โครงการปริญญานิพนธ์

PROJECT 2

การควบคุม PLC ด้วยคอมพิวเตอร์

(COMPUTER CONTROL PLC)

คณะจัดทำปริญญานิพนธ์

1. นาย สมมารถ คล่องพานิชภักดิ์ เลขที่ 34161236
2. นาย ภาณุมาศ ทิราชีพ เลขที่ 34161222

อาจารย์ที่ปรึกษา

อาจารย์ วิทยา ทิพยสุวรรณพร

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

ภาคเรียนที่ 2 ปีการศึกษา 2535

ปริญญานิพนธ์ปีการศึกษา 2535

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมพีแอลซีด้วยคอมพิวเตอร์

ผู้จัดทำ

1. นาย สมมารถ คล่องพานิชภักดิ์ 34161236
2. นาย ภานุมาศ ทิราชีพ 34161222

.....อาจารย์ที่ปรึกษา

(อาจารย์ วิทยา ทิพย์สุวรรณพร)

คำนำ

วิทยานิพนธ์นี้ได้เสนอเกี่ยวกับการควบคุม พี.แอล.ซี.ด้วยคอมพิวเตอร์ ซึ่งปกติแล้ว การเชื่อมต่อ พี.แอล.ซี. กับคอมพิวเตอร์เข้าด้วยกันจะสามารถทำได้โดยการผ่าน อาร์.เอส. 232 แต่มันไม่สามารถจะแสดงสภาวะของโหลดให้ปรากฏบนเครื่องคอมพิวเตอร์ในรูปแบบของกราฟพิคได้จึงได้มีการสร้างอินเตอร์เฟสการ์ดขึ้นมา เพื่อเป็นตัวกลางในการติดต่อระหว่าง พี.แอล.ซี.และคอมพิวเตอร์ ซึ่งสามารถแสดงสภาวะการทำงานของโหลดและอุปกรณ์อื่น ๆ ในรูปแบบของกราฟพิคได้ และสามารถสั่งการควบคุมจากคอมพิวเตอร์ได้โดยใช้ภาษา ซี.

ABSTRACT

This thesis is proposed computer control P.L.C. (Programmable Logic Controller). Normally, we will interface P.L.C. and computer together by R.S. 232 but it can not show the status of load to be graphic to appear in the monitor.

The interface card is fabricated to solve this problem which can show the status of load and other sensor to be graphic and it is controlled from computer by C language.

สารบัญ

บทที่ 1

ความมุ่งหมายของโครงงานนิพนธ์

1.1 บทนำ

ในปัจจุบันคอมพิวเตอร์ได้เข้ามามีบทบาทในชีวิตมนุษย์เพิ่มมากขึ้น ซึ่งช่วยในการอำนวยความสะดวกสบายต่างๆ แก่มนุษย์ ทั้งในด้านความรู้ความบันเทิงทั้งในด้านชีวิตประจำวันและในการทำงานต่างๆ ทั้งนี้เพราะคอมพิวเตอร์ทำงานได้อย่างรวดเร็วอีกทั้งยังถูกต้อง และไม่รู้จักเหนื่อยเหนื่อย นอกจากนั้น ในวงการอุตสาหกรรมปัจจุบัน คอมพิวเตอร์ได้เข้ามามีบทบาทเป็นอย่างมาก เครื่องจักรในปัจจุบันส่วนใหญ่จะใช้คอมพิวเตอร์ควบคุมการทำงาน ดังนั้นการศึกษาการทำงานของคอมพิวเตอร์ และศึกษาลักษณะการนำคอมพิวเตอร์มาใช้ในงานต่างๆ จึงมีความสำคัญและจำเป็นมากยิ่งขึ้น

1.2 จุดประสงค์ของโครงงานนิพนธ์

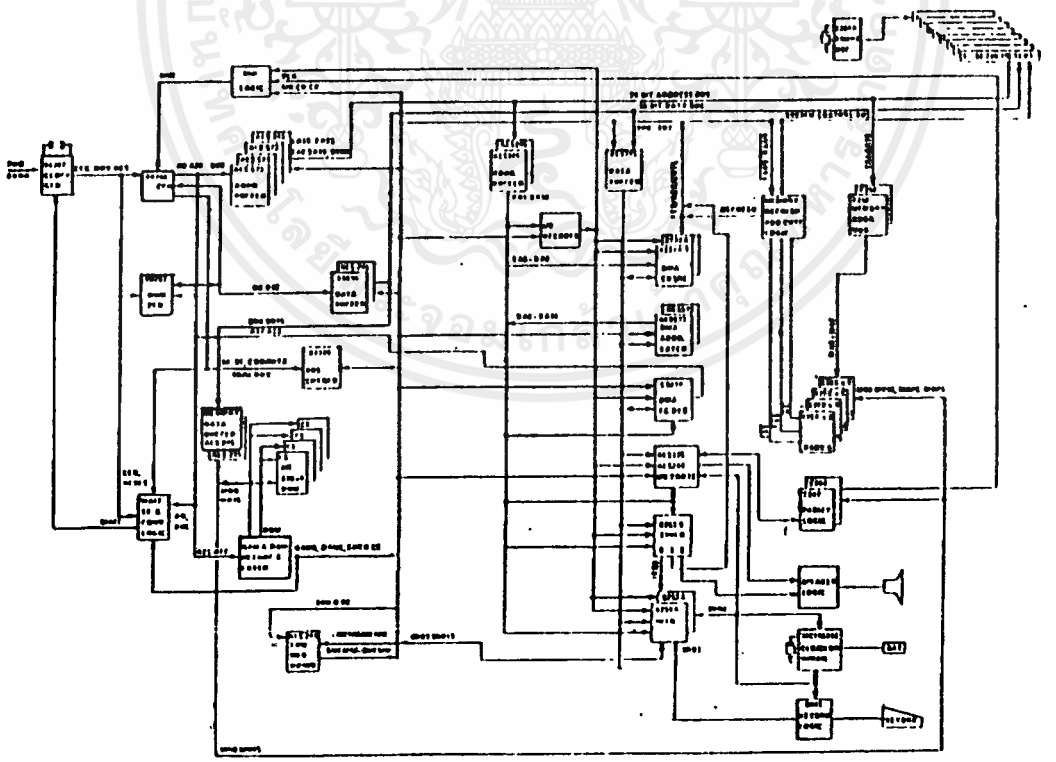
เพื่อศึกษาลักษณะการทำงานของเครื่อง เอบีเอ็ม พีซี ตลอดจนการนำเครื่อง เครื่อง เอบีเอ็ม พีซี มาใช้ในการควบคุมอุปกรณ์ต่าง ๆ โดยการต่อผ่านอุปกรณ์ที่ใช้สำหรับติดต่อกับอุปกรณ์ภายนอก(พอร์ต) รวมถึงการวัดสัญญาณจากหัววัดสัญญาณชนิดต่าง ๆ ทำให้สามารถควบคุมการทำงานของอุปกรณ์ให้เป็นไปอย่างอัตโนมัติตามโปรแกรมได้ และเพื่อให้การศึกษาครั้งนี้เป็นพื้นฐานสำหรับผู้ที่จะศึกษาเกี่ยวกับ การทำงานของเครื่องคอมพิวเตอร์และอุปกรณ์ที่ใช้เป็นตัวกลางในการติดต่อระหว่างคอมพิวเตอร์กับอุปกรณ์ภายนอก ตลอดจนผู้สนใจที่จะดัดแปลงและนำผลงานชิ้นต่าง ๆ นี้เพื่อให้เกิดประโยชน์มากยิ่งขึ้น

บทที่ 2

สถาปัตยกรรมของเครื่อง ไอบีเอ็มพีซี

2.1 เครื่องไอบีเอ็มพีซี

เครื่อง ไอบีเอ็มพีซีเป็นผลของการพัฒนาพีซีของบริษัทไอบีเอ็มในปี พ.ศ. 2528 พัฒนาการของ ไอบีเอ็มพีซีในขณะนั้นยังจัดได้ว่าเป็นการก้าวที่สำคัญในระดับพีซี ไอบีเอ็มพีซีใช้ซีพียู 80286 ทำงานที่ความถี่ของสัญญาณนาฬิกา 6 เมกะเฮิรตซ์ และทำงานร่วมกับโปรเซสเซอร์คณิตศาสตร์ 80287 ไอบีเอ็มพีซี มีโครงสร้างระบบบัสเป็น 16 บิตเต็ม และมีส่วนขยายของระบบทางฮาร์ดแวร์ที่เพิ่มเติมจากไอบีเอ็มพีซีหลายส่วน เพื่อให้เห็นรูปโครงของไอบีเอ็มพีซี จึงขอสรุปโครงสร้างดังแสดงในรูปที่ 2.1



รูปที่ 2.1 แกรมของเมนบอร์ด

ระบบไมโครคอมพิวเตอร์พีซีเอทีนี้ เป็นการปรับปรุงเทคโนโลยีจากเดิม โดยให้ระบบเอ็กซ์ทีเดิมยังคงใช้งานได้เหมือนเดิม ด้วยเหตุนี้ซอฟต์แวร์ทุกระบบที่เคยใช้ได้กับพีซีเอ็กซ์ทีก็ต้องใช้กับพีซีเอทีได้ โครงสร้างของระบบดังรูปที่ 2.2 ซึ่งอยู่บนเมนบอร์ดประกอบด้วย

ไมโครโปรเซสเซอร์เดอร์ 80286 ซึ่งเป็นไมโครโปรเซสเซอร์ที่มีระบบบัสข้อมูล 16 เส้น และแอดเดรส 24 เส้น การอ้างถึงแอดเดรสทำได้ถึง 16 เมกะไบต์ นอกจากนี้ยังทำงานร่วมกับโปรเซสเซอร์คณิตศาสตร์เบอร์ 80287 ได้อีกด้วย

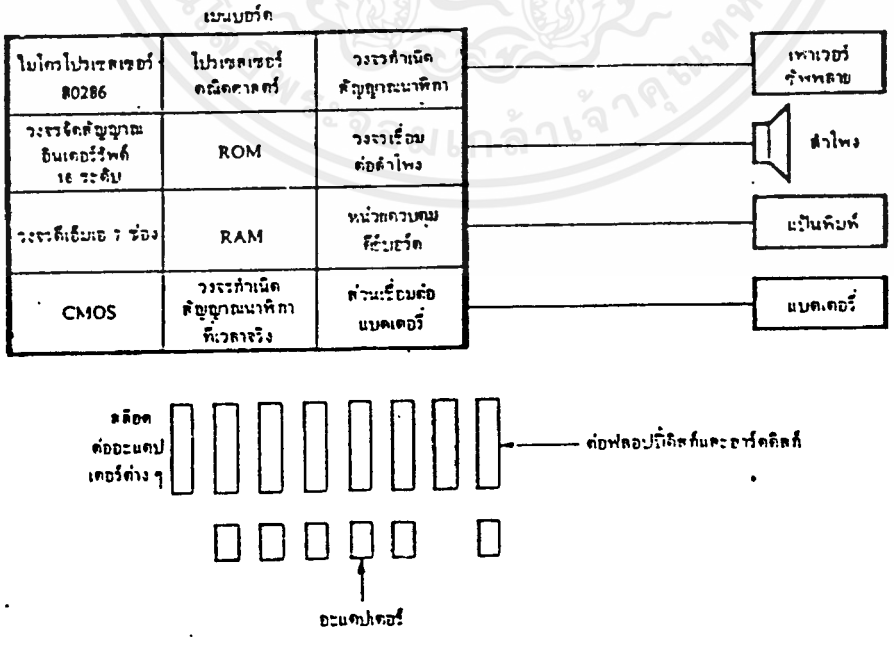
วงจรสนับสนุนระบบ ประกอบด้วย การจัดการดีเอ็มเอ ได้เพิ่มขยายช่องการทำดีเอ็มเอไปเป็น 7 ช่อง เพิ่มการจัดการอินเตอร์รัพต์จากเดิม 8 ระดับไปเป็น 16 ระดับ มีการจัดการระบบสัญญาณนาฬิกาที่โปรแกรมได้

ต่อเชื่อมรอม สำหรับเก็บโปรแกรมแบบถาวร เช่น ไบออสขนาด 64 กิโลไบต์ และเพิ่มขยายต่อเป็น 128 กิโลไบต์

ต่อหน่วยความจำแรม ระบบพีซีเอทีนี้ต่อกับหน่วยความจำแรมบนเมนบอร์ด 512 กิโลไบต์ หรือขยายต่อเป็น 1 เมกะไบต์ หรือมากกว่านั้น

วงจรควบคุมลำโพง สำหรับเปิด/ปิดเสียง ต่อกับลำโพงขนาดเล็ก

ซีมอสแรมเก็บข้อมูลระบบพีซีเอทีใช้สวิตช์สำหรับบอกการติดตั้งระบบพีซีเอที ใช้หน่วยความจำซีมอสขนาด 64 ไบต์ เป็นส่วนเก็บข้อมูลวันเดือนปีสำรองไว้



รูปที่ 2.2 โครงสร้างเมนบอร์ดของพีซีเอที

มีโปรแกรมทำงานตลอดเวลา นาฬิกานี้จะทำงานแม้ปิดเครื่อง เวลาของระบบจึงไม่ต้องเซตใหม่

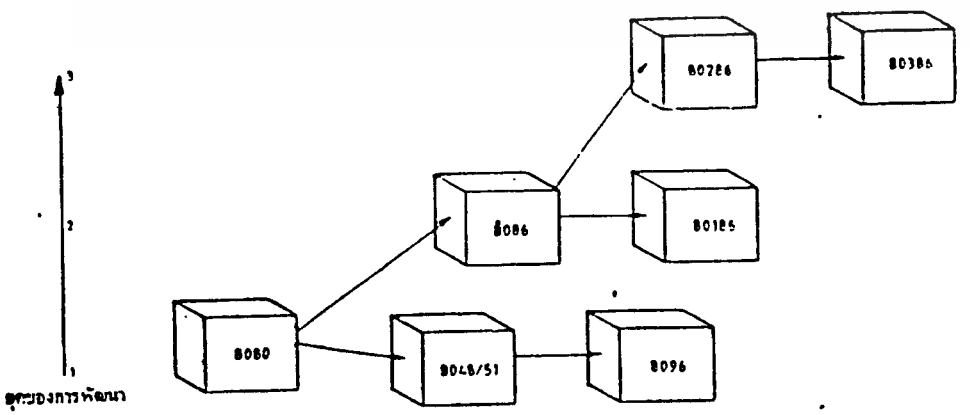
มีวงจรเชื่อมกับแตกเตอร์สำรอง ใช้สำหรับจ่ายไฟกับซีมอสแรม และนาฬิกา

ต่อสล็อตได้ 8 สล็อต ซึ่งเป็นการเพิ่มสล็อตเป็นแบบเอทีเอ็ม โดยเพิ่มซ็อกเกตขนาด 36 ขา ต่อจากของเดิม 62 ขา ในระบบเอ็กซีทีออกมาอีก 6 สล็อต

