคาว์เตอร์ไปยังแอดเดรสบัสและส่งสัญญาณ $\overline{RD}$ + $\overline{MERQ}$ เพื่อไปอ่านข้อมูลออพโค้ดที่อยู่ในหน่วยความจำเข้ามาใน IR (instruction register) เพื่อกระทำการตีความในการเอ็กซีคิ้วต่อไป เมื่อสิ้นสุดการเฟทช์ค่าของ PC จะเพิ่มขึ้นโดยอัตโนมัติเพื่อที่จะอ่านข้อมูลในตำแหน่งต่อไปในหน่วยความจำ

สภาวะการเอ็กซีคิ้ว เป็นการกระทำต่อเฟทช์ ส่วนของออปโค้ดที่อยู่ IR จะได้รับการถอดรหัสและตีความหมาย ซีพียูจะทราบความหมายของคำสั่งนั้นและจะกระทำตามคำสั่งนั้น ๆ เช่น การเคลื่อนย้ายข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่ง การกระทำทางคณิตศาสตร์และลอจิก เป็นต้น เมื่อกระทำในส่วนนี้เสร็จแล้วซีพียูก็จะกลับมากระทำการเฟทช์คำสั่งถัดไปอีกครั้งหนึ่ง

สลับไปเช่นนี้เรื่อย ๆ ไป

เพื่อให้เข้าใจบทบาทการทำงานของซีพียูได้ดียิ่งขึ้น เราจะพิจารณาการทำงานของ Z-80 ในแต่ละ T สเตททั้งขณะการเฟทช์และเอ็กซีคิ้วว่าแต่ละคำสั่งซีพียูจะต้องกระทำอะไรบ้าง

จากเหตุผลของการทำงานแต่ละคำสั่งมีความยุ่งยากต่อการทำงานไม่เท่ากัน ดังนั้นซีพียูจึงใช้เวลาทำงานในแต่ละครั้งไม่เท่ากัน แต่อย่างไรก็ตามการกำหนดรูปแบบทำงานของซีพียูเรากำหนดได้ในลักษณะของรูปแบบเป็นแมชีนไซเคิลต่าง ๆ ตามที่กล่าวมาแล้วได้เป็นหลายรูปแบบเช่น

IO (internal CPU operation) เป็นไซเคิลการทำงานภายในของซีพียูที่ไม่ต้องใช้สายสัญญาณจากบัสต่าง ๆ

MR (memory read) เป็นแมชีนไซเคิลการอ่านหน่วยความจำ ซึ่งได้เขียนไว้แล้วดังรูปที่ 3.8

MRH (memory read of high byte) เป็นแมชีนไซเคิลการอ่านหน่วยความจำเช่นกัน รายละเอียดการทำงานเหมือนรูปที่ 3.8

MRL (memory read of low byte) เป็นแมชีนไซเคิลการอ่านหน่วยความจำเช่นกัน รายละเอียดการทำงานเหมือนรูปที่ 3.8

MW - (memory write)เป็นแมชีนไซเคิลการเขียนหน่วยความจำ ในกรณีนี้มีไซเคิลการทำงาน ดังแสดงไว้ในรูปที่ 3.8

MWH (memory write of high byte) เป็นแมชีนไซเคิลการเขียนหน่วยความจำด้วยข้อมูล 8 บิท ที่มีนัยสำคัญสูง ไซเคิลการทำงานเหมือนรูปที่ 3.8

MWL (memory write of low byte) เป็นแมชีนไซเคิลการเขียนหน่วยความจำด้วยข้อมูล 8 บิท ที่มีนัยสำคัญต่ำ ไซเคิลการทำงานเหมือนรูปที่ 3.8

OCF (opcode fetch) เป็นแบบแมชีนไซเคิลที่แสดงด้วย $\overline{M}_1$ ดังรูปที่ 3.6 เป็นสภาวะการอ่านออพโค้ดจากหน่วยความจำ

ODH (operand data read of high byte) เป็นแมชีนไซเคิลการอ่านหน่วยความจำในส่วนของหน่วยความจำที่เป็นข้อมูลโอเปอร์แรนด์ 8 บิท ส่วนที่มีนัยสำคัญสูง ไซเคิลการทำงานเหมือนการอ่านหน่วยความจำดังรูปที่ 3.6

ODL (operand data read of low byte) เป็นแมชีนไซเคิลการอ่านหน่วยความจำในส่วนของหน่วยความจำที่เป็นข้อมูลโอเปอร์แรนด์ 8 บิท ส่วนที่มีความสำคัญต่ำ ไซเคิลการทำงานเหมือนการอ่านหน่วยความจำดังรูปที่ 3.8

SRH (stack read of high byte) เป็นการอ่านข้อมูลจากสแตคที่อยู่ในหน่วยความจำ 8 บิท ในส่วนที่เป็นข้อมูลบิทที่มีนัยสำคัญสูง แมชีนไซเคิลนี้เหมือนกับแมชีนไซเคิลการอ่านหน่วยความจำดังรูปที่ 3.8

SRL (stack read of low byte) เป็นการอ่านข้อมูลจากสแตค 8 บิท จากหน่วยความจำในส่วนที่เป็นบิทที่มีนัยสำคัญต่ำ แมชีนไซเคิลนี้เหมือนกับแมชีนไซเคิลในการอ่านหน่วยความจำ ดังรูปที่ 3.8

SWH (stack write of high byte) เป็นการเขียนข้อมูลลงหน่วยความจำที่เป็นสแตคในส่วนของข้อมูล 8 บิท ส่วนที่เป็นบิทมีนัยสำคัญสูง แมชีนไซเคิลนี้เหมือนกับแมชีนไซเคิลการเขียนข้อมูลลงในหน่วยความจำดังรูปที่ 3.8

SWL (stack write of low byte) เป็นการเขียนข้อมูลลงหน่วยความจำส่วนที่เป็นสแตคในส่วนของข้อมูล 8 บิท ส่วนที่เป็นบิทมีนัยสำคัญต่ำ แมชีนไซเคิลนี้เหมือนกับแมชีนไซเคิลการเขียนข้อมูลลงในหน่วยความจำ ดังรูปที่ 3.8

(T) (Number of Tstate in that machine cycle) ค่าในวงเล็บเป็นค่าของจำนวน สเตทในแมชีนไซเคิลนั้น ๆ

บทที่ 4

ชุดคำสั่งของ Z-80

การพัฒนา Z-80 มีจุดมุ่งหมายหลักเพื่อให้ใช้ชุดคำสั่งเดิมของ 8080 ได้ ดังนั้น ทุกคำสั่งที่เคยใช้ได้ในไมโครโปรเซสเซอร์เบอร์ 8080 จะยังคงใช้ได้ในกรณีใช้งานกับซีพียู Z-80 Z-80 มีคำสั่งได้มากถึง 158 คำสั่ง ซึ่งรวมเอาทั้ง 78 คำสั่งที่มีใน 8080 ไว้ด้วย กลุ่มคำสั่งของ Z-80 พอแบ่งแยกออกได้เป็นกลุ่ม ๆ ดังนี้

1. กลุ่มการโหลดและแลกเปลี่ยนข้อมูล (load and exchange)
2. กลุ่มการเคลื่อนย้ายและค้นหาข้อมูลเป็นบล็อก (block transfer and search)
3. กลุ่มคำสั่งทางคณิตศาสตร์และลอจิก (arithmetic and logical)
4. กลุ่มคำสั่งการเคลื่อนย้ายข้อมูลเป็นวงรอบและการชิฟท์ (rotate and shift)
5. กลุ่มคำสั่งการกระทำในส่วนบิท (bit maniputation set reset test)
6. กลุ่มคำสั่งการกระโดด การเรียกโปรแกรมย่อย การคืนกลับสู่โปรแกรมหลัก (jump, call and return)
7. กลุ่มคำสั่งเกี่ยวกับอินพุท-เอาท์พุท (input-output)
8. กลุ่มคำสั่งควบคุมซีพียู (basic cpu control)

ในการโหลดหรือแลกเปลี่ยนข้อมูลนี้ สามารถกระทำได้โดยใช้รีจิสเตอร์ต่าง ๆ ที่อยู่ภายในซีพียู โดยจากรีจิสเตอร์หนึ่งไปยังอีกรีจิสเตอร์หนึ่ง หรือเคลื่อนย้ายข้อมูลกับหน่วยความจำโดยจากรีจิสเตอร์ไปยังหน่วยความจำหรือจากหน่วยความจำเข้าหารีจิสเตอร์ก็ได้ นอกจากนี้ยังมีลักษณะคำสั่งที่สามารถโหลดข้อมูลตัวเลขใด ๆ ก็ได้เข้าไปในซีพียูโดยเฉพาะเจาะจงที่รีจิสเตอร์ใดรีจิสเตอร์หนึ่งหรือหน่วยความจำ การโหลดข้อมูลยังสามารถกระทำได้ทั้งแบบ 8 บิท และ 16 บิท ขีดความสามารถที่สำคัญอีกประการหนึ่งคือ สามารถแลกเปลี่ยนข้อมูลระหว่างรีจิสเตอร์สองตัวให้ข้อมูลสลับที่กันก็ได้

Z-80 มีคำสั่งที่เหนือกว่าของ 8080 หลายแบบ และหนึ่งในหลายแบบนี้ก็ได้แก่ กลุ่มคำสั่งในการย้ายข้อมูลเป็นบล็อกจากที่หนึ่งไปยังอีกที่หนึ่งด้วยคำสั่งเพียงคำสั่งเดียวเท่านั้น ลักษณะของคำสั่งนี้จะอาศัยวิธีการกำหนดกระทำคำสั่งเดิมซ้ำ ๆ ได้จนกว่าเงื่อนไขจะเกิดขึ้นจึงจะไปกระทำคำสั่งถัดไป นอกจากการเคลื่อนย้ายข้อมูลเป็นบล็อกแล้วยังมีการค้นหาข้อมูลในบล็อกของข้อมูลอีก โดยซีพียูจะตรวจสอบข้อมูลในบล็อกนั้นทีละตัว

กลุ่มคำสั่งเกี่ยวกับการกระทำทางคณิตศาสตร์และลอจิก โดยอาศัยรีจิสเตอร์หลักคือ แอคคิวมูเลเตอร์ กับรีจิสเตอร์อื่น ๆ การกระทำ กระทำภายในหน่วย ALU และให้ผลลัพธ์กลับเข้าไปยังแอคคิวมูเลเตอร์ โดยมีแฟลกรีจิสเตอร์ (F) เป็นตัวแสดงสถานะบางอย่างของผลการกระทำ ลักษณะของชุดคำสั่งทางคณิตศาสตร์มีเพียงคำสั่งพื้นฐานคือ การบวกและลบเท่านั้น การบวกอาจจะกระทำครั้งละ 8 บิท หรือ 16 บิทก็ได้

การเซท รีเซทและทดสอบค่าในบิทต่าง ๆของรีจิสเตอร์ภายในซีพียู หรือของหน่วยความจำก็เป็นอีกกลุ่มคำสั่งหนึ่งที่ทำให้ Z-80 มีชอบเขตการใช้งานได้ดีกว่า 8080 เราสามารถทำการเซทรีเซท หรือทดสอบเพียงบิทใดบิทหนึ่งได้โดยตรง ทำให้งานในระบบควบคุมบางอย่างใช้โปรแกรม การเขียนที่ไม่ยุ่งยากนัก

กลุ่มคำสั่งเกี่ยวกับ jump การ return เป็นการเคลื่อนย้ายตำแหน่งของโปรแกรมซึ่งอาจจะตามเงื่อนไขหรือไม่ก็ได้ การ jump เกือบทั้งหมดมักจะต้องเป็นคำสั่งหลายไบท์ แต่มีคำสั่ง jump อยู่ชนิดหนึ่งที่ใช้ขนาดคำสั่งเพียง 1 ไบท์เท่านั้น คำสั่งนั้นคือ restart (RST) ซึ่งจะทำให้เกิดการ jump มายังตำแหน่งในตอนต้นของแอดเดรสหน่วยความจำ นอกจากนี้การ jump ยังอาจเกิดขึ้นได้ด้วยการโหลดค่าจาก HL, IX, IY เข้าไปยัง PC โดยตรง

กลุ่มคำสั่งเกี่ยวกับอินพุทและเอาท์พุทในกรณีของ 8080 มีเพียงสองคำสั่งคือ IN และ OUT เท่านั้น IN คืออ่านค่าจากอุปกรณ์อินพุทมายังรีจิสเตอร์ A ส่วน OUT ก็จะตรงข้ามกัน แต่ในกรณีของ Z-80 มีชุดคำสั่งในการส่งถ่ายข้อมูลอินพุทและเอาท์พุทอีกหลายคำสั่งที่เพิ่มประสิทธิภาพกล่าวคือ การใช้รีจิสเตอร์ C เป็นรีจิสเตอร์กำหนดเบอร์พอร์ท

Z-80 ยังมีคำสั่งในการ IN หรือ OUT ข้อมูลเป็นบล็อกเข้ามาเก็บในหน่วยความจำได้โดยตรง ทำให้การใช้งานทางอินพุทและเอาท์พุทสะดวกยิ่งขึ้น

ในกลุ่มสุดท้ายคือ กลุ่มคำสั่งเกี่ยวกับการควบคุมซีพียู ซึ่งได้แก่ การอินเตอร์รัพท์ การอินเตอร์รัพท์ในชุดของ Z-80 มีได้หลายโหมด การเซทให้ขึ้นอยู่ในโหมดไหนขึ้นอยู่กับซอฟท์แวร์

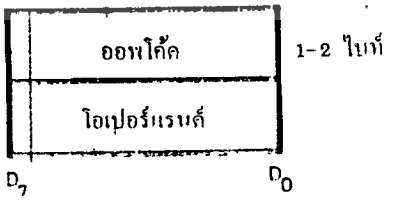
การอ้างแอดเดรสของ Z-80

ภายในซีพียูของ Z-80 มีรีจิสเตอร์หลายตัว การทำงานของ Z-80 จึงคล่องตัวได้มาก การที่จะให้ Z-80 เป็นระบบที่สมบูรณ์ต้องต่อกับหน่วยความจำและอุปกรณ์อินพุทเอาท์พุทในการต่อร่วม เพื่อให้ซีพียูทำงานได้สมบูรณ์ ในกรณีนี้ซีพียูจำเป็นจะต้องมีการเรียกแอดเดรสที่ต้องการ การอ้างแอดเดรส (addressing) ของซีพียูมีโหมด (mode) การอ้างได้หลายแบบหลายวิธี ในแต่ละวิธีก็จะมีวิธีการอ้างแอดเดรสได้เป็นกลุ่ม ๆ คือ

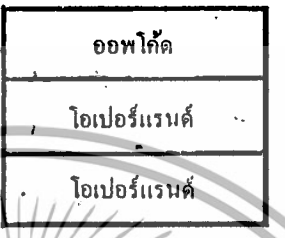
1. การอ้างแอดเดรสแบบอิมมีเดียท (Immediate addressing) ในโหมดนี้คำสั่งจะประกอบด้วยออปโค้ด 1-2 ไบท์ และตามด้วยโอเปอร์แรนด์อีก 1 ไบท์ ลักษณะคำสั่งจะเป็นดังนี้

ตัวอย่างการอ้างแอดเดรสแบบอิมมีเดียท เช่น การโหลดข้อมูลค่าคงที่เข้าไปเก็บไว้ในรีจิสเตอร์ หรือหน่วยความจำ เช่น

LD A, OFH

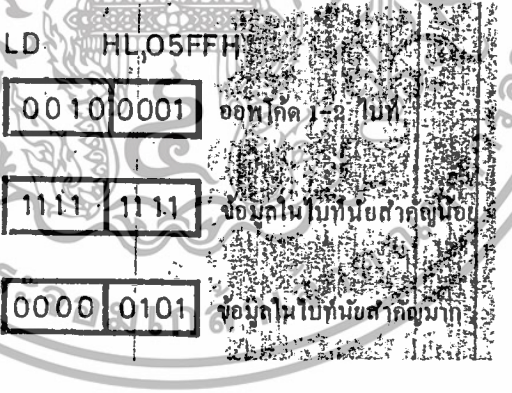


2. การขยายตัวโอเปอร์แรนด์ในกลุ่มคำสั่งการอ้างแอดเดรสแบบอิมมีเดียท (Immediate addressing extended) การอ้างแอดเดรสแบบนี้ก็เหมือนกับแบบแรก แตกต่างกันเพียงส่วนของโอเปอร์แรนด์เป็นข้อมูลขนาด 16 บิท ลักษณะรูปแบบของคำสั่งจะเป็นดังนี้



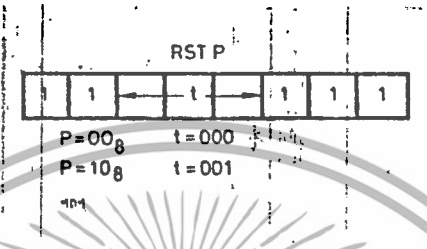
จากตัวอย่างนี้จะเป็นการโหลดข้อมูลสองไบท์เข้าไปในรีจิสเตอร์ภายในซีพียู

ในกรณีนี้เมื่อซีพียูกระทำการเอ็กซีคิว คำสั่งนี้ (คำสั่งขนาด 3 ไบท์) ซีพียูจะโหลดข้อมูล 05 เข้าไปเก็บไว้ที่รีจิสเตอร์ H และข้อมูล FF จะไปอยู่รีจิสเตอร์ L

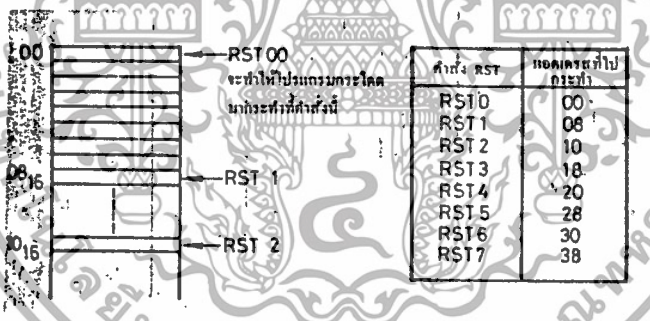


3. การอ้างแอดเดรสแบบ modified page zero ในคำสั่งของ Z-80 มีคำสั่งพิเศษอยู่คำสั่งหนึ่งที่ทำหน้าที่เหมือนคำสั่ง CALL คือคำสั่ง RST หรือ restart ลักษณะการทำงานในการอ้างแอดเดรสคือ มันสามารถกำหนดค่าให้กับโปรแกรมเคาน์เตอร์ได้โดยตรง ค่าที่มันกำหนดให้จะเป็นแอดเดรสที่อยู่ในเซคชัน ลักษณะของคำสั่งนี้ประกอบขึ้นเพียงไบท์เดียวเท่านั้นจึงมีประโยชน์หลายอย่าง เช่น ใช้เป็นมาส์ค (mask) สำหรับคำสั่งให้กระทำในขณะมีการอินเตอร์รัพท์

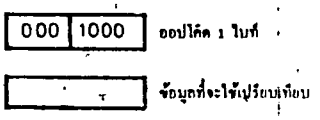
ข้อมูลเดิมในโปรแกรมเคาวน์เตอร์ก่อนการเปลี่ยนแปลงจะได้รับการเก็บรักษาไว้ได้อีกด้วย แต่เงื่อนไขในการกระโดดไปยังแอดเดรสในเรจศูนย์ยังมีขอบเขตจำกัด คือมันจะกระโดดไปได้เพียง 8 แอดเดรสเท่านั้น ค่าแอดเดรสที่มันจะไปได้คือแอดเดรส b_5 b_4 b_3 0 0 0 หรือ 00_8, 10_8,.....70_8 เท่านั้น ลักษณะของคำสั่งจะเป็น



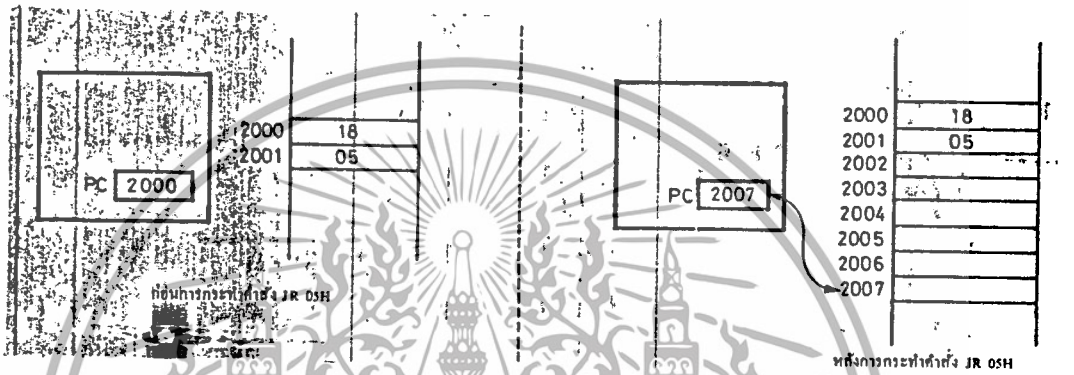
จะเห็นว่าการใช้คำสั่ง RST เพียงไบท์เดียวสามารถกำหนดคำสั่งในการทำให้เกิดการกระโดดไปยังแอดเดรสต่าง ๆ ได้ถึง 8 ค่าตามที่กำหนด ส่วนค่าในโปรแกรมเคาวน์เตอร์เดิมจะเก็บรักษาไว้ในชั้นของสแตค



4. การอ้างแอดเดรสแบบเปรียบเทียบ (relative addressing) วิธีการนี้จะใช้ข้อมูลไบท์ที่อยู่ตามหลังออปโค้ด เพื่อบอกตำแหน่งว่าแอดเดรสที่อ้างถึงอยู่ห่างจากค่าในโปรแกรมเคาวน์เตอร์เท่าใด เช่น



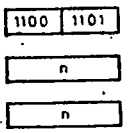
เมื่อ ซีพียูกระทำคำสั่งนี้เสร็จ ค่าในโปรแกรมเคาวน์เตอร์จะมีค่าเป็น PC + 2 + e นั่นคือ ค่าใหม่ที่มันจะอ้างถึงอยู่ห่างจากคำสั่งที่มันกระทำแล้วด้วยค่า e นั่นเอง ค่า e ที่ ซีพียูมองเห็นอาจจะเป็นลักษณะของตัวเลข 2's คอมพลีเมนท์ก็ได้ดังนั้นค่าที่มันจะอ้างถึงได้จึงอยู่ระหว่าง + 127 กับ - 128 จาก PC + 2 ดังตัวอย่างเช่น คำสั่ง JR 05H ลักษณะของคำสั่งจะเป็นดังนี้



5. การอ้างแอดเดรสแบบขยาย (extend address) วิธีการนี้จะใช้ข้อมูล 2 ไบท์ที่ตามหลังออพโค้ด เป็นตัวกำหนดค่าแอดเดรสเช่น ข้อมูลนี้คือ nn ซึ่งจะเป็นตัวกำหนดค่าของแอดเดรสที่จะกระทำใหม่ ยกตัวอย่าง คำสั่ง CALL nn หมายถึง การเรียกโปรแกรมย่อยที่ตำแหน่ง nn เช่น

CALL nn

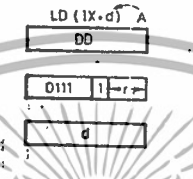
คำสั่งนี้ประกอบด้วยลักษณะทางภาษาเครื่องดังนี้



การอ้างแอดเดรสแบบนี้ เป็นการนำเอาข้อมูลในตัวโอเปอร์แรนด์เป็นแอดเดรสนั่นเอง โดยการนำเอาข้อมูลส่วนโอเปอร์แรนด์ไปไว้ยังโปรแกรมเคาวน์เตอร์โดยตรง ส่วน

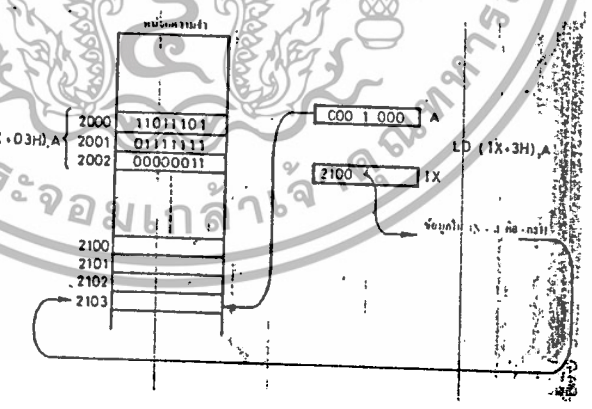
ข้อมูลเดิมของโปรแกรมเคาวน์เตอร์จะเก็บรักษาไว้ในส่วนของสแตค

6. การอ้างแอดเดรสโดยใช้อินเดครีจิสเตอร์ (index addressing) โดยที่ Z-80 มีอินเดครีจิสเตอร์ถึง 2 ตัว คือ IX และ IY วิธีการใช้อินเดครีจิสเตอร์ร่วมในการอ้างแอดเดรสนั้นจะใช้ค่า IX หรือ IY เป็นฐานเพื่อรวมกับค่าที่ตามหลังออปโค๊ดมารวมกัน เป็นแอดเดรสที่ต้องการ เช่น



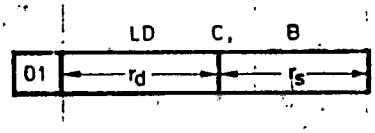
ลักษณะของคำสั่งนี้เป็นการนำเอาข้อมูลของรีจิสเตอร์ A ไปเก็บไว้ในหน่วยความจำที่อ้างแอดเดรสโดยที่ค่า IX รวมกับค่า d โดยปกติค่า d จะได้รับการรวมโดยดูในรูป 2's คอมพลีเมนท์ ดังนั้นจึงทำให้ค่า d แปรได้จากค่า (127 ถึง - 128)

ขอให้พิจารณาโคอะแกรมการทำงานดังรูป



7. การอ้างแอดเดรสแบบใช้รีจิสเตอร์ (register addressing) การออกแบบกลุ่มคำสั่งในชอบเซตของจำนวนบิทออปโค๊ดที่จำกัดต้องหาวิธีที่ให้ได้ประสิทธิภาพดีที่สุด วิธีหนึ่งที่ใช้ในกรณีของ Z-80 คือกำหนดรหัสของรีจิสเตอร์ เช่น มีรีจิสเตอร์ 8 ตัว ก็ใช้รหัส 3 บิท ดังนั้น กลุ่มคำสั่งโหลดสามารถใช้คำสั่งเพียง 1 ไบท์ เพื่อทำการโหลดข้อมูล

ระหว่างรีจิสเตอร์ได้ เช่น



โดยที่ส่วนสามบิทใน r_d คือ ชื่อรีจิสเตอร์ปลายทาง ส่วนสามบิทใน r_s คือ รีจิสเตอร์ต้นทาง เช่น LDC,B เขียนได้เต็ม 01 001 000 เมื่อการแทนรหัสสำหรับรีจิสเตอร์ต่าง ๆ เป็นดังนี้

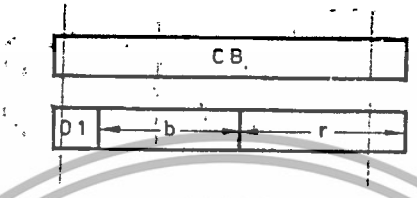
000 คือ รีจิสเตอร์ B
001 " C
010 " D
011 " E
100 " H
101 " L
110 " หน่วยความจำที่แอดเดรสด้วยข้อมูลในรีจิสเตอร์ H, L
111 " A

8. การอ้างแอดเดรสแบบ implied addressing ในกรณีนี้ ซี.พี.ยู. จะตีความหมายเองโดยตรงว่า รีจิสเตอร์ภายในซีพียูตัวหนึ่งจะเป็นโอเปอร์แรนด์ เช่น คำสั่ง

จากคำสั่งนี้จะเห็นว่า เราไม่ต้องกำหนดรหัสรีจิสเตอร์ A เลยแต่เครื่องจะตีความแล้วทราบเองว่า เป็นคำสั่งที่ต้องการกระทำร่วมกับรีจิสเตอร์ A กล่าวคือ การทำงานซีพียูจะนำเอารีจิสเตอร์ที่กำหนดในออปโค้ดมากระทำร่วมกับรีจิสเตอร์ A นั่นเอง

9. การอ้างแอดเดรสเข้าสู่บิทต่าง ๆ (bit addressing) ภายในตัว Z-80 มีคำสั่งพิเศษ ที่สามารถกระทำการเซทรีเซท หรือการทดสอบบิทใดบิทหนึ่งใน 8 บิท ได้ เช่น

RES b,r (reset bit b of operand r)



ในที่นี้เราใช้รหัส b แทนโอเปอร์แรนด์ ว่าเป็นบิทใดในรีจิสเตอร์ ถ้า b = 000 หมายถึง บิท 0 และ b = 111 หมายถึง บิท 7 ดังนั้น เราสามารถทำให้บิทใดในรีจิสรีจิสเตอร์ให้เป็น 0 ได้ เช่น RES 0,A หมายถึง ทำให้บิท 0 ของรีจิสเตอร์ A เป็น 0

หมายเหตุ เราสามารถใช้การอ้างแอดเดรสหลาย ๆ แบบที่กล่าวแล้วนี้มารวมกันได้

<u>แฟลก</u>

ใน Z-80ซีพียู จะประกอบด้วยแฟลกจำนวน 6 แฟลก และมีบิทที่ไม่ได้แสดงเป็นแฟลกอีก 2 บิท รวมเป็น 8 บิท เพื่อประกอบเป็นรีจิสเตอร์ F ส่วนของแฟลกแต่ละบิทสามารถที่จะเซทหรือรีเซทตามการกระทำของคำสั่งที่ซีพียูกำลังทำงาน นอกจากนี้ซีพียูยังสามารถใช้ตรวจสอบแฟลกเพื่อกระทำเงื่อนไขต่าง ๆ

ลักษณะการใช้แฟลกจะใช้ตัวอักษรย่อแทนแฟลก ดังนี้

C = แฟลกตัวทด

N = แฟลกแสดงการบวกหรือลบ

P/V = แฟลกแสดงพาริตี้และโอเวอร์โฟลว์

H = แสดงแฟลกตัวทดช่วย

Z = แฟลกแสดงค่าศูนย์

S = แฟลกเครื่องหมาย

X = ไม่ได้ใช้

ในการกระทำคำสั่งต่าง ๆ ของ Z-80 บางคำสั่งก็จะมีผลต่อแฟลก บางคำสั่งก็ไม่มีผลต่อแฟลก การที่คำสั่งบางคำสั่งมีผลต่อแฟลกทำให้เราสามารถตรวจสอบเงื่อนไขได้จากแฟลก เช่น

INC A (ให้เพิ่มค่ารีจิสเตอร์ A อีก 1)

ผลที่เกี่ยวกับแฟลก คือ

ถ้าผลลัพธ์เป็น 0 Z = 1

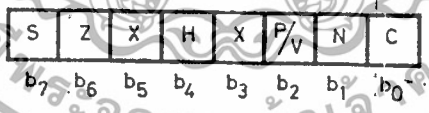
ถ้าผลลัพธ์เป็นลบ หรือบิท D_7 = 1 S = 1

ถ้ามีตัวทดในบิทที่ 3 H = 1

ถ้าผลลัพธ์เท่ากับ 80 P/V = 1

คำสั่งนี้จะไม่เกี่ยวข้องกับ N และ C แฟลก

ลักษณะของรีจิสเตอร์ F จะประกอบด้วยแฟลกแต่ละบิทเป็นดังนี้



ในบิท b_5 และ b_3 จะเป็นบิทที่ประกอบเพิ่มขึ้นมาโดยมิได้มีความหมายทางแฟลก แต่จะประกอบเพื่อให้รีจิสเตอร์ F ครบ 8 บิท การทำงานของซีพียูจึงสามารถโหลดข้อมูลจาก F ไปยัง A หรือโหลดจาก A กลับมายัง F ได้ รายละเอียดของแฟลกแต่ละตัวเป็นดังนี้

1. แฟลกตัวทด (C:Carry flag) แฟลกตัวนี้เป็นบิทที่ใช้สำหรับทดจากข้อมูลในรีจิสเตอร์ A เช่น เมื่อมีการบวกข้อมูล 8 บิท ผลบวกอาจเลยเป็น 9 บิท บิทที่เกินเลยจะทดเข้าไปเก็บไว้ที่บิท C นี้ ในทำนองเดียวกัน ถ้าซีพียูกระทำคำสั่งลบ และมีการขอ

ยีมค่าของแฟลกตัวนี้ก็จะได้รับการเซทให้มีค่าเป็น "1" เช่นกัน

2. แฟลกศูนย์ (z:zero flag) ในแฟลกบิทจะได้รับการเซทให้มีค่าเท่ากับ "1" ถ้าผลของการกระทำให้รีจิสเตอร์ A มีค่า "1" นอกเหนือจากนี้มันจะกระทำการรีเซท

3. แฟลกเครื่องหมาย (s:sign flag) ลักษณะของแฟลกนี้มีประโยชน์ในการบอกการกระทำของซีพียูว่าผลลัพธ์ที่เกิดขึ้นมีค่าเป็นบวก หรือเป็นลบ ถ้าเครื่องหมายของตัวเลขเป็นลบ คือ บท $D_7 = 1$ จะปรากฏค่า "1" ในบิทนี้

4. แฟลกพาริตี้/ค่าเกิน (parity/over flow flag) (p/v) แฟลกนี้เป็นแฟลกที่ใช้สำหรับเป็นตัวบอกพาริตี้ของผลลัพธ์ในแอคคิวมูเลเตอร์ เมื่อให้มีการกระทำทางบอจิก และยังใช้แสดงสภาวะของค่าที่เกินกำหนดใน 8 บิท (1 บิทเครื่องหมายรวมกับ 7 บิทที่เป็นขนาด) นั่นคือค่าสูงสุดที่เป็นไปได้คือ +127 และ -128 ถ้าค่ามากกว่า +127 หรือน้อยกว่า -128 ก็จะเกิดการเซทค่าที่ 1 ที่บิทนี้ขึ้นดังตัวอย่าง

```
+120     0111    1000
+105     0110    1001   +
   c = 0 1110    0001 = -95 (ผิด) เกิดโอเวอร์โฟลว์บิทพาริตี้จะได้รับการ
                                  set ให้มีค่าเป็น "1"
         ลองพิจารณาเลขจำนวนลบ

-5       1111    1011
-16      1111    0000
  c = 1  1110    1011 = -21  (ถูก)
```

ในกรณีนี้แฟลกนี้จะไม่ได้รับการเซท คือจะมีค่าเป็น "0" เพื่อบอกว่าผลลัพธ์ไม่เกิดโอเวอร์โฟลว์

นอกจากการใช้ในกรณีนี้แล้ว ในเรื่องการกระทำทางลอจิก เช่น AND, OR XOR, ผลลัพธ์ที่ได้จะได้รับการตรวจสอบว่ามีพาริตี้ คู่ หรือ คี่ โดยถ้าเป็นคู่ แฟลกนี้จะได้รับการเซทให้มีค่าเป็น "1"

5. แฟลกตัวทดช่วย (H:half carry) แฟลกนี้จะเป็นบิทที่ทำหน้าที่เป็นตัวทดหรือตัวยืมของตัวเลข BCD

6. แฟลกการลบ (N:subtract flag) เนื่องจากในการกระทำทางคณิตศาสตร์ของตัวเลข BCD เพื่อจะได้มีการรับรู้ในการปรับค่าเมื่อกระทำ DAA ได้ถูกต้อง แฟลกนี้จะเป็นตัวบอกว่าคำสั่งที่ถูกกระทำเป็นการบวกหรือลบ โดยถ้ากระทำคำสั่งลบแฟลกบิทนี้จะได้รับการเซทให้มีค่าเป็น "1"

กลุ่มคำสั่ง

คำสั่งของ Z-80 มีถึง 158 คำสั่ง โดยคำสั่งส่วนหนึ่งเป็นคำสั่งที่มีใน 8080 (78 คำสั่ง) คำสั่งเหล่านี้แบ่งออกเป็นกลุ่ม ๆ ได้ดังนี้

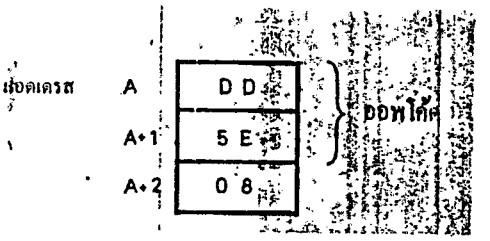
1. กลุ่มคำสั่งเกี่ยวกับการโหลดข้อมูลหรือแลกเปลี่ยนที่อยู่ข้อมูล
2. กลุ่มคำสั่งในการค้นหาข้อมูล และเคลื่อนย้ายข้อมูลเป็นกลุ่ม
3. กลุ่มคำสั่งในการกระทำทางคณิตศาสตร์และลอจิก
4. กลุ่มคำสั่งในการเลื่อนข้อมูลเป็นวงรอบและชิฟท์
5. กลุ่มคำสั่งในการทดสอบเซทและรีเซทบิทต่าง ๆ ของข้อมูล
6. กลุ่มคำสั่งในการอ้างถึงโปรแกรมย่อย
7. กลุ่มคำสั่งเกี่ยวข้องกับอินพุท-เอาท์พุท
8. กลุ่มคำสั่งควบคุมการทำงานของ ซีพียู

กลุ่มคำสั่งเกี่ยวกับการโหลดข้อมูลและแลกเปลี่ยนข้อมูล

การโหลดเป็นการเคลื่อนย้ายข้อมูลที่สำคัญ ใน Z-80 นั้น เราสามารถเคลื่อนย้ายข้อมูลได้อย่างมีประสิทธิภาพดีเพื่อทำความเข้าใจกลุ่มคำสั่งโหลดเราพิจารณาตัวอย่างต่าง ๆ ดังนี้

LD E, (IX + 08)

ชุดคำสั่งนี้ในหน่วยความจำจะเป็น



คำสั่งนี้เป็นการเคลื่อนย้ายข้อมูลจากหน่วยความจำแอดเดรสที่หาค่ามาจากค่าของ IX รวมกับตัวเลข 08 มาเก็บในรีจิสเตอร์ E

จะเห็นว่าเป็นคำสั่งขนาดเพียง 3 ไบท์เท่านั้น และจากคำสั่งขนาด 3 ไบท์เท่านั้น และจากคำสั่งขนาด 3 ไบท์ เราสามารถใช้วิธีการเพิ่มขนาดของแอดเดรส (extended addressing) ได้เช่น

LD A, (6F32H)

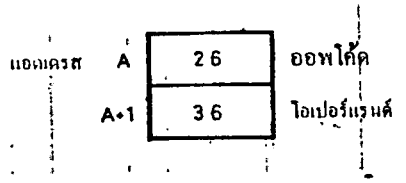
ชุดคำสั่งจะเป็น

ในกรณีนี้เป็นการโหลดข้อมูลในแอดเดรส 6F32H มาเก็บไว้ยังรีจิสเตอร์ A สิ่งที่น่าสังเกตคือ 32H จะอยู่ในไบท์ที่ 2 ส่วน 6FH จะอยู่ในไบท์ที่ 3

เรายังใช้วิธีการโหลดแบบอิมพีเดียทได้ดังนี้ เช่น

LD H, 36

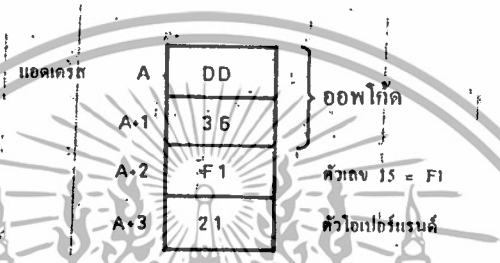
การวางรูปคำสั่งในหน่วยความจำจะเป็น



แต่การโหลดข้อมูลลงในหน่วยความจำที่มีความสามารถสูงวิธีหนึ่งคือ การโหลดใช้อินเดกรีจิสเตอร์เป็นตัวรับข้อมูล และใช้วิธีการอิมมีเดียทเป็นการกำหนดแหล่งเริ่มต้น (source) ของข้อมูล เช่น

LD (IX - 15), 21

การวางรูปคำสั่งในหน่วยความจำจะเป็นขนาด 4 ไบท์

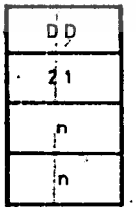


จากลักษณะคำสั่งนี้จะเป็นการเอาตัวเลข 21 ไปเก็บไว้ในหน่วยความจำตำแหน่งหน่วยความจำแอดเดรส IX - 15

เรายังสามารถใช้คำสั่งโหลดทำการโหลดข้อมูลในรูปแบบของการโหลดข้อมูล 16 บิท ได้อีกด้วย และเพื่อลดขนาดของคำสั่งให้สั้นขึ้น ตัวคำสั่งจึงเป็นลักษณะของการแลกเปลี่ยนข้อมูลของรีจิสเตอร์ต่าง ๆ

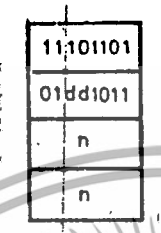
ตัวอย่างของคำสั่งการโหลดข้อมูลขนาด 16 บิท เช่น

LD IX, nn



เป็นคำสั่งที่ทำการโหลดตัวเลข nn ที่อยู่ในคำสั่งเข้าไปเก็บไว้ยังรีจิสเตอร์ IX หรือ IX nn เรายังโหลดข้อมูลจากหน่วยความจำทีละ 2 ไบท์ก็ได้ เช่น

LD dd, (nn)



dd
00 = BC
01 = DE
10 = HL
11 = SP

ในกรณีที่เราใช้รหัส dd เป็นรีจิสเตอร์ ซึ่งถ้าสมมุติว่า dd = 00 จะมีความหมายถึง การโหลดข้อมูลจากแอดเดรส nn และ nn + 1 โดยข้อมูลในแอดเดรส nn + 1 เก็บไว้ที่รีจิสเตอร์ B และข้อมูลในแอดเดรส nn เก็บไว้ที่รีจิสเตอร์ C

<u>กลุ่มคำสั่งในการค้นหาข้อมูลและเคลื่อนย้ายข้อมูลเป็นกลุ่ม</u>

Z-80 มีคำสั่งที่ทำให้การทำงานเป็นไปอย่างมีประสิทธิภาพโดยทำให้ลดขนาดของตัวโปรแกรมลงได้มาก มีลักษณะของกลุ่มคำสั่งในกลุ่มนี้จะอาศัยการทำงานรวมกันของคู่รีจิสเตอร์ ภายใน CPU ลงได้มาก มีลักษณะของกลุ่มคำสั่งในกลุ่มนี้จะอาศัยการทำงานรวมกันของคู่ รีจิสเตอร์ ภายใน CPU 3 คู่ คือ

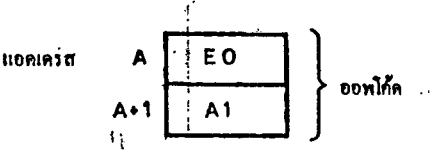
HL เป็นรีจิสเตอร์ที่อ้างตำแหน่งจุดต้นทาง

DE เป็นรีจิสเตอร์ที่อ้างถึงตำแหน่งจุดปลายทาง

BC เป็นตัวนับจำนวนไบท์

ในการค้นหาข้อมูล เราใช้วิธีการดังนี้ เช่น

CPI (compare with increment)



การทำงานจะเป็นไปได้ดังนี้ CPU จะนำข้อมูลจากหน่วยความจำที่มีแอดเดรสค่าอยู่ในคู่รีจิสเตอร์ HL มาเปรียบเทียบกับรีจิสเตอร์ A หลังจากนี้ค่าของคู่รีจิสเตอร์ HL จะเพิ่มค่าอีก 1 และค่าของคู่รีจิสเตอร์ BC จะลดค่าตัวเองลงไปทีละ 1 ลักษณะการเปรียบเทียบจะให้ผลลัพธ์ที่แฟลกจากคำสั่งนี้เราจะให้การเซทค่า BC เป็นจำนวนไบท์ที่ต้องการเปรียบค่าของ BC จะลดลงมาจนเป็น 0 แล้ว เราทดสอบค่า BC = 0 จะเห็นว่าทุกครั้งที่ทำคำสั่งนี้ค่าคู่รีจิสเตอร์ HL จะเพิ่มค่า ดังนั้นถ้าให้การกระทำคำสั่งนี้ใหม่ แอดเดรสในหน่วยความจำจะเพิ่มทีละหนึ่งเรื่อย ๆ ไป

เพื่อเพิ่มประสิทธิภาพของ Z-80 เรามีคำสั่งที่สามารถให้ กระทำคำสั่งจนครบตามจำนวนไบท์ที่วางไว้ เช่น

CPIR (Compare with increment and repeat)

การจัดวางคำสั่งในหน่วยความจำจะเป็น

การทำคำสั่งนี้จะเหมือนคำสั่ง CPI แต่ CPU จะเพิ่มการทดสอบค่าในคู่รีจิสเตอร์ BC ว่าเป็น 0 แล้วหรือยัง ถ้ายังก็จะกระทำคำสั่งเดิมนี้ซ้ำอีก ซึ่งค่าใน BC จะลดลงมาทีละ 1 จนถึงเป็น 0 จึงหยุดกระทำ

นอกจากนี้เรายังสามารถให้ค่าแอดเดรส ในคู่รีจิสเตอร์ HL ลดค่าลงทีละ 1 ได้เช่นกัน โดยใช้คำสั่ง CPD หรือ CPDR

สำหรับการเคลื่อนย้ายข้อมูลเป็นบล็อก (block) ก็กระทำเช่นเดียวกับการเปรียบเทียบโดยใช้คู่รีจิสเตอร์ HL เป็นตัวบอกแอดเดรสของหน่วยความจำที่ต้องการย้าย และคู่รีจิสเตอร์ DE เป็นตัวบอกว่าค่าแอดเดรสที่จะนำไปเก็บอยู่ที่ใด

การทำงานในลักษณะนี้ CPU นำข้อมูลในหน่วยความจำที่แอดเดรส (HL) ไปเก็บไว้ที่ในหน่วยความจำที่แอดเดรส (DE) และหลังจากนั้นค่าในรีจิสเตอร์ (BC) ลดลงไปหนึ่ง และค่าในคู่รีจิสเตอร์ HL และ DE จะเพิ่มค่าขึ้น 1 ในทำนองเดียวกันเราสามารถใช้ลำพังเพียงคำสั่งเดียวในการเคลื่อนย้ายข้อมูลทั้งบล็อกได้ด้วยการเอาจำนวนข้อมูลไว้ในคู่รีจิสเตอร์ BC แล้วใช้คำสั่ง LDIR การทำงานของคำสั่งนี้จะคล้ายกับ LDI แต่จะทำซ้ำ ๆ ซาก ๆไปเรื่อย ๆ จนกระทั่งค่าในคู่รีจิสเตอร์ BC เป็น 0 ก็จะหยุดไปกระทำคำสั่งถัดไป

กลุ่มคำสั่งชุดนี้เป็นกลุ่มคำสั่งที่ไม่มีใน 8080 แต่อย่างไรก็ตามเมื่อพิจารณาช่วงเวลาในการทำงานในแต่ละแมชีนไซเคิลดังได้กล่าวมาแล้วในบทก่อนจะเห็นว่า การกระทำคำสั่งที่กระทำซ้ำก็คือ การเฟทช์คำสั่งเดิมหลาย ๆ ครั้งนั่นเอง (ขอให้ย้อนกลับไปดูไดอะแกรมเวลาในการกระทำคำสั่งเหล่านี้)

<u>กลุ่มคำสั่งการกระทำทางคณิตศาสตร์และลอจิก</u>

คำสั่งนี้จะกระทำด้วยรีจิสเตอร์ A เป็นส่วนใหญ่ลักษณะของกลุ่มคำสั่งประกอบด้วยลักษณะการ ADD, SUB, ADC, SBC, INC, DEC, AND, OR, XOR ในการอ้างแอดเดรสของตัวโอเปอร์แรนด์ นั้นเราทำได้หลายแบบขึ้นอยู่กับผู้เขียนและผู้ใช้งาน

การกระทำในกลุ่มคำสั่งนี้จะให้ผลลัพธ์เก็บซ้ำที่ในรีจิสเตอร์ A และนอกจากนี้แล้วค่าของแฟลกจะมีผลต่อการกระทำของคำสั่งด้วย เช่น ถ้าค่าในรีจิสเตอร์ A เป็น 0 แฟลกตัวศูนย์ (Z flag) ก็ได้รับการเซทค่าไว้

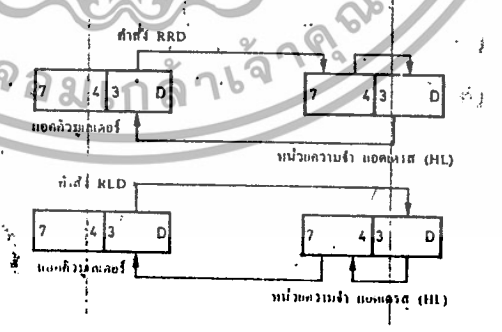
เป็นที่น่าสังเกตสำหรับการกระทำในกลุ่มคำสั่งนี้คือ เราแยกแยะการกระทำเป็นการกระทำทางคณิตศาสตร์ กลุ่มหนึ่ง (ADD, SUB) และการกระทำทางลอจิก (AND, OR, XOR) ซึ่งทั้งสองกลุ่มย่อยนี้จะให้ผลลัพธ์ที่มีผลต่อแฟลก กล่าวคือถ้าผลลัพธ์เป็นศูนย์ Z แฟลกจะมีค่าเป็น 1 ถ้าผลลัพธ์เป็นเลขลบ (บิท D_7 = 1) S แฟลกจะเป็น "1" ถ้ามีการทดก็จะให้ผลที่ C แฟลก แต่สำหรับ P/V แฟลกนี้ ในกรณีของกลุ่มคำสั่งทางคณิตศาสตร์แฟลกนี้ทำหน้าที่เป็นโอเวอร์โฟลว ในกรณีคำสั่งทางลอจิก แฟลกนี้ทำหน้าที่เป็นบิทพาริตี้ และที่น่าสังเกตอีกประการหนึ่งคือ คำสั่ง INC และ DEC จะไม่มีผลต่อแฟลกตัวทด

Z-80 ยังมีกลุ่มคำสั่งที่กระทำทางคณิตศาสตร์และลอจิกโดยเฉพาะบางอย่างเช่น การกระทำการเปลี่ยนรหัสตัวเลขไบนารีให้เป็นตัวเลข BCD ได้ด้วยคำสั่ง ADD คำสั่งเกี่ยวกับการทำคอมพลีเมนท์ การทำให้เป็นเลขลบ การเซทและรีเซทแฟลกบางตัว และยังมีคำสั่งในการควบคุมการทำงานของซีพียูในคำสั่ง NOP คือไม่ต้องทำอะไร HALT ให้หยุดโปรแกรม และนอกจากนี้ก็มีคำสั่งเกี่ยวกับอินเตอร์รัพท์ของซีพียู

การกระทำทางคณิตศาสตร์ของ Z-80 ยังสามารถกระทำในรูปการบวกลบตัวเลข 16 บิทได้อีกด้วย การบวกในกรณีนี้จะใช้การบวกระหว่างคู่รีจิสเตอร์ เช่น HL กับ BC เป็นต้น ในการบวกเช่นนี้จะทำให้การกระทำทางคณิตศาสตร์มีประสิทธิภาพเพิ่มขึ้น

กลุ่มคำสั่งในการเลื่อนข้อมูลเป็นวง (rotate) และการชิฟท์ (shift)

ความสามารถพื้นฐานของ Z-80 ในการทำข้อมูลและชิฟท์ก็เหมือนกันใน CPU ของคอมพิวเตอร์ทั่วไป เช่น มักจะเลื่อนบิทไปทางซ้ายไปทางขวาเลื่อนเป็นวงรอบ โดยผ่านบิทหดหรือไม่ผ่านบิทหด แต่ Z-80 มีคำสั่งเกี่ยวกับการเลื่อนข้อมูลตัวเลข BCD อยู่ 2 คำสั่ง คือ RRD และ RLD ทั้ง 2 คำสั่งนี้จะทำให้ข้อมูล 1 ตัวเลข (BCD) ในแอคคิวมูเลเตอร์เลื่อนเป็นวงรวมกับข้อมูล 2 ตัวเลข (BCD) ในหน่วยความจำที่อ้างแอดเดรสโดย HL ลักษณะของการกระทำเป็นดังนี้



การกระทำในกลุ่มคำสั่งนี้เกิดขึ้นภายใน ALU โดยมีการกระทำร่วมกับแฟลก ดังนั้นจึงมีผลเกี่ยวข้องโดยตรงกับแฟลกบางตัว เช่น แฟลกการหด แฟลกเครื่องหมายและแฟลกศูนย์เป็นต้น

กลุ่มของคำสั่งเกี่ยวกับการกระทำภายในข้อมูลบิทต่าง ๆ

ใน Z-80 มีความสามารถพิเศษในการ เซท รีเซท หรือทดสอบบิทใดบิทหนึ่งใน รีจิสเตอร์ต่าง ๆ และในหน่วยความจำได้รายละเอียดของการ เซท รีเซท และการทดสอบ สามารถแยกเป็นคำสั่งย่อย ๆ ในกลุ่มนี้ได้มาก เพราะแต่ละคำสั่งจะเท่ากับมีคำสั่งย่อย ๆ ได้ถึง 8 คำสั่ง คือการกระทำในแต่ละบิทต่าง ๆ ของรีจิสเตอร์ต่างนั่นเอง

ในการทดสอบบิทต่าง ๆ ว่าเป็น 0 หรือ 1 นั้น เราใช้แฟลก 0 เป็นตัวเก็บข้อมูล เช่น ถ้าบิทที่ทดสอบเป็น 0 ก็จะเซทแฟลก 0 ให้เป็น 1

ดังตัวอย่างเช่น BIT b, r หมายถึงการทดสอบบิทที่ b ของรีจิสเตอร์ r ว่าเป็น "0" หรือเป็น "1" วิธีการก็เป็นเพียงการโหลดข้อมูลของบิทที่ b ของรีจิสเตอร์ r มาไว้ยัง Z แฟลกนั้นก็ทำการทดสอบได้ว่า Z แฟลกจะเป็น "0" หรือ "1"

ในทำนองเดียวกัน เราสามารถทำการ เซทและรีเซทบิทใดบิทหนึ่งในรีจิสเตอร์ และหน่วยความจำต่าง ๆ ได้ด้วยคำสั่ง SET b, r หรือ RES b, r

การกระทำในกรณีนี้เรากระทำได้ในทุกรีจิสเตอร์ ในหน่วยความจำ การอ้างแอดเดรสไปยังหน่วยความจำก็ทำได้ ทั้งใช้แบบทางอ้อมผ่านรีจิสเตอร์ HL หรือใช้อินแดครีจิสเตอร์ IX, IY ร่วมด้วย

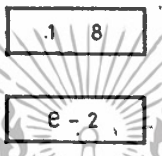
กลุ่มคำสั่งในการกระโดด

Z-80 มีกลุ่มคำสั่งเกี่ยวกับกระโดดที่มีประสิทธิภาพ คือ คำสั่ง JP และ JR การกระโดดไปยังที่ใดก็ได้ในที่นี้ก็คือ การโหลดเปลี่ยนค่าโปรแกรมเคาน์เตอร์นั่นเอง เช่น JP nn ก็คือ การโหลดข้อมูล nn เข้าไปยัง PC

การกระโดดของคำสั่ง JP ยังสามารถตรวจสอบเงื่อนไขก่อน ถ้าเป็นจริงจึงมีการโหลด PC ด้วยค่า nn แต่ถ้าเงื่อนไขไม่เป็นจริง ค่า PC จะเพิ่มตามปกติ คือซีพียูจะกระทำคำสั่งถัดไปอย่างปกติโดยไม่มีการกระโดด เช่น JP NZ, nn หมายถึงว่าถ้า Z แฟลก

เป็น "0" PC จะมีค่าเป็น nn แต่ถ้า z แฟลกเป็น "1" ค่า PC จะมีค่าเพิ่มตามปกติ คือ PC+3

การ JP ของกลุ่มคำสั่งที่มีพิเศษนอกเหนือจากที่ 8080 มีคือ JR หรือการกระโดดไปยังแอดเดรสที่อ้างเปรียบเทียบกับค่า PC เดิมของมัน ลักษณะคำสั่งนี้มีข้อดีคือใช้จำนวนไบท์น้อยกว่า คือใช้เพียง 2 ไบท์เท่านั้น ดังเช่น JR e โดยลักษณะของออพโค้ดจะเป็น



การกระทำในกรณีนี้ ค่า PC ใหม่จะเป็น PC + e

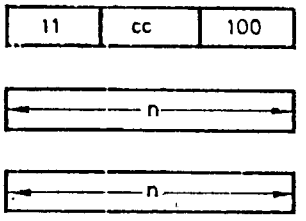
<u>กลุ่มคำสั่งที่อ้างกับโปรแกรมย่อย</u>

เพื่อให้การเรียกโปรแกรมย่อยมีประสิทธิภาพสูง Z-80 ใช้วิธีการเรียกโดยคำสั่ง CALL และ RET คำสั่ง CALL จะกระทำให้ค่า PC เดิมที่มีอยู่ไปเก็บไว้ที่สแตคแล้วโหลดค่าแอดเดรสของโปรแกรมย่อยกลับคืนมาให้ PC และในการ RET ก็จะกระทำกลับกันนั่นคือ จะเป็นการ POP เอาข้อมูลในสแตคมาให้ PC

นอกจากนี้ตัว Z-80 จะมีคำสั่งพิเศษ คือ

CALL cc, nn

ซึ่งเขียนเป็นรูปคำสั่งในหน่วยความจำจะเป็น



คำสั่งนี้จะเป็นการทดสอบเงื่อนไขใน cc ถ้าเงื่อนไขเป็นจริงการกระทำ CALL ก็จะเกิดขึ้น คือเครื่องจะ PUSH ข้อมูลใน PC ไปไว้ในสแตค แล้วนำค่า nn ไปไว้ใน PC ถ้าเงื่อนไขใน cc ไม่เป็นจริง CPU ก็จะกระทำคำสั่งถัดไป การใช้รหัสแทนเงื่อนไข 3 บิท ทำให้สามารถแทนเงื่อนไขได้กับ 8 แบบ ดังนี้

cc	เงื่อนไข	ดูจากแฟลก
000	ไม่เป็นศูนย์	X
001	เป็นศูนย์	X
010	ไม่มีตัวทด	C
011	มีตัวทด	C
100	พาริตี้ เป็นคี่	P
101	พาริตี้ เป็นคู่	P
110	มีเครื่องหมายบวก	S
111	มีเครื่องหมายลบ	S

ในทำนองเดียวกัน Z-80 ก็มีคำสั่งที่ใช้ RET ทั้งแบบไม่มีเงื่อนไข และแบบมีเงื่อนไข เช่น

RET cc

คือเมื่อ cc เป็นจริงจึงจะกระทำการกระโดดกลับโปรแกรมหลัก ถ้า cc ไม่เป็นจริงก็จะกระทำคำสั่งถัดไป

นอกจากการเรียกโปรแกรมย่อยแล้ว Z-80 ยังมีคำสั่งในลักษณะ 8080 แบบต่าง ๆ อีกซึ่งอาจเป็นแบบมีเงื่อนไขหรือไม่มีเงื่อนไข

จากลักษณะของการ RETURN มีสองคำสั่งที่ไม่มีใน 8080 คือ RETI และ RETN ในกรณีนี้จะเป็นการคำรงสถานะบางอย่างเกี่ยวกับ IFF ซึ่งจะได้กล่าวในรายละเอียดเรื่องเกี่ยวกับการอินเตอร์รัพท์

การ CALL และ RETURN ก็เหมือนคับคำสั่ง JP แต่ต่างกันตรงที่มีการเก็บข้อมูล PC เดิมไว้ที่สแตค ในกรณีนี้เรายังมีการ CALL ที่ใช้คำสั่งไบท์เดียวได้ คือ RST P ซึ่งเป็นคำสั่งที่มีรหัสเพียง 1 ไบท์ คือ

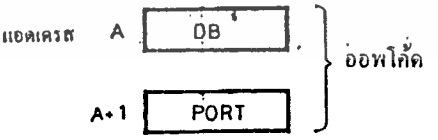
เป็นการเก็บรักษาค่า PC เดิมไว้ในสแตค และโหลดค่าแอดเดรสใหม่ให้กับ PC โดยค่าแอดเดรสใหม่จะเป็นไปตามตาราง

<u>กลุ่มคำสั่งเกี่ยวกับการติดต่อระหว่างอินพุท-เอาท์พุทกับซีพียู</u> ใน 8080 กลุ่มคำสั่งนี้มีเพียง 2 คำสั่งเท่านั้น คือ IN และ OUT แต่ในของ Z-80 ได้เพิ่มขยายการติดต่อกับอินพุท เอาท์พุทมากยิ่งขึ้น ทำให้ประสิทธิภาพการติดต่อไปเป็นได้ดียิ่งขึ้น

คำสั่งที่ซ้ำกับ 8080 คือ

IN A, PORT

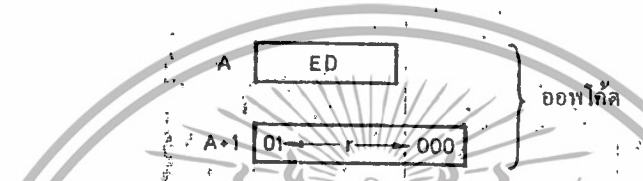
ลักษณะการจัดรูปโครงสร้างคำสั่งในหน่วยความจำเป็น



การกระทำคำสั่งนี้ จะเป็นการนำข้อมูลจากพอร์ทที่เป็นอินพุทโหลดข้อมูลมาที่รีจิสเตอร์ สิ่งที่เพิ่มมาของ Z-80 คือ มันสามารถเลือกการโหลดให้กับข้อมูลไว้ที่รีจิสเตอร์ใดก็ได้ เช่น

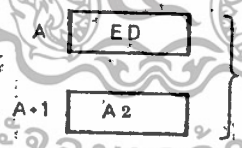
IN r, (C)

การจัดรูปคำสั่งในหน่วยความจำเป็น



ในกรณีนี้ ซีพียู จะใช้รีจิสเตอร์ C เป็นตัวกำหนดชื่อพอร์ทแล้วโหลดข้อมูลจากอินพุทมาเก็บไว้ที่รีจิสเตอร์ในซีพียูที่กำหนดด้วย 3 บิทในออปโค้ด

การโหลดข้อมูลจาก I/O สามารถทำเป็นบลอกได้เช่นกัน โดยใช้คำสั่ง INI, INIR, INI



ในกรณีที่จะใช้รีจิสเตอร์ C เป็นตัวกำหนดชื่อพอร์ทและใช้รีจิสเตอร์ B เป็นตัวนับข้อมูลโดยค่าของรีจิสเตอร์ B จะลดลงมา 1 ส่วนรีจิสเตอร์ HL จะเป็นตัวชี้ตำแหน่งในหน่วยความจำที่จะนำข้อมูลจากอินพุทมาเก็บไว้ จะเห็นว่าถ้าเราให้ทำคำสั่งนี้ซ้ำ ๆ ค่ารีจิสเตอร์ B จะลดค่าลงมาเรื่อย ๆ ทำให้การเก็บข้อมูลเรียงกันไปได้

สำหรับ Z-80 เรามีคำสั่ง INIR ซึ่งสามารถกระทำคำสั่ง INI ซ้ำจนกระทั่งค่ารีจิสเตอร์ B มีค่าเป็น 0 จึงหยุด ทำให้เราสามารถโหลดข้อมูลจากอินพุทเป็นบล็อกได้ถึง 256 ไบท์ ส่วนทางด้านเอาท์พุทก็เช่นเดียวกัน

บทที่ 5

การใช้ 8255 PIA กับ Z-80

รายละเอียดเกี่ยวกับ 8255

8255 เป็นอุปกรณ์ LSI (LARGE SCALE INTEGRATED CIRCUIT) บรรจุอยู่ใน PACKAGE 40 ขาแบบ DIP (DUAL-IN-LINE PACKAGE) เริ่มผลิตโดยบริษัท INTEL COORPERATION ผู้ผลิตไมโครโปรเซสเซอร์เบอร์ 8080 จุดประสงค์เพื่อใช้งานร่วมกับ 8080 โดยเฉพาะแต่ในภายหลังได้มีการนำ 8255 ไปประยุกต์ใช้งานร่วมกับไมโครโปรเซสเซอร์เบอร์อื่น ๆ รวมทั้ง Z-80 ด้วย หากผู้อ่านเข้าใจการใช้งาน 8255 กับ Z-80 ที่จะกล่าวถึงในบทนี้แล้วก็จะนำไปประยุกต์ใช้งานในลักษณะอื่น ๆ ได้ไม่ยากนัก

รูป 5.1 นี้แสดงบล็อกไดอะแกรม ของ 8255 ซึ่งหน้าที่ของแต่ละบล็อกมีดังต่อไปนี้คือ

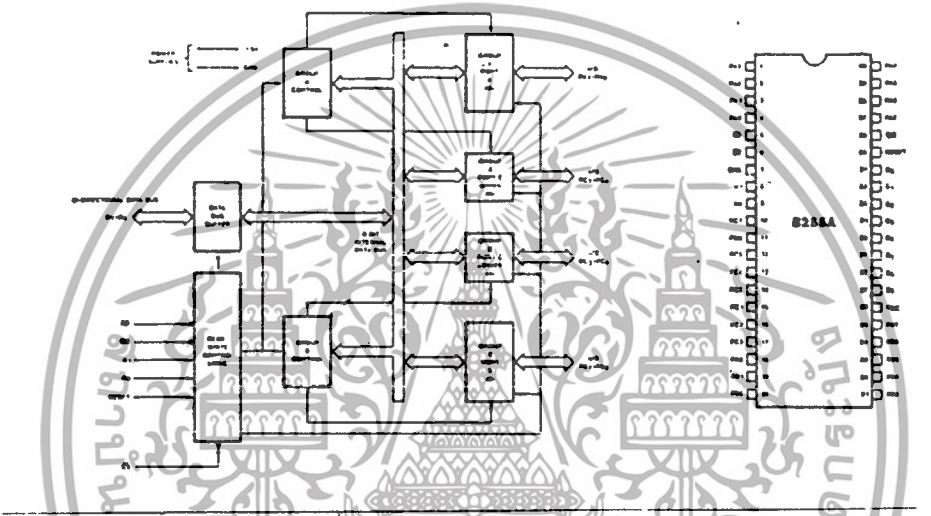
บล็อกกลุ่มแรกที่เราจะพูดถึงนี้ ได้แก่ บล็อกจำนวน 4 บล็อก ที่อยู่ทางด้านขวาของรูป ซึ่งจะเป็นส่วนที่เชื่อมต่อกับอุปกรณ์ภายนอกอื่น ๆ โดยมีสาย PA0-PA7, PB0-PB7 และ PC0-PC7 เป็นทางผ่านของข้อมูลระหว่างอุปกรณ์ภายนอกกับ 8255 สายสัญญาณเหล่านี้จะถูกแบ่งออกเป็น 3 I/O พอร์ทได้แก่ พอร์ท A (PA), พอร์ท B (PB) และพอร์ท C (PC) พอร์ทเหล่านี้แต่ละพอร์ทสามารถเป็นได้ทั้งพอร์ทอินพุทและเอาท์พุท และแต่ละบล็อกจะมีสายสัญญาณเชื่อมเข้ากับบัสข้อมูลภายในของ 8255

บล็อกกลุ่มถัดมา ได้แก่ GROUP A CONTROL และ GROUP B CONTROL ซึ่งจะเป็นตัวกำหนดลักษณะการทำงานของทั้ง 3 I/O พอร์ท (8255 ลักษณะการทำงานที่แตกต่างกันอยู่ 3 โหมด สามารถกำหนดได้โดยการโปรแกรมส่ง CONTROL WORD ให้กับ 8255 ซึ่งจะกล่าวถึงในภายหลัง) จากรูป 5.1 จะเห็นว่า พอร์ท C นี้จะประกอบด้วยพอร์ทขนาด 4 บิท 2 พอร์ท กลุ่มหนึ่งจะถูกควบคุมโดย GROUP A CONTROL และอีกกลุ่มหนึ่งจะถูกควบคุมโดย GROUP B CONTROL สำหรับเหตุผลนั้นจะกล่าวถึงในภายหลัง

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel® Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40-Pin Dual In-Line Package
- Reduces System Package Count
- Improved DC Driving Capability

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.