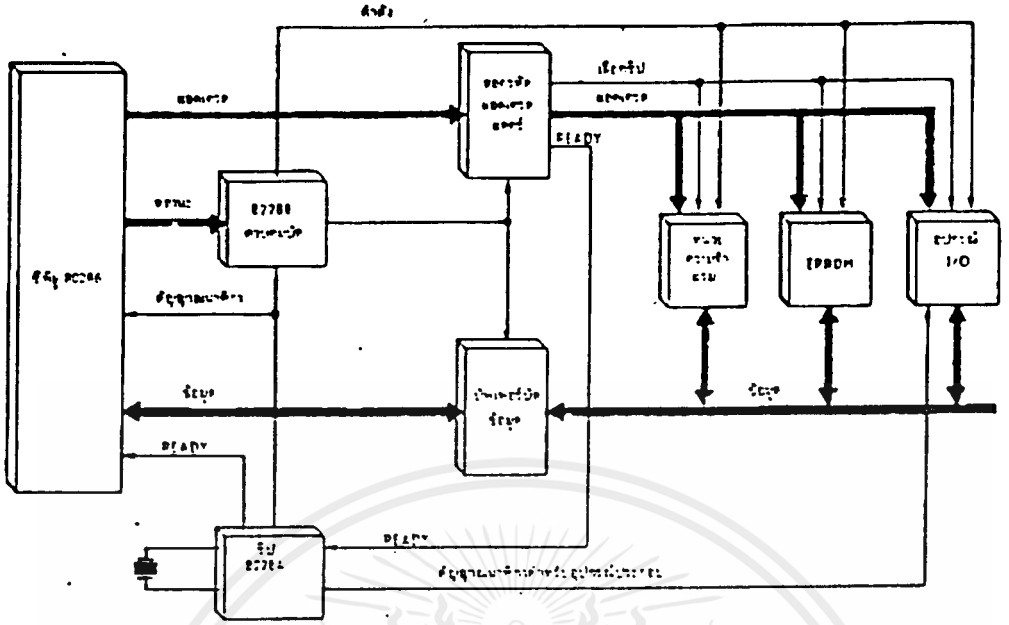
เมนบอร์ดของพีซีเอทีที่กล่าวมาทั้งหมดนี้เมื่อเขียนแสดง โครงสร้างพื้นฐานทั้งหมดของวงจร แสดงดังรูปที่ 2.2

2.2 วงจรซีพียู

ซีพียู 80286 เป็นชิปในยุคที่สามของการพัฒนาไมโครโปรเซสเซอร์ของบริษัทอินเทล การพัฒนาไมโครโปรเซสเซอร์นั้น อินเทลได้ให้โครงสร้างการทำงานที่ทำงานร่วมกันได้ ยึดความสามารถที่เพิ่มขึ้นเรื่อยเป็นซุปเปอร์เซตของเดิม ยุคของไมโครโปรเซสเซอร์ที่อินเทลพัฒนามาเป็นดังรูปที่2.3 ดังนั้นฐานของ 80286 จึงก้าวต่อเข้าสู่ 80386 ได้เป็นอย่างดี เพื่อที่จะเข้าใจระบบเมนบอร์ดของพีซีเอทีได้ดีผู้เขียนขอเข้าในระบบพื้นฐานของตัว 80286 ก่อน เพื่อให้เห็นระบบบัสและการเชื่อมต่อกับอุปกรณ์อื่นๆของ 80286 โดยปกติ 80286 จะทำงานร่วมกับชิปทำเนิดสัญญาณนาฬิกาคือ 82284 และชิปควบคุมบัส 82288 ดังนั้นแก่นของซีพียูจึงประกอบด้วยชิป 3 ตัวคือ 80286 82284 และ 82288 ประกอบรวมกันเข้าเป็นระบบเพื่อสร้างบัสเชื่อมโยงกับอุปกรณ์หน่วยความจำ หรืออุปกรณ์ประกอบจำพวกเพอริเฟอรัล หากเขียนระบบไมโครคอมพิวเตอร์พื้นฐานที่ใช้ซีพียู 80286 เป็นหลักจะได้ดังรูปที่ 2.4 โครงสร้างในรูปนี้จึงเป็นเสมือนโครงสร้างที่ระบบพีซีเอที จะต้องนำไปทำการสร้างระบบบัสเพื่อเชื่อมต่อกับสิ่งต่างๆ



รูปที่ 2.3 พัฒนาการของไมโครโปรเซสเซอร์ในกลุ่ม 80xxx



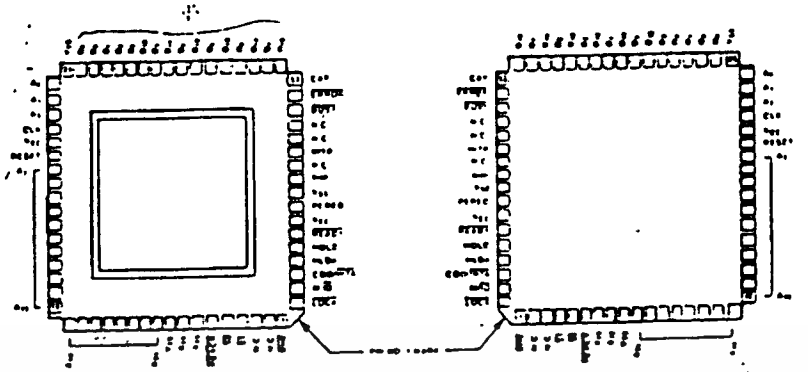
รูปที่ 2.4 ระบบไมโครคอมพิวเตอร์ที่มีซีพียูเบอร์ 80286

จากระบบไมโครคอมพิวเตอร์ทุกระบบที่จัดโครงสร้างเป็นบัส จะต้องประกอบด้วย บัสข้อมูล บัสแอดเดรสและสัญญาณสำหรับการควบคุมระบบต่าง ๆ ในระบบ 80286 ก็เช่นเดียวกันที่ต้องมีส่วนของการควบคุมระบบที่ประกอบด้วยบัสทั้งสามแบบนี้ ไมโครโปรเซสเซอร์ 80286 เป็นไอซีที่ต้องใช้สัญญาณนาฬิกาที่สร้างมาจากชิป 82284 ส่วนชิป 82288 เป็นชิปที่ทำการสร้างสัญญาณควบคุมระบบทั้งหมดก่อนที่จะต่อเข้ากับหน่วยความจำภายนอกแรมและรอม

จากระบบดังกล่าวนี้เราจึงควรมารู้จักกับชิปทั้งสามนี้ก่อน ชิป 80286 เป็นชิปที่มีรูปร่างลักษณะชิปแบบ pin grid array ตัวชิปมีลักษณะเป็นตัวสี่เหลี่ยมจตุรัสมีขาทั้งสิ้น 68 ขา เรียงล้อมรอบทั้งสี่ด้าน ส่วน 82284 เป็นไอซีแบบดีนคะขาบ 18 ขาและ 82288 เป็นแบบ 20 ขา

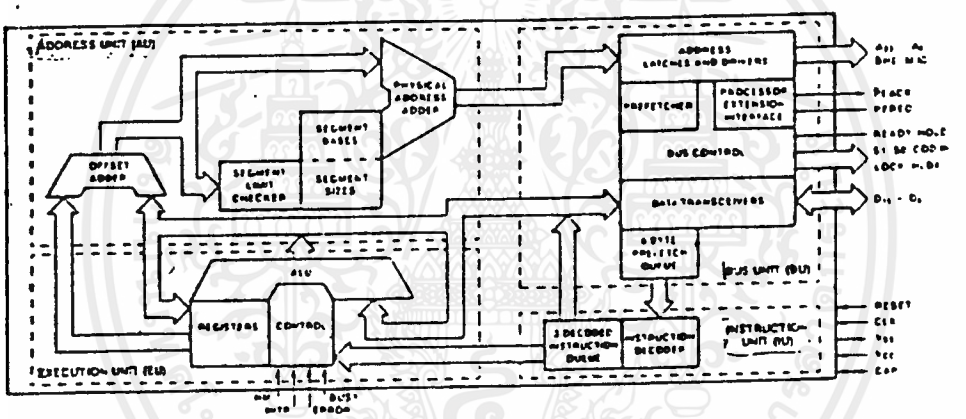
2.3 ไอซี 80286

ไอซีไมโครโปรเซสเซอร์ 80286 มีขีดความสามารถสูงกว่า 8088 ที่ใช้ในพีซีธรรมดามาก อินเทลเปรียบเทียบไว้ในคู่มือว่ามีขีดความสามารถสูงกว่า 8086 มากกว่า 6 เท่า 80286 มี โครงสร้างภายในที่พัฒนามาให้มีสถาปัตยกรรมใช้กับตระกูล 8086 ได้ การจัดวางขา ของไอซี 80286 แสดงดังรูปที่ 2.5



ก. มองทางด้านตัวอุปกรณ์ ข. มองทางด้าน PC บอร์ด

รูปที่ 2.5 การจัดเรียงขาของ 80286



รูปที่ 2.6 บล็อกไดอะแกรมภายในของ 80286

สำหรับบล็อกไดอะแกรมภายในของ 80286 แสดงดังรูปที่ 2.6 ภายในแบ่งออกเป็นหน่วยจัดการแอดเดรส (AU-address unit) หน่วยการเอ็กซีคิวต์ (EU-execution unit) หน่วยควบคุมบัส (BU-bus unit) และหน่วยจัดการวงจรด้วยการถอดรหัสคำสั่ง (IU-instruction unit)

การทำงานของ 80286 ใช้สัญญาณที่ประกฏที่ขาต่าง ๆ ในขั้นต้นนี้ลองมาทำความเข้าใจกับตัว 80286 ในฟังก์ชันการทำงานของแต่ละขาดูดังนี้

CLK เป็นอินพุต รับสัญญาณนาฬิกาจากภายนอกเข้ามา สัญญาณนี้จะได้รับการหารด้วย 2 เพื่อกำหนดเป็นสัญญาณนาฬิกาของระบบ การหารสองจะกระทำภายในชิป และจะทำ

การซิงโครนัสกับการเริ่มต้นทำงานของสัญญาณนาฬิกา โดยเริ่มจาก "0" ไป "1" ของสัญญาณนาฬิกา

D_{15}-D_0 เป็นบัสสองทิศทาง ที่จะทำหน้าที่เป็นอินพุต คือเมื่อสัญญาณของการอ่านจากอินพุตเอาต์พุต จากหน่วยความจำหรืออุปกรณ์อินเตอร์รัพต์ ส่วนจะเป็นเอาต์พุตขณะทำการเขียนข้อมูลให้กับอุปกรณ์เอาต์พุตหรือหน่วยความจำ

A_{23}-A_0 เป็นบัสแอดเดรส ทำหน้าที่เป็นเอาต์พุตแอดเดรสให้กับบัสแอดเดรส

BHE ที่ถูกสัญญาณ bus high enable เป็นสัญญาณเอาต์พุตบอกสถานะการติดต่อระหว่างซีพียูกับอุปกรณ์ภายนอกฝั่งทางบัสข้อมูลจำนวนแปดบิต $(D_{15}$-$D_8)$ การทำงานจะร่วมกับ A_0 ดังตารางที่ 2.1

ตารางที่ 2.1

BHE	A_0	ฟังก์ชัน
0	0	การติดต่อเป็นเวิร์ด
0	1	ติดต่อผ่านทาง 8 บิต D_{15}-D_8
1	0	ติดต่อผ่านทาง 8 บิต D_7-D_0
1	1	ไม่ได้ใช้

S_1, S_0 เป็นสัญญาณเอาต์พุต ที่บอกสถานะของบัส ซึ่งต้องถอดรหัสร่วมกับสัญญาณเลือก ระหว่างหน่วยความจำและอินพุตเอาต์พุต และสัญญาณ COD/INTA สัญญาณ S_1 S_0 เป็นลอจิกสามสถานะ และจะเป็นอิมพีแดนซ์สูง เมื่อมีการขอใช้บัส (bus acknowledge) เมื่อถอดรหัสจะได้ดังตารางที่ 2.2

M/IO เป็นสัญญาณเอาต์พุตบอกสถานะ การติดต่อระหว่าง ซีพียูกับอินพุตเอาต์พุต (I/O) หรือหน่วยความจำ

COD/INTA เป็นสัญญาณเอาต์พุต บอกการเฟตช์ หรือการตอบสนองต่ออินเตอร์รัพต์

LOCK เป็นสัญญาเอาต์พุต เพื่อบอกสถานะการล็อกของบัสเอาไว้ใช้งาน

READY เป็นอินพุต เมื่อหยุดทำงานหรือยืดเวลาเพื่อคอยการอ่านบัสออกไปอีก เพื่อการซิงโครนัสกับอุปกรณ์ที่ทำงานได้ช้ากว่าซีพียู

HOLD เป็นอินพุต เป็นสัญญาณการขอใช้บัสจากภายนอก

ตารางที่ 2.2

COD/ $\overline{INTA}$	M/ $\overline{S_0}$	$\overline{S_1}$	$\overline{S_0}$	ไซเคิลของบัสเป็น
0	0	0	0	ตอบสนองอินเตอร์รัพต์ (INTA)
0	0	0	1	สงวนไว้
0	0	1	0	สงวนไว้
0	0	1	1	ไม่ใช้
0	1	0	0	ถ้า $A_1 = 1$ จะเป็น Halt ถ้าไม่อยู่ในสถานะ shut down
0	1	0	1	อ่านจากหน่วยความจำ
0	1	1	0	เขียนหน่วยความจำ
0	1	1	1	ไม่ใช้
1	0	0	0	สงวนไว้
1	0	0	1	อ่านจาก I/O
1	0	1	0	เขียน I/O
1	0	1	1	ไม่ใช้
1	1	0	0	สงวนไว้
1	1	0	1	เฟตช์
1	1	1	0	สงวนไว้
1	1	1	1	ไม่ใช้

HOLA เป็นเอาท์พุตเป็นสัญญาณการตอบรับให้ใช้บัสจากภายนอก เช่นในสภาวะDMA

INTR เป็นอินพุต เป็นการขออินเตอร์รัพต์ซีพียู

PEREQ เป็นอินพุต มีชื่อว่า processor extension operand request ทำหน้าที่ในการจัดการหน่วยความจำในการทยายหน่วยความจำออกไป

PEACK เป็นเอาท์พุต เป็นสัญญาณตอบสนองของ PEREQ

BUSY เป็นสัญญาณอินพุต มีชื่อว่า processor extension busy เป็นสัญญาณที่ใช้ร่วมกับการทำงานภายนอก เช่น 80287

ERROR เป็นสัญญาณอินพุต มีชื่อว่า processor extension error เป็นระบบการทำงานจากโปรเซสเซอร์ภายนอก