รูปที่ 5.1 แสดงบล็อกไดอะแกรมและการวางตำแหน่งขาของ 8255

บล็อกกลุ่มสุดท้ายที่จะกล่าวถึง ได้แก่ DATA BUS BUFFER และ READ/WRITE CONTROL LOGIC ซึ่งบล็อกเหล่านี้จะเป็นส่วนที่ติดต่อกับ CPU, DATA BUS BUFFER นี้จะเป็นบัฟเฟอร์ให้กับบัสข้อมูลของ CPU ส่วน READ/WRITE CONTROL LOGIC จะเป็นส่วนที่ควบคุมให้ข้อมูลเข้าหรือออกจากรีจิสเตอร์ภายใน ตัวที่ถูกต้อง และในเวลาที่เหมาะสม

รายละเอียดการจัดเรียงขาของ 8255

ในส่วนนี้เราจะพิจารณาหน้าที่ของขาแต่ละขาของ 8255 ซึ่งข้อมูลเหล่านี้จะมีประโยชน์ในการเชื่อมต่อเข้ากับระบบบัสของ CPU สำหรับการจัดขาแสดงไว้ในรูปที่ 5.1 รายละเอียดของแต่ละขามีดังนี้ คือ

D0-D7 : เป็นสายข้อมูลอินพุท/เอาท์พุทแบบสองทิศทาง (BI-DIRECTIONAL BUS) จะเป็นทางผ่านของข้อมูลระหว่างพอร์ทต่าง ๆ ของ 8255 กับบัสข้อมูลของ Z-80

$\overline{CS}$ (CHIP SELECT INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" CPU จะสามารถที่จะอ่านหรือเขียนข้อมูลกับ 8255 ได้

$\overline{RD}$ (READ INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" และสัญญาณ $\overline{CS}$ มีลอจิกเป็น "0" ข้อมูลจาก 8255 จะปรากฏสู่ระบบบัสข้อมูล CPU ก็จะสามารถอ่านข้อมูลออกไปได้ (ในการตั้งชื่อของขาสัญญาณนี้จะถือเอา CPU เป็นหลัก)

$\overline{WR}$ (WRITE INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" และขาสัญญาณ $\overline{CS}$ มีลอจิกเป็น "0" ข้อมูลจากระบบบัสข้อมูลจะถูกเขียนเข้าไปยัง 8255 ได้

A0-A1 (ADDRESS INPUT) : จะเป็นตัวกำหนดการเลือกใช้รีจิสเตอร์ภายในของ 8255 ซึ่งจะกล่าวรายละเอียดในภายหลัง

RESET : เมื่อขานี้มีสถานะเป็น "1" 8255 จะอยู่ในสภาวะรีเซ็ทพุก ๆ พอร์ทของ 8255 จะถูกเซ็ทให้อยู่ในโหมดอินพุท

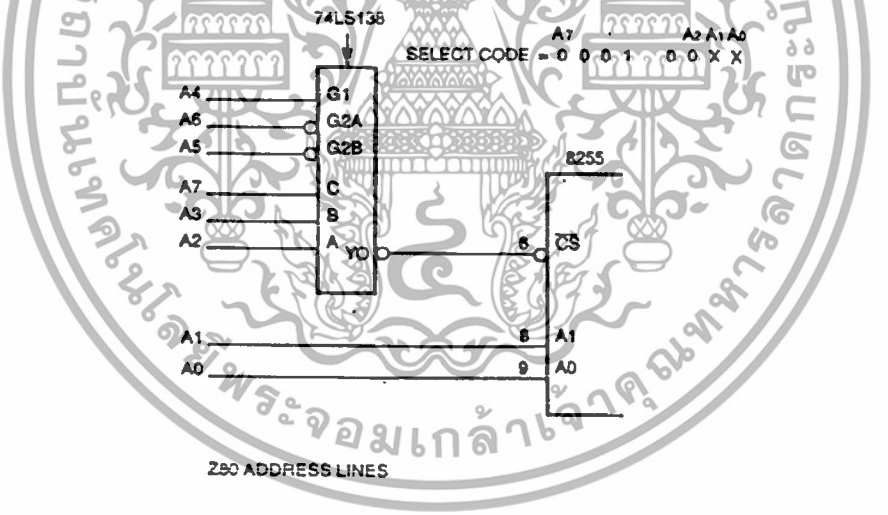
PA0-PA7,PB0-PB7 : ขาสัญญาณเหล่านี้จะถูกใช้เป็นพอร์ท I/O ขนาด 8 บิท ใช้ต่อเข้ากับอุปกรณ์ภายนอกอื่น ๆ

PC0-PC7 : ขาสัญญาณนี้ถูกใช้เป็นพอร์ท I/O ขนาด 8 บิท เช่นเดียวกับ PA0-PA7 และ PB0-PB7 แต่กลุ่มของขาสัญญาณเหล่านี้สามารถแบ่งออกเป็น 2 กลุ่ม โดยแต่ละกลุ่มมีขนาด 4 บิทได้ กลุ่มแรกจะใช้ควบคุม PB0-PB7 และกลุ่มที่ 2 ใช้ควบคุม PA0-PA7 (ซึ่งจะกล่าวถึงรายละเอียดในภายหลัง)

การต่อ 8255 เข้ากับ Z-80

ในการต่อ 8255 เข้ากับระบบของ Z-80 นั้น สัญญาณต่าง ๆ ที่เกิดขึ้นจะเหมือนกับขบวนการติดต่อกับ I/O ดังที่เคยกล่าวมาแล้ว โดยจะต้องเอาสัญญาณ A0-A7 จาก Z-80 มาถอดรหัสเพื่อสร้างสัญญาณเลือกพอร์ท แต่เนื่องจาก 8255 มีขา ADDRESS INPUT อยู่แล้ว 2 ขา (A0,A1) ซึ่งโดยปกติแล้ว ขา A0,A1 นี้จะต่อเข้าโดยตรงกับ A0,A1 จากบัสแอดเดรส นั่นคือ 8255 หนึ่งตัวจะใช้ค่าพอร์ทแอดเดรสถึง 4 ค่า (2^2) ส่วนสัญญาณอีก 6 เส้น (A2-A7) จะนำไปถอดรหัสเพื่อทำสัญญาณเลือกชิพ (CHIP SELECT) ให้แก่ 8255

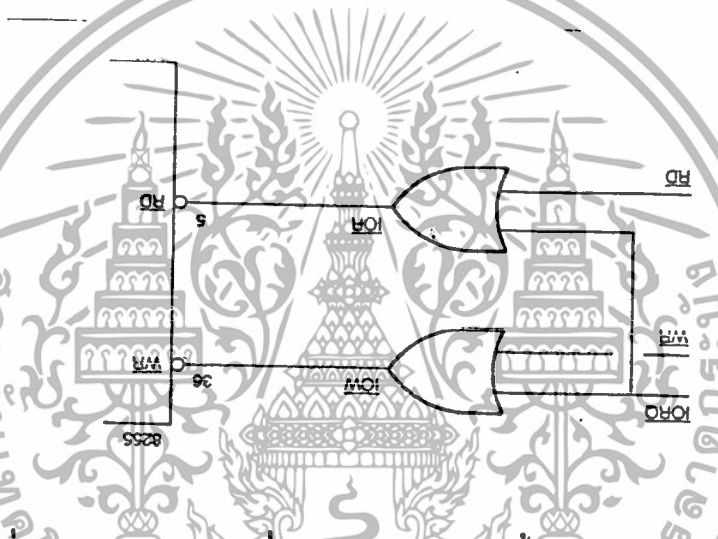
ในที่นี้เราจะสมมติให้ 8255 มีพอร์ทแอดเดรสอยู่ที่ 10H, 11H, 12H และ 13H ซึ่งวิธีหนึ่งที่สามารถจะถอดรหัสพอร์ทเหล่านี้ได้ แสดงไว้ในรูป 5.2



รูปที่ 5.2 แสดงผังวงจรการถอดรหัสการเลือกพอร์ทที่ติดต่อกับ 8255

จากรูปที่ 5.2 นี้ จะเห็นว่าขาอินพุท $\overline{CS}$ จะแอคทีฟก็ต่อเมื่อ A7-A2 มีเท่ากับ 000100XXB (2 บิทล่างจะใช้เพื่อเลือกใช้รีจิสเตอร์ภายใน 4 ตัว)

ขั้นต่อไปที่เราจะต้องทำคือ การต่อขา $\overline{RD}$ และ $\overline{WR}$ ของ 8255 เข้ากับสัญญาณควบคุม $\overline{IOR}$ และ $\overline{IOW}$ของระบบ การที่เราไม่ต่อขา $\overline{RD}$ และ $\overline{WR}$ เข้าโดยตรง เพราะในตัวอย่างวิธีการถอดรหัสของเรานี้ อาจจะเกิดกรณีที่ A7-A0 มีค่าตรงกับ 000100XXB ซึ่งจะทำให้เกิดการอ่านหรือเขียนข้อมูลกับ 8255 โดยไม่ต้องการได้ ในการแก้ปัญหานี้เราจึงใช้สัญญาณ $\overline{IORQ}$ จากCPU มาทำเป็นสัญญาณ $\overline{IOR}$ และ $\overline{IOW}$ เพื่อแยกว่าเป็นการติดต่อกับ I/O ไม่ใช่หน่วยความจำ ดังแสดงในรูป 5.3



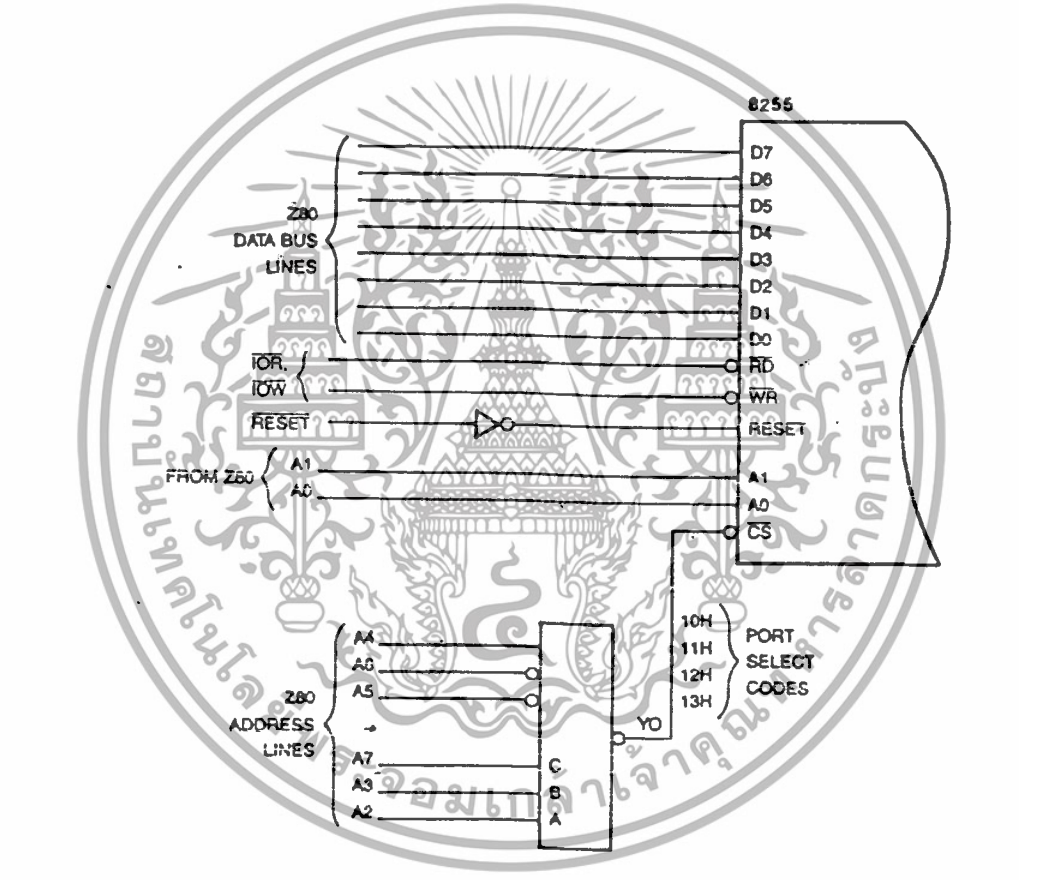
รูปที่ 5.3 แสดงวิธีการต่อขา $\overline{WR}$ และ $\overline{RD}$ เข้ากับระบบของ Z-80

ในการต่อขา RESET ของ 8255 ซึ่งจะแอคทีฟที่ลอจิก "1" เข้ากับขา $\overline{RESET}$ ของ Z-80 ซึ่งแอคทีฟที่ลอจิก "0" นั้นจะต้องใช้ INVERTER คั่นกลางเสียก่อน

ในการต่อสายข้อมูล D0-D7 ของ 8255 เข้ากับระบบบัสข้อมูลของระบบ เราจะสมมติว่าไม่มีการโหลดบนบัสข้อมูล ดังนั้นเราจึงสามารถต่อสายสัญญาณเหล่านี้เข้าโดยตรงกับระบบบัสข้อมูล ดังแสดงวงจรสมบูรณ์ของการเชื่อมต่อ 8255 เข้ากับระบบของ Z-80 ในรูป 5.4

8255 READ และ WRITE REGISTER

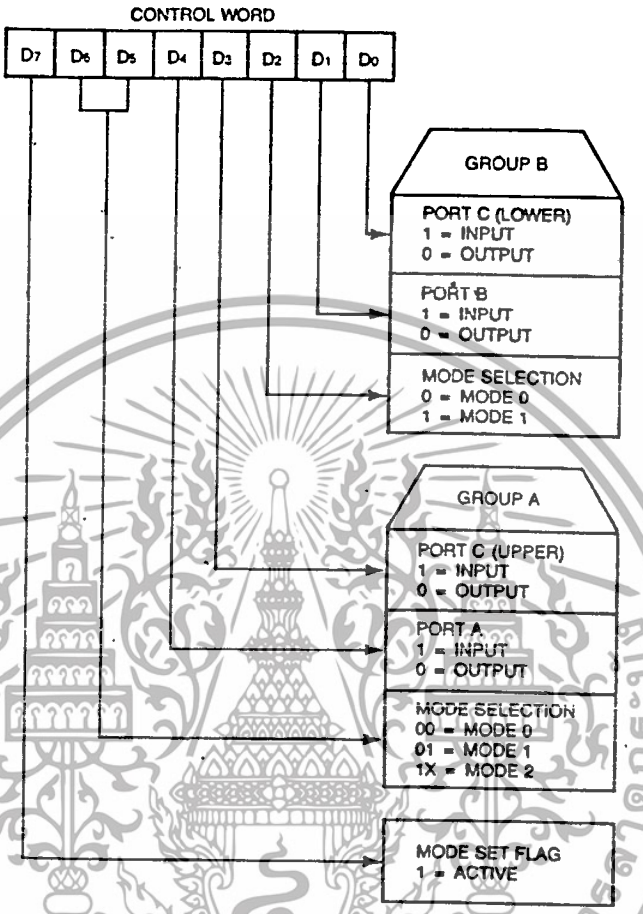
ขณะนี้เราได้ทำการต่อ 8255 เข้ากับระบบของ z-80แล้ว ต่อไปเราจะศึกษาการโปรแกรมใช้งาน 8255 เพื่อที่จะให้ทำงานตามที่เราต้องการได้ จะเริ่มต้นพิจารณาที่รีจิสเตอร์ภายใน 4 ตัว ของ 8255 สำหรับในตัวอย่างการถอดรหัสของเรานี้ ตำแหน่งของรีจิสเตอร์จะอยู่ที่แอดเดรส 10H, 11H, 12H และ 13H ซึ่งรายละเอียดของรีจิสเตอร์เหล่านี้มีดังนี้คือ



รูปที่ 5.4 แสดงผังวงจรสมบูรณ์ของการเชื่อมต่อ 8255 เข้ากับระบบของ z-80

DEVICE PIN				REGISTER NAME
RD	WR	A1	A0	
1	0	0	0	WRITE PORT A DATA
0	1	0	0	READ PORT A DATA
1	0	0	1	WRITE PORT B DATA
0	1	0	1	READ PORT B DATA
1	0	1	0	WRITE PORT C DATA
0	1	1	0	READ PORT C DATA
1	0	1	1	WRITE CONTROL DATA
0	1	1	1	ILLEGAL READ REGISTER

หน้าที่ของรีจิสเตอร์หมายเลข 0-2 จะถูกกำหนดลักษณะการทำงานจากรีจิสเตอร์หมายเลข 3 (รีจิสเตอร์ควบคุม) รูปที่ 5.5 จะแสดงรายละเอียดของแต่ละบิทของรีจิสเตอร์ควบคุมนี้ต่อไป เราจะกล่าวถึงลักษณะการทำงานของ 8255 ทั้ง 3 โหมด และการโปรแกรมให้อยู่ในโหมดต่าง ๆ ได้ดังต่อไปนี้ คือ



รูปที่ 5.5 แสดงรายละเอียดแต่ละบิทของรีจิสเตอร์ควบคุมของ 8255

โหมด 0 : BASIC REGISTER I/O

ในการเซ็ท 8255 ให้อยู่ในโหมด 0 นั้น เราจะต้องส่งคำสั่งควบคุม (CONTROL WORD) ให้แก่รีจิสเตอร์ควบคุมก่อนคำสั่งควบคุมนี้จะกำหนดลักษณะการทำงานให้แก่แต่ละพอร์ทของ 8255 ตัวอย่างหนึ่งของคำสั่งควบคุมที่จะสั่งให้ 8255 ทำงานอยู่ในโหมด 0 นี้ ได้แก่

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

จากรูปที่ 5.5 เราจะเห็นว่า

บิท D7 เป็นตัวกำหนดว่าเป็นคำสั่งควบคุม (CONTROL WORD)

บิท D6 และ D5 กำหนดโหมดการทำงานของ พอร์ท A D6,D5 มีค่าเป็น "0" แสดงว่าอยู่ในโหมด 0

บิท D4 "0" กำหนดให้ พอร์ท A เป็นพอร์ทเอาท์พุท

บิท D3 "0" เซ็ทพอร์ท C 4 บิทบนเป็นพอร์ทเอาท์พุท

บิท D2 "0" เซ็ทโหมดของพอร์ท B ให้พอร์ท อยู่ในโหมด 0

บิท D1 "0" เซ็ทพอร์ท B เป็นพอร์ทเอาท์พุท

บิท D0 "0" เซ็ทพอร์ท C ให้ 4 บิทล่างเป็นพอร์ทเอาท์พุท

คำสั่งควบคุมนี้จะกำหนดให้พอร์ทั้ง 3 ของ 8255 ทำงานอยู่ในโหมด 0 และเป็นพอร์ทเอาท์พุทซึ่งจะได้สายสัญญาณซึ่งสามารถติดต่อกับอุปกรณ์ภายนอกได้ถึง 24 สาย คำสั่งของ Z-80 ที่จะเซ็ทให้ 8255 อยู่ในลักษณะดังกล่าวได้แก่

```
LD A, 32H     : เซ็ทข้อมูลให้พอร์ท A
OUT (10H), A  : ส่งข้อมูลให้พอร์ท A
LD A, 41H     : เซ็ทข้อมูลให้พอร์ท B
OUT (11H), A  : ส่งข้อมูลให้พอร์ท B
LD A, 73H     : เซ็ทข้อมูลให้พอร์ท C
OUT (12H), A  : ส่งข้อมูลให้พอร์ท C
```

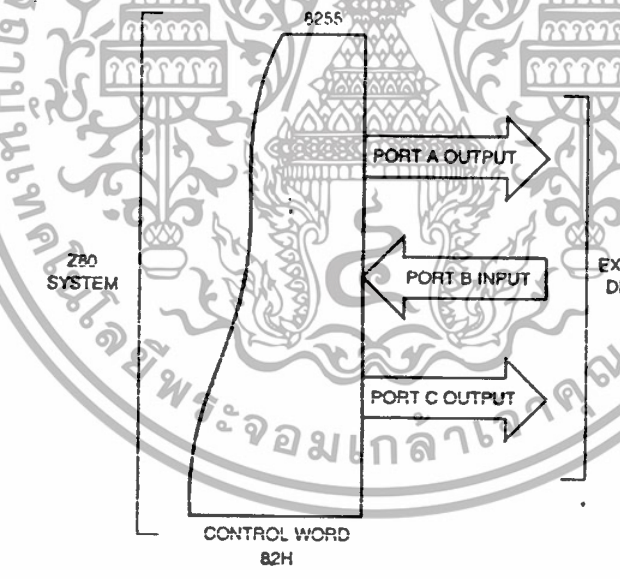
หลังจากที่คำสั่งเหล่านี้ถูก EXECUTE แล้วพอร์ท A, B และ C ของ 8255 จะมีข้อมูลต่าง ๆ ที่ส่งไปให้ปรากฏอยู่

ในการทำงานในโหมด 0 ของ 8255 นี้อาจจะสั่งให้พอร์ทของ 8255 เป็นอินพุทหรือเอาท์พุทก็ได้ อย่างเช่น ให้พอร์ท A และพอร์ท C เป็นพอร์ทเอาท์พุทและพอร์ท B เป็นพอร์ทอินพุท เราจะต้องส่งคำสั่งควบคุมให้แก่รีจิสเตอร์ควบคุมในลักษณะดังนี้คือ

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0

หลังจากที่ส่งคำสั่งควบคุมให้แก่รีจิสเตอร์ควบคุมแล้ว 8255 จะถูกเซ็ทให้มีลักษณะการทำงานดังรูป 5.6 เราจะใช้คำสั่ง IN อ่านข้อมูลมาจากพอร์ท B ได้

IN A, (11H) : อ่านข้อมูลจากพอร์ท B



รูปที่ 5.6 บล็อกไดอะแกรมแสดงลักษณะการทำงานของ 8255 ในโหมด 0 หลังจากส่งคำสั่งควบคุมให้ 8255 แล้ว

บทที่ 6

การออกแบบและการทำงาน

ในการออกแบบเครื่อง AUTOMATIC LOADER IC นี้ เราจะนำไปใช้กับเครื่องที่ทำงานเกี่ยวกับไอซี โดยให้ไอซีบ่านขบวนการได้ทุกเครื่อง เพราะฉะนั้น เครื่องที่ออกแบบนี้จึงสามารถนำไปใช้งานกับเครื่องที่เกี่ยวข้องได้อย่างกว้างขวางเช่น เครื่องคัดชาไอซี, เครื่องพิมพ์เบอร์ไอซี และเครื่องอื่น ๆ

เครื่อง AUTOMATIC LOADER IC แบ่งได้เป็น 2 ส่วนคือ ส่วนการปล่อยไอซี (Output Shuttle) และส่วนของการรับไอซี (Input Shuttle) ซึ่งแต่ละส่วนมีลำดับขั้นการทำงานดังนี้

การทำงานของส่วนรับไอซี (Input Shuttle)

1. เริ่มแรกจะเป็นการเซ็ทข้อมูลต่าง ๆ เช่น เซ็ทการทำงานของ PPI-8255 เพื่อให้ PORT A เป็น OUTPUT PORT และ PORT B และ C เป็น INPUT PORT แล้ว CLEAR OUTPUT PORT

2. ON-MAN & DIS เพื่อให้ผู้ใช้ทราบว่าขณะนี้เครื่องเริ่มทำงานแล้ว

3. OFF-MAN & DIS เพื่อให้ผู้ใช้ทราบว่าขณะนี้เครื่องกำลังทำงานจะไม่รับทราบข้อมูลที่ป้อนให้ขณะนี้

4. SCAN คีย์ เพื่อตรวจสอบว่ามีการกดคีย์สั่งงานอะไรให้กับเครื่องหรือไม่ โดยจะป้องกันการดีเบลาร์ของสวิทช์ด้วยการ DELAY การรับค่าไว้เป็นเวลา 10 ms เพื่อให้แน่ใจว่าเป็นการกดคีย์จริง ไม่ใช่สัญญาณ transient จากที่อื่นเข้ามารบกวน

โดยจะแยกคีย์ไว้ 3 คีย์คือ

4.1 CK VALVE เพื่อตรวจสอบการทำงานของ VALVE แต่ละตัวโดย ON และ OFF VALVE แต่ละตัว เป็นเวลาตัวละ 30 ms แล้ว ON-DIS LIGHT พร้อม ๆ กันด้วย

4.2 DISABLE (DIS) เพื่อหยุดการทำงานของเครื่องไว้ชั่วขณะกรณีที่เกิดการผิดพลาดขึ้นขณะทำงาน โดยใช้ SW แบบกดติดกดดับ (ใช้ไฟสีแดง)

4.3 MANUAL (MAN) เพื่อใช้สำหรับการ RUN PROCESS แบบธรรมดา คือ ผู้ปฏิบัติงานคอยกดคีย์เองทีละครั้ง เมื่อครบ Function การทำงาน (ใช้ไฟสีเหลือง)

5. RETUBE ทำหน้าที่ควบคุมให้ VALVE ลมต่าง ๆปิดเปิดตามจังหวัดที่เหมาะสมเพื่อให้หลอดที่ใช้แล้วถูกดันออกไป แล้วนำหลอดใหม่เข้ามาโดยไม่มีการหล่นลงมาทับกัน

6. CHK TUBE เป็นการตรวจสอบว่าขณะนี้มีหลอดรอรับ IC หรือไม่ (เพื่อป้องกันการผิดพลาดอันเกิดจาก IC ถูกส่งไปขณะไม่มีหลอด) ถ้าไม่มีหลอดไฟสีแดงจะกระพริบเพื่อแสดงให้ทราบว่ายังไม่มีหลอด ถ้ามีหลอดแล้วจะข้ามขั้นตอนนี้ไป

7. CHK IC เป็นการตรวจสอบว่ามี IC ไหลผ่านเข้ามา ยังจุดดักจับหรือยัง ถ้ายังก็จะ LOOP คอยตรวจสอบต่อไป ถ้ามี IC แล้วก็จะทำงานในขั้นต่อไป

8. อ่านข้อมูลตัวเลข BCD จาก PORT B ซึ่งถูกกำหนดให้เป็นตัวตั้งจำนวน IC ในแต่ละหลอด แล้วแสดงผลไปพร้อม ๆ กัน โดยใช้ 7-Sigments 2 ตัวแล้วแปลงตัวเลข BCD 2 หลักให้อยู่ในรูปของเลขฐาน 16 เพื่อให้ง่ายต่อการคำนวณของ Z-80 โดยจะ Set ค่าที่ได้เป็นจำนวน IC ที่ต้องการในแต่ละหลอด

9. ตรวจสอบว่ามี IC มาหรือไม่ ถ้ามีจะปล่อย IC ไปทีละตัว ไปยังหลอดแล้วลดค่าตัวนับที่ได้จากข้อ 8 ลงหนึ่ง พร้อมทั้งตรวจสอบว่า ครบจำนวนแล้วหรือยัง โดยจะทำงานจนครบจำนวน (ยกเว้นมีการสั่งจากคีย์อื่นมาขัดจังหวะในขณะทำงานนี้) แล้วจึงไปปล่อยหลอดที่มี IC เต็มแล้ว (ครบจำนวนตัวที่ตั้งไว้) ในข้อที่ 3 แล้ววนอยู่เช่นนี้ตลอดการใช้งาน

การทำงานของส่วนปล่อยไอซี (Output Shuttle)

1. เป็นการเริ่มต้นโปรแกรมโดยจะเซ็ทข้อมูลต่าง ๆ แล้ว CLEAR OUTPUT PORT เพื่อให้พร้อมที่จะรับงานต่อไป

2. ON-MAN & DIS เพื่อให้ผู้ใช้ทราบว่า เครื่องเริ่มทำงานแล้ว

3. OFF DIS ; ON MAN เพื่อแสดงว่า พร้อมที่จะรับคำสั่งจากคีย์ต่าง ๆ แล้ว

4. SCAN คีย์ ซึ่งจะเหมือนกันกับในข้อที่ 4 ของทางด้าน INPUT

5. SHK ON เป็นการเคาะหลอดจำนวน 6 ครั้ง เพื่อให้ IC ไหลลงสู่ PROCESS ที่ต้องการ

6. IC CHK เป็นการตรวจสอบว่ามี IC ไหลผ่านมายังตัวคัดจับหรือไม่ ถ้าไม่มีจะรอจนกว่าจะมี (และเพื่อป้องกันกรณีที่เครื่องทำงานโดยไม่มีหลอด IC ทางด้าน OUTPUT)

7. ตรวจสอบ IC อีกครั้งว่าที่มีแล้วนั้นผ่านตัวคัดจับไปหรือยัง (นำไปเข้า Product Line หรือยัง) ถ้ายังจะรอจนกว่าจะผ่านไป

8. SHAKE เป็นการเคาะหลอดจำนวน 6 ครั้ง โดยในการเคาะแต่ละครั้งจะตรวจสอบด้วยว่ามี IC ผ่านมาหรือยัง ถ้ายังจะเคาะต่อไปจนครบ 6 ครั้ง ถ้ายังไม่มีอีกก็ปล่อยหลอด (ถือว่าหลอดนั้นไม่มี IC แล้ว) ถ้าในระหว่างที่เคาะยังไม่ครบ 6 ครั้ง เกิดมี IC ผ่านมาเครื่องจะเซ็ทการนับใหม่คือ ไปเริ่มที่ศูนย์

9. RETUBE เป็นการปล่อยหลอด ซึ่งจะทำโดยการควบคุมให้ VALVE แต่ละตัวสลับกันปิด เปิด เพื่อให้หลอดที่ไม่มี IC แล้ว ถูกดันออกไป และหลอดที่มี IC หลอดต่อไปจะถูกปล่อยลงมาแทนที่เดิมแล้วกลับไปที่ข้อ 3 ใหม่ โดยจะวนอยู่เช่นนี้ตลอดการใช้งาน

ชุด IC

2	ตัว	Z80	CPU
2	ตัว	6264	RAM
4	ตัว	74LS138	
2	ตัว	8255	I/O PORT
2	ตัว	74LS86	
2	ตัว	7448	
1	ตัว	74374	
1	ตัว	74244	
1	ตัว	74132	
2	ตัว	74139	

ชุด SOCKET

4	ตัว	40 - PINS SOCKET
6	ตัว	16 - PINS SOCKET
4	ตัว	28 - PINS SOCKET

ชุด RESISTOR

23	ตัว	10KΩ
2	ตัว	4.7KΩ
4	ตัว	330 Ω
2	ตัว	220 Ω
2	ตัว	680 Ω
6	ตัว	1K Ω

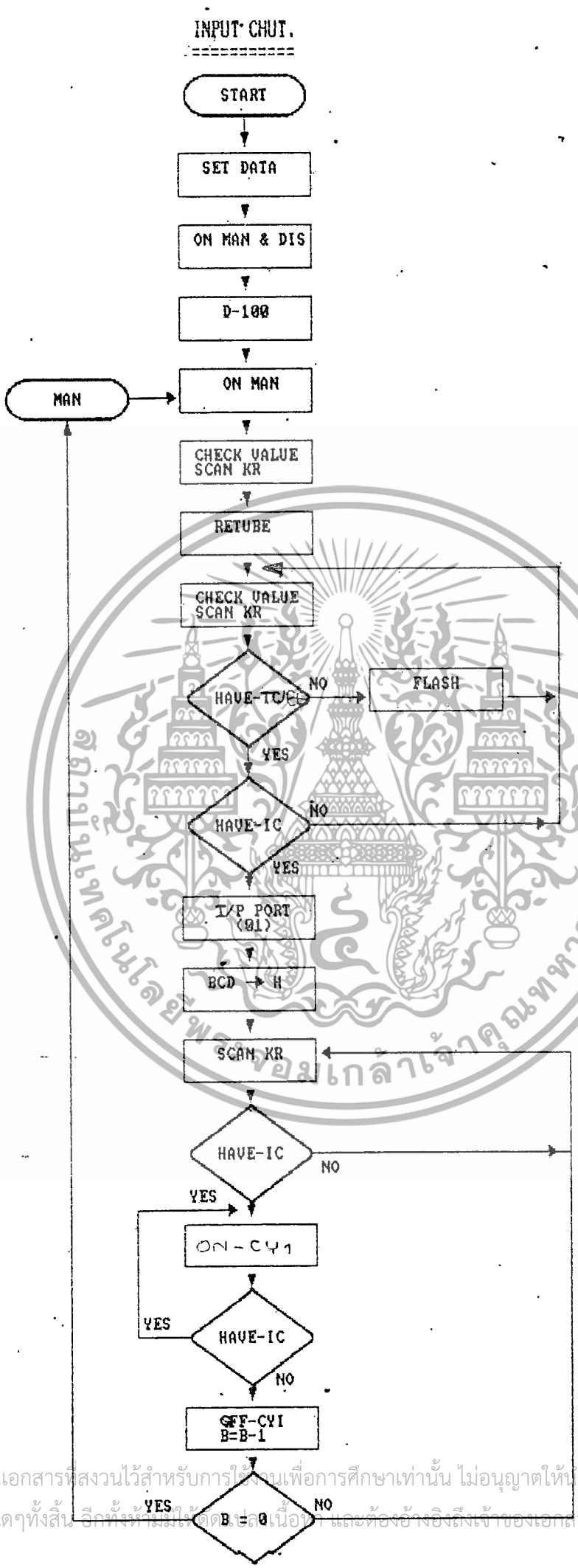
ชุด CAPACITOR

2	ตัว	220	μF	ELE
2	ตัว	22	μF	ELE
2	ตัว	1	μF	TAN
2	ตัว	0.01	μF	ไมล่าร์
2	ตัว	83	μF	ELE
2	ตัว	4.7	μF	TAN
12	ตัว	0.1	μF	TAN
2	ตัว	10	μF	CER

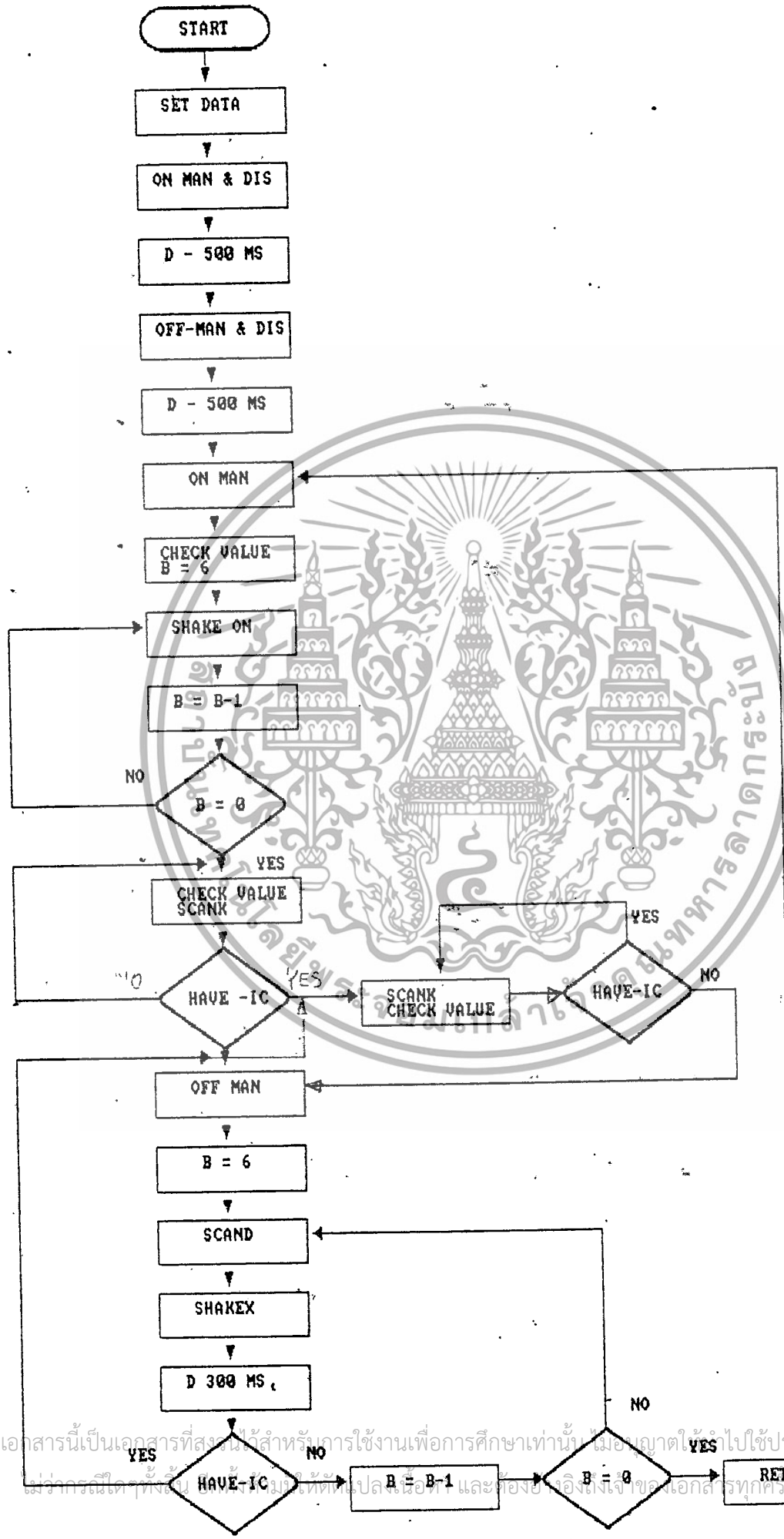
ชุดเบ็ดเตล็ด

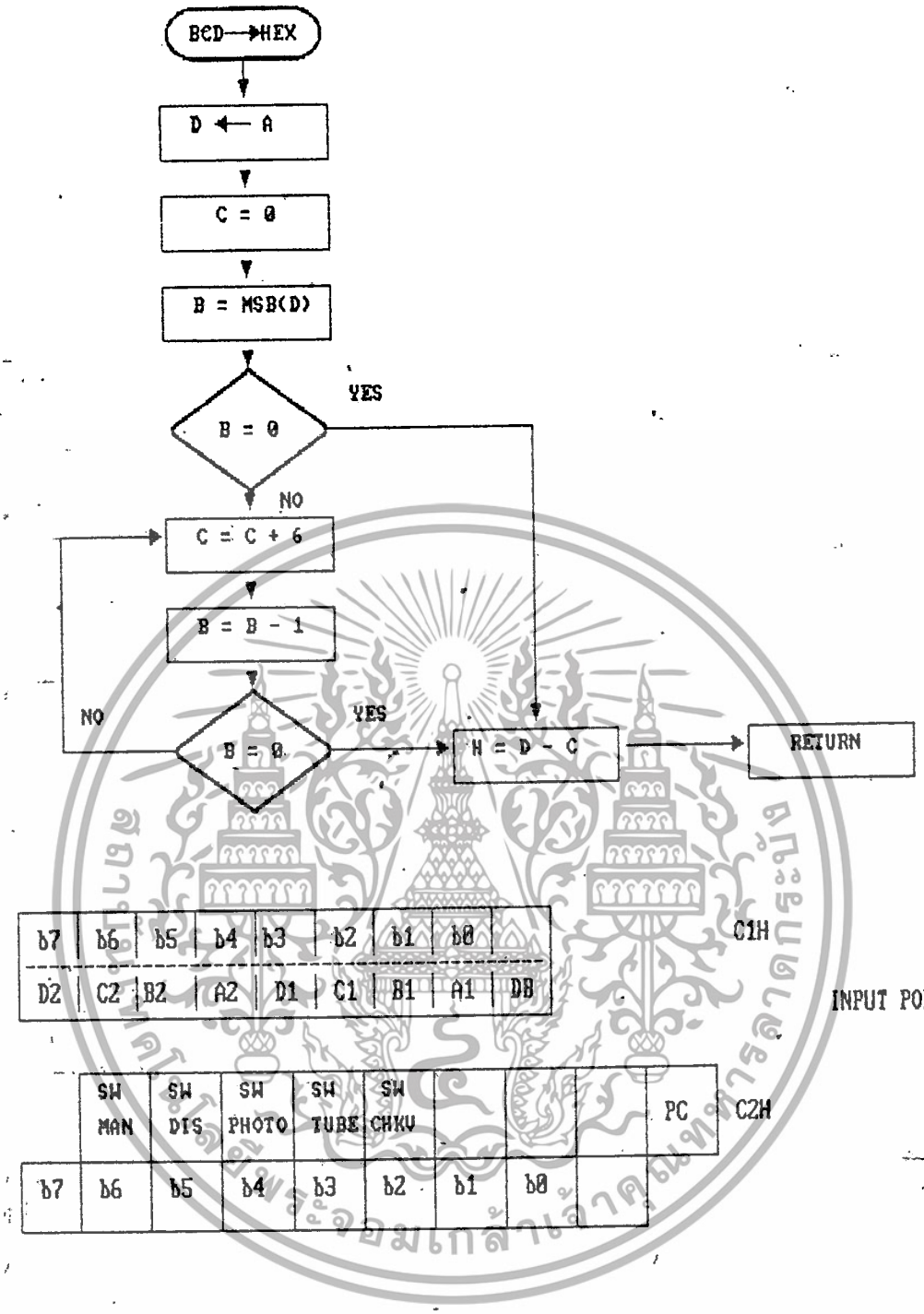
6	ตัว	LED (แดง 4 ตัว, เขียว 2 ตัว)
4	ตัว	1N60
2	ตัว	7805
2	ตัว	1N4148
2	ตัว	HEAT SINK
4	ตัว	CONNECTOR 40 PINS
2	ตัว	DIP SWITCH
2	ตัว	X'TAL 3.579545 MHZ
2	ชุด	ข้าต่อ SUPPLY
2	ตัว	RESET SWITCH
2	ตัว	BC 547

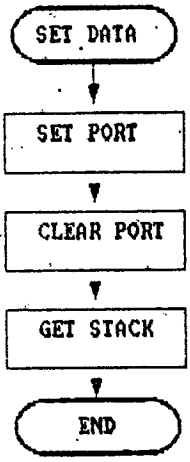
SOLINOID VALUE	*	16 x 15	7	EA
"----------"	*	16 x 60	4	"
"----------"	#	15 x 30	4	"
"----------"	#	15 x 15	1	"

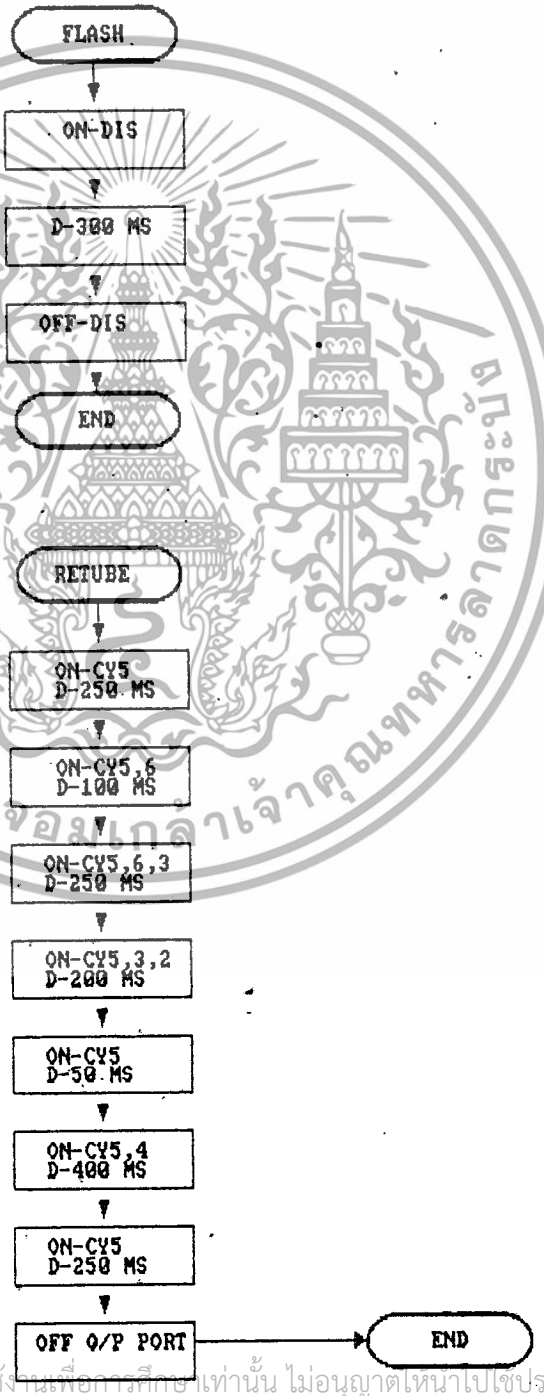


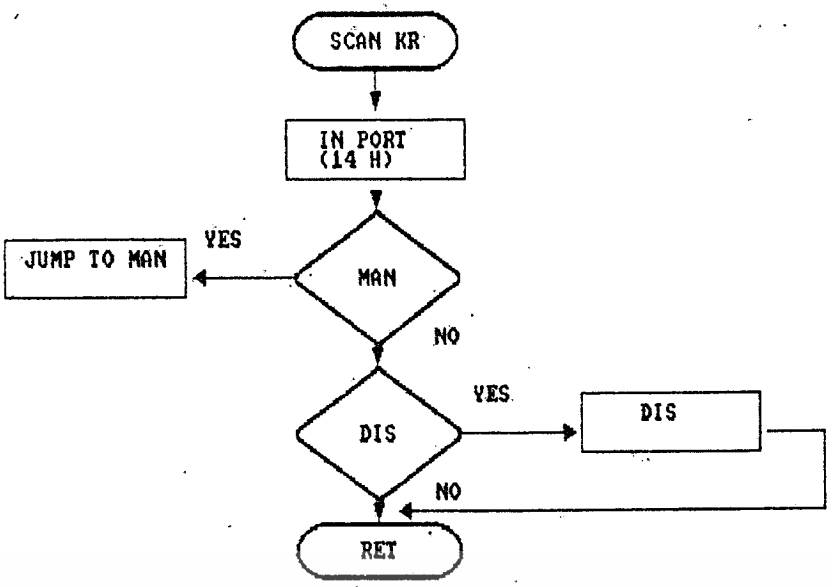
OUTPUT CHUT
==========

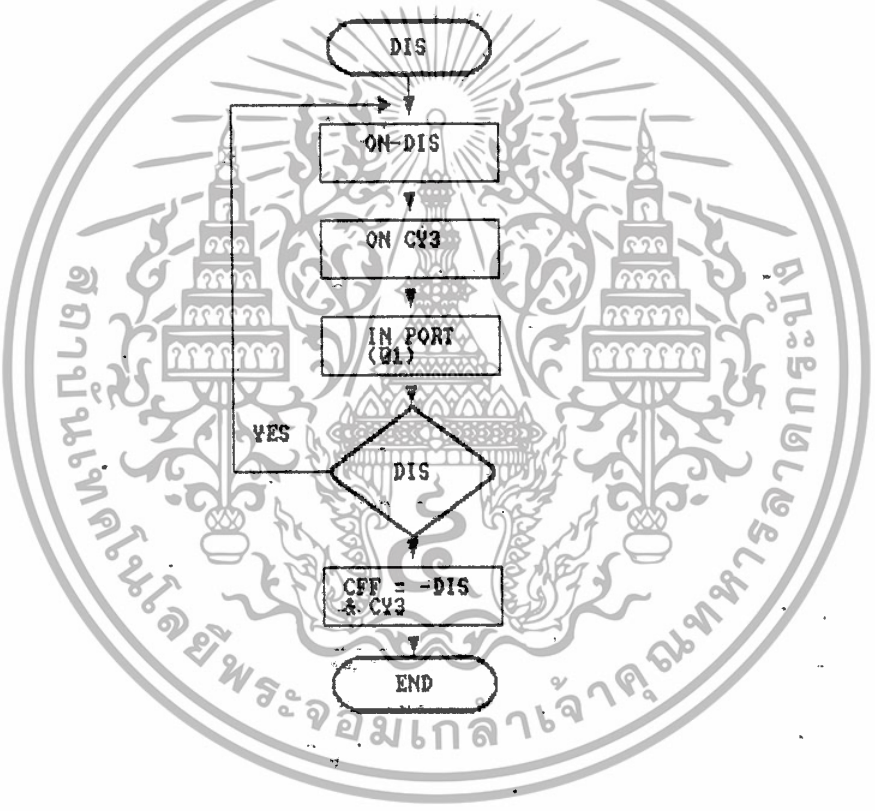


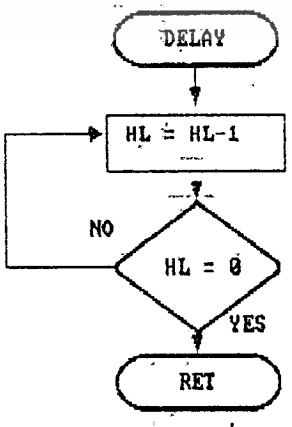


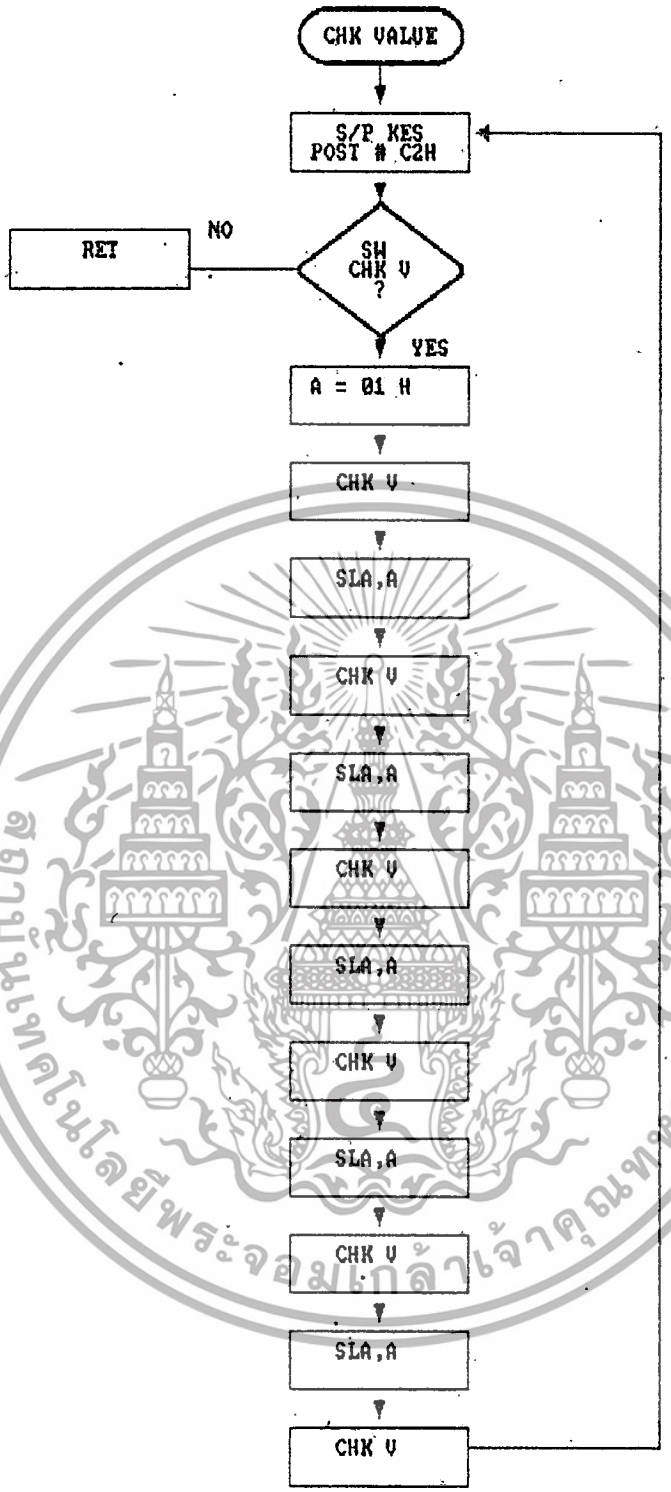


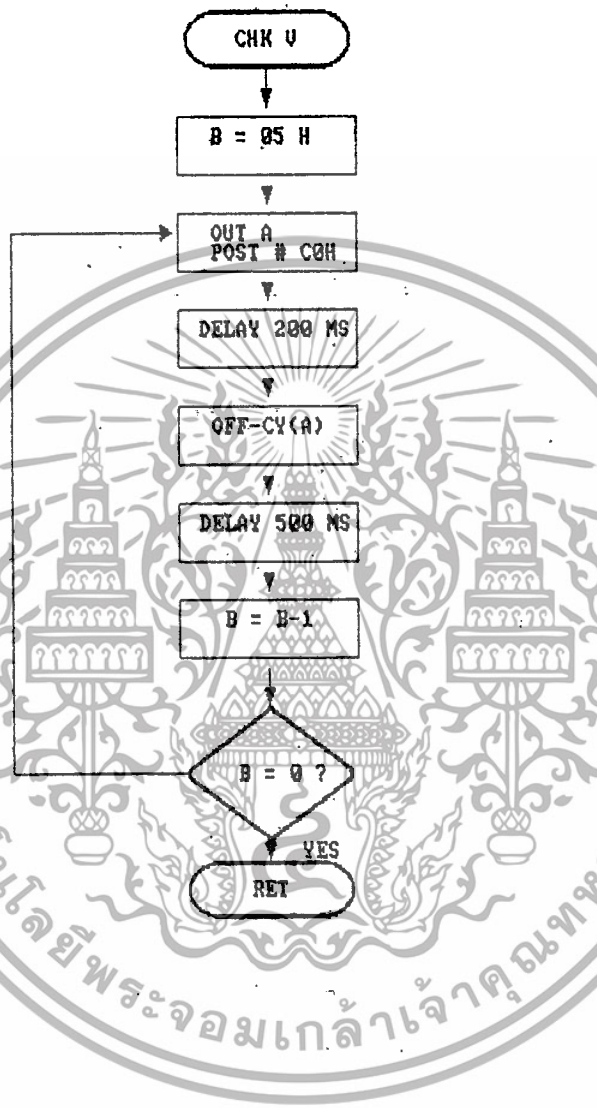


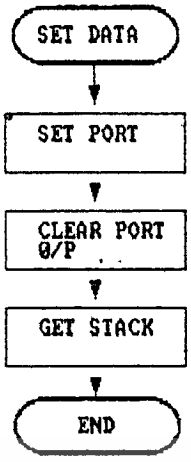


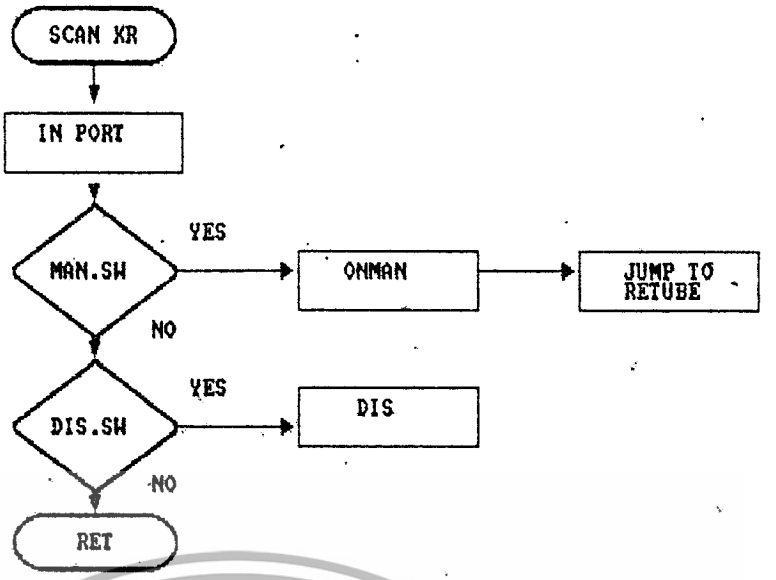


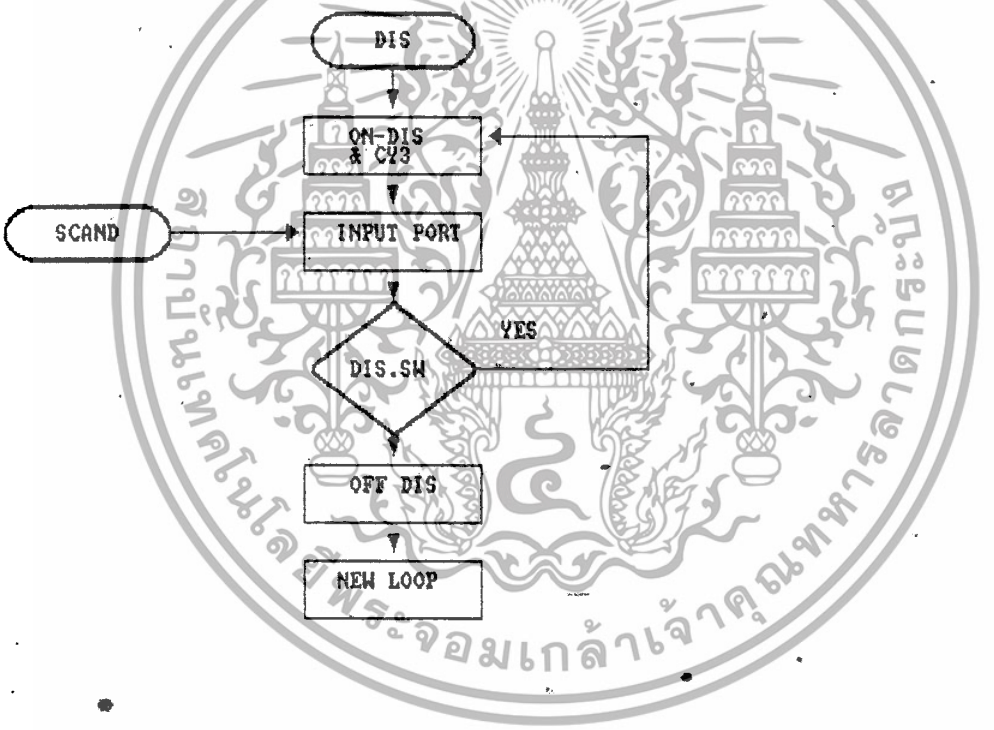


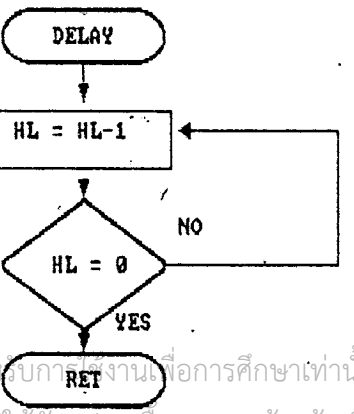


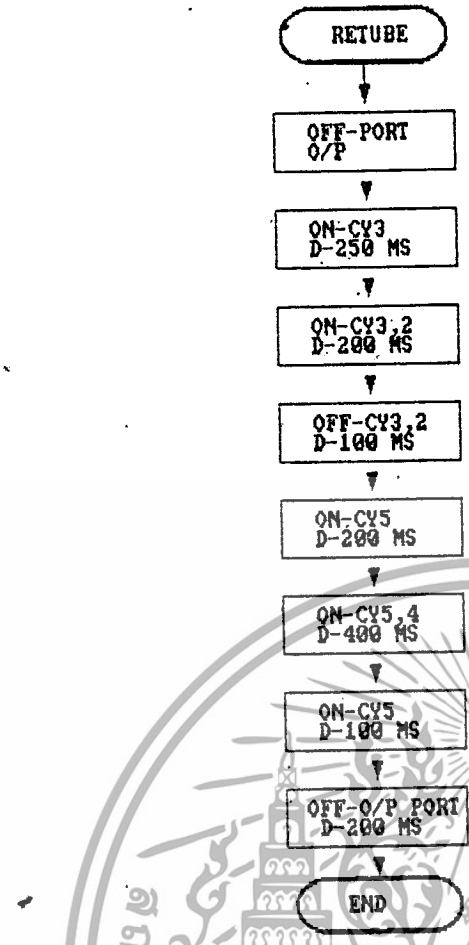












(COH)
PA-OUTPUT PORT (00H)

b7	b6	b5	b4	b3	b2	b1	b0
MAN	DIS		CY5	CY4	CY3	CY2	CY1

PC-INPUT PORT (10H)
(C2H)