RESET เป็นสัญญาณอินพุต สำหรับการรีเซตซีพียู

V_{SS} แหล่งจ่ายไฟ 0 โวลต์

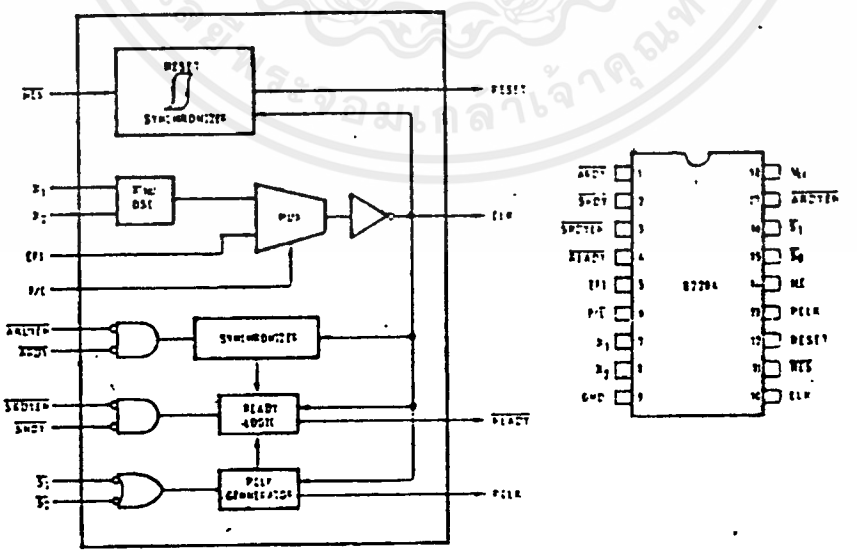
V_{CC} แหล่งจ่ายไฟ 5 โวลต์

CAP ต่อกับตัวเก็บประจุภายนอก ค่าตัวเก็บประจุใช้ขนาด 0.047 ไมโครฟาราด

2.4 ชิป 82284

ชิป 82284 เป็นชิปสร้างสัญญาณนาฬิกาใช้กับ 80286 เช่น ถ้าใช้คริสตอล 16 เมกะเฮิรตซ์ ใส่เข้ากับ 82284 82284 จะสร้างสัญญาณนาฬิกาขนาด 16 เมกะเฮิรตซ์ให้กับ 80286 ส่วนของ 80286 จะมีวงจรแบ่งสองทำการลดความถี่ 16 เมกะเฮิรตซ์ให้เหลือ 8 เมกะเฮิรตซ์ และจัดว่าเป็นสัญญาณนาฬิกาหลักของระบบ เช่นกันถ้าใช้คริสตอล 12 เมกะเฮิรตซ์ จึงใช้กับความเร็วของสัญญาณนาฬิกาในระบบเท่ากับ 6 เมกะเฮิรตซ์ แต่เครื่องที่ทำคล้ายคลึงส่วนใหญ่จะใช้สัญญาณนาฬิกาสูงกว่า ในปัจจุบันรุ่นเอทีจะใช้สัญญาณนาฬิการะบบที่ 10,12 หรือ 16 เมกะเฮิรตซ์ นั่นคือต้องใช้ความถี่คริสตอลสูงเป็นสองเท่า

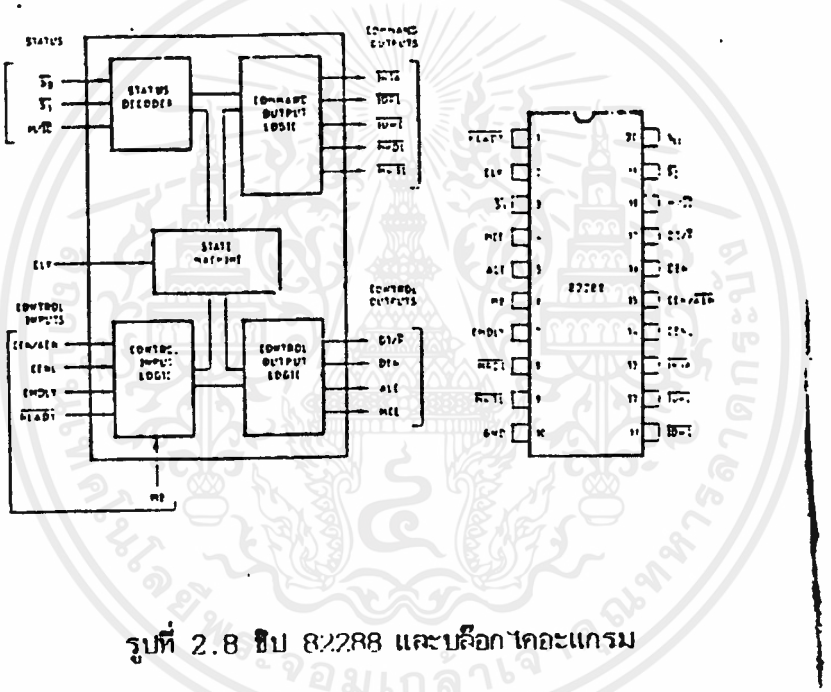
ชิปและบล็อกไดอะแกรมของ 82284 โดยสัญญาณต่าง ๆ นี้อินเทลได้ออกแบบมาให้ใช้กับ 80286 พอดี ดังรูปที่ 2.7



รูปที่ 2.7 การจัดทางขา 82284 และวงจรบล็อกไดอะแกรม

2.5 ชิป 82288 ตัวควบคุมบัส

หากให้สัญญาณนาฬิกาของระบบเป็น 8 เมกะเฮิรตซ์ 80286 สามารถทำงานได้ประมาณ 1.5 ล้านคำสั่งในเวลา 1 วินาที หรือมีความเร็วขนาด 1.5 MIPS การที่ 80286 จะทำงานควบคุมติดต่อกับหน่วยความจำ หรือ อินพุต-เอาต์พุตจำเป็นต้องแยกสัญญาณควบคุมออกมาให้อย่างชัดเจนก่อน อินเทลได้ออกชิป 82288 เป็นชิปสำหรับควบคุมบัส เพื่อแยกสัญญาณควบคุมต่างๆ ดังจะกล่าวต่อไปออกมา ชิป 82288 บล็อกไดอะแกรมแสดงไว้ดังรูปที่ 2.8



รูปที่ 2.8 ชิป 82288 และบล็อกไดอะแกรม

ซีพียูจะควบคุมฮาร์ดแวร์ระบบต่างๆ ได้นั้นจะต้องสร้างไซเคิลของบัสขึ้นมา แต่ละไซเคิลจะแทนการทำงานอย่างใดอย่างหนึ่งขึ้นอยู่กับซีพียูขณะนั้น ว่าได้รับคำสั่งอะไรเข้ามา ซีพียูไซเคิลพื้นฐานประกอบด้วย

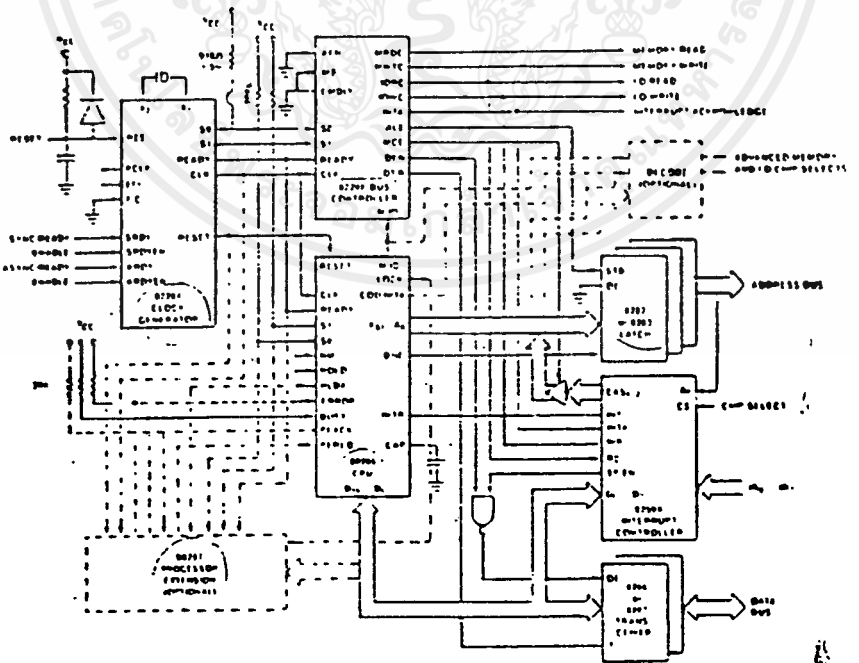
- ไซเคิลการอ่านหน่วยความจำ
- ไซเคิลการเขียนหน่วยความจำ
- การอ่านอินพุต
- การเขียนเอาต์พุต
- การตอบสนองต่ออินเตอร์รัพท์

2.6 การต่อ 80286 82288 และ 82284 แบบพื้นฐาน

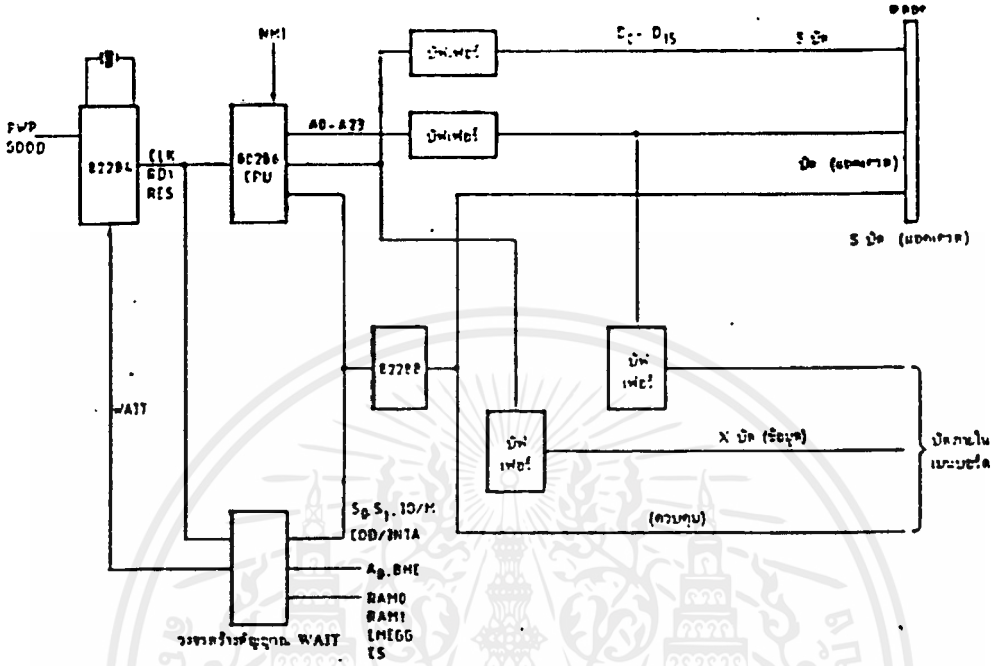
การทำงานเป็นระบบของ 80286 ต้องประกอบชิปทั้งสามชิปนี้ประกอบกันบนซีพีเอชีที มีการต่อชิปทั้งสามนี้เป็นระบบเช่นเดียวกัน 82288 จะอยู่ในตำแหน่งระหว่างซีพียูกับระบบที่จะต่อเชื่อมกับซีพียู 82288 จะทำการแปลงสัญญาณ S_0 S_1 M/IO เพื่อใช้ในการทำงานกับระบบ

สำหรับในซีพีเอชีที การต่อซีพียูกับชิป 82284 ก็มีลักษณะเดียวกัน บนส่วนของ 82284 นี้ จะรีเซตมาจากสัญญาณ POWER GOOD ซึ่งแอกทีฟ "1" นั่นคือ ถ้าหากมีปัญหาทางด้านแหล่งจ่ายไฟเลี้ยงจะทำให้ 82284 หยุดจ่ายสัญญาณนาฬิกาให้กับ 80286 วงจรส่วนของการสร้างบัสเพื่อต่อกับอุปกรณ์ภายนอกของซีพีเอชีทีด้วยกัน 4 แผ่น ดังที่แสดงในตอนท้ายหนังสือนี้ อนึ่งวงจรทั้งหมดของซีพีเอชีทีเฉพาะส่วนเมนบอร์ดมีทั้งหมด 22 แผ่น ซึ่งประกอบต่อกัน ในตอนนี้จะขอเน้นเฉพาะส่วนของซีพียูก่อน

สำหรับวงจรพื้นฐานการต่อระหว่าง 80286 กับ 80288 และ 80284 และโครงสร้างระบบบัสของซีพีเอชีที แสดงดังรูปที่ 2.9 และรูปที่ 2.10



รูปที่ 2.9 วงจรพื้นฐานการต่อระหว่าง 80286 กับ 80288 และ 80284



รูปที่ 2.10 โครงสร้างระบบบัสของพีซีเอที

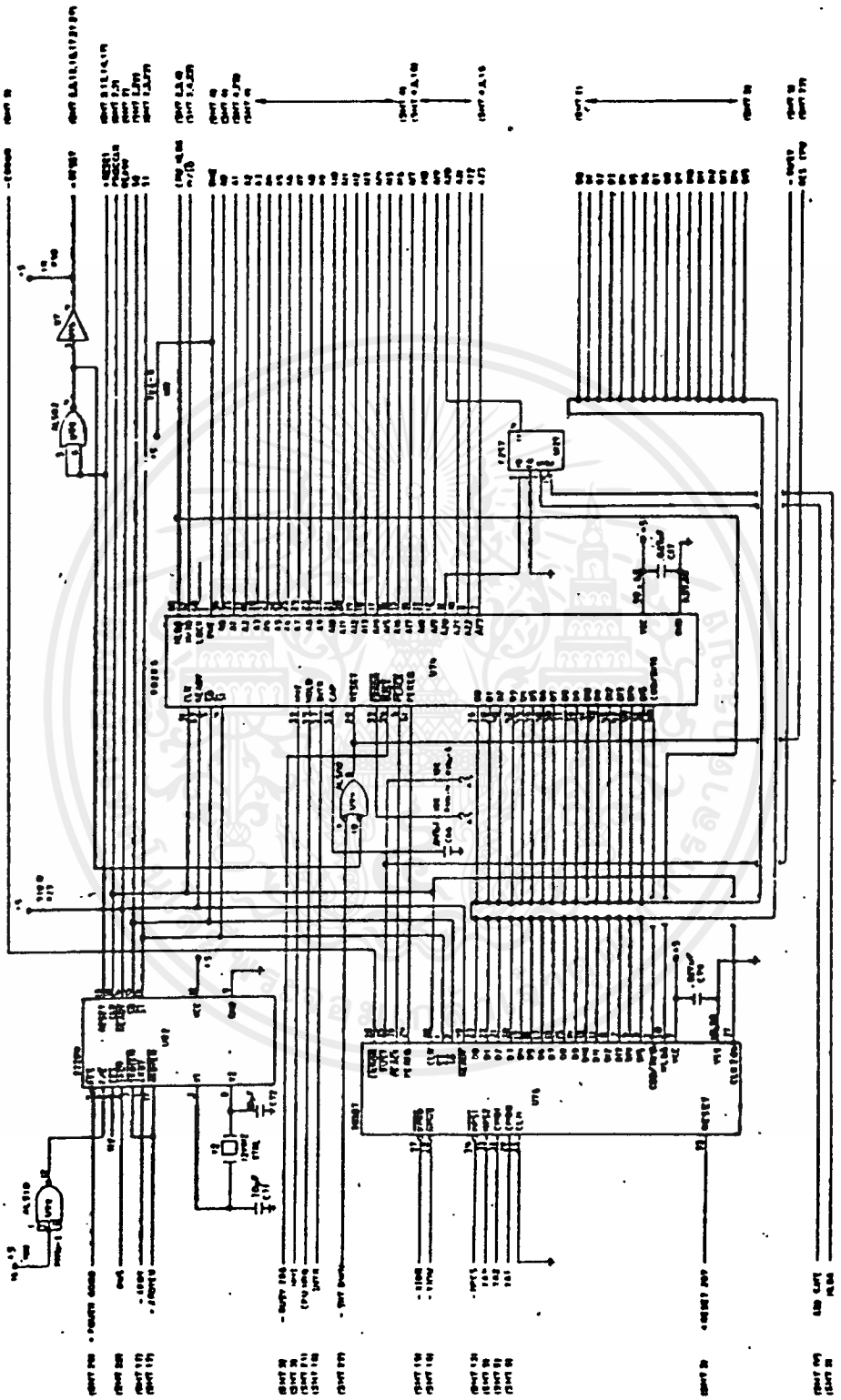
รูปที่ 2.11 วงจรแผ่นที่ 1 เป็นการต่อ 82284 กับ 80286 โดยมี 80287 ต่อร่วมอยู่ด้วยสัญญาณเอาต์พุตของแผ่นนี้ส่วนใหญ่เป็นสัญญาณบัสของระบบ