	SW MAN	SW DIS	SW PHOTO		SW CHKV		
b1	b6	b5	b4	b3	b2	b1	b0

```
:ASM
```

```
0000 *
0010 *
0020 ********************************
0030 *                              *
0040 *                              *
0050 *      PROGRAM MICROPROCESSOR  *
0060 *                              *
0070 *          # Z-80              *
0080 *                              *
0090 *                              *
0100 *                              *
0110 *      AUTOMATIC LOADER  IC    *
0120 *                              *
0130 *        (OUT PUT SHUTLE)      *
0140 *                              *
0150 ********************************
0160 *
0170 *
0180          .OR   0000H
0190          .TF   AUTO
0200 *
0210 *--------------------------------*
0220 *                                *
0230 *       SET PARAMETER            *
0240 *                                *
0250 *--------------------------------*
0260 *
0270 *
0000- 3E BB   0280          LD    A,8BH    ;SET PORT
0002- D3 C3   0290          OUT   (0C3H),A
0004- 3E 00   0300          LD    A,00H    ;CLEAR O/P PORT
0006- D3 C0   0310          OUT   (0C0H),A
0008- 31 FF 0D 0320         LD    SP,0DFFH
000B- 3E C0   0330          LD    A,0C0H   ;ON-MAN&DIS
000D- D3 C0   0340          OUT   (0C0H),A
000F- 21 7D AA 0350         LD    HL,0AA7DH  ;DELAY 500
0012- CD E6 00 0360         CALL  DELAY
0015- 3E 00   0370          LD    A,00H    OFF-MAN&DIS
0017- D3 C0   0380          OUT   (0C0H),A
0019- 21 7D AA 0390         LD    HL,0AA7DH  ;DELAY 500
001C- CD E6 00 0400         CALL  DELAY
0410 *
0420 *
0430 *--------------------------------*
0440 *                                *
0450 *         NEW LOOP               *
0460 *                                *
0470 *--------------------------------*
0480 *
0490 *
001F- 3E 80   0500 NWLOOP   LD    A,80H    ;ON-MAN
0021- D3 C0   0510          OUT   (0C0H),A
0023- CD F1 00 0520         CALL  CKVALE
0026- 06 06   0530          LD    B,06H
0540 *
0550 *
0560 *--------------------------------*
0570 *                                *
0580 *         SHAKE ON               *
0590 *                                *
0600 *--------------------------------*
0610 *
0620 *
```

```
0028- 3E 81      0630 SHKON  LD   A,81H    ;ON-CY1,MAN
002A- D3 C0      0640         OUT  (0C0H),A
002C- 21 32 44   0650         LD   HL,4432H    ;DELAY 200
002F- CD E6 00   0660         CALL DELAY
0032- 3E 80      0670         LD   A,80H    ;OFF-CY1
0034- D3 C0      0680         OUT  (0C0H),A
0036- 21 4B 66   0690         LD   HL,664BH    ;DELAY 300
0039- CD E6 00   0700         CALL DELAY
003C- 10 EA      0710         DJNZ SHKON
                 0720 *
                 0730 *-------------------------------*
                 0740 *                               *
                 0750 *          LOOP IC              *
                 0760 *                               *
                 0770 *-------------------------------*
                 0785 *
                 0790 *
003E- CD F1 00   0800 LOOPIC CALL CKVALE
0041- CD C4 00   0810        CALL SCANK
0044- DB C2      0820        IN   A,(0C2H)
0046- CB 67      0830        BIT  4,A      ;CHECK IC
0048- 20 F4      0840        JR   NZ,LOOPIC
                 0856 *
                 0860 *
                 0870 *-------------------------------*
                 0880 *                               *
                 0890 *          HAS IC               *
                 0900 *                               *
                 0910 *-------------------------------*
                 0620 *
                 0930 *
004A- CD C4 00   0940 HASIC  CALL SCANK
004D- CD F1 00   0950        CALL CKVALE
0050- DB C2      0960        IN   A,(0C2H)
0052- CB 67      0970        BIT  4,A
0054- 28 F4      0980        JR   Z,HASIC;HAS-IC
0056- 06 06      0990        LD   B,06H
                 1000 *
                 1010 *
                 1020 *-------------------------------*
                 1030 *                               *
                 1040 *          SHAKE                *
                 1050 *                               *
                 1060 *-------------------------------*
                 1070 *
                 1080 *
0058- CD C4 00   1090 SHAKE  CALL SCANK
005B- 3E 01      1100        LD   A,01H    ;ON-CY1
005D- D3 C0      1110        OUT  (0C0H),A
005F- 21 32 44   1120        LD   HL,4432H    ;DELAY 200
0062- CD E6 00   1130        CALL DELAY
0065- 3E 00      1140        LD   A,00H
0067- D3 C0      1150        OUT  (0C0H),A
0069- 21 4B 66   1160        LD   HL,664BH    ;DELAY 300
006C- CD E6 00   1170        CALL DELAY
006F- DB C2      1180        IN   A,(0C2H)
0071- CB 67      1190        BIT  4,A
0073- 28 D5      1200        JR   Z,HASIC
0075- 10 E1      1210        DJNZ SHAKE    ;SHAKE 6 TIME
                 1220 *
                 1230 *
                 1240 *-------------------------------*
                 1250 *                               *
                 1260 *          RETUBE               *
                 1270 *                               *
                 1280 *-------------------------------*
```

```
                        1290 *
                        1300 *
0077- 3E 00             1310 RETUBE LD    A,00H    ;CLEAR O/P PORT
0079- D3 C0             1320        OUT   (0C0H),A
007B- 3E 04             1330        LD    A,04H    ;ON-CY3
007D- D3 C0             1340        OUT   (0C0H),A
007F- 21 EE 53          1350        LD    HL,553EEH    ;DELAY 250
0082- CD E6 00          1360        CALL  DELAY
0085- 3E 06             1370        LD    A,06H    ;ON-CY3,CY2
0087- D3 C0             1380        OUT   (0C0H),A
0089- 21 32 44          1390        LD    HL,4432H     ;DELAY 200
008C- CD E6 00          1400        CALL  DELAY
008F- 3E 00             1410        LD    A,00H    ;OFF-CY3,CY2
0091- D3 C0             1420        OUT   (0C0H),A
0093- 21 18 22          1430        LD    HL,2218H     ;DELAY 100
0096- CD E6 00          1440        CALL  DELAY
0099- 3E 01             1450        LD    A,01H    ;NO-CY5
009B- D3 C0             1460        OUT   (0C0H),A
009D- 21 32 44          1470        LD    HL,4432H     ;DELAY 200
00A0- CD E6 00          1480        CALL  DELAY
00A3- 3E 18             1490        LD    A,18H    ;ON-CY5,CY4
00A5- D3 C0             1500        OUT   (0C0H),A
00A7- 21 64 88          1510        LD    HL,8864H     ;DELAY 400
00AA- CD E6 00          1520        CALL  DELAY
00AD- 3E 10             1530        LD    A,10H    ;ON-CY5
00AF- D3 C0             1540        OUT   (0C0H),A
00B1- 21 18 22          1550        LD    HL,2218H     ;DELAY 100
00B4- CD E6 00          1560        CALL  DELAY
00E7- 3E 00             1570        LD    A,00H    ;OFF-O/P PORT
00B9- D3 C0             1580        OUT   (0C0H),A
00BB- 21 32 44          1590        LD    HL,4432H     ;DELAY 200
00BE- CD E6 00          1600        CALL  DELAY
00C1- C3 1F 00          1610        JP    NWLOOP
                        1620 *
                        1630 *
                        1640 *-------------------------------*
                        1650 *                               *
                        1660 *          SCANK                *
                        1670 *-------------------------------*
                        1680 *
                        1690 *
                        1700 *
00C4- DB C2             1710 SCANK  IN    A,(0C2H)
00C6- CB 77             1720        BIT   6,A      ;MAN SW
00C8- 28 00             1730        JR    Z,DISCHK ;NON PRESS
00CA- 3E 80             1740        LD    A,80H    ;ON-MAN
00CC- D3 C0             1750        OUT   (0C0H),A
00CE- 21 18 22          1760        LD    HL,2218H     ;DELAY 100
00D1- CD E6 00          1770        CALL  DELAY
00D4- 18 A1             1780        JR    RETUBE
                        1790 *
00D6- CB 6F             1800 DISCHK BIT   5,A      ;DIS SW
00D8- C8                1810        RET   Z        ;DON'KEY
                        1820 *
00D9- 3E 44             1830 DIS    LD    A,44H    ;ON-DIS,CY3
00DB- D3 C0             1840        OUT   (0C0H),A
00DD- DB C2             1850        IN    A,(0C2H)
00DF- CB 6F             1860        BIT   5,A      ;DIS SW
00E1- 20 F6             1870        JR    NZ,DIS
00E3- C3 1F 00          1880        JP    NWLOOP   ;DON'T PRESS
                        1890 *
                        1900 *
                        1910 *-------------------------------*
                        1920 *                               *
                        1930 *          DELAY                *
                        1940 *                               *
```

```
                                1950 *-------------------------------*
                                1960 *                               *
                                1970 *
00E6- 00                        1980 DELAY   NOP
00E7- 00                        1990         NOP
00E8- 00                        2000         NOP
00E9- 00                        2010         NOP
00EA- 00                        2020         NOP
00EB- 3B                        2030         DEC   HL
00EC- 7C                        2040         LD    A,H
00ED- B5                        2050         OR    L
00EE- 20 F8                     2060         JR    NZ,DELAY
00F0- C9                        2070         RET
                                2080 *
                                2090 *
                                2100 *-------------------------------*
                                2110 *                               *
                                2120 *                               *
                                2130 *          CHECK VALVE          *
                                2140 *                               *
                                2150 *-------------------------------*
                                2160 *
00F1- DB C2                     2170 CKVALE  IN    A,(0C2H)
00F3- CB 57                     2180         BIT   2,A          ;VALVE SW
00F5- C8                        2190         RET   Z            ;DON'T PRESS
00F6- 3E 01                     2200         LD    A,01H        ;TEST-CY1
00F8- CD 11 01                  2210         CALL  CHKV
00FB- CB 27                     2220         SLA   A            ;TEST-CY2
00FD- CD 11 01                  2230         CALL  CHKV
0100- CB 27                     2240         SLA   A            ;TEST-CY3
0102- CD 11 01                  2250         CALL  CHKV
0105- CB 27                     2260         SLA   A            ;TEST-CY4
0107- CD 11 01                  2270         CALL  CHKV
010A- CB 27                     2280         SLA   A            ;TEST-CY5
010C- CD 11 01                  2290         CALL  CHKV
010F- 18 E0                     2300         JR    CKVALE
                                2310 *
                                2320 *
                                2330 *-------------------------------*
                                2340 *                               *
                                2350 *             CHKV              *
                                2360 *                               *
                                2370 *-------------------------------*
                                2380 *
                                2390 *
0111- 06 05                     2400 CHKV    LD    B,05H        ;TEST 5 TIME
                                2410 *
0113- D3 C0                     2420 CHKLP   OUT   (0C0H),A
0115- 4F                        2430         LD    C,A
0116- 21 32 44                  2440         LD    HL,4432H     ;DELAY 200
0119- CD E6 00                  2450         CALL  DELAY
011C- 3E 00                     2460         LD    A,00H
011E- D3 C0                     2470         OUT   (0C0H),A
0120- 21 91 CC                  2480         LD    HL,0CC91H    ;DELAY 600
0123- CD E6 00                  2490         CALL  DELAY
0126- 79                        2500         LD    A,C
0127- 10 EA                     2510         DJNZ  CHKLP
0129- C9                        2520         RET
                                2530 *
                                2540 *
                                2550 *********************************
                                2560 *                               *
                                2570 *                               *
                                2580 *     END OF OUTPUT SHUTLE      *
                                2590 *                               *
                                2600 *                               *
```

```
2610 ********************************
2620 *                              *
2630 *                              *
2640 *      PROJECT  BY:            *
2650 *                              *
2660 *      1)MR.JARUAK MEECHUEI    *
2670 *                              *
2680 *            NUMBER 31.3604    *
2690 *                              *
2700 *      2)MR.CHUCHAT JITNOM     *
2710 *                              *
2720 *            NUMBER 31.3606    *
2730 *                              *
2740 *      3)MR.SUMRITH BUMRUNGSUK *
2750 *                              *
2760 *            NUMBER 31.3630    *
2770 *                              *
2780 ********************************
2790 *
2800 *
2810 *
2820 END
```

```
SYMBOL TABLE

0113- CHKLP
0111- CHKV
00F1- DKVALE
00E6- DELAY
0009- DTS
0002- DTSCHK
012A- END
004A- HASIC
003E- LOOPIC
001F- NWLOOP
0077- RETUBE
0024- SCANK
0058- SHAKE
0028- SHKON

0000 ERRORS IN ASSEMBLY
```

```
:
:ASM
```

```
0000 *
0010 *
0020 ************************************
0030 *                                  *
0040 *                                  *
0050 *   PROGRAM FOR MICROPROCESSOR     *
0060 *                                  *
0070 *          # Z-80                  *
0080 *                                  *
0090 *     AUTOMATIC LOADER  IC         *
0100 *                                  *
0110 *          (INPUT SHUTLE)          *
0120 *                                  *
0130 ************************************
0140 *
0150 *
0160          .OR 0000H
0170          .TF AUTO
0180 *
0190 *-----------------------------------*
0200 *                                   *
0210 *     SET PARAMETER                 *
0220 *                                   *
0230 *-----------------------------------*
0240 *
0250 *
0260          LD    A,8BH  ;SET PORT
0270          OUT   (0C3H),A
0280          LD    A,00H  ;CLEAR O/P PORT
0290          OUT   (0C0H),A
0300          LD    SP,0DFFH
0310          LD    A,0C0H ;ON MAN&DIS
0320          OUT   (0C0H),A
0330          LD    HL,0AA7DH
0340          CALL  DELAY
0345 *
0353 MANUAL   LD    A,80H  ;OFF MAN&DIS
0360          OUT   (0C0H),A
0370          LD    HL,1000H
0380          CALL  DELAY
0390          CALL  CKVALE
0400          CALL  SCANKR
0410          LD    A,80H  ;CLEAR O/P PORT
0420          OUT   (0C0H),A
0430          LD    A,10H  ;ON-CY5
0440          OUT   (0C0H),A
0450          LD    HL,553EH ;DELAY 250
0460          CALL  DELAY
0470          LD    A,30H    ;ON-CY5,CY6
0480          OUT   (0C0H),A
0490          LD    HL,2218H ;DELAY 100
0500          CALL  DELAY
0510          LD    A,34H    ;ON-CY5,CY6,CY3
0520          OUT   (0C0H),A
0530          LD    HL,553EH ;DELAY 250
0540          CALL  DELAY
0550          LD    A,16H    ;ON-CY5,CY3,CY2
0560          OUT   (0C0H),A
0570          LD    HL,4432H ;DELAY 200
0580          CALL  DELAY
0590          LD    A,10H    ;ON-CY5
0600          OUT   (0C0H),A
0610          LD    HL,1100H ;DELAY 50
0620          CALL  DELAY
0630          LD    A,18H    ;ON-CY5,CY4
0640          OUT   (0C0H),A
```

```
0000- 3E 8B
0002- D3 C3
0004- 3E 00
0006- D3 C0
0008- 31 FF 0D
000B- 3E C0
000D- D3 C0
000F- 21 7D AA
0012- CD E0 00
0015- 3E 80
0017- D3 C0
0019- 21 00 10
001C- CD E0 00
001F- CD 05 01
0022- CD C7 00
0025- 3E 80
0027- D3 C0
0029- 3E 10
002B- D3 C0
002D- 21 3E 55
0030- CD E0 00
0033- 3E 30
0035- D3 C0
0037- 21 18 22
003A- CD E0 00
003D- 3E 34
003F- D3 C0
0041- 21 3E 55
0044- CD E0 00
0047- 3E 16
0049- D3 C0
004B- 21 32 44
004E- CD E0 00
0051- 3E 10
0053- D3 C0
0055- 21 00 11
0058- CD E0 00
005B- 3E 18
005D- D3 C0
```

```
005F- 21 64 88 0650                LD    HL,8864H ;DELAY 400
0062- CD E0 00 0660                CALL  DELAY
0065- 3E 10    0670                LD    A,10H    ;ON-CY5
0067- D3 C0    0680                OUT   (0C0H),A
0069- 21 34 55 0690                LD    HL,5534H ;DELAY 250
006C- CD E0 00 0700                CALL  DELAY
006F- 3E 80    0710                LD    A,80H    ;ON-MAN
0071- D3 C0    0720                OUT   (0C0H),A
               0725 *
0073- CD 05 01 0730 LOOP1          CALL  CKVALE
0076- CD C7 00 0740                CALL  SCANKR
0079- DB C2    0750                IN    A,(0C2H)
007B- CB 5F    0760                BIT   3,A      ;CHK TUBE
007D- 20 14    0770                JR    NZ,ICCHK
007F- 3E 40    0780                LD    A,40H    ;ON-DIS
0081- D3 C0    0790                OUT   (0C0H),A
0083- 21 4B 66 0800                LD    HL,664BH ;DELAY 300
0086- CD E0 00 0810                CALL  DELAY
0089- 3E C0    0820                LD    A,0C0H
008B- D3 C0    0830                OUT   (0C0H),A ;OFF-DIS
008D- 21 4B 66 0840                LD    HL,664BH ;DELAY 300
0090- CD E0 00 0850                CALL  DELAY
0093- 18 DE    0860                JR    LOOP1
               0870 *
               0880 *
               0890 *-------------------------------------------*
               0900 *                                           *
               0910 *              CHECK IC                     *
               0920 *                                           *
               0930 *-------------------------------------------*
               0940 *
               0950 *
0095- CB 67    0960 ICCHK  BIT   4,A      ;CHK PHOTO(IC)
0097- 20 DA    0970        JR    NZ,LOOP1
0099- 3E 00    0980        LD    A,00H    ;OFF-MAN
009B- D3 C0    0990        OUT   (0C0H),A
009D- DB C1    1000        IN    A,(0C1H)
009F- CD E8 00 1010        CALL  DTOH
00A2- 44       1020        LD    B,H
               1025 *
00A3- 58       1030 LOOP2  LD    E,B
00A4- CD 05 01 1040        CALL  CKVALE
00A7- CD C7 00 1050        CALL  SCANKR
00AA- 43       1060        LD    B,E
00AB- DB C2    1070        IN    A,(0C2H)
00AD- CB 67    1080        BIT   4,A      ;CHK IC
00AF- 20 F2    1090        JR    NZ,LOOP2 ;NO-IC
               1100 *
               1110 *
               1120 *-------------------------------------------*
               1130 *                                           *
               1140 *              REWORK                       *
               1150 *                                           *
               1160 *-------------------------------------------*
               1170 *
               1180 *
00B1- 3E 01    1190 REWORK LD    A,01H    ;ON-CY1
00B3- D3 C0    1200        OUT   (0C0H),A
00B5- DB C2    1210        IN    A,(0C2H)
00B7- CB 67    1220        BIT   4,A
00B9- 28 F6    1230        JR    Z,REWORK ;HAVE IC
00BB- 3E 00    1240        LD    A,00H    ;OFF-CY1
00BD- D3 C0    1250        OUT   (0C0H),A
00BF- 10 E2    1260        DJNZ  LOOP2
00C1- CD 05 01 1270        CALL  CKVALE
00C4- C3 15 00 1280        JP    MANUAL
               1290 *
               1300 *
               1310 *-------------------------------------------*
               1320 *                                           *
```

```
                        ;*-------------------------------*
                        ;*          SCAN KEY             *
                        ;*-------------------------------*
00C7- DB C2             SCANKR  IN    A,(0C2H)
00C9- CB 77                     BIT   6,A         ;MAN SW
00CB- C2 15 00                  JP    NZ,MANUAL
00CE- CB 6F                     BIT   5,A         ;DIS SW
00D0- C8                        RET   Z
                        ;
00D1- 3E 44             DIS     LD    A,44H       ;ON-DIS,CY3
00D3- D3 C0                     OUT   (0C0H),A
00D5- DB C2                     IN    A,(0C2H)
00D7- CB 6F                     BIT   5,A         ;DIS SW
00D9- 20 F6                     JR    NZ,DIS
00DB- 3E 80                     LD    A,80H
00DD- D3 C0                     OUT   (0C0H),A
00DF- C9                        RET
                        ;
                        ;*-------------------------------*
                        ;*          DELAY LOOP           *
                        ;*-------------------------------*
                        ;
00E0- 00                DELAY   NOP
00E1- 00                        NOP
00E2- 00                        NOP
00E3- 00                        NOP
00E4- 00                        NOP
00E5- 2B                        DEC   HL
00E6- 7C                        LD    A,H
00E7- B5                        OR    L
00E8- 20 F6                     JR    NZ,DELAY
00EA- C9                        RET
                        ;
                        ;*-------------------------------*
                        ;*       CONV BCD TO BINARY      *
                        ;*-------------------------------*
                        ;
00EB- 7A                DTOH    LD    A,D
00EC- 0E 00                     LD    C,00H
00EE- E6 F0                     AND   0F0H
00F0- CB 0F                     RRC   A
00F2- CB 0F                     RRC   A
00F4- CB 0F                     RRC   A
00F6- CB 0F                     RRC   A
00F8- 28 07                     JR    Z,CONV
00FA- 47                        LD    B,A
                        ;
00FB- 79                SUM     LD    A,C
00FC- C6 06                     ADD   A,06H
00FE- 4F                        LD    C,A
00FF- 10 FA                     DJNZ  SUM
                        ;
0101- 7A                CONV    LD    A,D
0102- 91                        SUB   C
0103- 67                        LD    H,A
0104- C9                        RET
                        ;
                        ;*-------------------------------*
                        ;*          CHECK VALVE          *
                        ;*-------------------------------*
```

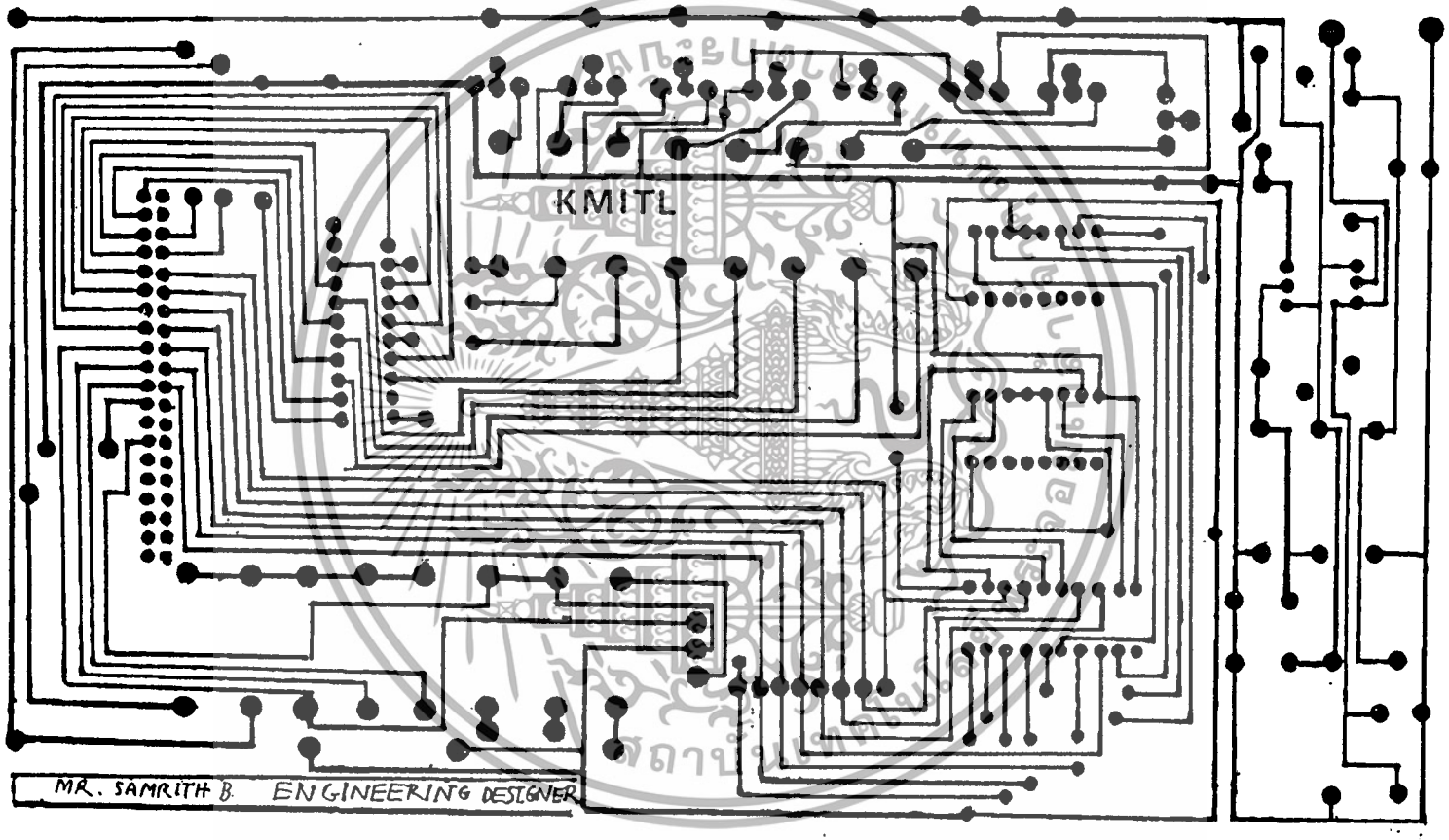
```
                        2000 *                                              *
                        2010 *----------------------------------------------*
                        2020 *
                        2030 *
0105- DB C2             2040 CKVALE IN   A,(0C2H)
0107- CB 57             2050        BIT  2,A        ;CHK VALVA SW
0109- C8                2060        RET  Z
010A- 3E 01             2070        LD   A,01H
010C- CD 3A 01          2080        CALL CHKV
010F- CB 07             2090        RLC  A
0111- CD 3A 01          2100        CALL CHKV
0114- CB 07             2110        RLC  A
0116- CD 3A 01          2120        CALL CHKV
0119- CB 07             2130        RLC  A
011B- CD 3A 01          2140        CALL CHKV
011E- CB 07             2150        RLC  A
0120- CD 3A 01          2160        CALL CHKV
0123- CB 07             2170        SLA  A
0125- CD 3A 01          2180        CALL CHKV
0128- 18 E6             2190        JR   CKVALE
                        2195 *
012A- 06 05             2200 CHKV   LD   B,05H      ;DEMO 5 TIME
012C- D3 C0             2210 CHKVLP OUT  (0C0H),A
                        2215 *
012E- 4F                2220        LD   C,A
012F- 21 32 44          2230        LD   HL,4432H ;DELAY 200
0132- CD E0 00          2240        CALL DELAY
0135- 3E 00             2250        LD   A,00H
0137- D3 C0             2260        OUT  (0C0H),A
0139- 21 91 CC          2270        LD   HL,0CC91H ;DELAY 600
013C- CD E0 00          2280        CALL DELAY
013F- 79                2290        LD   A,C
0140- 10 EA             2300        DJNZ CHKVLP
0142- C9                2310        RET
                        2320 ********************************************
                        2330 *                                            *
                        2340 *      END OF INPUT CHUTLE                   *
                        2350 *                                            *
                        2360 ********************************************
                        2370 END
                        2380 *                                            *
```

SYMBOL TABLE

```
012A- CHKV
012C- CHKVLP
0105- CKVALE
0101- CONV
00E0- DELAY
00D1- DIS
00EB- DTOH
0143- END
0095- ICCHK
0073- LOOP
00A3- LOOP2
0015- MANUAL
00B1- REWORK
00C7- SCANNR
00FB- SUM
```