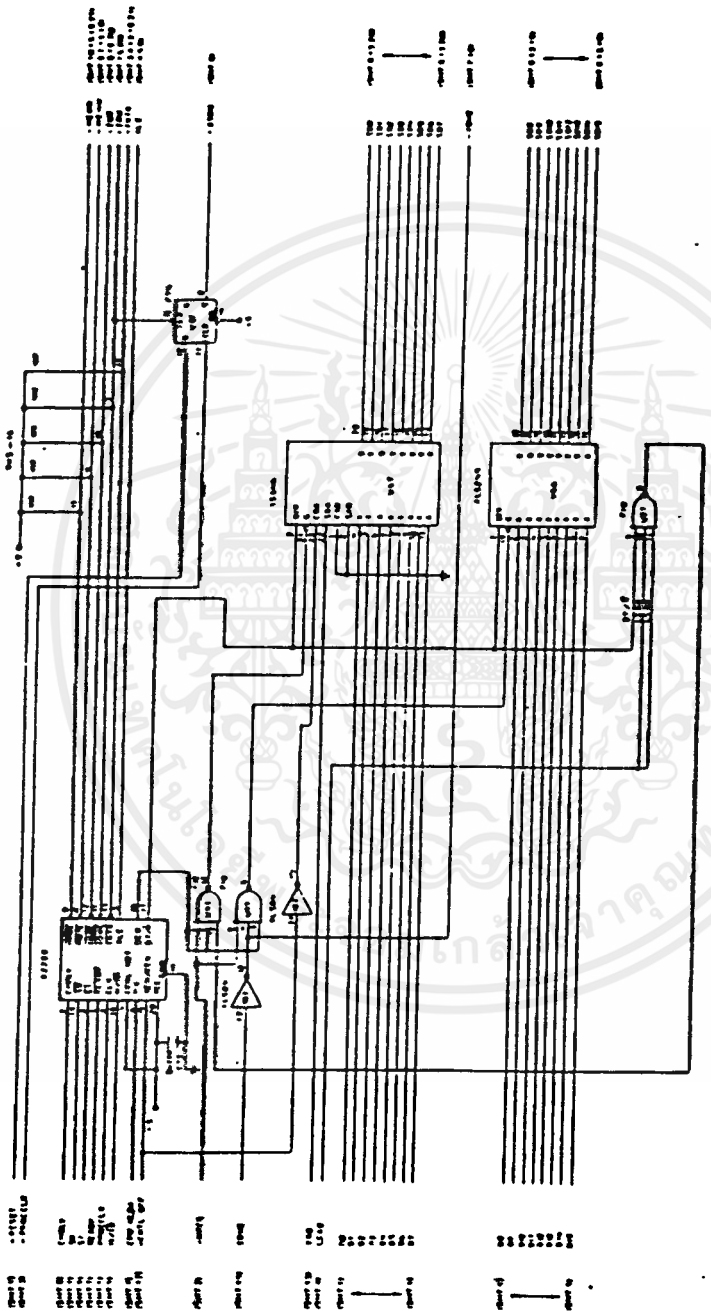
รูปที่ 2.12 วงจรแผ่นที่ 2 เป็นการต่อ 82288 เข้ากับระบบเพื่อถอดรหัสสัญญาณควบคุมระบบทั้งหมดคือสัญญาณการอ่านหน่วยความจำ การเขียนหน่วยความจำ การเขียนอ่านอินพุต/เอาต์พุต การครอบสนองต่อสิทเอดร์รัพท์ และการแลกช์แอดเดรสในแผ่นที่ 2 นี้มีจุดที่น่าสนใจคือการใส่บัฟเฟอร์ให้กับบัสข้อมูล ในที่นี้ใช้ไอซี LS646 เป็นบัฟเฟอร์สำหรับข้อมูลไบต์ต่ำ (D_0-D_7) และ LS245 เป็นบัฟเฟอร์ข้อมูลทางไบต์สูง (D_8-D_{15}) สิ่งที่สนใจคือวงจรถอดรหัสที่ใช้ 74F10 หรือแนนเกต 3 อินพุต เพื่อเลือกการติดต่อข้อมูลเป็นเวิร์ดหรือเป็นไบต์ ทั้งไบต์บนและไบต์ล่างทั้งนี้เพราะซีพียูมีสัญญาณการอินาเบิลไบต์สูง BHE และ A_0 ร่วมกันในการคัดเลือกการติดต่อกับบัสข้อมูล สำหรับสัญญาณบางส่วนที่ปรากฏเป็นสัญญาณที่จะไปปรากฏที่สล็อต

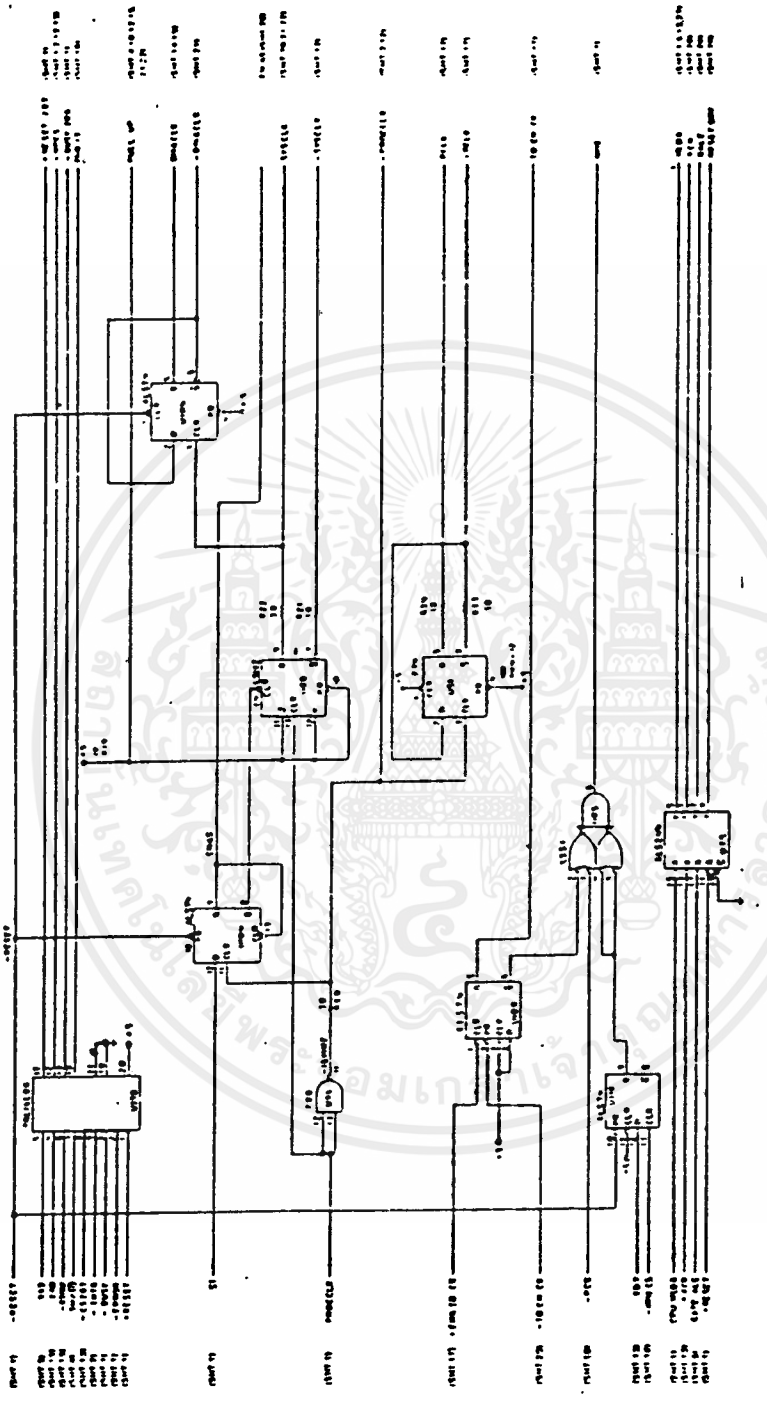
อนึ่ง 82288 มีสัญญาณ DT/R เป็นตัวบอกทิศทางของการรับส่งข้อมูลในบัสข้อมูลซึ่งนำมาใช้กำหนดในตัวบัฟเฟอร์ข้อมูลนี้โดยตรง

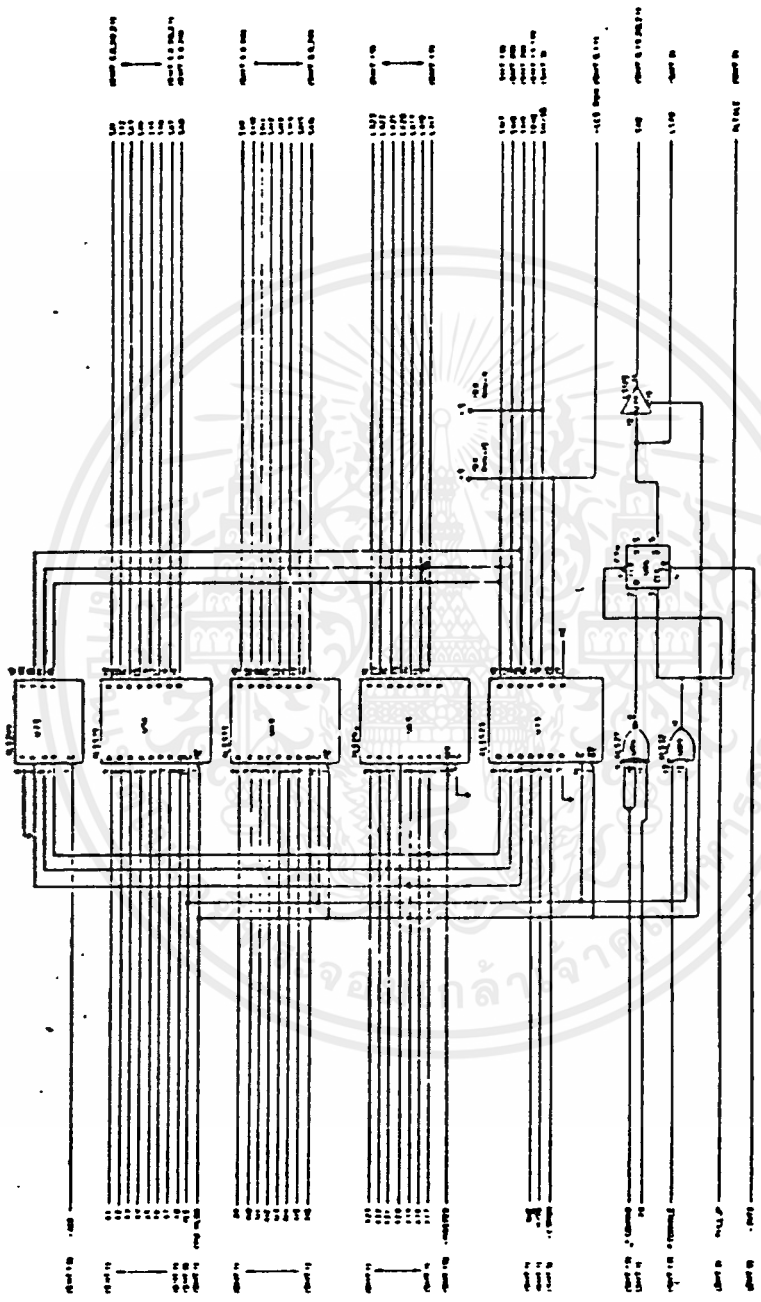
รูปที่ 2.13 วงจรแผ่นที่ 3 เป็นการสร้างสัญญาณควบคุมระบบเฉพาะต่าง ๆ สัญญาณควบคุมส่วนนี้ จะได้นำไปใช้งานต่อไป สิ่งที่น่าสังเกตคือ มีการใช้ PAL (programmble array logic) ทำการถอดรหัสสัญญาณต่าง ๆ เพื่อการสร้างสัญญาณควบคุมระบบขึ้น

ส่วนในรูปที่ 2.14 วงจรแผ่นที่ 4 เป็นวงจรบัฟเฟอร์ของแอดเดรสบัส สำหรับเชื่อมต่อกับสล็อต โครงสร้างของการเชื่อมแอดเดรสเข้ากับสล็อตนี้การจัดการบัสของระบบพีซีเอที่นี้แยกบัสเป็นสองชุด ชุดหนึ่งเมื่อผ่านบัฟเฟอร์ก็จะส่งต่อไปยังช่องต่อสล็อต ส่วนบัสที่ใช้ในเมนบอร์ดเราเรียกว่า X บัส ส่วนของ X บัสนี้จะเป็นบัสที่ผ่านบัฟเฟอร์อีกชุดหนึ่ง สำหรับใช้ภายในเมนบอร์ด