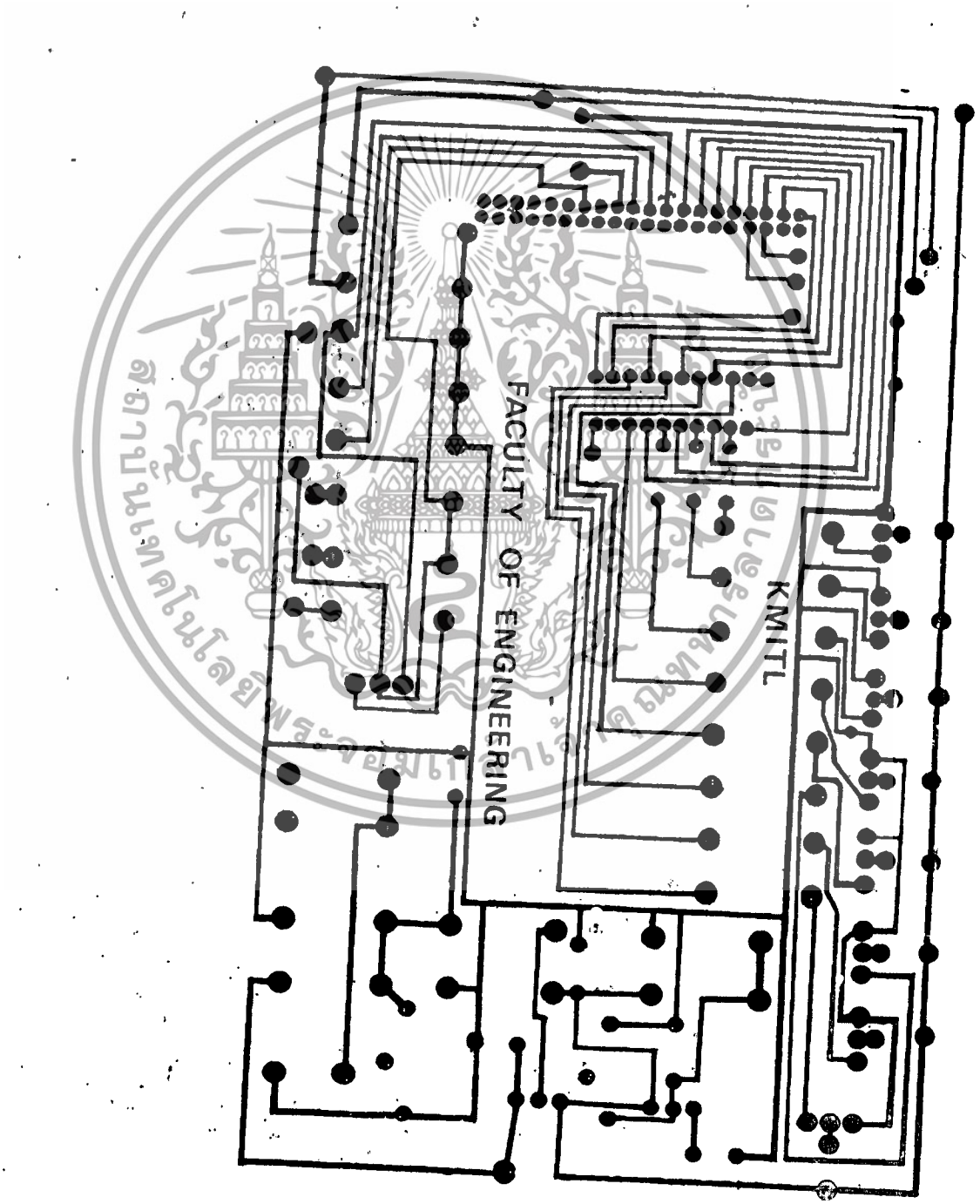
0000 ERRORS IN ASSEMBLY

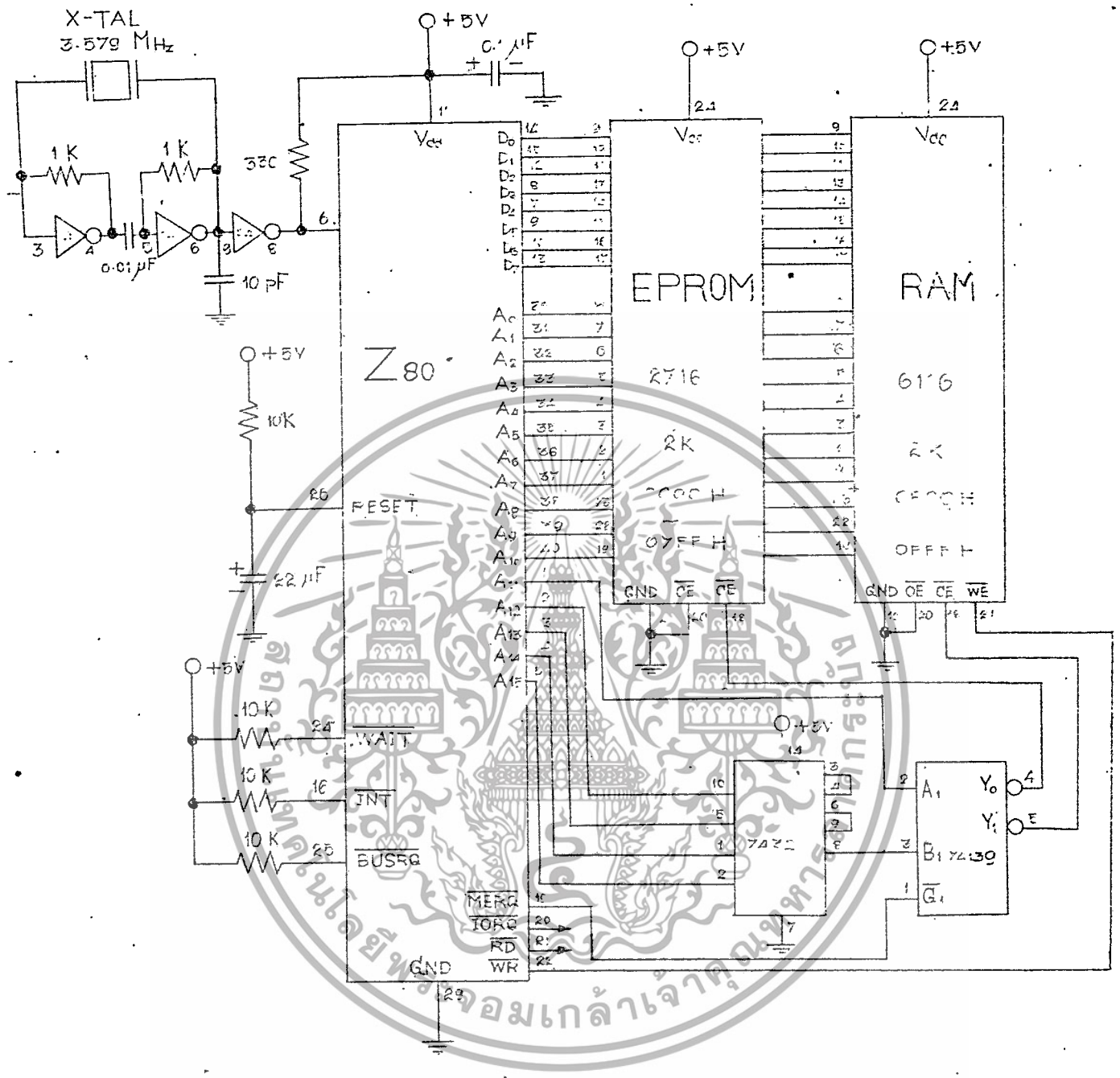
INPUT CHUT

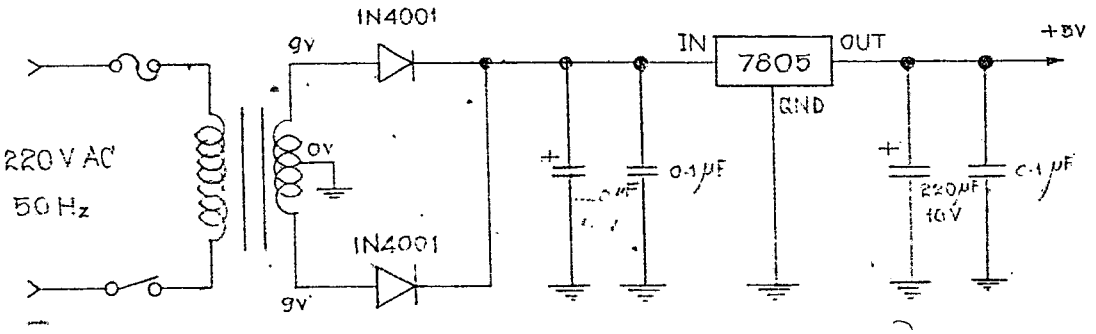


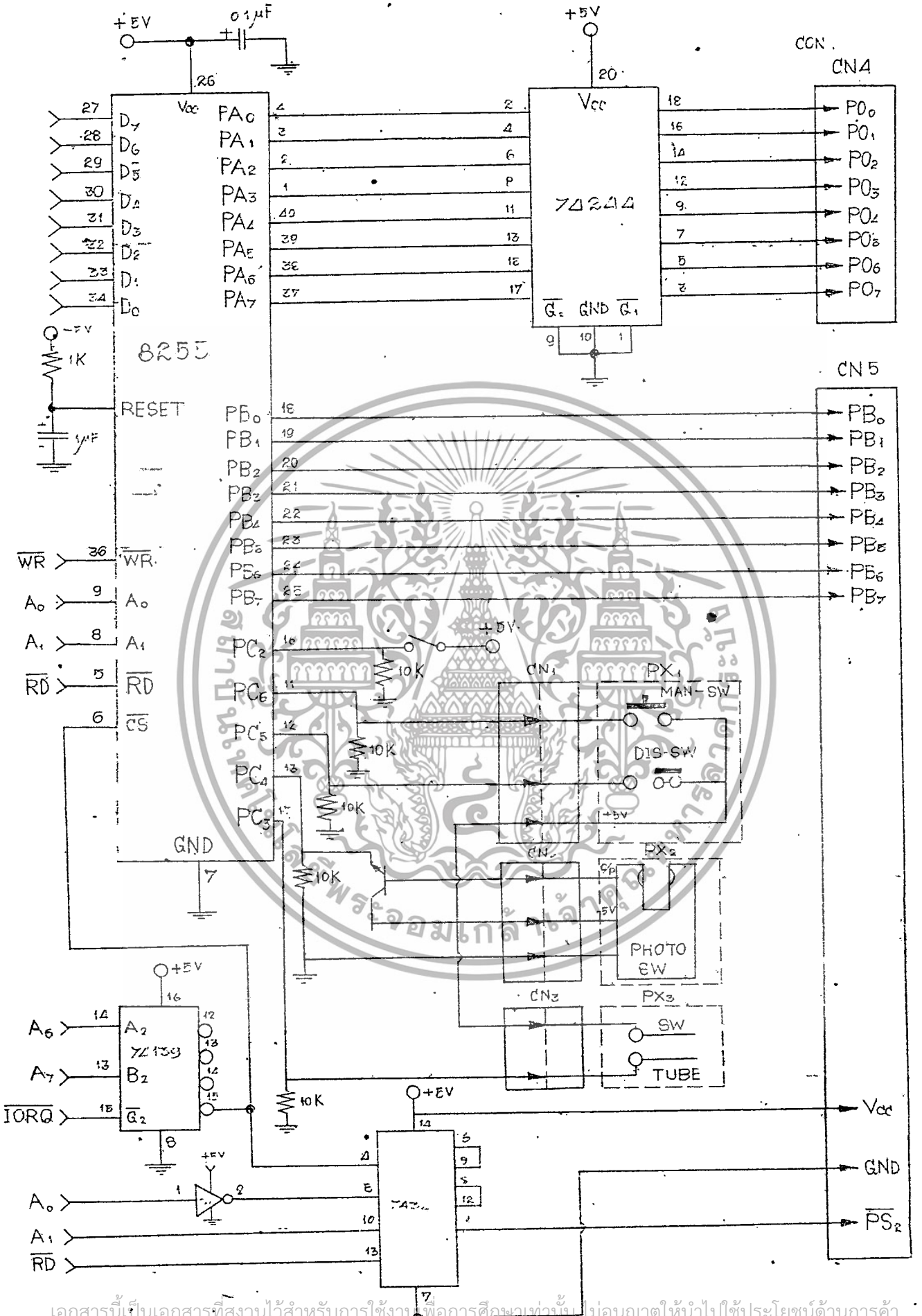
FACULTY

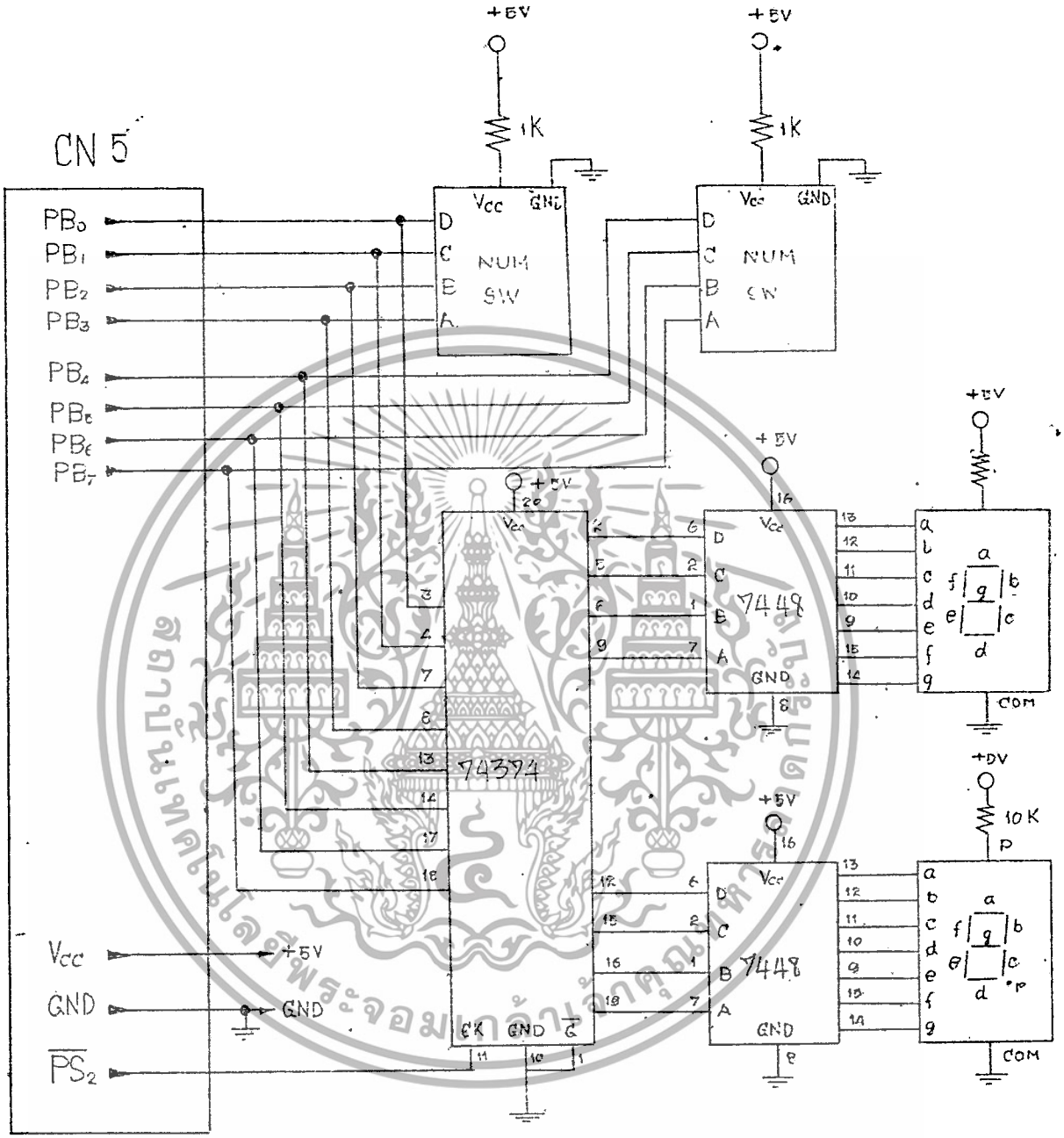
OUT PUT CHUT

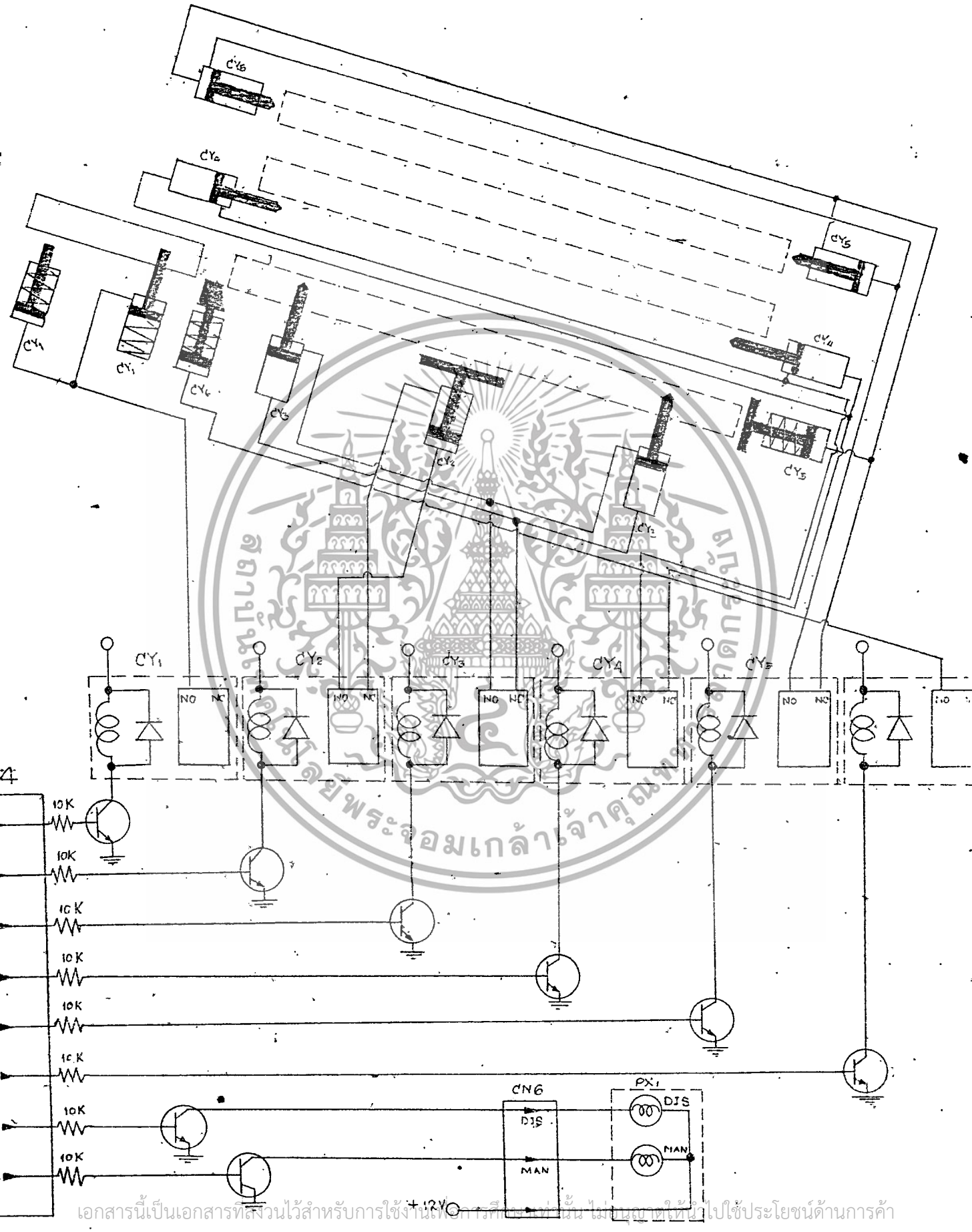


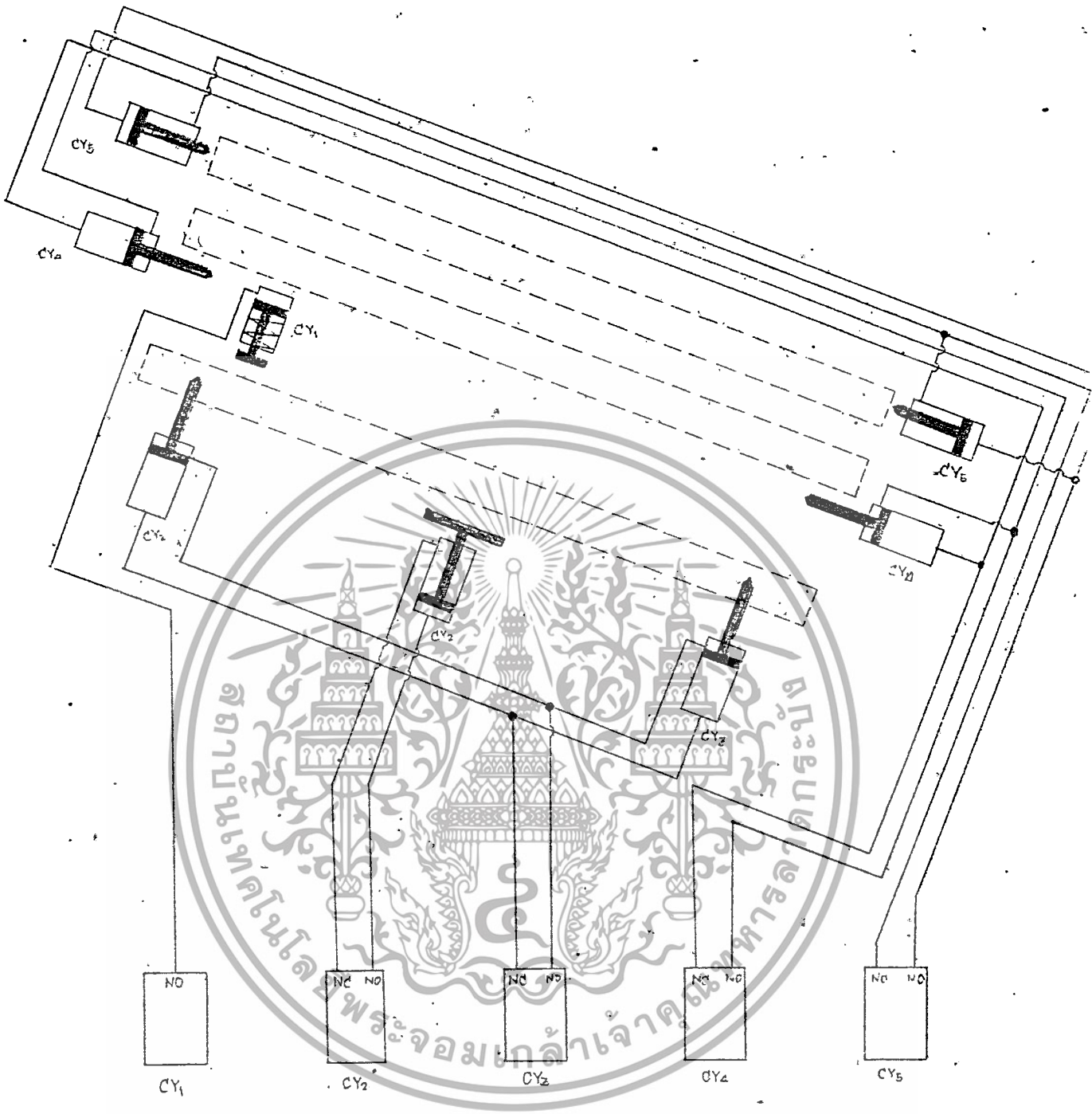


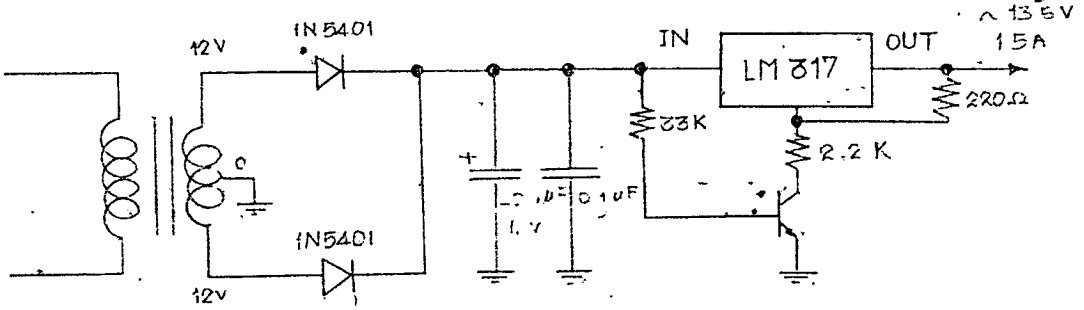




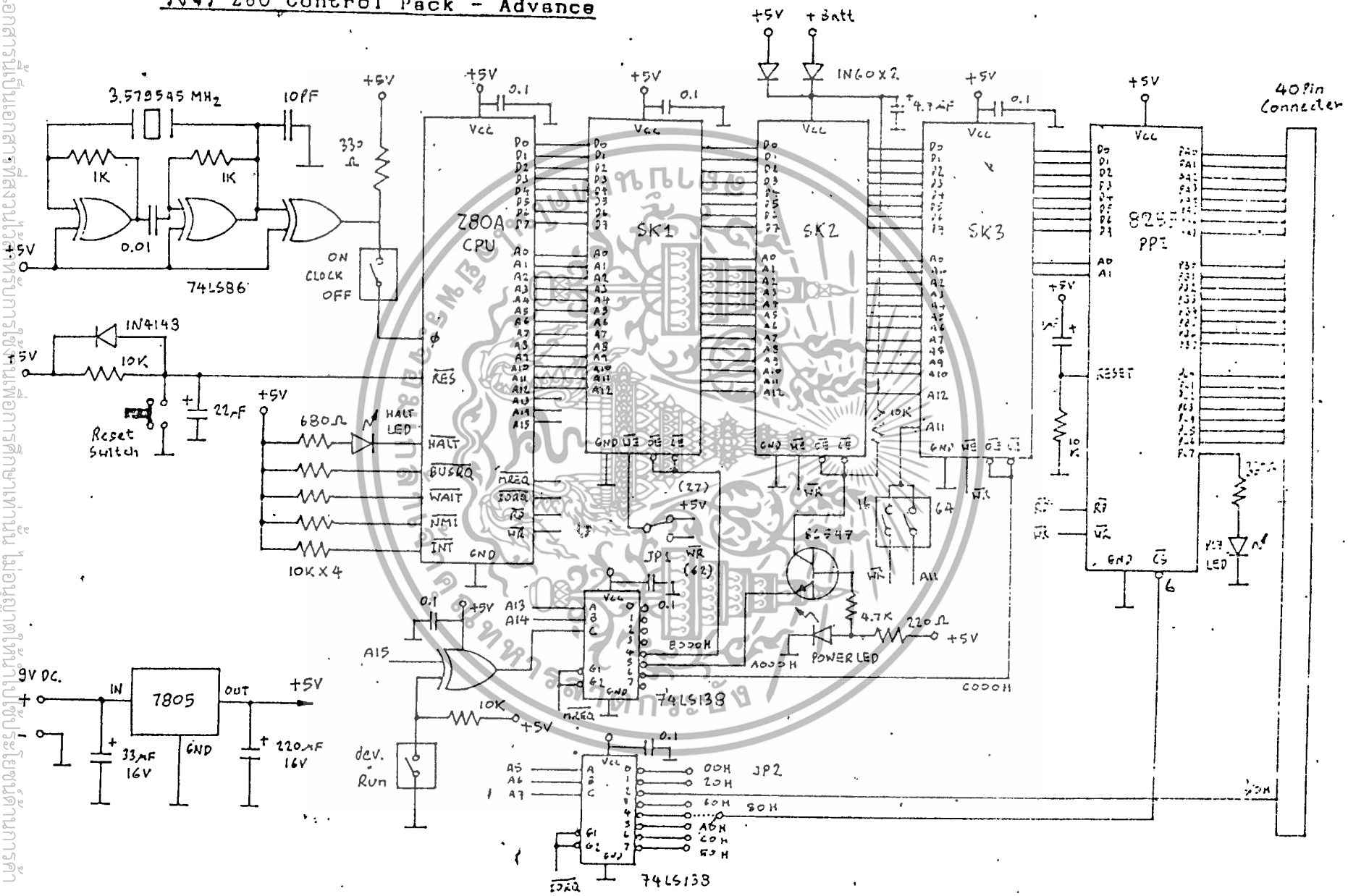








๒๐๙๗ Z80 Control Pack - Advance

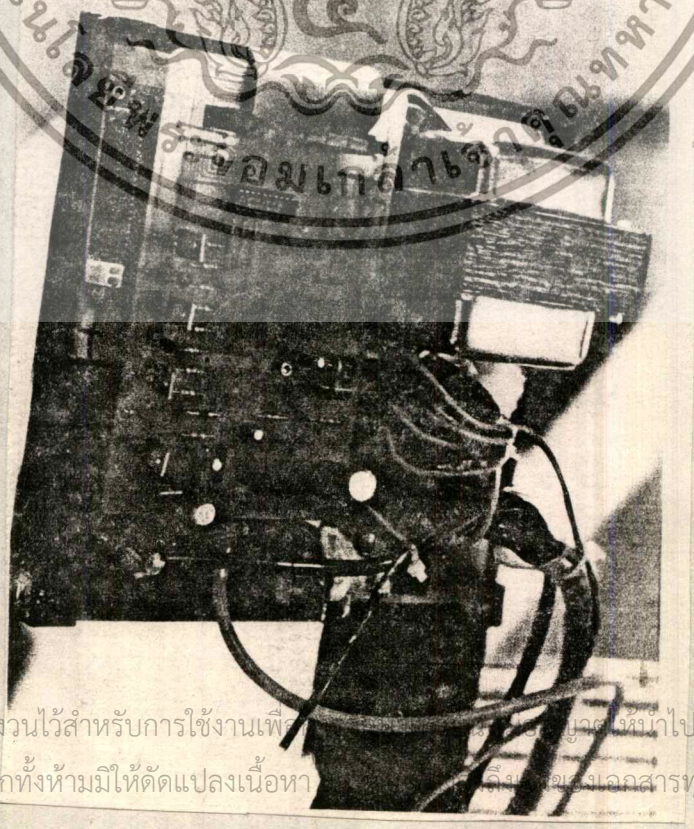


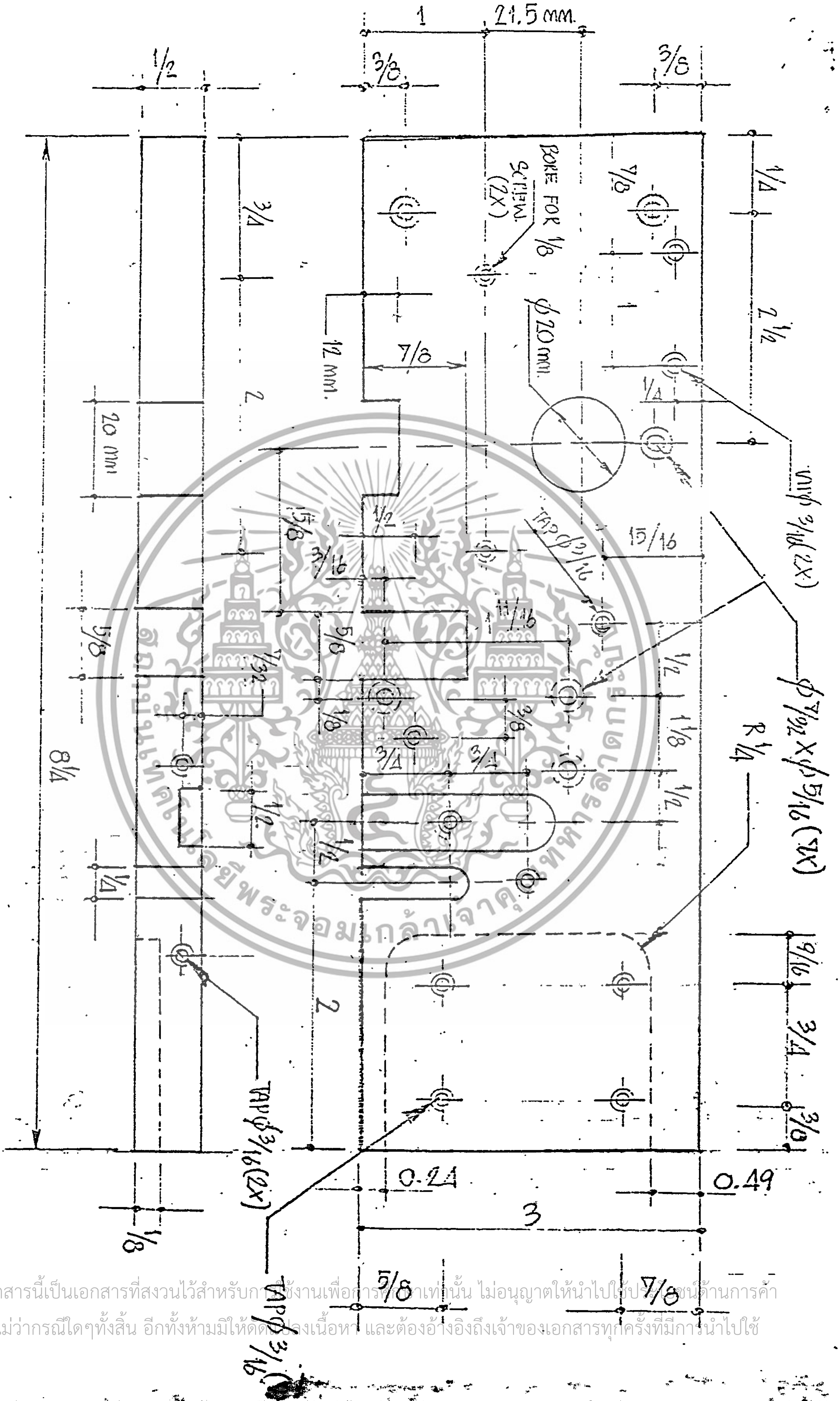
สรุปผลการทดลอง

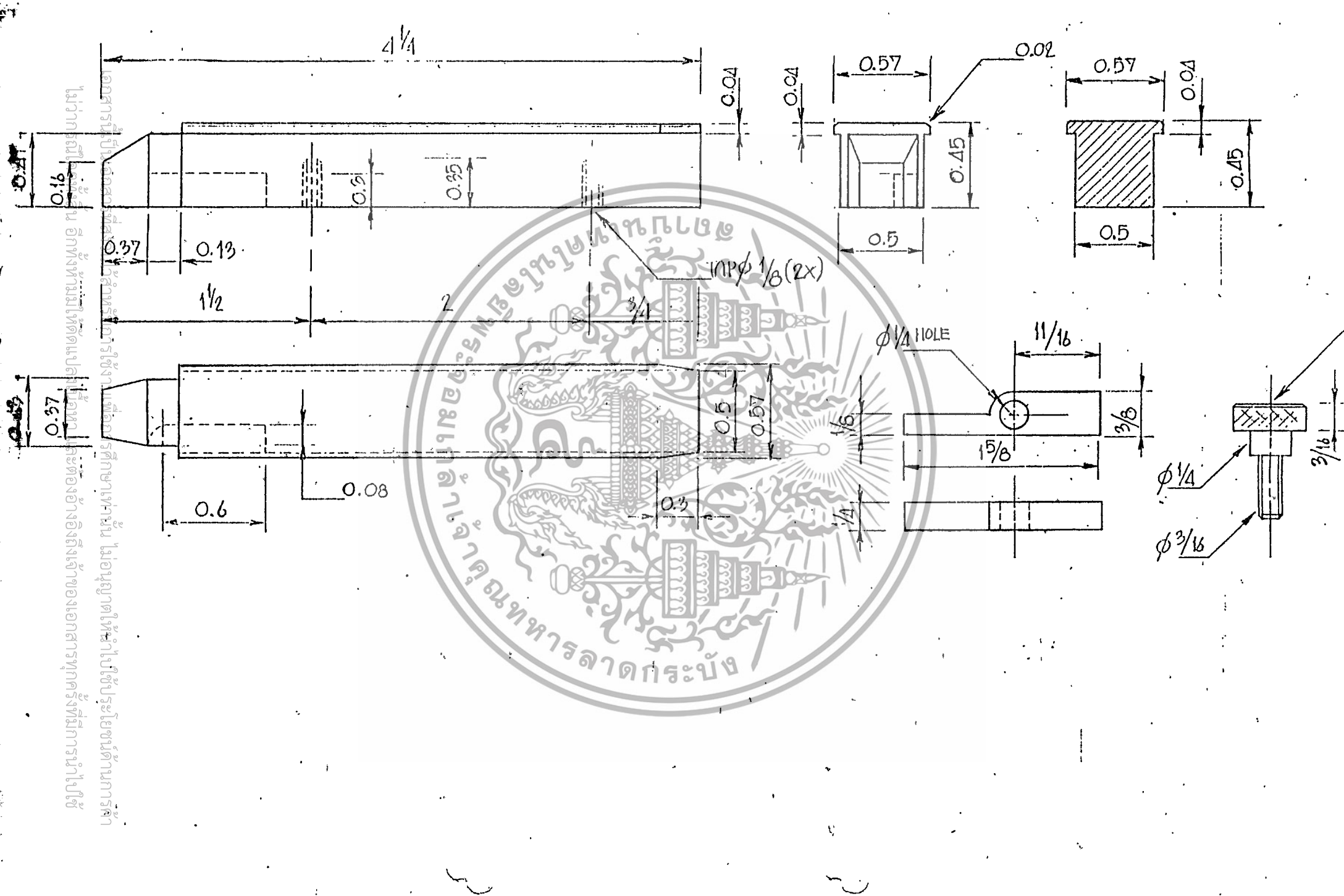
การทดลองโครงการนี้ได้ประสบผลสำเร็จตามจุดมุ่งหมายที่ตั้งไว้ และได้นำเครื่องที่ได้นี้ไปใช้งานจริงในโรงงานอุตสาหกรรมประกอบ IC บริษัทงานทวีอิเล็คทรอนิคส์ ซึ่งขณะนี้ก็ยังใช้งานอยู่ทุกวัน โดยให้พนักงาน (OPERATOR) เป็นผู้ควบคุม ส่วนการบำรุงรักษาได้มอบหมายให้ฝ่ายบำรุงรักษาของโรงงานเป็นผู้ดูแล

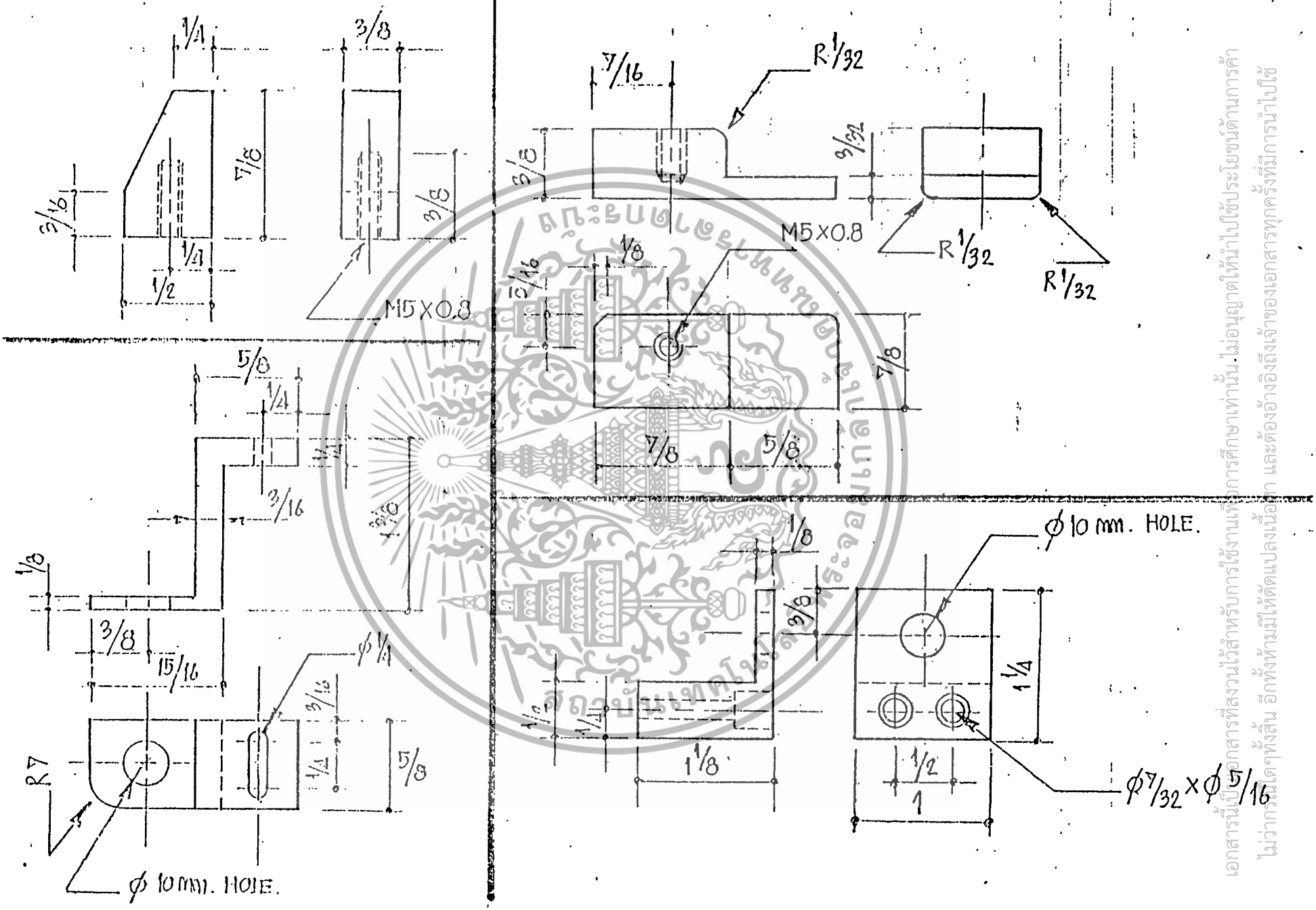
ปัญหาจากการทดลองโครงการนี้ก็มีในส่วนของ MECHANIC ที่ส่วนประกอบบางตัวไม่สามารถประกอบลงตัวได้จึงต้องมีการเปลี่ยนแปลงแบบบางส่วนเพื่อความเหมาะสมและสามารถหาวัสดุอุปกรณ์ได้ จึงเป็นผลให้ส่วนของ MECHANIC DESIGN ใช้ช้ากว่าส่วนของ ELECTRONIC ส่วนทางด้าน ELECTRONIC นั้นเกิดปัญหาน้อยเพราะสามารถหาอุปกรณ์ได้ง่ายกว่าและผู้ออกแบบถนัดทางด้าน ELECTRONIC มากกว่า

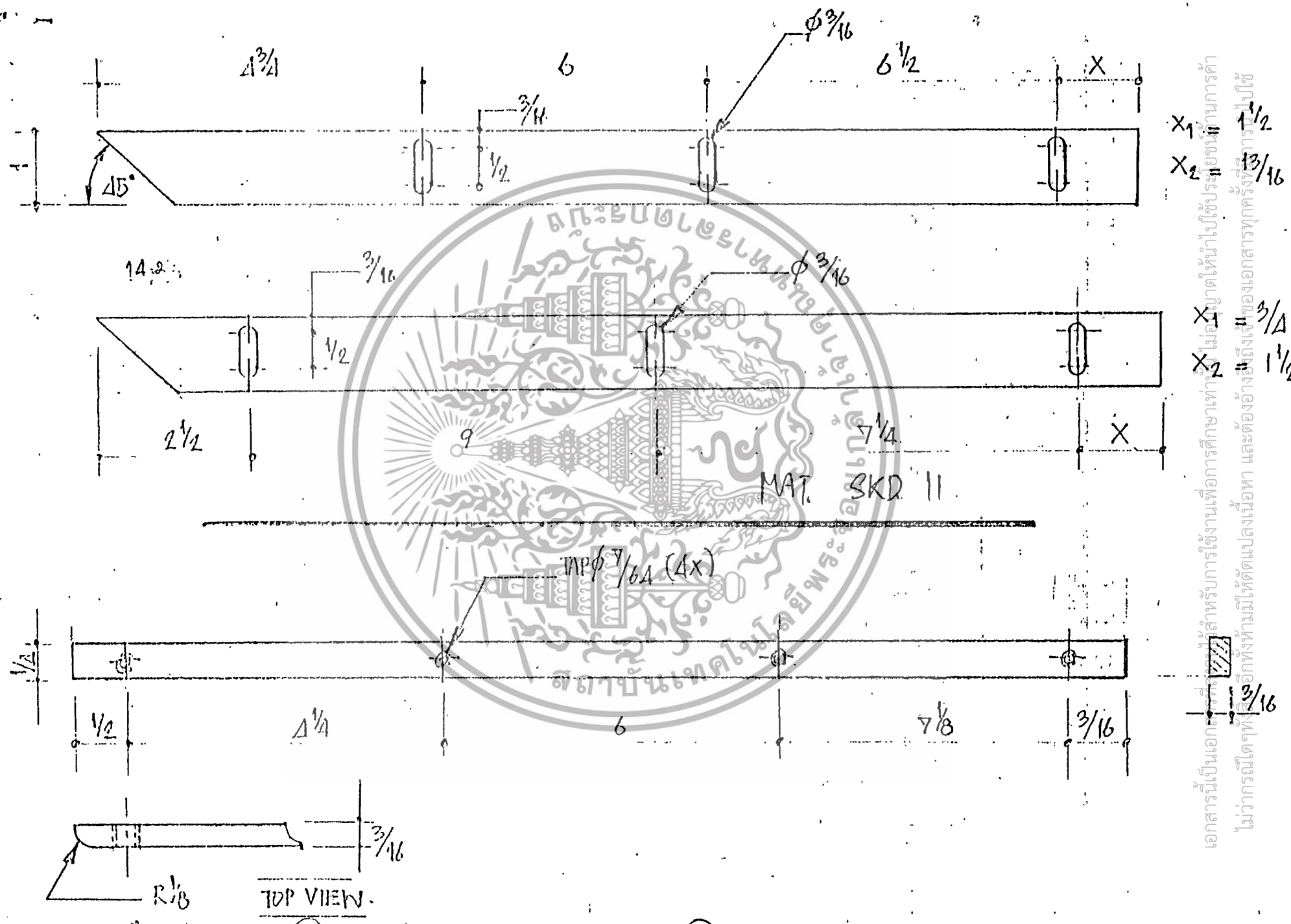
ปัญหาของเครื่องในการใช้งานจริง ๆ ส่วนใหญ่จะเกิดจากการ ALIGNMENT ส่วนของ MECHANIC ให้ได้ตามต้องการและการเปลี่ยนเครื่องเพื่อให้สามารถ RUN งานตามต้องการ ดังนั้นผู้ออกแบบต้องรับคำปรึกษาจากผู้ที่เกี่ยวข้องทุกฝ่าย