2.7 พอร์ตอินพุตและเอาต์พุตของพีซี เอที

โครงสร้างการวางพอร์ตอินพุตและเอาต์พุต

เพื่อให้โครงสร้างทางฮาร์ดแวร์ของพีซี เอที ใช้ซอฟต์แวร์ร่วมกับพีซี เอ็กซ์ทีได้ จำเป็นต้องให้โครงสร้างอินพุตและเอาต์พุตเหมือนกัน หมายเลขพอร์ตที่ใช้ของพีซี เอทีแสดงดังตารางที่2.3

ตารางที่ 2.3 หมายเลขพอร์ตที่ใช้กับพีซี เอที

หมายเลขพอร์ตรหัสฐานสิบหก	ชื่ออุปกรณ์
000-01F	ดีเอ็มเอคอนโทรลเลอร์หมายเลข 1, 8237A-5
020-03F	อินเตอร์รัพต์คอนโทรลเลอร์หมายเลข 1, 8259A ตัวหลัก
040-05F	ไทเมอร์ 8254-2
060-06F	8042 คีย์บอร์ด
070-07F	นาฬิกา และ NMI และซีมอสแรม
080-09F	DMA เพจรีจิสเตอร์
0A0-0BF	อินเตอร์รัพต์คอนโทรลเลอร์หมายเลข 2, 8259A
0C0-0DF	ดีเอ็มเอคอนโทรลเลอร์หมายเลข 2, 8237A-5
0F0	เคลียร์โปรเซสเซอร์คณิตศาสตร์
0F1	รีเซตโปรเซสเซอร์คณิตศาสตร์
0F8-0FF	โปรเซสเซอร์คณิตศาสตร์
1F0-1F8	ฮาร์ดดิสก์
200-207	เกมไอโอ
278-27F	พอร์ตเครื่องพิมพ์หมายเลข 2
2F8-2FF	พอร์ตอนุกรมหมายเลข 2
300-31F	โปรโตไทป์การ์ด
360-36F	สำรอง
378-37F	พอร์ตเครื่องพิมพ์หมายเลข 1
380-38F	SDLC, ไบซิงก์ 2
3A0-3AF	ไบซิงก์ 1
3B0-3BF	โมโนโครมและเครื่องพิมพ์
3C0-3CF	สำรอง
3D0-3DF	จอภาพสี
3F0-3F7	ควบคุมดิสเกตต์
3F8-3FF	พอร์ตอนุกรมหมายเลข 1

2.8 วงจรไทเมอร์

ไทเมอร์หรือวงจรกำหนดเวลาเหล่านี้ระบบสำหรับไมโครคอมพิวเตอร์เอทีแตกต่างจากพีซีเอ็กซ์ทีบ้างเล็กน้อย ในพีซีเอ็กซ์ทีใช้ชิป 8253 แต่สำหรับพีซีเอทีใช้ชิป 8254-2 วงจรที่แสดงส่วนของการต่อใช้นี้แสดงไว้ดังรูปที่ 2.19 วงจรแผ่นที่ 9 โครงสร้างของชิป 8254 นี้มีวงจรตั้งเวลาอยู่สามช่อง โดยทั้งสามช่องใช้สัญญาณนาฬิกาอินพุตหรือที่เรียกว่า CLK1N ด้วยความถี่ 1.19 MHz แต่ละช่องมีการเชื่อมต่อระหว่างอินพุตกับสัญญาณดังนี้

สัญญาณเชื่อมต่อของช่อง 0

GATE0 เป็นขาที่เชื่อมต่อกับแรงดัน V_{CC}

CLK IN0 เป็นสัญญาณอินพุต 1.19 MHz

CLK OUT0 เป็นสัญญาณที่ส่งเข้าอินเตอร์รัพท์ทางช่อง 0 ของ 8259A

สัญญาณเชื่อมต่อของช่อง 1

GATE1 ต่อเข้ากับ V_{CC}

CLK IN1 เป็นสัญญาณอินพุต 1.19 MHz

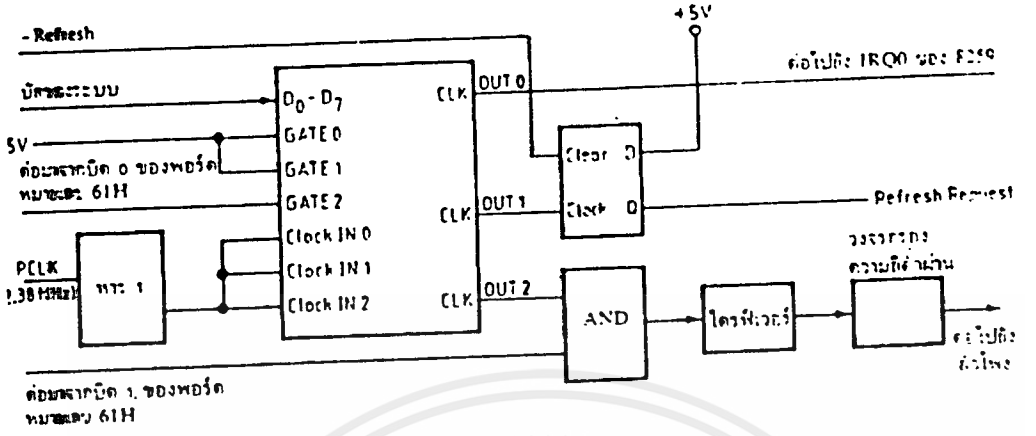
CLK OUT1 เป็นการต่อเข้าที่ใช้ในการกำหนดรอบการรีเฟรชสัญญาณช่อง 1 ที่เอาต์พุตนี้ จะได้รับการโปรแกรม ให้ส่งเมื่อการรีเฟรชด้วยคาบเวลา ประมาณ 15 ไมโครวินาที

สัญญาณเชื่อมต่อของช่อง 2

GATE2 ถูกควบคุมโดยบิต 0 ของพอร์ตหมายเลข 61H ของสัญญาณควบคุมชิปไมโครคอมพิวเตอร์ 8042 เพื่อใช้ในการเปิดปิดสัญญาณเอาต์พุต

CLK IN2 ต่อกับสัญญาณอินพุตขนาด 1.19 MHz

CLK OUT2 ใช้เป็นสัญญาณเอาต์พุตในการขับลำโพง



รูปที่ 2.15 แสดงไดอะแกรมของ 8254

2.8 อินเตอร์รัพท์

ผู้ออกแบบอินเตอร์รัพท์สำหรับเครื่องพีซีเอ็กซ์ที่ได้วางโครงสร้างของการอินเตอร์-รัพท์ไว้ 8 ระดับ โดยใช้ไอซี 8259A โดยมี IRQ2 ว่างไว้ ครั้นมาออกแบบพีซีเอทีจึงจำเป็นต้องใช้อินเตอร์รัพท์เพิ่มเติมมาจากเดิม จึงต้องขยายอินเตอร์รัพท์เพิ่มเติมออกมาอีก โดยจัดวางอินเตอร์รัพท์ตามระดับความสำคัญดังตารางที่ 2.4

ตารางที่ 2.4 การวางอินเตอร์รัพท์ตามระดับความสำคัญ

ระดับความสำคัญ	หน้าที่
นอนมาสเคเบิล NMI	รับสัญญาณจากตรวจสอบพาริตี้ และ ตรวจสอบแชนแนลไอโอ
IRQ0	ไทเมอร์เอาต์พุต 0
IRQ1	คีย์บอร์ด (เมื่อบัฟเฟอร์เต็ม)
IRQ2	อินเตอร์รัพต์มาจาก CTRL 2
IRQ8	สัญญาณกำหนดเวลา
IRQ9	เรียกมาจากซอฟต์แวร์ INT 0AH หรือ IRQ2 เดิม
IRQ10	สงวนไว้
IRQ11	สงวนไว้
IRQ12	สงวนไว้
IRQ13	โปรเซสเซอร์ร่วม (coprocessor)

ตารางที่ 2.4 (ต่อ) การวางอินเตอร์รัพท์ตามระดับความสำคัญ

ระดับความสำคัญ	หน้าที่
IRQ14	จากแผงควบคุมฮาร์ดดิสก์
IRQ15	สงวนไว้
IRQ3	พอร์ตอนุกรม 2
IRQ4	พอร์ตอนุกรม 1
IRQ5	วงจรควบคุมฮาร์ดดิสก์
IRQ6	วงจรควบคุมดิสก์
IRQ7	พอร์ตขนาน 1

สังเกตุว่าการขยายอินเตอร์รัพท์ออกมาอีก 8 ช่องนี้ทำโดยสัญญาณ CTRL2 เป็นตัวส่งให้กับ IRQ2 ของ 8259A ตัวแรก ดังนั้นวงจรที่ใช้จึงใช้ 8259 สองตัว โดยให้เอาท์พุตของตัวหนึ่งเข้าที่ขา IRQ2 ของอีกตัวหนึ่ง วิธีการขยายแบบนี้ทำให้ระดับความสำคัญของ IRQ8-IRQ15 มีความสำคัญอยู่ในระดับ 2 เดิม หรือแทรก 8 ระดับอยู่ระหว่างระดับ 2 และ 3 เดิม

การจัดหมายเลขพอร์ตนั้นเป็นตามข้อกำหนดเดิมคือให้ 8259A ตัวแรก มีหมายเลขพอร์ตเป็น 020-03F และอีกตัวหนึ่งมีหมายเลข 0A0-0BF วงจรของการต่อ 8259 ทั้งสองตัว แสดงดังรูปที่ 2.20 วงจรแผ่นที่ 9

2.9 ดีเอ็มเอ

ดีเอ็มเอที่ใช้กับเครื่องพีซีเอ็กซ์ที ใช้ชิป 8237A เพียงชิปเดียว แต่สำหรับเอทีนี้ได้เพิ่มขยายช่องของดีเอ็มเอ ซึ่งจากเดิมมี 4 ช่อง ให้เป็น 8 ช่อง ดังนั้นจึงต้องใช้ชิป 8237A เพิ่มขึ้นอีกหนึ่งตัว โดยมีการเปลี่ยนแปลงโครงสร้างบ้างเล็กน้อย ช่องของดีเอ็มเอเดิมมี 4 ช่องคือ แชนเนล 0-3 และในเครื่องพีซีเอทีได้เพิ่มขยายช่อง 4-7 เพิ่มเติมจากเดิม จึงมีทั้งหมด 8 ช่อง โดยแต่ละช่องทำหน้าที่ดังตารางที่ 2.5

เมื่อให้แชนเนลดีเอ็มเอเดิม 0-3 ทำงานเหมือนเดิม แต่เมื่อเครื่องพีซีเอทีไม่ต้องการรีเฟรชจากดีเอ็มเอ ดังนั้นแชนเนล 0 จึงว่างไว้ ดีเอ็มเอคอนโทรลเลอร์ตัวที่ 1 สนับสนุนการรับส่งข้อมูลแบบ 8 บิตระหว่างอินพุต เอาท์พุต กับหน่วยความจำของระบบ ในขณะที่ตัวให้รับส่งกับหน่วยความจำและรับส่งแบบ 16 บิตได้ ในแต่ละแชนเนลถูกกำหนดด้วยโปรแกรมที่ระบบจะส่งไป การโปรแกรมคือโปรแกรมให้เกิดการส่งถ่ายข้อมูลที่ละบล็อกด้วยขนาดบล็อกที่ใหญ่ที่สุด

64 กิโลไบต์ โดยติดต่อกับหน่วยความจำได้ตลอดสายแอดเดรส 24 เส้น หรือ 16 MB

ตารางที่ 2.5 รายละเอียดหน้าที่ช่องของดีเอ็มเอ

8237 ตัวแรก	หน้าที่
แชนเนล 0	สงวนไว้ (แต่เดิมของเครื่องพีซีเอ็กซ์ทีใช้สำหรับรีเฟรช)
แชนเนล 1	ใช้สำหรับการส่งผ่านกับ SDLC
แชนเนล 2	ใช้สำหรับการติดต่อรับ ส่งข้อมูลกับดิสเกตต์
แชนเนล 3	สงวนไว้
8237 ตัวที่สอง	
แชนเนล 4	ต่อร่วมสำหรับชิป 8237 ตัวแรก
แชนเนล 5	สงวนไว้
แชนเนล 6	สงวนไว้
แชนเนล 7	สงวนไว้

สำหรับดีเอ็มเอคอนโทรลเลอร์ตัวที่สอง มีช่องการส่งรับข้อมูลคือ แชนเนล 4 ถึง แชนเนล 7 แชนเนล 4 ใช้สำหรับการต่อร่วม (cascade) กับแชนเนล 0-3 แชนเนล 4-7 นี้ได้รับการออกแบบให้ส่งถ่ายข้อมูลแบบ 16 บิตระหว่างอุปกรณ์อินพุตเอาต์พุตกับหน่วยความจำระบบโดยกำหนดขนาดข้อมูลเป็นบล็อกได้สูงสุดถึง บล็อกละ 128 กิโลไบต์ แต่อย่างไรก็ตาม ช่อง 5,6,7 ที่จะใช้งานนั้นตัววงจรได้ใช้ขา XA1 ต่อจากแอดเดรส ขา A0 ของ 8237 ทำให้การส่งถ่ายข้อมูลจะไม่สามารถส่งถ่ายในลักษณะเริ่มจากไบต์คี่ได้

ตัววงจรของ 8237 แสดงไว้ในรูปที่ 2.18 วงจรแผ่นที่ 7 การกำหนดแอดเดรสอินพุต-เอาต์พุตที่เกี่ยวข้องกับดีเอ็มเอทั้งสองชุดคือ แอดเดรส 000-01F ติดต่อกับดีเอ็มเอ โดยมีช่วงแอดเดรส 080-09F เป็นอินพุตเอาต์พุตแอดเดรสของเพจรีจิสเตอร์ และอินพุตเอาต์พุตของดีเอ็มเอคอนโทรลเลอร์ของตัวที่สองอยู่ที่แอดเดรส 0C0-0DF

การกำหนดหมายเลขแอดเดรสของพอร์ตอินพุตเอาต์พุตทำบนชิป 74ALS138 ซึ่งอยู่ในรูปที่ 12.17 วงจรแผ่นที่ 6 โดยเอาต์พุตกำหนดกลุ่มแอดเดรสของดีเอ็มเอ อินเตอร์รัพท์คอนโทรลเลอร์ไทเมอร์ พอร์ตขนาย สำหรับการกำหนดสถานะ (ในที่นี้ใช้ชิปไมโครคอมพิวเตอร์ชิปเดี่ยว 8042) พอร์ตกำหนดเพจรีจิสเตอร์และโปรเซสเซอร์คณิตศาสตร์

ชิป 74ALS138 ที่ถอดรหัสเป็นสัญญาณ PG REGCS นี้ส่งต่อไปยังชิป LS612 เพื่อกำหนดเพจรีจิสเตอร์ให้ทำงานร่วมกับดีเอ็มเอคอนโทรลเลอร์ ในการกำหนดว่าการดีเอ็มเอจะ