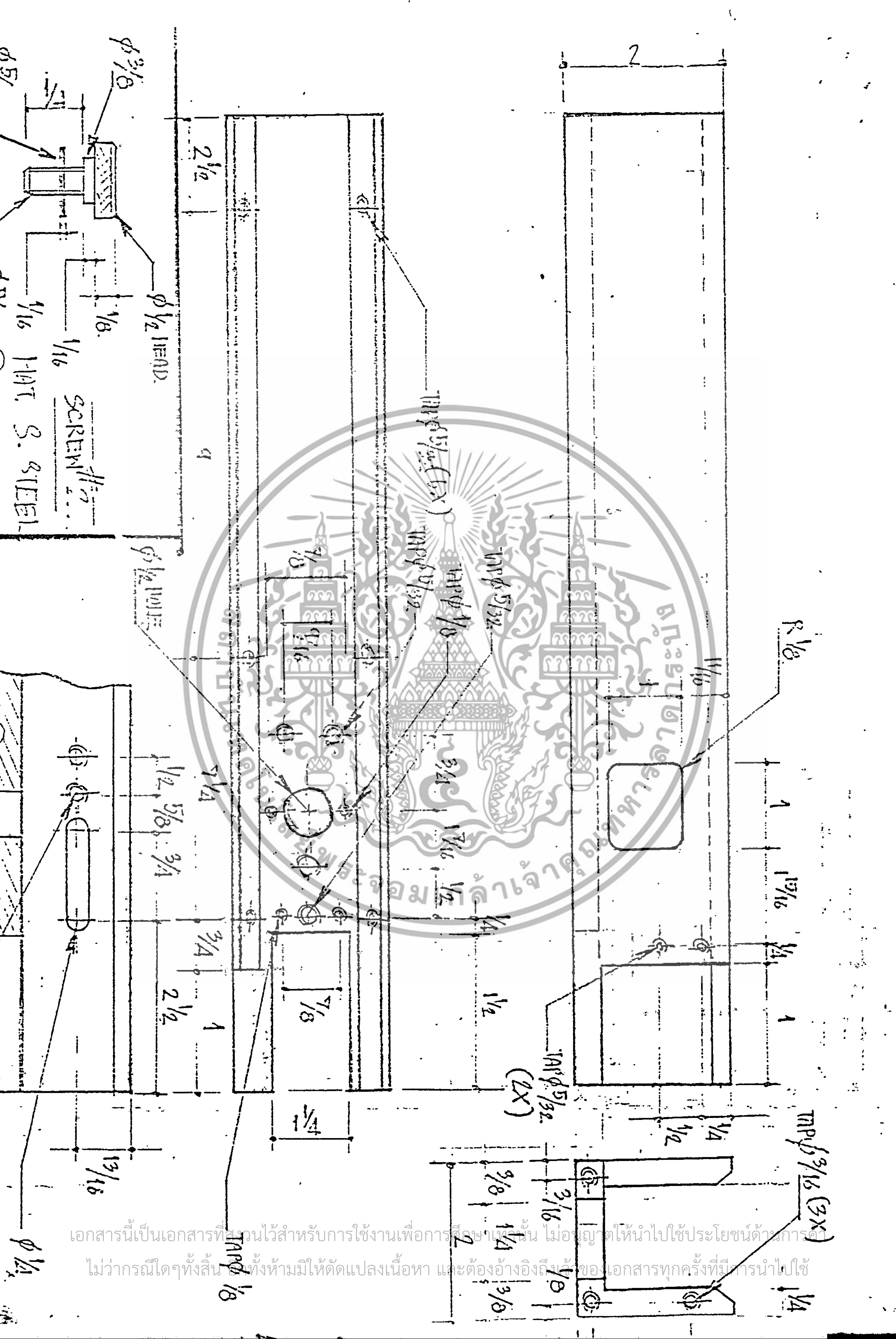


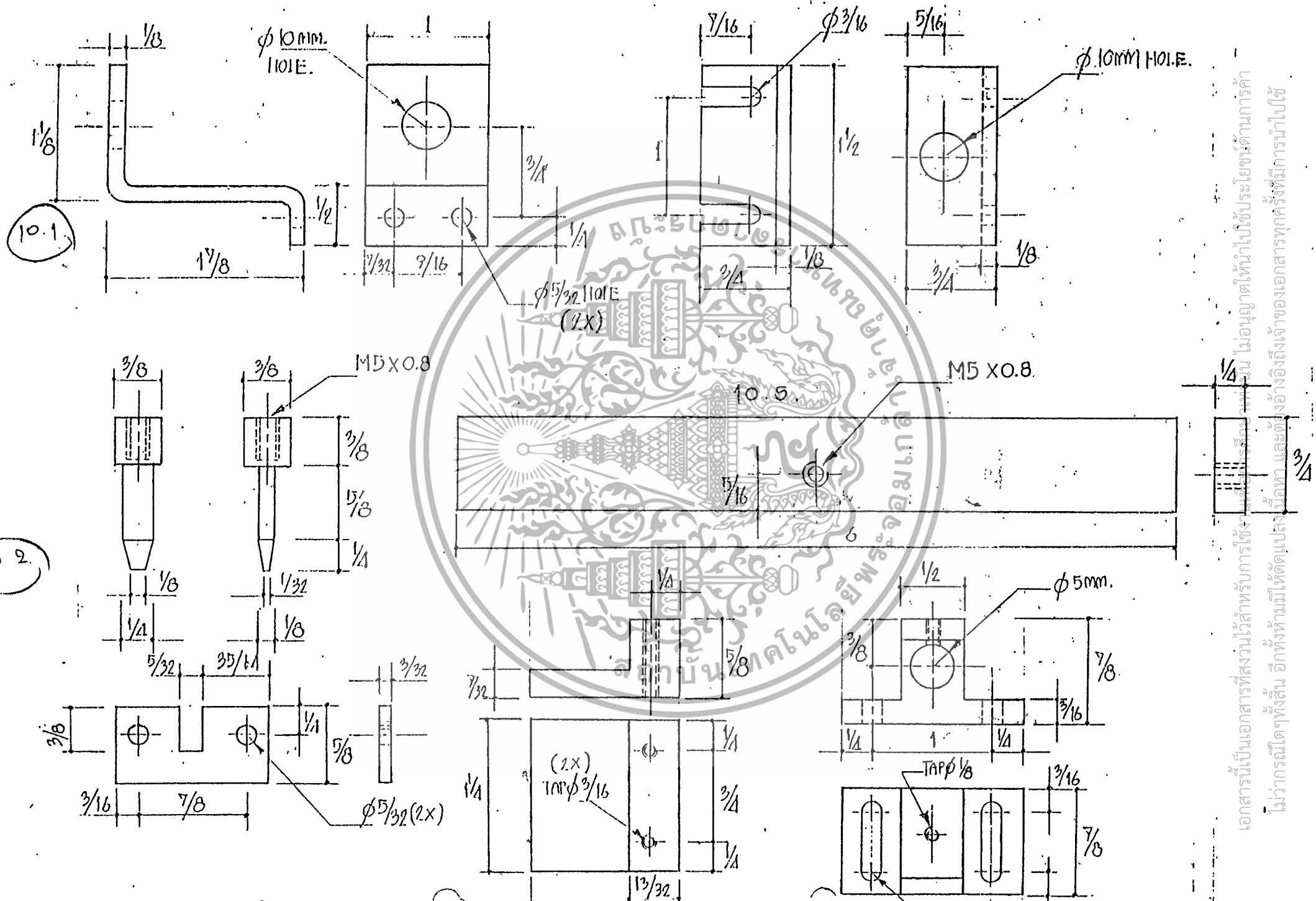


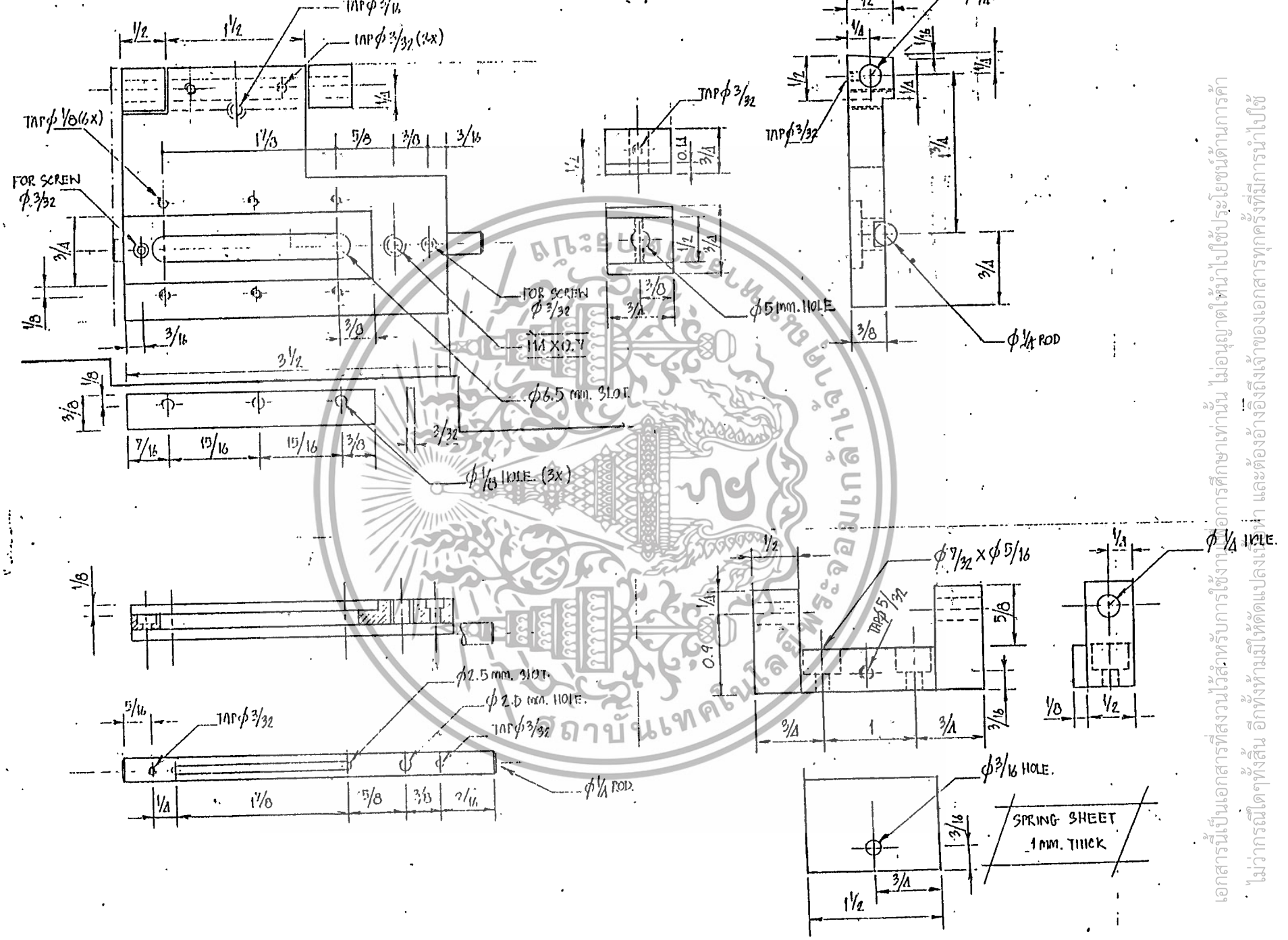

4 1/4
0.04
0.21
0.16
0.21
0.35
0.37
0.13
1 1/2
2
3/4
TAP φ 1/8 (2X)
0.04
0.57
0.02
0.45
0.5
0.57
0.04
0.45
0.5
φ 1/4 HOLE
11/16
3/8
1 5/8
1/16
1/4
0.37
0.69
0.5
0.57
0.08
0.6
0.3
φ 1/4
φ 3/16
3/16

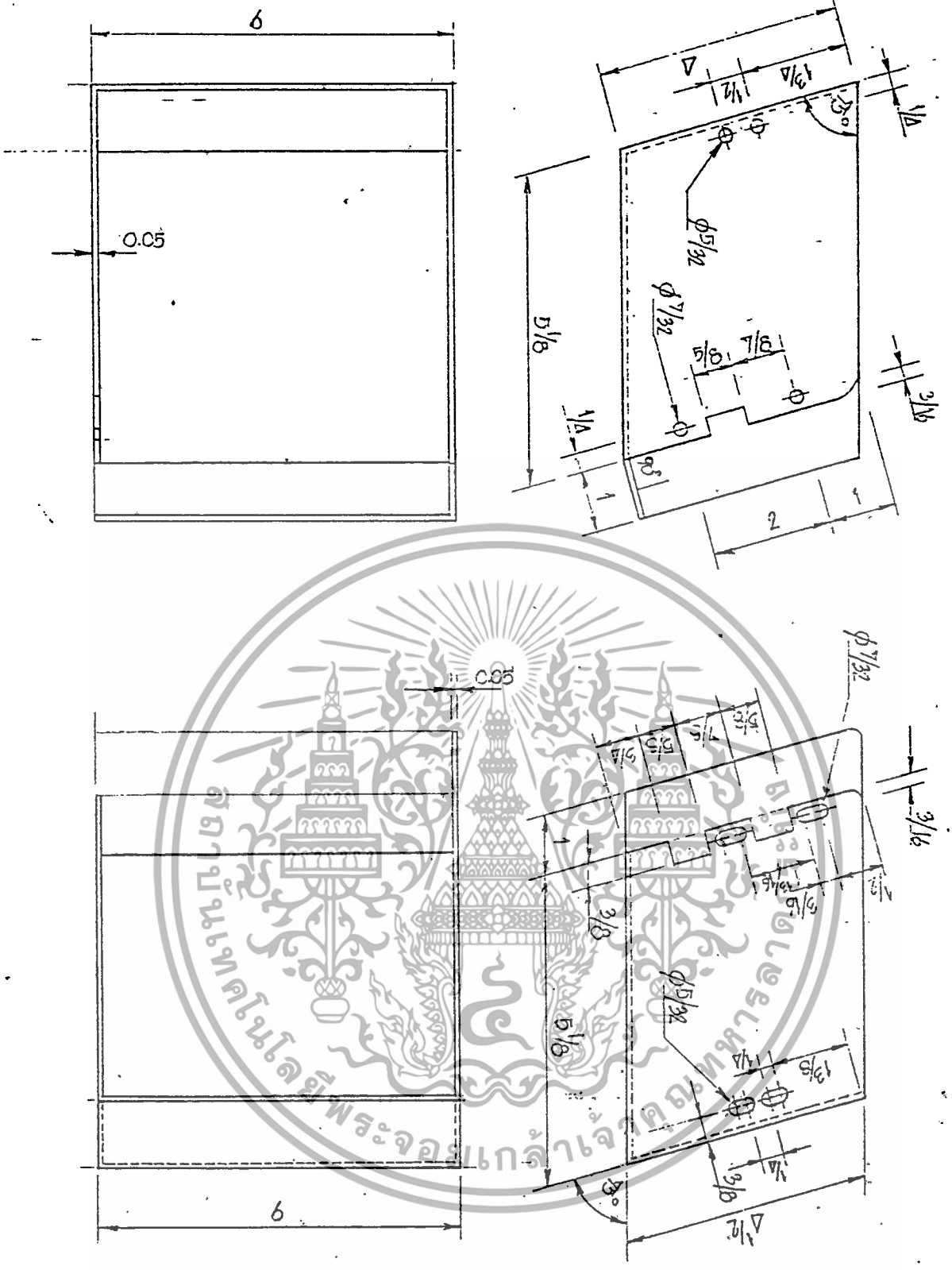




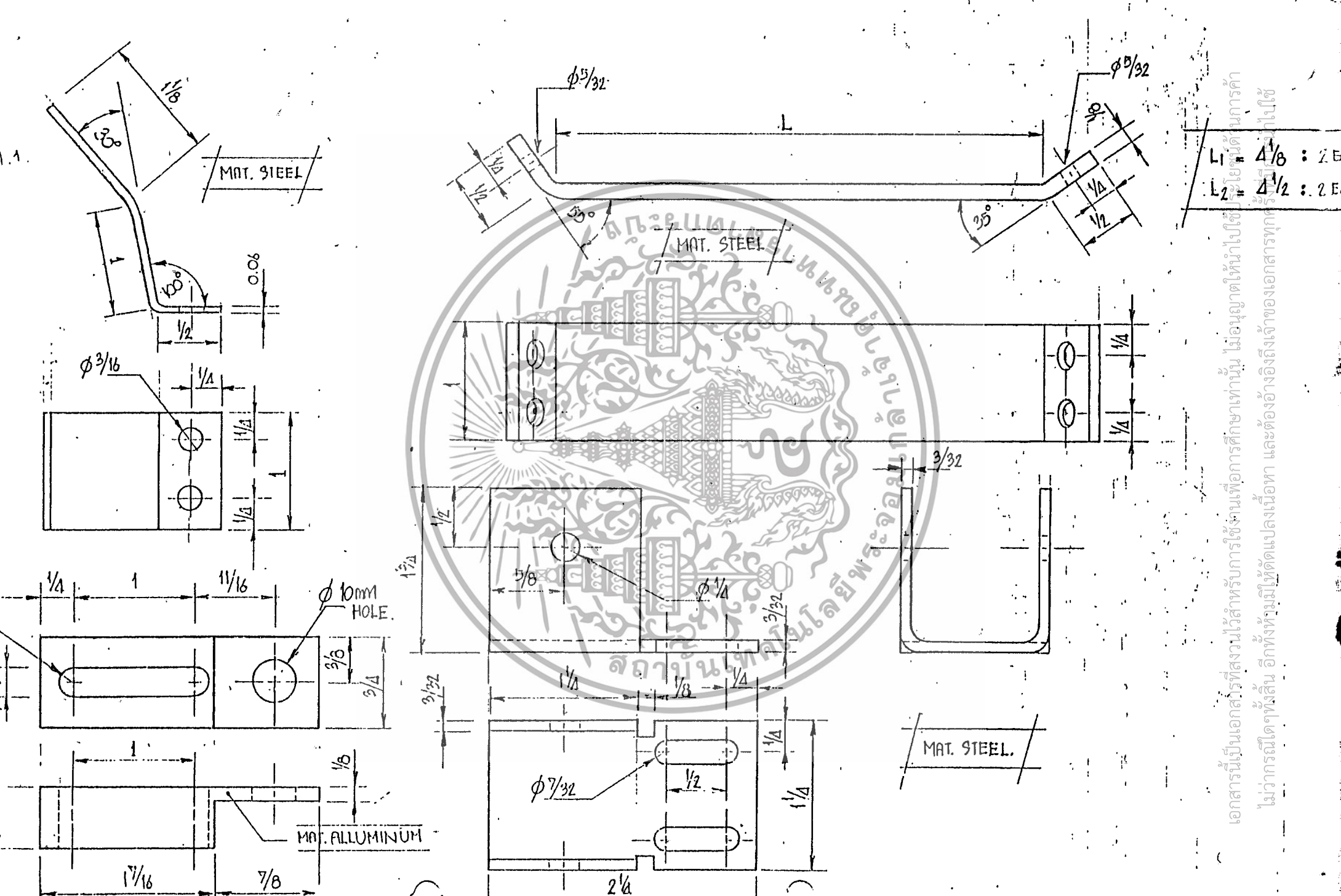


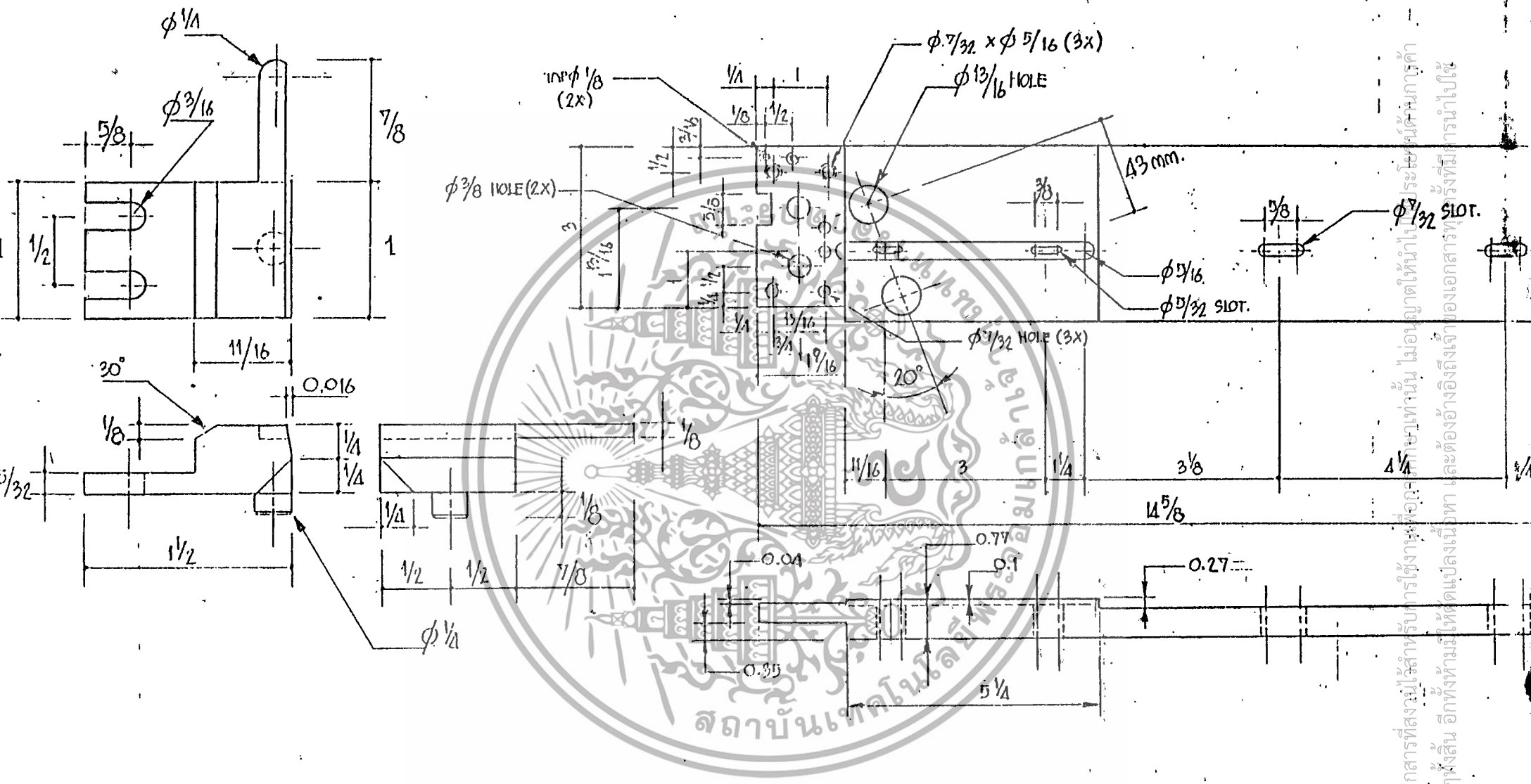


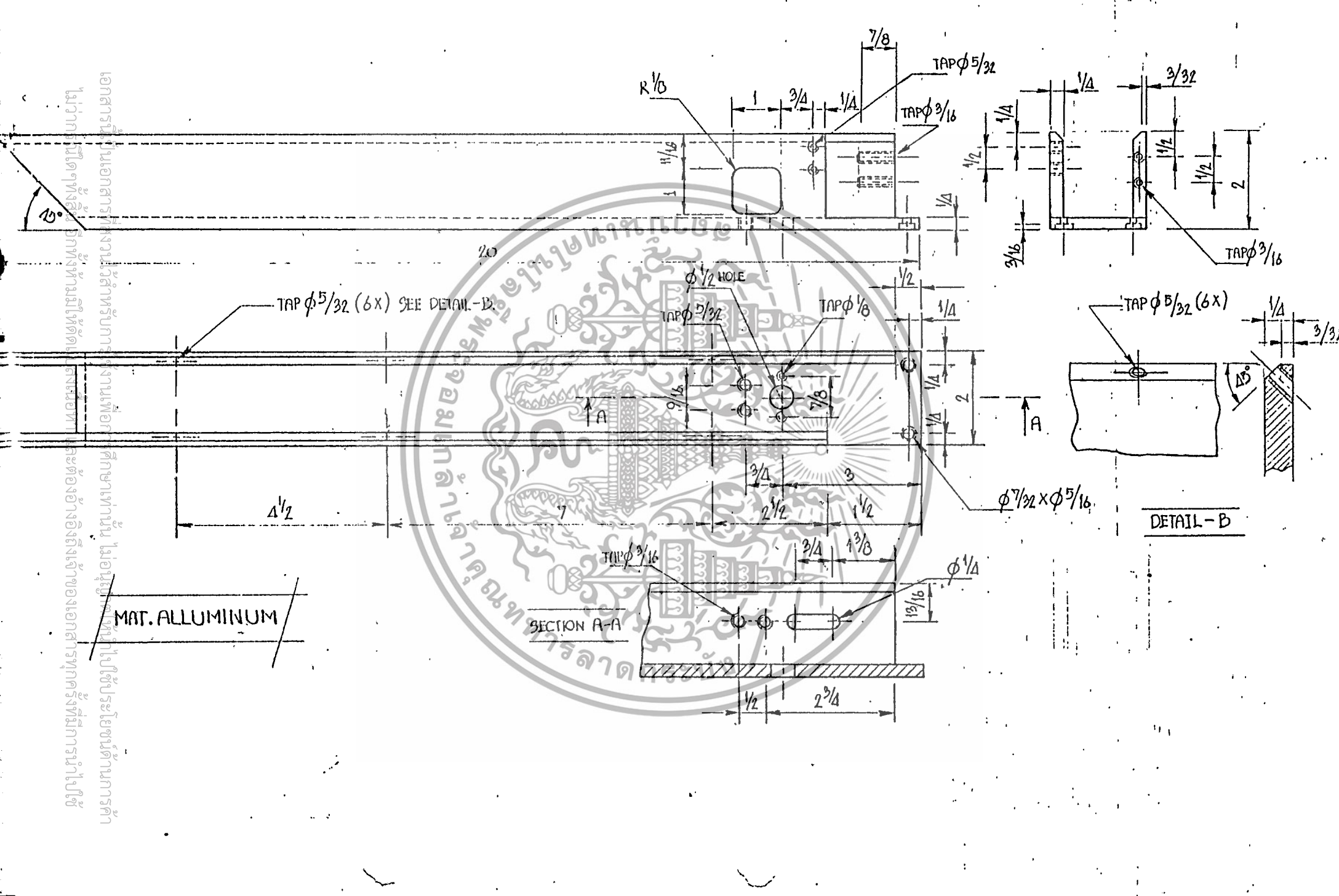


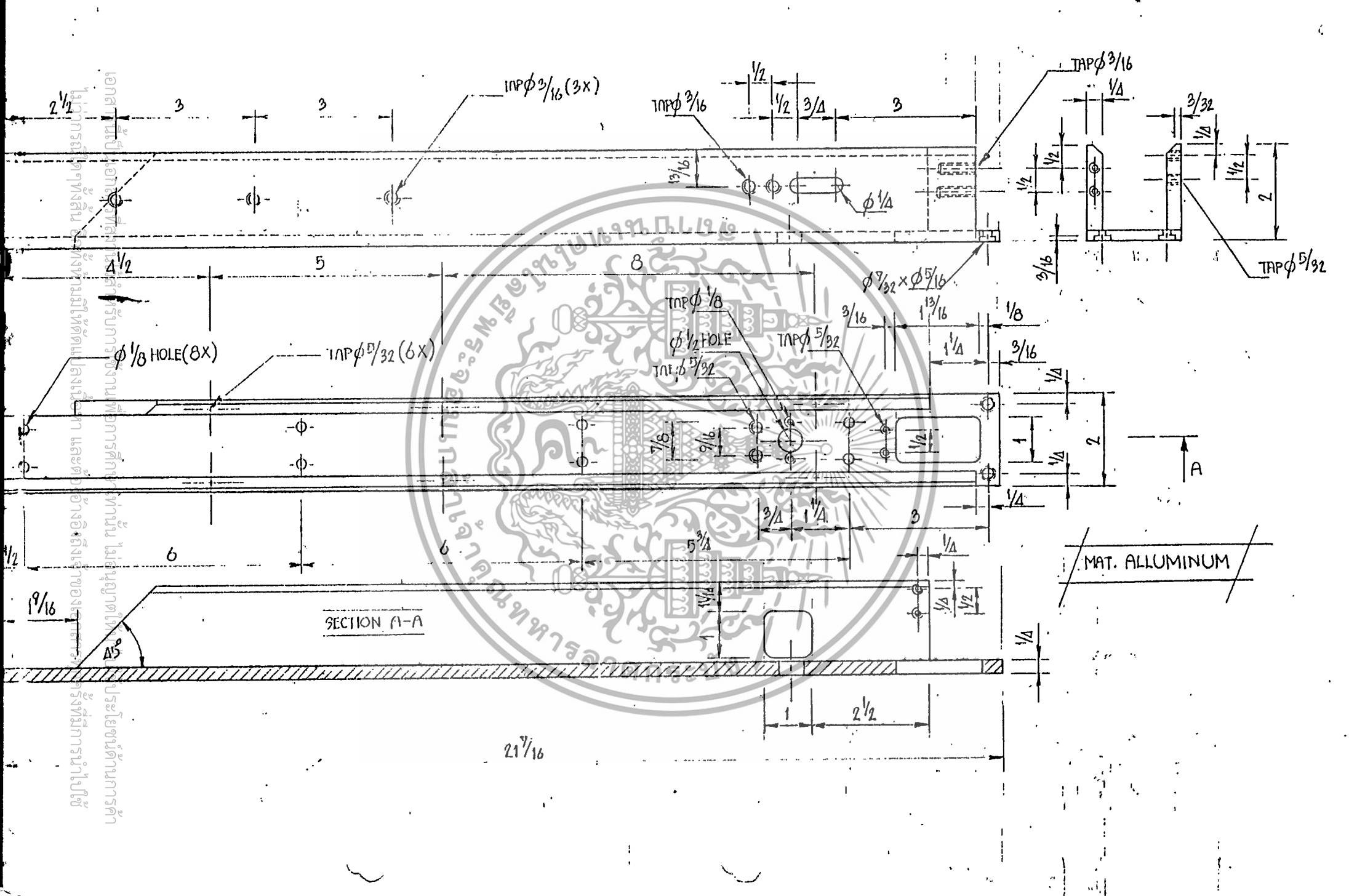


MAT. STEEL SHEET


1 1/8
30°
MAT. STEEL
L
100°
0.06
1/2
φ 3/16
1/4
1 1/2
1
1/2
φ 3/32
1/4
1/2
30°
35°
φ 3/32
1/4
1/2
L1 = 4 1/8 : 2 E
L2 = 4 1/2 : 2 E
MAT. STEEL
1/4
1
1 1/16
φ 10mm HOLE.
3/8
3/4
1
1/8
MAT. ALLUMINUM
1 7/16
7/8
1 3/4
1/2
5/8
φ 1/4
3/32
3/32
1 1/4
1/8
1/4
φ 7/32
1/2
1 1/4
2 1/4
3/32
MAT. STEEL.




7/8
TAP φ 5/32
R 1/8
1
3/4
1/4
TAP φ 3/16
11/16
1/4
15°
20
TAP φ 5/32 (6X) SEE DETAIL-B.
φ 1/2 HOLE
TAP φ 5/32
TAP φ 1/8
1/2
1/4
1/4
2
1/2
9/16
7/8
A
A
3/4
3
2 1/2
1 1/2
4 1/2
7
φ 7/32 x φ 5/16
3/4
1 3/8
TAP φ 3/16
φ 1/4
13/16
SECTION A-A
1/2
2 3/4
MAT. ALLUMINUM
1/4
3/32
1/4
1/2
1 1/2
2
3/16
TAP φ 3/16
TAP φ 5/32 (6X)
1/4
3/32
45°
DETAIL-B


TAP ϕ 3/16 (3X)
TAP ϕ 3/16
TAP ϕ 3/16
TAP ϕ 5/32
ϕ 1/4
ϕ 7/32 × ϕ 5/16
TAP ϕ 1/8
ϕ 1/2 HOLE
TAP ϕ 5/32
TAP ϕ 5/32
ϕ 1/8 HOLE (8X)
TAP ϕ 5/32 (6X)
A
MAT. ALLUMINUM
SECTION A-A
45°
2 1/2
3
3
4 1/2
5
8
6
6
1/2
3/4
3
1 3/16
1/8
3/16
1 1/4
3/16
1/4
2
1
1/4
1/2
3/4
1 1/4
3
5 3/4
1/4
1 9/16
1 1/4
1
1/4
1/2
1/4
1
2 1/2
21 7/16
7/8
9/16
1/2
3/32
1/4
1/2
1/2
2
3/16

รหัสคำสั่ง 8080 และ Z-80

8080 Mnemonic	Z-80 Mnemonic
ACI	ADC A,N
ADC M	ADC A,(HL)
ADC r	ADC A,R
ADD M	ADD A,(HL)
ADD r	ADD A,R
ADI	ADD A,N
ANA M	AND (HL)
ANA r	AND R
ANI	AND N
CALL	CALL NN
CC	CALL C,NN
CM	CALL M,NN
CMA	CPL
CMC	CCF
CMP M	CP (HL)
CMP r	CP R
CNC	CALL NC,NN
CNZ	CALL NZ,NN
CP	CALL P,NN
CP E	CALL PE,NN
CPI	CP N
CPO	CALL PO,NN
CZ	CALL Z,NN
DAA	DAA
DAD B	ADD HL,BC
DAD D	ADD HL,DE
DAD H	ADD HL,HL
DAD SP	ADD HL,SP
DCR M	DEC (HL)
DCR r	DEC R
DCX B	DEC BC
DCX D	DEC DE
DCX H	DEC HL
DCX SP	DEC SP
DI	DI
EI	EI
HLT	HALT

8080 Mnemonic	Z-80 Mnemonic
IN	IN A,(N)
INR M	INC (HL)
INR r	INC R
INX B	INC BC
INX D	INC DE
INX H	INC HL
INX SP	INC SP
JC	JP C,NN
JM	JP M,NN
JMP	JP NN
JNC	JP NC,NN
JNZ	JP NZ,NN
JP	JP P,NN
JPE	JP PE,NN
JPO	JP PO,NN
JZ	JP Z,NN
LDA	LD A,(NN)
LDAX B	LD A,(BC)
LDAX D	LD A,(DE)
LHLD	LD HL,(NN)
LXI B	LD BC,NN
LXI D	LD DE,NN
LXI H	LD HL,NN
LXI SP	LD SP,NN
MVI M	LD (HL),N
MVI r	LD R,N
MOV M,r	LD (HL),R
MOV r,M	LD R,(HL)
MOV r1,r2	LD R,R'
NOP	NOP
ORA M	OR (HL)
ORA r	OR R
ORI	OR N
OUT	OUT (N),A
PCHL	JP (HL)
POP B	POP BC
POP D	POP DE

8080 Mnemonic	Z-80 Mnemonic
POP H	POP HL
POP PSW	POP AF
PUSH B	PUSH BC
PUSH D	PUSH DE
PUSH H	PUSH HL
PUSH PSW	PUSH AF
RAL	RLA
RAR	RRA
RC	RET C
RET	RET
RLC	RLCA
RM	RET M
RNC	RET NC
RNZ	RET NZ
RP	RET P
RPE	RET PE
RPO	RET PO
RRC	RRCA
RST	RST P
RZ	RET Z
SBB M	SBC A,(HL)
SBB r	SBC A,R
SBI	SBC A,N
SHLD	LD (NN),HL
SPHL	LD SP,HL
STA	LD (NN),A
STAX B	LD (BC),A
STAX D	LD (DE),A
STC	SCF
SUB M	SUB (HL)
SUB r	SUB R
SUI	SUB N
XCHG	EX DE,HL
XRA M	XOR (HL)
XRA r	XOR R
XRI	XOR N
XTHL	EX (SP),HL

รายละเอียดข้อมูลไมโครโปรเซสเซอร์และชิพสนับสนุน

Z80®-CPU
Z80A-CPU

Product Specification

MARCH 1978

The Zilog Z80 product line is a complete set of microcomputer components, development systems and support software. The Z80 microcomputer component set includes all of the circuits necessary to build high-performance microcomputer systems with virtually no other logic and a minimum number of low cost standard memory elements.

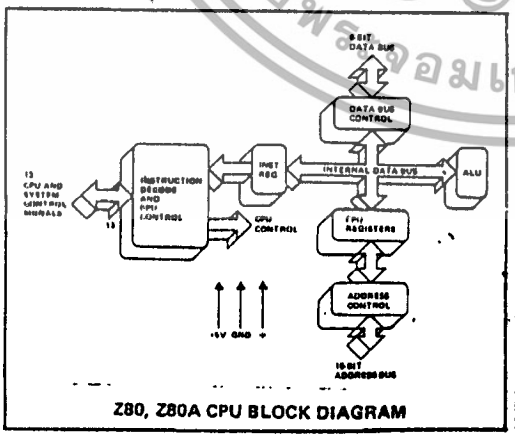
The Z80 and Z80A CPU's are third generation single chip microprocessors with unrivaled computational power. This increased computational power results in higher system through-put and more efficient memory utilization when compared to second generation microprocessors. In addition, the Z80 and Z80A CPU's are very easy to implement into a system because of their single voltage requirement plus all output signals are fully decoded and timed to control standard memory or peripheral circuits. The circuit is implemented using an N-channel, ion implanted, silicon gate MOS process.

Figure 1 is a block diagram of the CPU. Figure 2 details the internal register configuration which contains 208 bits of Read/Write memory that are accessible to the programmer. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or as 16-bit register pairs. There are also two sets of accumulator and flag registers. The programmer has access to either set of main or alternate registers through a group of exchange instructions. This alternate set allows foreground/background mode of operation or may be reserved for very fast interrupt response. Each CPU also contains a 16-bit stack pointer which permits simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data handling.

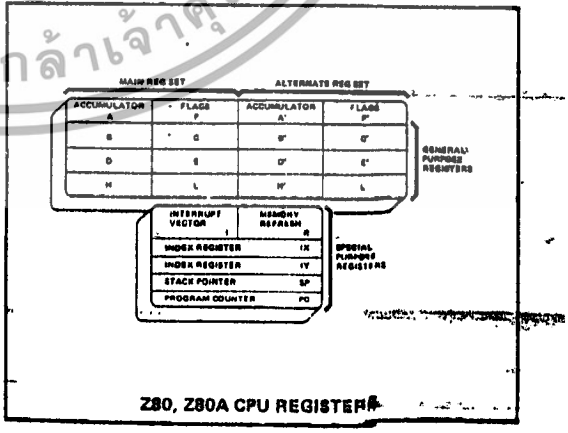
The two 16-bit index registers allow tabular data manipulation and easy implementation of relocatable code. The Refresh register provides for automatic, totally transparent refresh of external dynamic memories. The I register is used in a powerful interrupt response mode to form the upper 8 bits of a pointer to a interrupt service address table, while the interrupting device supplies the lower 8 bits of the pointer. An indirect call is then made to this service address.

FEATURES

- Single chip, N-channel Silicon Gate CPU.
- 158 instructions—includes all 78 of the 8080A instructions with total software compatibility. New instructions include 4-, 8- and 16-bit operations with more useful addressing modes such as indexed, bit and relative.
- 17 internal registers.
- Three modes of fast interrupt response plus a non-maskable interrupt.
- Directly interfaces standard speed static or dynamic memories with virtually no external logic.
- 1.0 μs instruction execution speed.
- Single 5 VDC supply and single-phase 5 volt Clock.
- Out-performs any other single chip microcomputer in 4-, 8-, or 16-bit applications.
- All pins TTL Compatible
- Built-in dynamic RAM refresh circuitry.

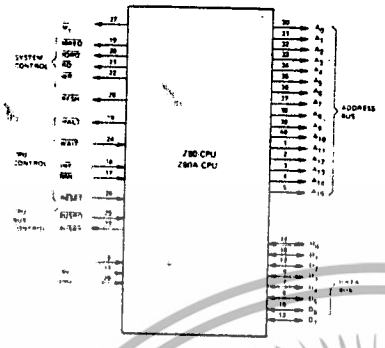


Z80, Z80A CPU BLOCK DIAGRAM



Z80, Z80A CPU REGISTERS

Z80, Z80A-CPU Pin Description



Z80, Z80A CPU PIN CONFIGURATION

A_0-A_{15} (Address Bus) Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges.