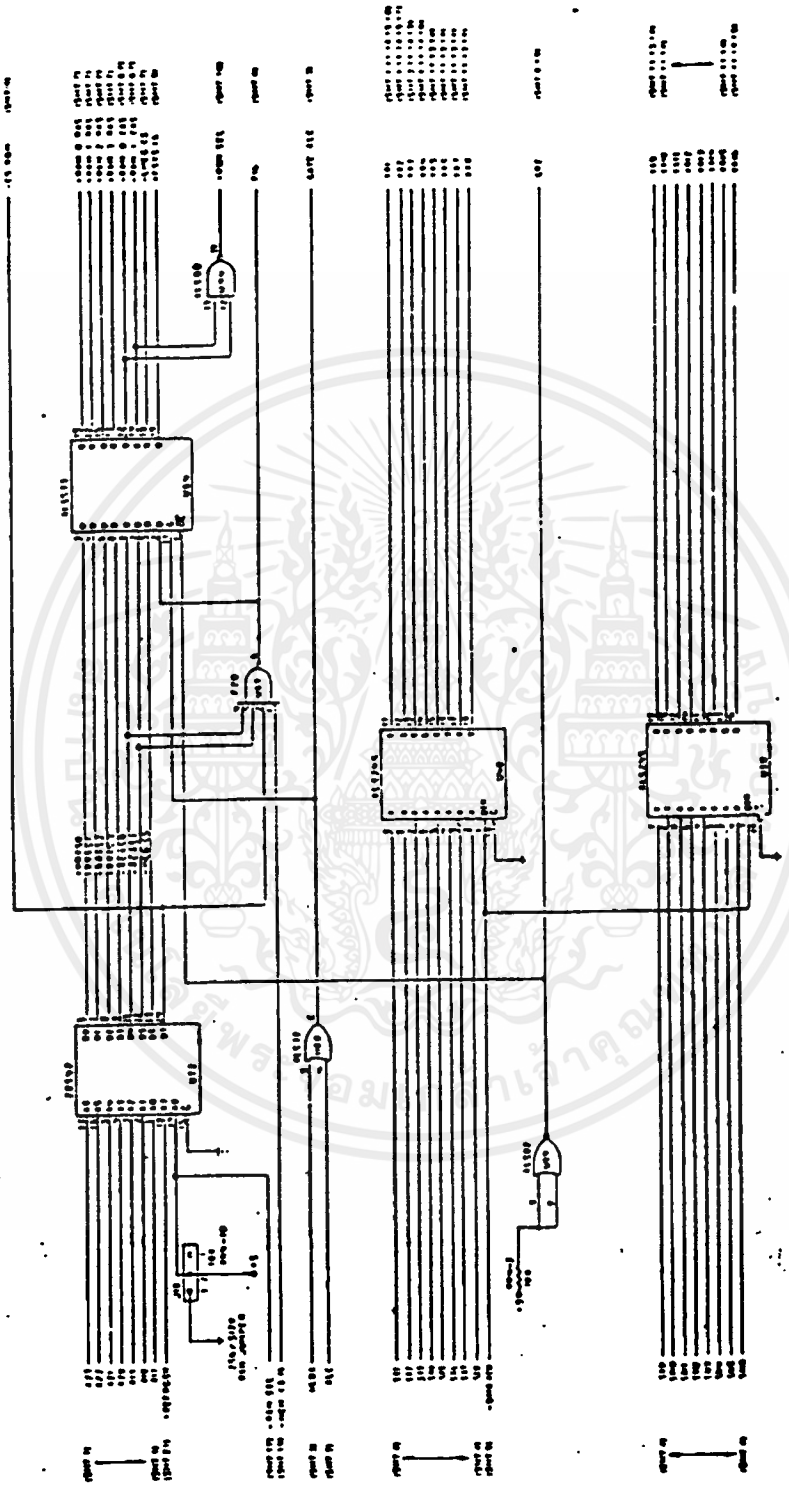
เริ่มที่เพจใด โดยแต่ละเพจจะมีขนาดใหญ่ได้ถึง 64 กิโลไบต์ ดังนั้น LS612 จึงให้แอดเดรสของการดีเอ็มเอ ด้วยแอดเดรสตั้งแต่ A16-A23 นั่นหมายความว่าการกำหนดตำแหน่งหน่วยความจำสำหรับการดีเอ็มเอใด ๆ ต้องกำหนดแอดเดรสไบต์ A16-A23 หรือเพจแอดเดรสนี้ด้วย วงจรการใช้ LS612 แสดงไว้ในรูปที่ 2.19 วงจรแผ่นที่ 8

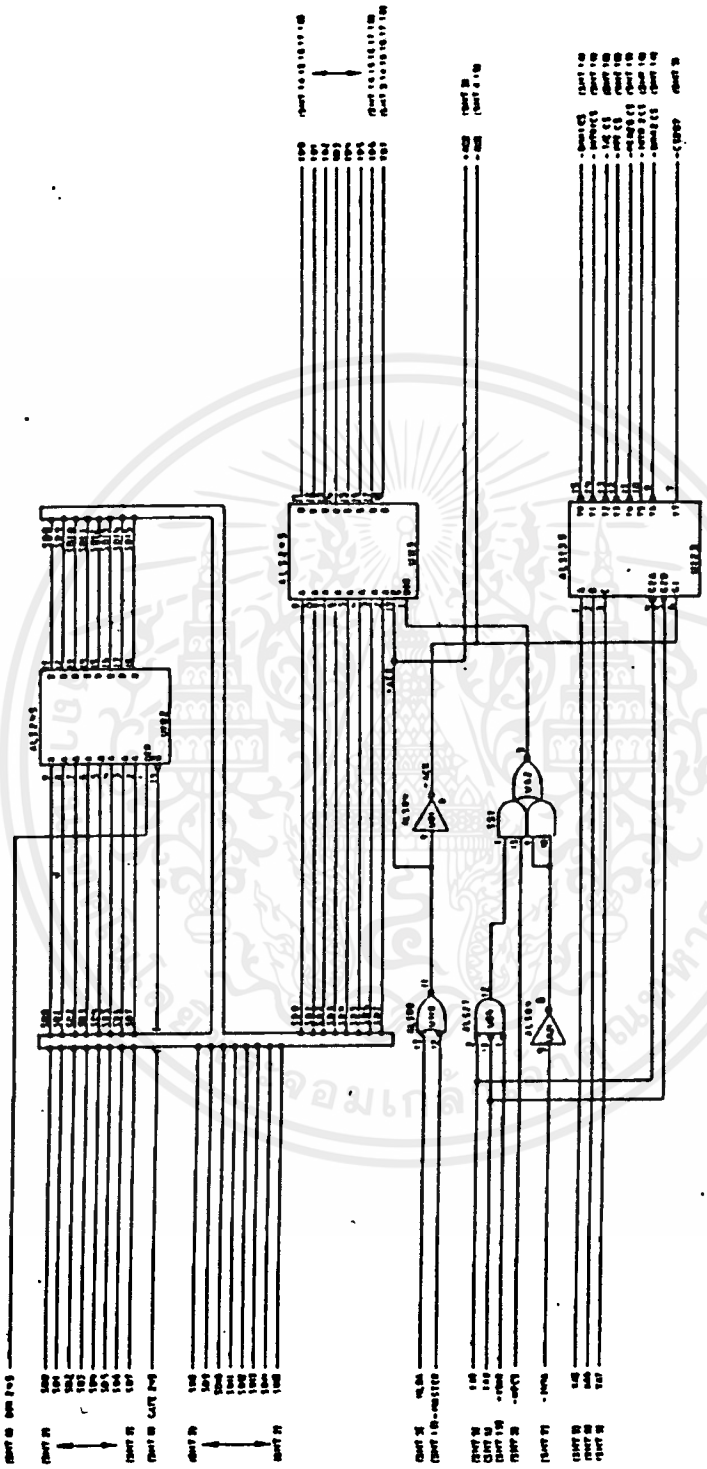
ผู้ใช้สามารถกำหนดค่าแอดเดรส A16-A23 ให้กับชิป LS612 นี้โดยผ่านทางพอร์ตหมายเลข 080-08F โดยแต่ละหมายเลขของพอร์ตมีความหมายดังต่อไปนี้

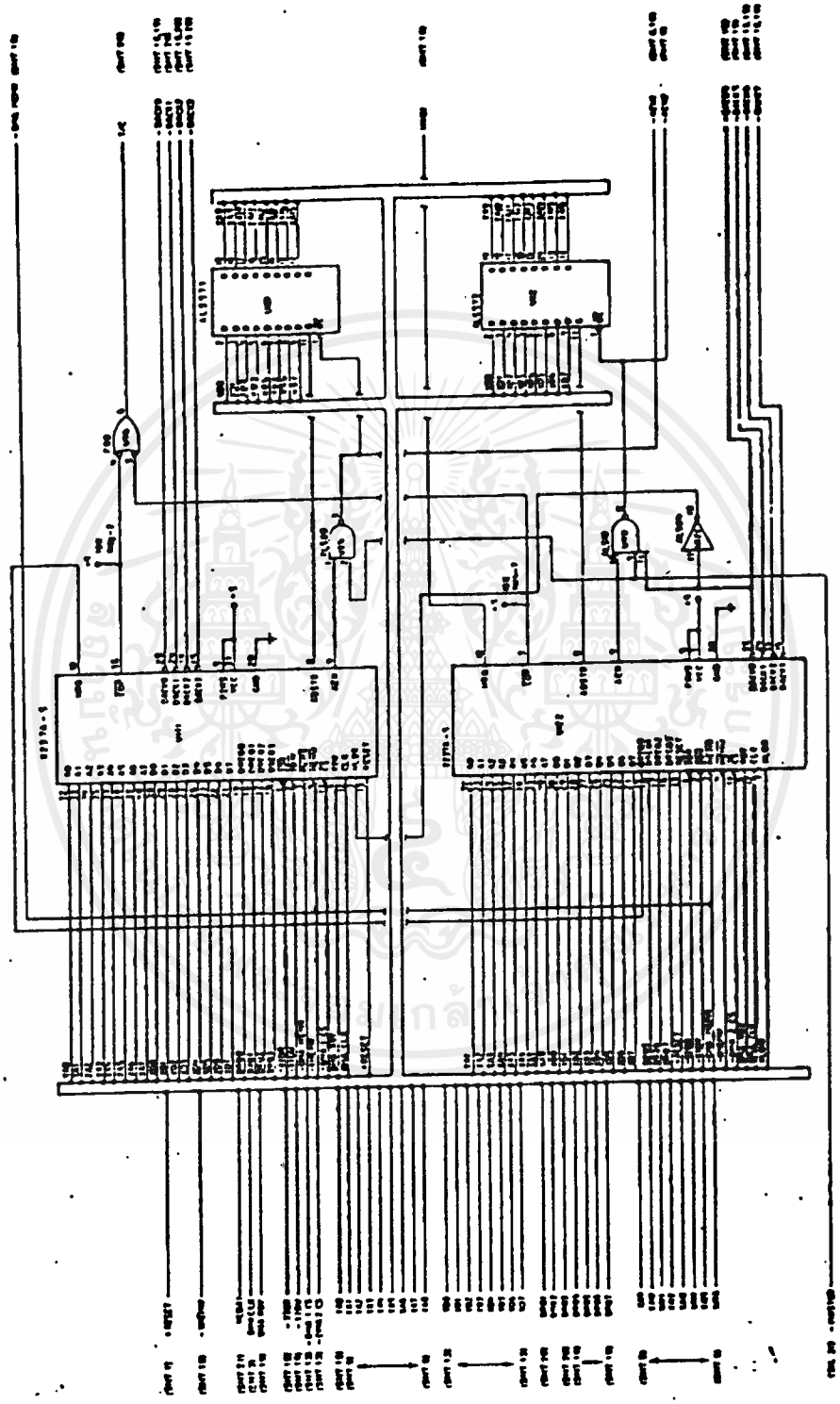
ดีเอ็มเอแชนเนล 0	ใช้พอร์ตหมายเลข 0087
ดีเอ็มเอแชนเนล 1	ใช้พอร์ตหมายเลข 0083
ดีเอ็มเอแชนเนล 2	ใช้พอร์ตหมายเลข 0081
ดีเอ็มเอแชนเนล 3	ใช้พอร์ตหมายเลข 0082
ดีเอ็มเอแชนเนล 5	ใช้พอร์ตหมายเลข 008A
ดีเอ็มเอแชนเนล 6	ใช้พอร์ตหมายเลข 0089
ดีเอ็มเอแชนเนล 7	ใช้พอร์ตหมายเลข 008B
รีเฟรชแอดเดรส	ใช้พอร์ตหมายเลข 008F