D_0-D_7 (Data Bus) Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.

$\overline{M_1}$ (Machine Cycle one) Output, active low. $\overline{M_1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution.

$\overline{MREQ}$ (Memory Request) Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.

$\overline{IORQ}$ (Input/ Output Request) Tri-state output, active low. The $\overline{IORQ}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{IORQ}$ signal is also generated when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus.

$\overline{RD}$ (Memory Read) Tri-state output, active low. $\overline{RD}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

$\overline{WR}$ (Memory Write) Tri-state output, active low. $\overline{WR}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

$\overline{RFSH}$ (Refresh) Output, active low. $\overline{RFSH}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current $\overline{MREQ}$ signal should be used to do a refresh read to all dynamic memories.

$\overline{HALT}$ (Halt state) Output, active low. $\overline{HALT}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non-maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.

$\overline{WAIT}$ (Wait) Input, active low. $\overline{WAIT}$ indicates to the Z-80 CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active.

$\overline{INT}$ (Interrupt Request) Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled.

$\overline{NMI}$ (Non Maskable Interrupt) Input, active low. The non-maskable interrupt request line has a higher priority than $\overline{INT}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{NMI}$ automatically forces the Z-80 CPU to restart to location 0066_H.

$\overline{RESET}$ Input, active low. $\overline{RESET}$ initializes the CPU as follows: reset interrupt enable flip-flop, clear PC and registers I and R and set Interrupt to 8080A mode. During reset time, the address and data bus go to a high impedance state and all control output signals go to the inactive state.

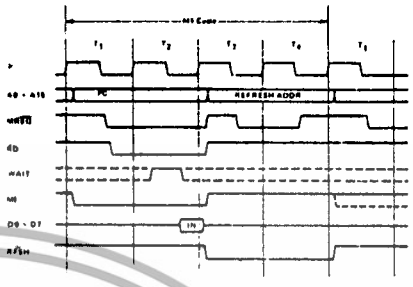
$\overline{BUSRQ}$ (Bus Request) Input, active low. The bus request signal has a higher priority than $\overline{NMI}$ and is always recognized at the end of the current machine cycle and is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these busses.

$\overline{BUSAK}$ (Bus Acknowledge) Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.

Timing Waveforms

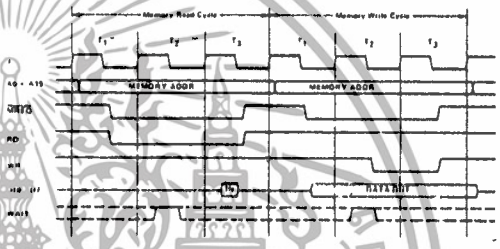
INSTRUCTION OP CODE FETCH

The program counter content (PC) is placed on the address bus immediately at the start of the cycle. One half clock time later $\overline{MREQ}$ goes active. The falling edge of $\overline{MREQ}$ can be used directly as a chip enable to dynamic memories. $\overline{RD}$ when active indicates that the memory data should be enabled onto the CPU data bus. The CPU samples data with the rising edge of the clock state T_3. Clock states T_3 and T_4 of a fetch cycle are used to refresh dynamic memories while the CPU is internally decoding and executing the instruction. The refresh control signal $\overline{RFSH}$ indicates that a refresh read of all dynamic memories should be accomplished.



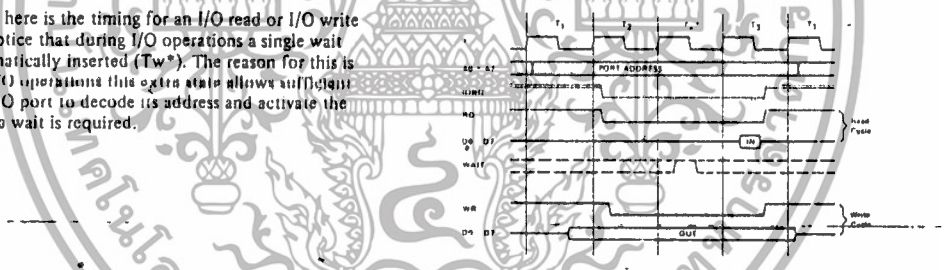
MEMORY READ OR WRITE CYCLES

Illustrated here is the timing of memory read or write cycles other than an OP code fetch (M_1 cycle). The $\overline{MREQ}$ and $\overline{RD}$ signals are used exactly as in the fetch cycle. In the case of a memory write cycle, the $\overline{MREQ}$ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The $\overline{WR}$ line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory.



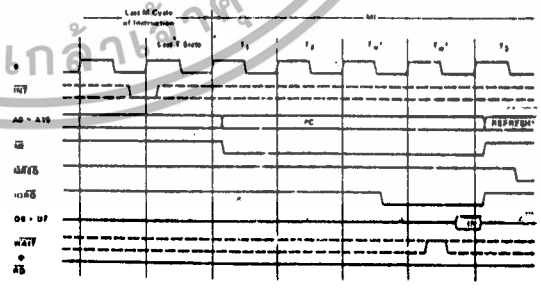
INPUT OR OUTPUT CYCLES

Illustrated here is the timing for an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted (Tw*). The reason for this is that during I/O operations this extra state allows sufficient time for an I/O port to decode its address and activate the $\overline{WAIT}$ line if a wait is required.



INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

The interrupt signal is sampled by the CPU with the rising edge of the last clock at the end of any instruction. When an interrupt is accepted, a special M_1 cycle is generated. During this M_1 cycle, the $\overline{IORQ}$ signal becomes active (instead of $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. Two wait states (Tw*) are automatically added to this cycle so that a ripple priority interrupt scheme, such as the one used in the Z80 peripheral controllers, can be easily implemented.



Z80, Z80A Instruction Set

The following is a summary of the Z80, Z80A instruction set showing the assembly language mnemonic and the symbolic operation performed by the instruction. A more detailed listing appears in the Z80-CPU technical manual, and assembly language programming manual. The instructions are divided into the following categories:

- 8-bit loads
- 16-bit loads
- Exchanges
- Memory Block Moves
- Memory Block Searches
- 8-bit arithmetic and logic
- 16-bit arithmetic
- General purpose Accumulator & Flag Operations
- Miscellaneous Group
- Rotates and Shifts
- Bit Set, Reset and Test
- Input and Output
- Jumps
- Calls
- Restarts
- Returns

In the table the following terminology is used.

- b ≡ a bit number in any 8-bit register or memory location
- cc ≡ flag condition code
  - NZ ≡ non zero
  - Z ≡ zero
  - NC ≡ non carry
  - C ≡ carry
  - PO ≡ Parity odd or no over flow
  - PE ≡ Parity even or over flow
  - P ≡ Positive
  - M ≡ Negative (minus)
- d ≡ any 8-bit destination register or memory location
- dd ≡ any 16-bit destination register or memory location
- e ≡ 8-bit signed 2's complement displacement used in relative jumps and indexed addressing
- L ≡ 8 special call locations in page zero. In decimal notation these are 0, 8, 16, 24, 32, 40, 48 and 56
- n ≡ any 8-bit binary number
- nn ≡ any 16-bit binary number
- r ≡ any 8-bit general purpose register (A, B, C, D, E, H, or L)
- s ≡ any 8-bit source register or memory location
- sb ≡ a bit in a specific 8-bit register or memory location
- ss ≡ any 16-bit source register or memory location
- subscript "L" ≡ the low order 8 bits of a 16-bit register
- subscript "H" ≡ the high order 8 bits of a 16-bit register
- ( ) ≡ the contents within the ( ) are to be used as a pointer to a memory location or I/O port number

8-bit registers are A, B, C, D, E, H, L, I and R
16-bit register pairs are AF, BC, DE and HL
16-bit registers are SP, PC, IX and IY

Addressing Modes implemented include combinations of the following:

- Immediate
- Immediate extended
- Modified Page Zero
- Relative
- Extended
- Indexed
- Register
- Implied
- Register Indirect
- Bit

8-BIT LOADS

Mnemonic	Symbolic Operation	Comments
LD r, s	r ← s	s ≡ r, n, (HL), (IX+e), (IY+e)
LD d, r	d ← r	d ≡ (HL), r (IX+e), (IY+e)
LD d, n	d ← n	d ≡ (HL), (IX+e), (IY+e)
LD A, s	A ← s	s ≡ (BC), (DE), (nn), I, R
LD d, A	d ← A	d ≡ (BC), (DE), (nn), I, R

16-BIT LOADS

Mnemonic	Symbolic Operation	Comments
LD dd, nn	dd ← nn	dd ≡ BC, DE, HL, SP, IX, IY
LD dd, (nn)	dd ← (nn)	dd ≡ BC, DE, HL, SP, IX, IY
LD (nn), ss	(nn) ← ss	ss ≡ BC, DE, HL, SP, IX, IY
LD SP, ss	SP ← ss	ss ≡ HL, IX, IY
PUSH ss	$(SP-1) \leftarrow ss_H$; $(SP-2) \leftarrow ss_L$	ss ≡ BC, DE, HL, AF, IX, IY
POP dd	$dd_L \leftarrow (SP)$; $dd_H \leftarrow (SP+1)$	dd ≡ BC, DE, HL, AF, IX, IY

EXCHANGES

Mnemonic	Symbolic Operation	Comments
EX DE, HL	DE ↔ HL	
EX AF, AF'	AF ↔ AF'	
EXX	$\begin{pmatrix} BC \\ DE \\ HL \end{pmatrix} \leftrightarrow \begin{pmatrix} BC' \\ DE' \\ HL' \end{pmatrix}$	
EX (SP), ss	$(SP) \leftarrow ss_L$, $(SP+1) \leftarrow ss_H$	ss ≡ HL, IX, IY

MEMORY BLOCK MOVES

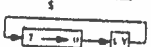
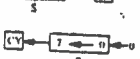
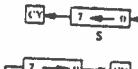
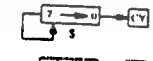
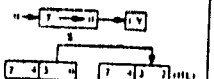
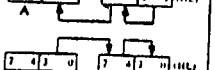
Mnemonic	Symbolic Operation	Comments
LDI	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC-1	
LDIR	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC-1 Repeat until BC = 0	
LDD	(DE) ← (HL), DE ← DE-1 HL ← HL-1, BC ← BC-1	
LDDR	(DE) ← (HL), DE ← DE-1 HL ← HL-1, BC ← BC-1 Repeat until BC = 0	

MEMORY BLOCK SEARCHES

Mnemonic	Symbolic Operation	Comments
CPI	A-(HL), HL ← HL+1 BC ← BC-1	A-(HL) sets the flags only, A is not affected
CPIR	A-(HL), HL ← HL+1 BC ← BC-1, Repeat until BC = 0 or A = (HL)	
CPD	A-(HL), HL ← HL-1 BC ← BC-1	
CPDR	A-(HL), HL ← HL-1 BC ← BC-1, Repeat until BC= 0 or A = (HL)	

8-BIT ALU

Mnemonic	Symbolic Operation	Comments
ADD s	A ← A + s	CY is the carry flag s ≡ r, n, (HL) (IX+e), (IY+e)
ADC s	A ← A + s + CY	
SUB s	A ← A - s	
SBC s	A ← A - s - CY	
AND s	A ← A ∧ s	
OR s	A ← A ∨ s	
XOR s	A ← A ⊕ s	

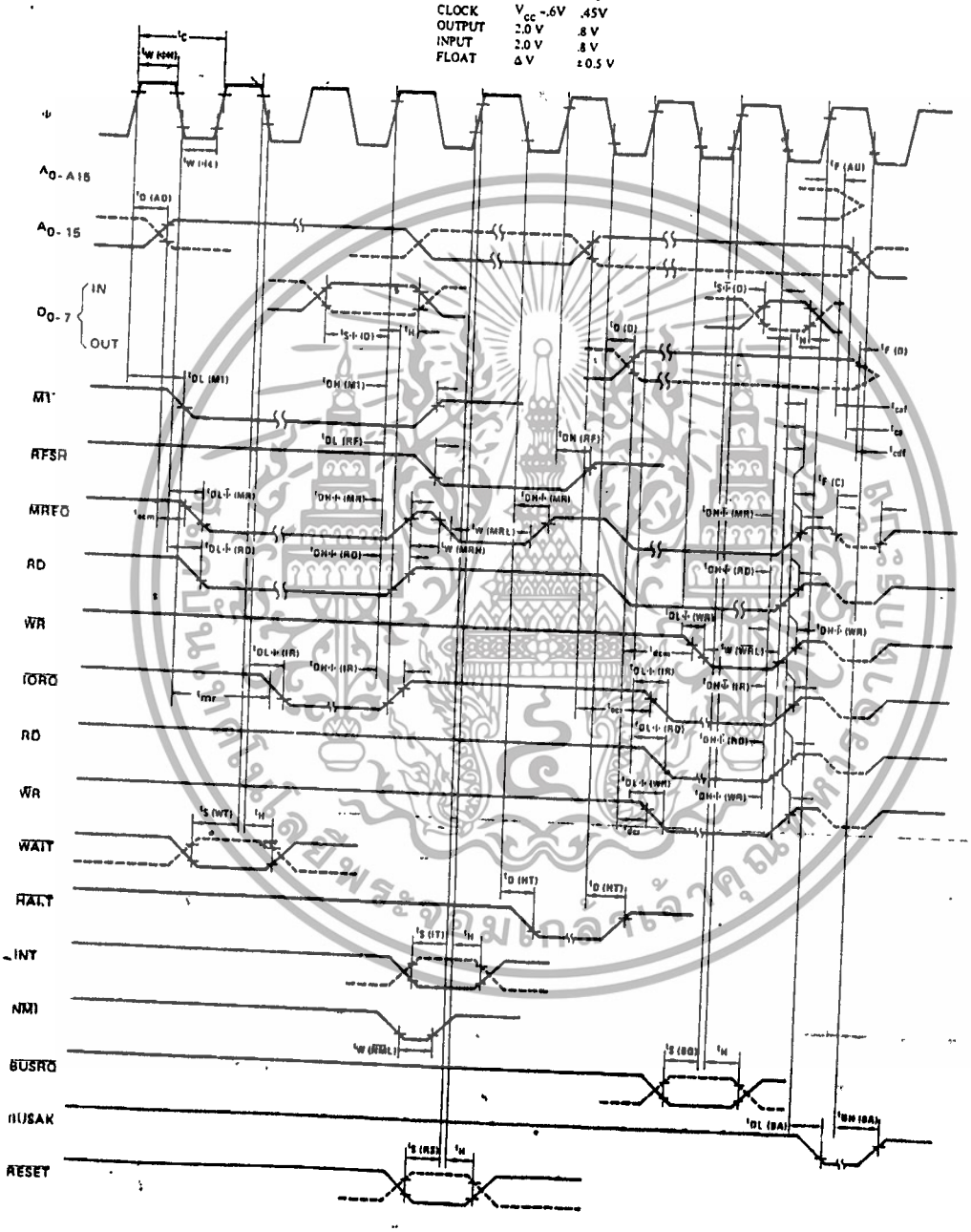
	Mnemonic	Symbolic Operation	Comments
8-BIT ALU	CP s	A - s	s = r, n (HL) (IX+e), (IY+e)
	INC d	d ← d + 1	d = r, (HL) (IX+e), (IY+e)
	DEC d	d ← d - 1	
16-BIT ARITHMETIC	ADD HL, ss	HL ← HL + ss	ss ≡ BC, DE HL, SP
	ADC HL, ss	HL ← HL + ss + CY	
	SBC HL, ss	HL ← HL - ss - CY	
	ADD IX, ss	IX ← IX + ss	ss ≡ BC, DE, IX, SP
	ADD IY, ss	IY ← IY + ss	ss ≡ BC, DE, IY, SP
	INC dd	dd ← dd + 1	dd ≡ BC, DE, HL, SP, IX, IY
	DEC dd	dd ← dd - 1	dd ≡ BC, DE, HL, SP, IX, IY
GP ACC. & FLAG	DAA	Converts A contents into packed BCD following add or subtract.	Operands must be in packed BCD format
	CPL	$A \leftarrow \overline{A}$	
	NEG	A ← 00 - A	
	CCF	$CY \leftarrow \overline{CY}$	
	SCF	CY ← 1	
MISCELLANEOUS	NOP	No operation	
	HALT	Halt CPU	
	DI	Disable Interrupts	
	EI	Enable Interrupts	
	IM 0	Set interrupt mode 0	8080A mode
	IM 1	Set interrupt mode 1	Call to 0038_H
	IM 2	Set interrupt mode 2	Indirect Call
ROTATES AND SHIFTS	RLC s		s ≡ r, (HL) (IX+e), (IY+e)
	RL s		
	RRC s		
	RR s		
	SLA s		
	SRA s		
	SRL s		
	RLD		
	RRD		

	Mnemonic	Symbolic Operation	Comments
BIT S, R, & T	BIT b, s	$Z \leftarrow \overline{s_b}$	Z is zero flag
	SET b, s	$s_b \leftarrow 1$	s ≡ r, (HL) (IX+e), (IY+e)
	RES b, s	$s_b \leftarrow 0$	
INPUT AND OUTPUT	IN A, (n)	A ← (n)	Set flags
	IN r, (C)	r ← (C)	
	INI	(HL) ← (C), HL ← HL + 1 B ← B - 1	
	INIR	(HL) ← (C), HL ← HL + 1 B ← B - 1 Repeat until B = 0	
	IND	(HL) ← (C), HL ← HL - 1 B ← B - 1	
	INDR	(HL) ← (C), HL ← HL - 1 B ← B - 1 Repeat until B = 0	
	OUT(n), A	(n) ← A	
	OUT(C), r	(C) ← r	
	OUTI	(C) ← (HL), HL ← HL + 1 B ← B - 1	
	OTIR	(C) ← (HL), HL ← HL + 1 B ← B - 1 Repeat until B = 0	
	OUTD	(C) ← (HL), HL ← HL - 1 B ← B - 1	
	OTDR	(C) ← (HL), HL ← HL - 1 B ← B - 1 Repeat until B = 0	
JUMPS	JP nn	PC ← nn	cc { NZ PO, Z PE, NC P, C M
	JP cc, nn	If condition cc is true PC ← nn, else continue	
	JR e	PC ← PC + e	kk { NZ NC, Z C
	JR kk, e	If condition kk is true PC ← PC + e, else continue	ss ≡ HL, IX, IY
	JP (ss)	PC ← ss	
	DJNZ e	B ← B - 1, if B ≠ 0 continue, else PC ← PC + e	
CALLS	CALL nn	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$, PC ← nn	cc { NZ PO, Z PE, NC P, C M
	CALL cc, nn	If condition cc is false continue, else same as CALL nn	
RESTARTS	RST L	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$, $PC_H \leftarrow 0$ $PC_L \leftarrow L$	
RETURNS	RET	$PC_L \leftarrow (SP)$, $PC_H \leftarrow (SP+1)$	cc { NZ PO, Z PE, NC P, C M
	RET cc	If condition cc is false continue, else same as RET	
	RETI	Return from interrupt, same as RET	
	RETN	Return from non-maskable interrupt	

A.C. Timing Diagram

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	V_{CC} -.6V	.45V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	Δ V	±0.5 V



T_A = 0°C to 70°C, Vcc = +5V ± 5%. Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
Φ	t_c	Clock Period	4	1121	μsec	
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	180	[E]	nsec	
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	180	2000	nsec	
	$t_{r,f}$	Clock Rise and Fall Time		10	nsec	
A_{0-15}	$t_{D(AD)}$	Address Output Delay		145	nsec	C_L = 50pF
	$t_{F(AD)}$	Delay to Float		110	nsec	
	t_{acm}	Address Stable Prior to MREQ (Memory Cycle)	[1]		nsec	
	t_{aci}	Address Stable Prior to IORQ, RD or WR (I/O Cycle)	[2]		nsec	
	t_{ca}	Address Stable from RD, WR, IORQ or MREQ	[3]		nsec	
	t_{cal}	Address Stable From RD or WR During Float	[4]		nsec	
D_{0-7}	$t_{D(D)}$	Data Output Delay		230	nsec	C_L = 50pf
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{s\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
	$t_{s\overline{\Phi}(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	60		nsec	
	t_{dcm}	Data Stable Prior to WR (Memory Cycle)	[5]		nsec	
	t_{dci}	Data Stable Prior to WR (I/O Cycle)	[6]		nsec	
	t_{cdf}	Data Stable From WR	[7]		nsec	
	t_H	Any Hold Time for Setup Time	0		nsec	
MREQ	$t_{DL\overline{\Phi}(MR)}$	MREQ Delay From Falling Edge of Clock, MREQ Low		100	nsec	C_L = 50pF
	$t_{DH\Phi(MR)}$	MREQ Delay From Rising Edge of Clock, MREQ High		100	nsec	
	$t_{DH\overline{\Phi}(MR)}$	MREQ Delay From Falling Edge of Clock, MREQ High		100	nsec	
	$t_{w(MRL)}$	Pulse Width, MREQ Low	[8]		nsec	
	$t_{w(MRH)}$	Pulse Width, MREQ High	[9]		nsec	
IORQ	$t_{DL\Phi(IR)}$	IORQ Delay From Rising Edge of Clock, IORQ Low		90	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(IR)}$	IORQ Delay From Falling Edge of Clock, IORQ Low		110	nsec	
	$t_{DH\Phi(IR)}$	IORQ Delay From Rising Edge of Clock, IORQ High		100	nsec	
	$t_{DH\overline{\Phi}(IR)}$	IORQ Delay From Falling Edge of Clock, IORQ High		110	nsec	
RD	$t_{DL\Phi(RD)}$	RD Delay From Rising Edge of Clock, RD Low		100	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(RD)}$	RD Delay From Falling Edge of Clock, RD Low		130	nsec	
	$t_{DH\Phi(RD)}$	RD Delay From Rising Edge of Clock, RD High		100	nsec	
	$t_{DH\overline{\Phi}(RD)}$	RD Delay From Falling Edge of Clock, RD High		110	nsec	
WR	$t_{DL\Phi(WR)}$	WR Delay From Rising Edge of Clock, WR Low		80	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(WR)}$	WR Delay From Falling Edge of Clock, WR Low		90	nsec	
	$t_{DH\Phi(WR)}$	WR Delay From Rising Edge of Clock, WR High		100	nsec	
	$t_{w(WRL)}$	Pulse Width, WR Low	[10]		nsec	
M1	$t_{DL(M1)}$	M1 Delay From Rising Edge of Clock, M1 Low		130	nsec	C_L = 50pF
	$t_{DH(M1)}$	M1 Delay From Rising Edge of Clock, M1 High		130	nsec	
RFSH	$t_{DL(RF)}$	RFSH Delay From Rising Edge of Clock, RFSH Low		180	nsec	C_L = 50pF
	$t_{DH(RF)}$	RFSH Delay From Rising Edge of Clock, RFSH High		150	nsec	
WAIT	$t_{s(WT)}$	WAIT Setup Time to Falling Edge of Clock	70		nsec	
HALT	$t_{D(HT)}$	HALT Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF
INT	$t_{s(IT)}$	INT Setup Time to Rising Edge of Clock	80		nsec	
NMI	$t_{w(NML)}$	Pulse Width, NMI Low	80		nsec	
BUSRQ	$t_{s(BQ)}$	BUSRQ Setup Time to Rising Edge of Clock	80		nsec	
BUSAK	$t_{DL(BA)}$	BUSAK Delay From Rising Edge of Clock, BUSAK Low		120	nsec	C_L = 50pF
	$t_{DH(BA)}$	BUSAK Delay From Falling Edge of Clock, BUSAK High		110	nsec	
RESET	$t_{s(RS)}$	RESET Setup Time to Rising Edge of Clock	90		nsec	
	$t_{F(C)}$	Delay to Float (MREQ, IORQ, RD and WR)		100	nsec	
	t_{mr}	M1 Stable Prior to IORQ (Interrupt Ack.)	[11]		nsec	

[12] $t_c = t_{w(\Phi H)} + t_{w(\Phi L)} + t_r + t_f$

[1] $t_{acm} = t_{w(\Phi H)} + t_f - 75$

[2] $t_{aci} = t_c - 80$

[3] $t_{ca} = t_{w(\Phi L)} + t_f - 40$

[4] $t_{cal} = t_{w(\Phi L)} + t_f - 70$

[5] $t_{dcm} = t_c - 210$

[6] $t_{dci} = t_{w(\Phi L)} + t_f - 210$

[7] $t_{cdf} = t_{w(\Phi L)} + t_f - 80$

[8] $t_{w(MRL)} = t_c - 40$

[9] $t_{w(MRH)} = t_{w(\Phi H)} + t_f - 30$

[10] $t_{w(WRL)} = t_c - 40$

[11] $t_{mr} = 2t_c + t_{w(\Phi H)} + t_f - 80$

NOTES

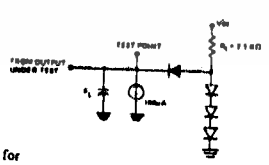
A. Data should be enabled onto the CPU data bus when RD is active. During interrupt acknowledge data should be enabled when M1 and IORQ are both active.

B. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.

C. The RFSH signal must be active for a minimum of 3 clock cycles.

D. Output Delay vs. Loaded Capacitance
TA = 70°C Vcc = +5V ±5%
Add 10nsec delay for each 50pf increase in load up to a maximum of 200pf for the data bus & 100pf for address & control lines

E. Although static by design, testing guarantees $t_{w(\Phi H)}$ of 200 μsec maximum



Load circuit for Output

T_A = 0°C to 70°C, Vcc = +5V ± 5%, Unless Otherwise Noted

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition	
Φ	t_c	Clock Period	.25	1121	μsec		[12] $t_c = t_{w(\Phi H)} + t_{w(\Phi L)} + t_r + t_f$
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	110	[E]	nsec		
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	110	2000	nsec		
	t_r, t_f	Clock Rise and Fall Time		30	nsec		
A_{0-15}	$t_{D(AD)}$	Address Output Delay		110	nsec	C_L = 50pF	[1] $t_{acm} = t_{w(\Phi H)} + t_f - 65$
	$t_{F(AD)}$	Delay to Float		90	nsec		[2] $t_{acl} = t_c - 70$
	t_{acm}	Address Stable Prior to MREQ (Memory Cycle)	[1]		nsec		[3] $t_{ca} = t_{w(\Phi L)} + t_r - 50$
	t_{acl}	Address Stable Prior to IORQ, RD or WR (I/O Cycle)	[2]		nsec		[4] $t_{caf} = t_{w(\Phi L)} + t_r - 45$
	t_{ca}	Address Stable from RD, WR, IORQ or MREQ	[3]		nsec		[5] $t_{dcm} = t_c - 170$
	t_{caf}	Address Stable From RD or WR During Float	[4]		nsec		[6] $t_{dcl} = t_{w(\Phi L)} + t_r - 170$
D_{0-7}	$t_{D(D)}$	Data Output Delay		150	nsec	C_L = 50pF	[7] $t_{cdf} = t_{w(\Phi L)} + t_r - 70$
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec		
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	35		nsec		[8] $t_{w(\overline{MRL})} = t_c - 30$
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	50		nsec		[9] $t_{w(\overline{MRH})} = t_{w(\Phi H)} + t_r - 20$
	t_{dcm}	Data Stable Prior to WR (Memory Cycle)	[5]		nsec		
	t_{dcl}	Data Stable Prior to WR (I/O Cycle)	[6]		nsec		
	t_{cdf}	Data Stable From WR	[7]		nsec		
	t_H	Any Hold Time for Setup Time		0	nsec		
MREQ	$t_{DL\overline{\Phi}(MR)}$	MREQ Delay From Falling Edge of Clock, MREQ Low		85	nsec	C_L = 50pF	
	$t_{DH\Phi(MR)}$	MREQ Delay From Rising Edge of Clock, MREQ High		85	nsec		
	$t_{DH\overline{\Phi}(MR)}$	MREQ Delay From Falling Edge of Clock, MREQ High		85	nsec		
	$t_{w(\overline{MRL})}$	Pulse Width, MREQ Low	[8]		nsec		
	$t_{w(\overline{MRH})}$	Pulse Width, MREQ High	[9]		nsec		
IORQ	$t_{DL\Phi(IR)}$	IORQ Delay From Rising Edge of Clock, IORQ Low		75	nsec	C_L = 50pF	
	$t_{DL\overline{\Phi}(IR)}$	IORQ Delay From Falling Edge of Clock, IORQ Low		85	nsec		
	$t_{DH\Phi(IR)}$	IORQ Delay From Rising Edge of Clock, IORQ High		85	nsec		
	$t_{DH\overline{\Phi}(IR)}$	IORQ Delay From Falling Edge of Clock, IORQ High		85	nsec		
RD	$t_{DL\Phi(RD)}$	RD Delay From Rising Edge of Clock, RD Low		85	nsec	C_L = 50pF	
	$t_{DL\overline{\Phi}(RD)}$	RD Delay From Falling Edge of Clock, RD Low		95	nsec		
	$t_{DH\Phi(RD)}$	RD Delay From Rising Edge of Clock, RD High		85	nsec		
	$t_{DH\overline{\Phi}(RD)}$	RD Delay From Falling Edge of Clock, RD High		85	nsec		
WR	$t_{DL\Phi(WR)}$	WR Delay From Rising Edge of Clock, WR Low		65	nsec	C_L = 50pF	
	$t_{DL\overline{\Phi}(WR)}$	WR Delay From Falling Edge of Clock, WR Low		80	nsec		
	$t_{DH\overline{\Phi}(WR)}$	WR Delay From Falling Edge of Clock, WR High		80	nsec		
	$t_{w(\overline{WRL})}$	Pulse Width, WR Low	[10]		nsec		[10] $t_{w(\overline{WRL})} = t_c - 30$
M1	$t_{DL(M1)}$	M1 Delay From Rising Edge of Clock, M1 Low		100	nsec	C_L = 50pF	
	$t_{DH(M1)}$	M1 Delay From Rising Edge of Clock, M1 High		100	nsec		
RFSH	$t_{DL(RF)}$	RFSH Delay From Rising Edge of Clock, RFSH Low		130	nsec	C_L = 50pF	
	$t_{DH(RF)}$	RFSH Delay From Rising Edge of Clock, RFSH High		120	nsec		
WAIT	$t_{s(WT)}$	WAIT Setup Time to Falling Edge of Clock	70		nsec		
HALT	$t_{D(HT)}$	HALT Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF	
INT	$t_{s(IT)}$	INT Setup Time to Rising Edge of Clock	80		nsec		
NMI	$t_{w(\overline{NML})}$	Pulse Width, NMI Low	80		nsec		
BUSRQ	$t_{s(BQ)}$	BUSRQ Setup Time to Rising Edge of Clock	50		nsec		
BUSAK	$t_{DL(BA)}$	BUSAK Delay From Rising Edge of Clock, BUSAK Low		100	nsec	C_L = 50pF	
	$t_{DH(BA)}$	BUSAK Delay From Falling Edge of Clock, BUSAK High		100	nsec		
RESET	$t_{s(RS)}$	RESET Setup Time to Rising Edge of Clock	60		nsec		
	$t_{F(C)}$	Delay to Float (MREQ, IORQ, RD and WR)		80	nsec		
	t_{mr}	M1 Stable Prior to IORQ (Interrupt Ack.)	[11]		nsec		[11] $t_{mr} = 2t_c + t_{w(\Phi H)} + t_f - 65$

NOTES:

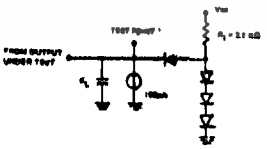
A. Data should be enabled onto the CPU data bus when RD is active. During interrupt acknowledge data should be enabled when M1 and IORQ are both active.

B. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.

C. The RESET signal must be active for a maximum of 3 clock cycles.

D. Output Delay vs. Loaded Capacitance
TA = 70°C Vcc = +5V ±5%
Add 10nsec delay for each 50pf increase in load up to maximum of 200pf for data bus and 100pf for address & control lines.

E. Although static by design, testing guarantees $t_{w(\Phi H)}$ of 200 μsec maximum



Load circuit for Output

Absolute Maximum Ratings

Temperature Under Bias	Specified operating range.
Storage Temperature	-65°C to +150°C
Voltage On Any Pin with Respect to Ground	-0.3V to +7V
Power Dissipation	1.5W*

*Comment

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note: For Z80-CPU all AC and DC characteristics remain the same for the military grade parts except I_{CC}.

I_{CC} = 200 mA

Z80-CPU D.C. Characteristics

T_A = 0°C to 70°C, V_{CC} = 5V ± 5% unless otherwise specified

Symbol	Parameter	Min	Typ.	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC} -.6		V_{CC} +.3	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250μA
I_{CC}	Power Supply Current			150	mA	
I_{LI}	Input Leakage Current			10	μA	V_{IN}=0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT}=2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \le V_{IN} \le V_{CC}$

Capacitance

T_A = 25°C, f = 1 MHz,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_ϕ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80-CPU Ordering Information

C = Ceramic
P = Plastic
S = Standard 5V ±5% 0° to 70°C
E = Extended 5V ±5% -40° to 85°C
M = Military 5V ±10% -55° to 125°C

Z80A-CPU D.C. Characteristics

T_A = 0°C to 70°C, V_{CC} = 5V ± 5% unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC} -.6		V_{CC} +.3	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250μA
I_{CC}	Power Supply Current		90	200	mA	
I_{LI}	Input Leakage Current			10	μA	V_{IN}=0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT}=2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \le V_{IN} \le V_{CC}$

Capacitance

T_A = 25°C, f = 1 MHz,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_ϕ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80A-CPU Ordering Information

C = Ceramic
P = Plastic
S = Standard 5V ±5% 0° to 70°C

หนังสืออ้างอิง

ซีเอ็ดยูเคชั่น จำกัด, บริษัท. คู่มือ/เทียบเบอร์ TTL. กรุงเทพฯ. บริษัท ซีเอ็ดยูเคชั่น จำกัด. พ.ศ. 2531.

วัฒนา เอี่ยงกูล, ยืน ภู่วรวรรณ. ไมโครโปรเซสเซอร์และไมโครคอมพิวเตอร์. กรุงเทพฯ. บริษัท ซีเอ็ดยูเคชั่น จำกัด. พ.ศ. 2529.

ซีเอ็ดยูเคชั่น จำกัด, บริษัท. คู่มือ/เทียบเบอร์ทรานซิสเตอร์. กรุงเทพฯ. บริษัท ซีเอ็ดยูเคชั่น จำกัด. พ.ศ. 2528

PHILIPS ECG, INC. ECG SEMICONDUCTORS. PHILIPS ECG, INC. 1987.

ศูนย์ภาษาคอมพิวเตอร์. การใช้งาน Z-80 กรุงเทพฯ.

PESTO PNEUMATIC. PNEUMATICS PRODUCT RANGE.