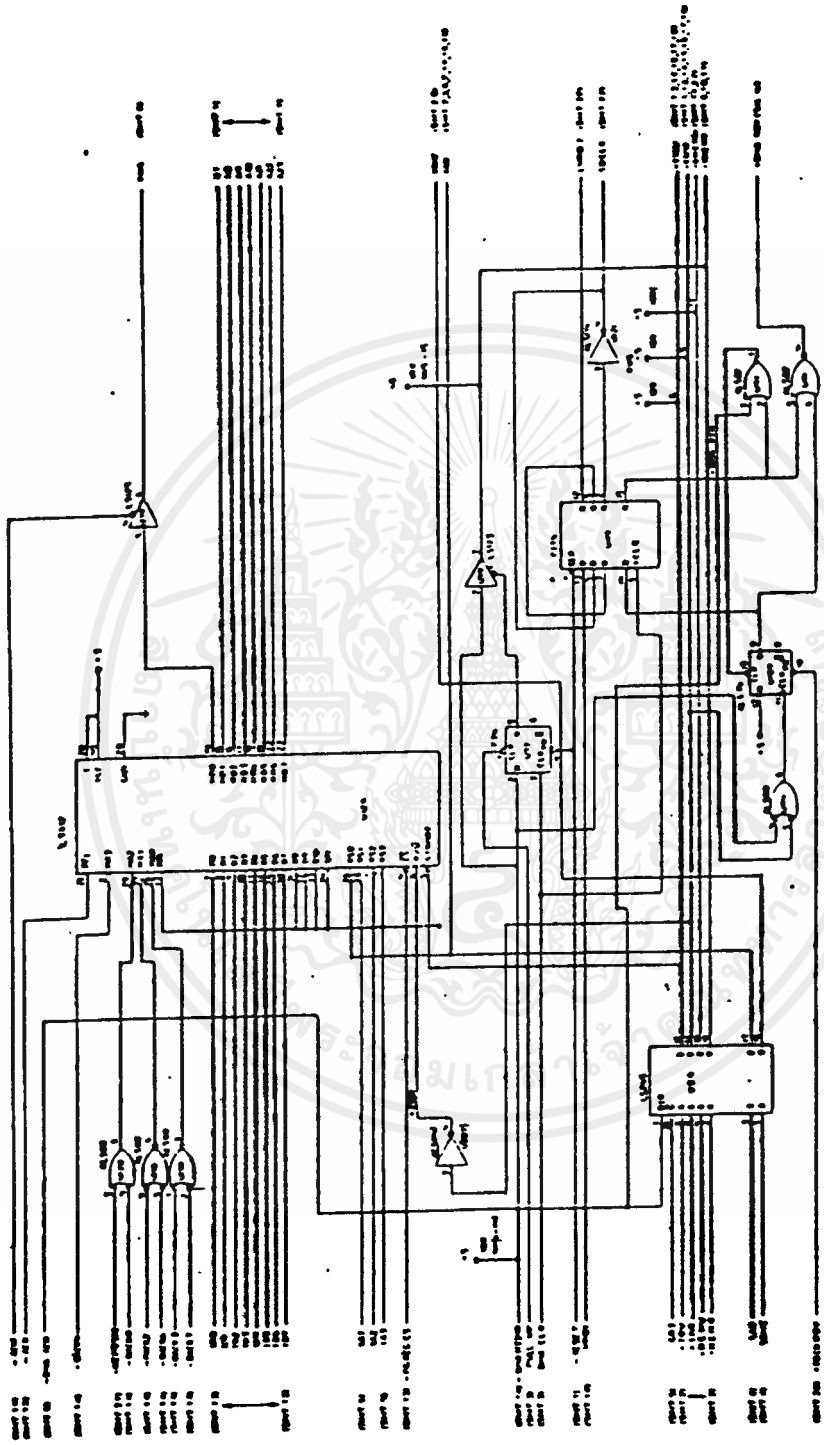
ในการกำหนดแอดเดรสเพื่อทำการดีเอ็มเอ ผู้ใช้ต้องกำหนดแอดเดรสเริ่มต้นให้กับ 8237A และ LS612 ด้วย หากกำหนดให้กับดีเอ็มเอแชนเนล 0-3 การกำหนดจะกระทำโดยบอกแอดเดรสเริ่มต้นที่ 8237A บิต A0-A15 และบอกไปที่ LS612 ให้กับ 8237 และการกำหนดแอดเดรสบิต A17-A23 ให้กับดีเอ็มเอเพจรีจิสเตอร์ หรือ LS612 นั่นเอง สังเกตความแตกต่างกันระหว่างดีเอ็มเอ 8237A ทั้งสองตัว โดยตัวแรกจะทำงานเหมือนกับเครื่องเอ็กซีที แต่ตัวหลังเพิ่มขีดความสามารถให้ทำงานได้เต็ม 16 บิต สำหรับการส่งถ่ายข้อมูลแทนที่จะเป็น 8 บิต ตามระบบบัสข้อมูลของ 8088 ทั้งนี้เพราะบัสข้อมูลของ 80286 มี 16 เส้นแล้ว อย่างไรก็ตามเมื่อทำการดีเอ็มเอแบบ 16 บิตของชิป 8237 ตัวใหม่คือ ช่อง 5-7 จะต้องให้สัญญาณ BHE และ A0 เป็นลอจิก "0"

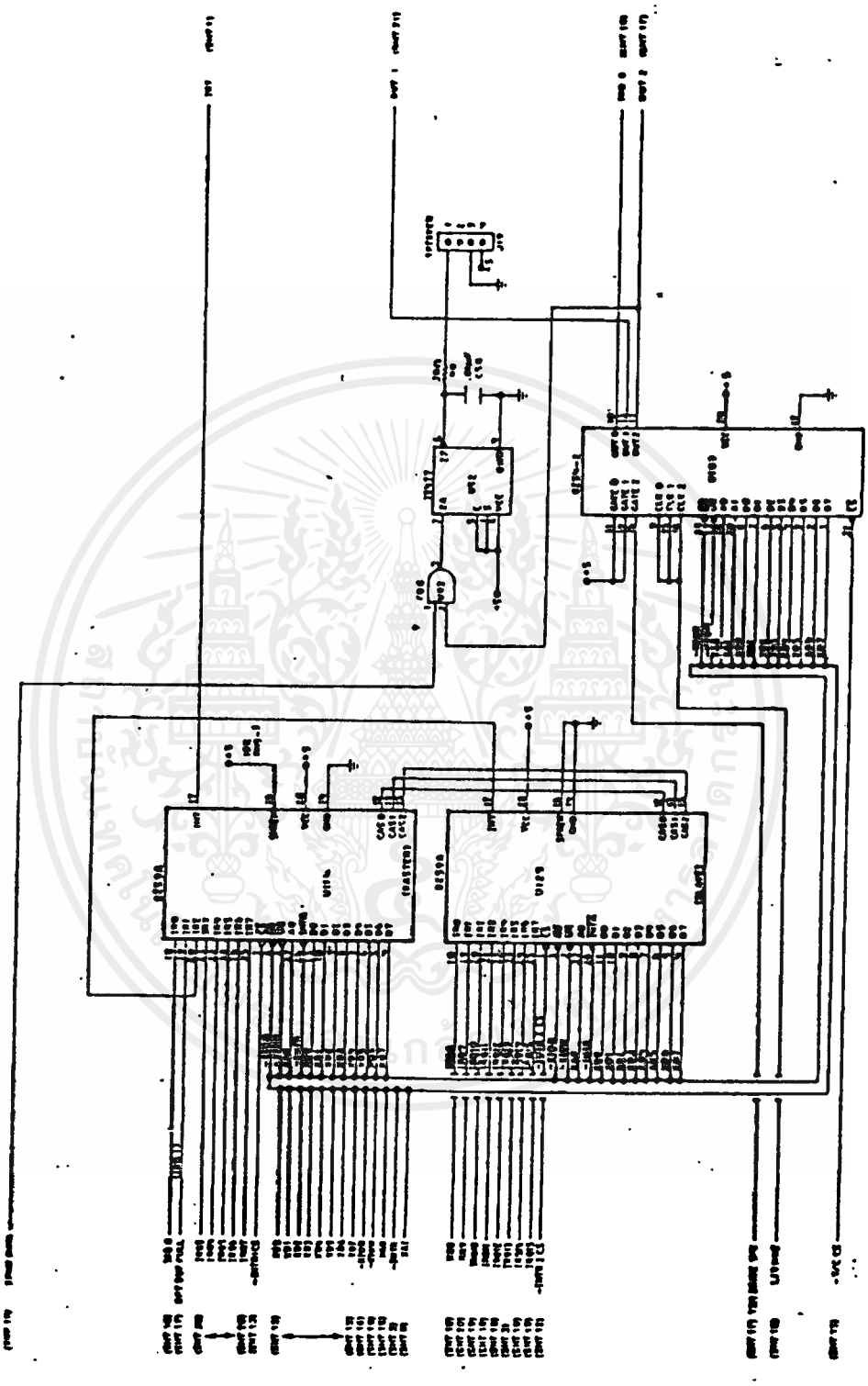
เนื่องจากการทำดีเอ็มเอต้องมีการกำหนดเพจแอดเดรสเสียก่อน ดังนั้นการเคลื่อนย้ายข้อมูลจะทำงานในตัว 8237 จึงทำให้ค่าแอดเดรสที่ตั้งไว้เปลี่ยนค่าเฉพาะในชิป 8237 เท่านั้นด้วยเหตุนี้เอง ทำให้เราจะกำหนดขนาดบล็อกเกินกว่า 64 กิโลไบต์ สำหรับตัวแรกไม่ได้











2.10 การโปรแกรมดีเอ็มเอแบบ 16 โต

จากการที่กล่าวแล้วว่า แชนเนล 5-7 สำหรับเครื่องพีซีเอทีนี้ได้ออกแบบไว้สำหรับการทำดีเอ็มเอแบบ 16 บิต การควบคุมจะทำได้ด้วยการกำหนดค่าลงไปในพอร์ตแอดเดรส 0C0-0DF โดยแต่ละพอร์ตมีความหมายดังนี้

พอร์ต 0C0 เบสและแอดเดรสปัจจุบันของแชนเนล 0

พอร์ต 0C2 เบสแอดเดรสและตัวนับเวิร์ดของแชนเนล 0

พอร์ต 0C4 เบสและแอดเดรสปัจจุบันของแชนเนล 1

พอร์ต 0C6 เบสแอดเดรสและตัวนับเวิร์ดของแชนเนล 1

พอร์ต 0C8 เบสและแอดเดรสปัจจุบันของแชนเนล 2

พอร์ต 0CA เบสแอดเดรสและตัวนับเวิร์ดของแชนเนล 2

พอร์ต 0CC เบสและแอดเดรสปัจจุบันของแชนเนล 3

พอร์ต 0CE เบสแอดเดรสและตัวนับเวิร์ดของแชนเนล 3

พอร์ต 0D0 พอร์ตรีจิสเตอร์สถานี (อ่าน)/รีจิสเตอร์คำสั่ง (เขียน) พอร์ต 0D6 โหมดรีจิสเตอร์ (เขียน)

พอร์ต 0D8 ฟลิปฟลอปสำหรับเคลียร์ไบต์พอยน์เตอร์

พอร์ต 0DA รีจิสเตอร์ชั่วคราว (อ่าน)/มาสเตอร์เคลียร์ (เขียน)

พอร์ต 0DC เคลียร์มาส์กรีจิสเตอร์

พอร์ต 0DE เขียนมาส์กรีจิสเตอร์บิต

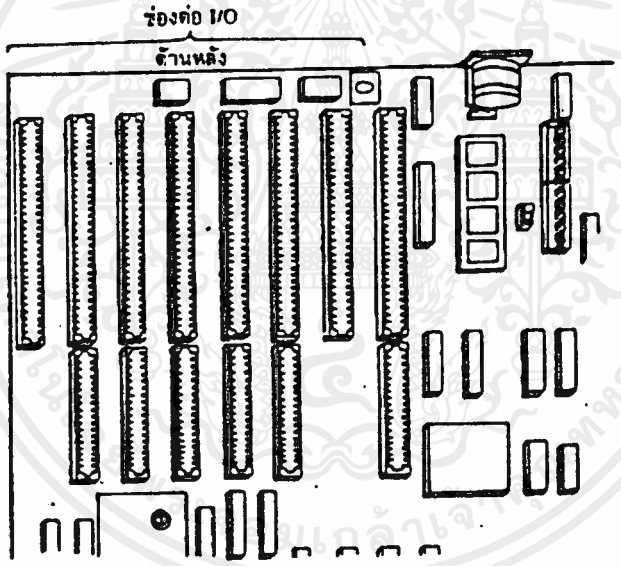
ในการเริ่มต้นทำงาน โปรแกรมไบออสจะเริ่มต้นตั้งค่าให้กับ 8237 เพื่อกำหนดโหมดการทำงาน การกำหนดโหมดการทำงานนี้จะต้องคล้องจองกับการทำงานในส่วนต่าง ๆด้วย

2.11 สล็อตหรือช่องต่อสำหรับอินพุตและเอาต์พุต

ไมโครคอมพิวเตอร์แบบเอ็กซีที มีสล็อตแบบ 62 จำนวน 8 สล็อตไว้ต่อเชื่อมกับอินพุต-เอาต์พุต แต่เมื่อพัฒนามาเป็นเครื่องแบบเอที ทำให้มีความสามารถบางอย่างเพิ่มขึ้น ดังนั้นจึงจำเป็นต้องปรับปรุงสล็อตเพิ่มเติมและเพื่อให้เข้ากันได้กับของเดิมได้บริษัทไอบีเอ็มจึงกำหนด

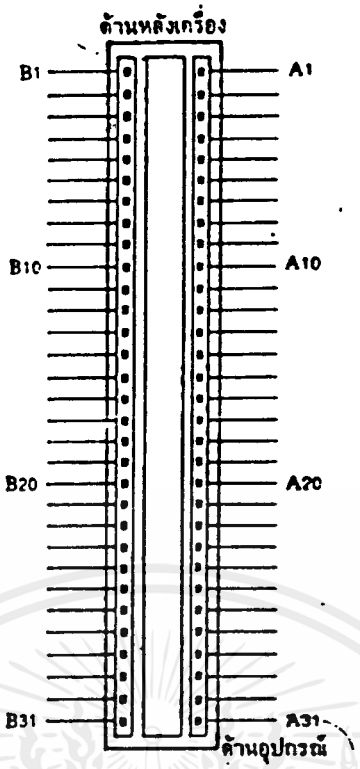
สล็อตเพิ่มเติมจากเดิมโดยมีโครงสร้างรูปแบบของจริงดังรูปที่2.21 สำหรับจุดมุ่งหมายของช่วงต่ออินพุตและเอาต์พุต หรือสล็อตนี้มีเพื่อสนับสนุนดังนี้

- แอดเดรสหมายเลขพอร์ตจากพอร์ตหมายเลข 100 ถึง 3FF
- ใช้มีแอดเดรสครบ 24 เส้นตามโครงสร้างของ 80286 เพื่ออ้างอิงหน่วยความจำได้ 16 MB
- แซนเนลดีเอ็มเอ
- สร้างสถานะการรอของอินพุตหรือเอาต์พุต (I/O wait state)
- เปิดสถานะของการเชื่อมต่อ เพื่อให้อุปกรณ์ภายนอกเชื่อมโยงกับระบบในส่วนต่างๆได้ง่าย
- รีเฟรชหน่วยความจำจากแซนเนลของไมโครโปรเซสเซอร์ภายใน

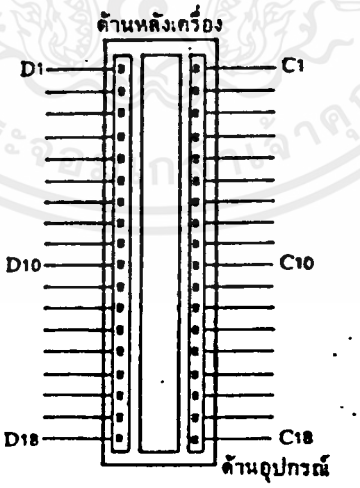


รูปที่ 2.22 การเพิ่ม J10-J16 เป็นสล็อตเพิ่มจากเดิม

จากรูปที่ 2.22 ได้แสดงจำนวนของช่องอินพุต/เอาต์พุตโดยแบ่งเป็น 2 ส่วน คือ ส่วนแรกมีขนาด 62 ขา ส่วนที่ 2 มีขนาด 36 ขา สำหรับวงจรการเชื่อมต่อกับสล็อตแสดงไว้ในรูปที่ 2.28 วงจรแผ่นที่ 13 ซึ่งเป็นสล็อตแบบ 62 ขา และรูปที่ 2.27 วงจรแผ่นที่ 12 เป็นส่วนขยายแบบ 36 ขา ส่วนสล็อตแบบ 62 ขา และสล็อตแบบ 36 ขา แสดงได้จากรูปที่ 2.23 และรูปที่ 2.24 ตามลำดับ และตำแหน่งขาบนสล็อตแสดงดังตารางที่ 2.6



รูปที่ 2.23 การนับขาของสล็อตแบบ 62 ขา



รูปที่ 2.24 การนับขาของสล็อตแบบ 36 ขา

ตารางที่ 2.6 ชื่อของสัญญาณขาต่าง ๆ ของสล็อต

ขาอินพุต เอาต์พุต	ชื่อสัญญาณ	อินพุต/ เอาต์พุต
A1	-I/O CH CK	I
A2	SD7	I/O
A3	SD6	I/O
A4	SD5	I/O
A5	SD4	I/O
A6	SD3	I/O
A7	SD2	I/O
A8	SD1	I/O
A9	SD0	I/O
A10	-I/O CH RDY	I
A11	AEN	O
A12	SA19	I/O
A13	SA18	I/O
A14	SA17	I/O
A15	SA16	I/O
A16	SA15	I/O
A17	SA14	I/O
A18	SA13	I/O
A19	SA12	I/O
A20	SA11	I/O
A21	SA10	I/O
A22	SA9	I/O
A23	SA8	I/O

ขาอินพุต/ เอาต์พุต	ชื่อสัญญาณ	อินพุต/ เอาต์พุต
B1	GND	กราวนด์
B2	RESET DRV	O
B3	+5 Vdc	แหล่งจ่ายไฟเลี้ยง
B4	IRQ9	I
B5	-5 Vdc	แหล่งจ่ายไฟเลี้ยง
B6	DRQ2	I
B7	-12 Vdc	แหล่งจ่ายไฟเลี้ยง
B8	OWS	I
B9	+12 Vdc	แหล่งจ่ายไฟเลี้ยง
B10	GND	กราวนด์
B11	-SMEMW	O
B12	-SMEMR	O
B13	-IOW	I/O
B14	-IOR	I/O
B15	-DACK3	O
B16	DRQ3	I
B17	-DACK1	O
B18	DRQ1	I
B19	-Refresh	I/O
B20	CLK	O
B21	IRQ7	I
B22	IRQ6	I
B23	IRQ5	I

ตารางที่ 2.6 (ต่อ) ชื่อของสัญญาณขาต่าง ๆ ของสล็อต

ขาอินพุต/เอาต์พุต	ชื่อสัญญาณ	อินพุต/เอาต์พุต
A24	SA7	I/O
A25	SA6	I/O
A26	SA5	I/O
A27	SA4	I/O
A28	SA3	I/O
A29	SA2	I/O
A30	SA1	I/O
A31	SA0	I/O

อินพุต/เอาต์พุตแชนเนลด้าน A J1 ถึง J8

ขาอินพุต/เอาต์พุต	ชื่อสัญญาณ	อินพุต/เอาต์พุต
B24	IRQ4	I
B25	IRQ3	I
B26	-DACK2	O
B27	T/C	O
B28	BALE	O
B29	+5 Vdc	แหล่งจ่ายไฟเลี้ยง
B30	OSC	O
B31	GND	กราวนด์

อินพุต/เอาต์พุตแชนเนลด้าน B J1 ถึง J8

ขาอินพุต/เอาต์พุต	ชื่อสัญญาณ	อินพุต/เอาต์พุต
C1	SBHE	I/O
C2	LA23	I/O
C3	LA22	I/O
C4	LA21	I/O
C5	LA20	I/O
C6	LA19	I/O
C7	LA18	I/O
C8	LA17	I/O
C9	-MEMR	I/O
C10	-MEMW	I/O
C11	SD08	I/O
C12	SD09	I/O
C13	SD10	I/O
C14	SD11	I/O
C15	SD12	I/O
C16	SD13	I/O

ขาอินพุต/เอาต์พุต	ชื่อสัญญาณ	อินพุต/เอาต์พุต
D1	-MEM CS16	I
D2	-I/O CS16	I
D3	IRQ10	I
D4	IRQ11	I
D5	IRQ12	I
D6	IRQ15	I
D7	IRQ14	I
D7	-DACK0	O
D9	DRQ0	I
D10	-DACK5	O
D11	DRQ5	I
D12	-DACK6	O
D13	DRQ6	I
D14	-DACK7	O
D15	DRQ7	I
D16	+5 Vdc	แหล่งจ่ายไฟเลี้ยง

ตารางที่ 2.6 (ต่อ) ชื่อสัญญาณขาต่าง ๆ ของสล็อต

ขาอินพุต/เอาต์พุต	ชื่อสัญญาณ	อินพุต/เอาต์พุต
C17	SD14	I/O
C18	SD15	I/O

อินพุต/เอาต์พุตแชนเนลด้าน C
J10 ถึง J14 และ J16

ขาอินพุต/เอาต์พุต	ชื่อสัญญาณ	อินพุต/เอาต์พุต
D17	-MASTER	I
D18	GND	กราวนด์

อินพุต/เอาต์พุตแชนเนลด้าน D
J10 ถึง J14 และ J16

สัญญาณที่ต่อเชื่อมกับอุปกรณ์อินพุต/เอาต์พุต เป็นสัญญาณที่มีขนาด 5 โวลต์ ตามมาตรฐาน TTL โดยที่แต่ละสล็อตจะเชื่อมต่อกับ ทีทีแอลแบบ LS ได้ 2 อินพุต ดังนั้นการต่อกับสล็อตจำเป็นต้องคำนึงถึงโหลดดังกล่าวนี้ด้วย สัญญาณที่ขาต่าง ๆ ของสล็อตมีความหมายดังนี้

SA0-SA19 (อินพุต/เอาต์พุต) เป็นแอดเดรสของระบบที่ใช้ติดต่อกับหน่วยความจำและอุปกรณ์อินพุต/เอาต์พุต สายสัญญาณนี้จะต่อกับหน่วยความจำได้ 1 MB แต่ถ้าต้องการเชื่อมขยายแอดเดรสจะต้องใช้สายแอดเดรส LA17-LA23 การใช้สัญญาณ SA0-SA19 จะต้องแอกทีฟขณะที่สัญญาณ BALE เป็น "1" และจะแลตช์ไปใช้ขณะเปลี่ยนจาก "1" ไป "0" สัญญาณ BALE เป็นสัญญาณที่มาจากไมโครโปรเซสเซอร์ หรือดีเอ็มเอคอนโทรลเลอร์

LA17-LA23 (อินพุต/เอาต์พุต) สัญญาณนี้เป็นสัญญาณที่ไม่ผ่านการแลตช์มาเลยเป็นสัญญาณที่ขยายเพื่อให้ระบบใช้กับหน่วยความจำได้เต็มที่ 16 MB สัญญาณนี้จะใช้ได้ต่อเมื่อ BALE เป็น "1" สัญญาณนี้จะไม่มีการแลตช์มาเลยจากไมโครโปรเซสเซอร์ ทั้งนี้เพื่อให้ใช้สำหรับการสร้างสถานะการรอ (wait state)ได้ สัญญาณนี้ได้รับการควบคุมโดยโปรเซสเซอร์และดีเอ็มเอคอนโทรลเลอร์ เพื่อควบคุมการเข้าถึงข้อมูล

CLK (เอาต์พุต) เป็นสัญญาณนาฬิกาของระบบ ในการใช้ของไอบีเอ็มเอทีจะส่งสัญญาณนี้เป็นสัญญาณขนาด 6 MHz โดยมีช่วงเวลาประมาณ 167 นาโนวินาที สัญญาณเป็นรูปสี่เหลี่ยมมี duty cycle 50 เปอร์เซ็นต์ สัญญาณนี้มีจุดมุ่งหมายเพื่อใช้ในการซิงโครนัสระบบ มีใช้มีจุดมุ่งหมายสำหรับให้ใช้เป็นฐานเวลา

RESET DRV (เอาต์พุต) สัญญาณนี้ใช้สำหรับรีเซตระบบในขณะปิดเครื่อง หรือขณะที่แหล่งจ่ายไฟเสื่อมขาด หรือไฟตก สัญญาณนี้จะแอกทีฟเมื่อเป็นลอจิก "1"

SD0-SD15 (อินพุต/เอาต์พุต) เป็นสัญญาณข้อมูลขนาด 16 บิต ที่ใช้ติดต่อกับหน่วยความจำไมโครโปรเซสเซอร์ และอุปกรณ์อินพุต/เอาต์พุต บิต D0 เป็นบิตที่มีนัยสำคัญน้อยที่สุดในการติดต่อกับอุปกรณ์บางอย่างที่ต้องใช้ 8 บิต จะมีวิธีการแปลงบิตข้อมูลจาก SD8-SD15 เข้ามาใน 8 บิตล่างได้ เพื่อให้การติดต่อเป็นไปได้ทั้งแบบ 16 บิตและ 8 บิต

BALE (เอาต์พุต) เป็นสัญญาณที่ใช้ สำหรับการแลตช์แอดเดรสของระบบสัญญาณนี้มาจาก 82288 ตัวควบคุมบัสสัญญาณที่จะใช้แลตช์แอดเดรส เมื่อเปลี่ยนจาก "1" กับ "0" และสัญญาณนี้จะได้รับการทำให้เป็น "1" ขณะที่กำลังทำดีเอ็มเอ

I/O CHK (อินพุต)สัญญาณตรวจสอบของ อินพุต/เอาต์พุต เพื่อบอกข้อมูลกับระบบเช่นเดียวกับการตรวจสอบพาริตี้ดังนั้นถ้าบน อินพุต/เอาต์พุตมีข้อผิดพลาดสัญญาณนี้จะแอกทีฟ เพื่อให้ส่งสัญญาณเตือนในลักษณะ parity error

I/O CHRDY (อินพุต) สัญญาณนี้จะได้รับการทำให้เป็น "0" ด้วยหน่วยความจำหรืออุปกรณ์อินพุต/เอาต์พุตการใช้สัญญาณนี้ก็เพื่อให้อุปกรณ์อินพุต/เอาต์พุตที่ช้าจะได้ติดต่อกับระบบ ด้วยการส่งสัญญาณมายังซีพียู เพื่อชิงโครนัสระบบได้

IRQ3-IRQ7,IRQ9-IRQ12 และ IRQ14-IRQ15 (อินพุต) สัญญาณอินเตอร์รัพท์เหล่านี้ต่อเข้าเป็นสัญญาณอินพุต8259A สองตัว เพื่อให้สัญญาณ INT เข้าสู่ไมโครโปรเซสเซอร์ การจัดลำดับความสำคัญเป็นไปตามที่กล่าวมาแล้วในเรื่องของวงจรอินเตอร์รัพท์ โดยมีIRQ7 มีลำดับความสำคัญน้อยที่สุด IRQ9 มีลำดับความสำคัญสูงสุด

IOR (อินพุต/เอาต์พุต) สัญญาณ อินพุต/เอาต์พุต เป็นสัญญาณที่ส่งมาจากซีพียูการควบคุมสัญญาณนี้มาจาก 80286 และดีเอ็มเอคอลโทรลเลอร์ สัญญาณนี้แอกทีฟ "0"

IOW (อินพุต/เอาต์พุต) สัญญาณเขียนข้อมูลลงบนอุปกรณ์ อินพุต/เอาต์พุตสัญญาณนี้ควบคุมจากไมโครโปรเซสเซอร์หรือดีเอ็มเอคอลโทรเลอร์ สัญญาณนี้แอกทีฟด้วยลอจิก "0"

SMEAR (เอาต์พุต) MEMR (อินพุต/เอาต์พุต) สัญญาณนี้เป็นสัญญาณควบคุมการอ่านข้อมูลจากหน่วยความจำ SMEMR ใช้สำหรับ ติดต่อกับหน่วยความจำ ในส่วน 1 MB แรกหรือถอดรหัสมาจากแอดเดรสส่วนล่าง ส่วน MEMR นี้แอกทีฟกับหน่วยความจำได้หมด 16 MB

DRQ0-DRQ3 และ DRQ5-DRQ7 (อินพุต) สัญญาณการขอดีเอ็มเอแชลเนล 0-3 และ 5-7 สัญญาณนี้จะมาจากอุปกรณ์อินพุต/เอาต์พุต DRQ0 มีลำดับความสำคัญสูงสุดและ DRQ7 มีลำดับความสำคัญต่ำสุด DRQ0-DRQ3 ใช้กับดีเอ็มเอแบบ 8 บิตส่วน DRQ5-7

AEN (เอาต์พุต) มีหน้าที่เปิดเอดเดรส เพื่อเป็นสัญญาณเพื่อใช้สำหรับการแยกบัสเอดเดรสในการทำดีเอ็มเอ เมื่อสัญญาณนี้แอกทีฟจะเป็นการทำให้ดีเอ็มเอคอนโทรลเลอร์สามารถควบคุมการทำงานแอดเดรสแทนการควบคุมของซีพียู

REFRESH (อินพุต/เอาต์พุต) เป็นสัญญาณที่ใช้ในการแสดงสัญญาณรีเฟรชไซเคิลสัญญาณนี้ส่งมาจากไมโครโปรเซสเซอร์ผ่านทางช่องอินพุต/เอาต์พุต

T/C (เอาต์พุต) สัญญาณ Terminal Count เป็นสัญญาณพัลซ์เมื่อดีเอ็มเอนับจำนวนมาครบตามที่กำหนด

SBHE (อินพุต/เอาต์พุต) ชื่อสัญญาณ Bus High Enable เป็นสัญญาณบ่งบอกการถ่ายข้อมูลจาก SD8-SD15 เข้าสู่บัฟเฟอร์

MASTER (อินพุต) สัญญาณนี้ใช้กับ DRQ เพื่อควบคุมระบบ สัญญาณนี้มีจุดมุ่งหมายบ่งบอกการควบคุมบัสทั้งหมดว่ามาจากระบบซีพียูหลักนี้ หรือมาจากที่อื่น ถ้าหากสัญญาณนี้แอกทีฟ หมายความว่า ซีพียูเดิมส่งอำนาจการควบคุมให้กับสล็อต ซึ่งอาจจะมีซีพียูอื่น เข้ามาควบคุมระบบก็ได้ อนึ่งหากสัญญาณนี้แอกทีฟเกินกว่า 15 ไมโครวินาที โดยไม่มีการรีเฟรชช่วยอาจทำให้ข้อมูลในหน่วยความจำหายได้

MEM CS16 (อินพุต) สัญญาณนี้เป็นตัวส่งมาบอกเมนบอร์ด ถ้าหากการถ่ายเทข้อมูล

IO CS16 (อินพุต) สัญญาณนี้เป็นตัวส่งบอกเมนบอร์ดว่าอินพุต/เอาต์พุตต้องการสถานะรอ

OSC (เอาต์พุต) สัญญาณนาฬิกา 70 นาโนวินาที หรือประมาณ 14.31818 เมกะเฮิรตซ์สัญญาณนี้ไม่ได้ซิงโครนัสกับระบบ

OWS (อินพุต) เป็นสัญญาณที่จะบอกซีพียูว่าการทำงานในหนึ่งรอบของบัสไม่ต้องรอ

2.12 วงจรลำโพง

วงจรสัญญาณเสียงออกมาทาง OIT2 ของ 8254 ผ่าน AND เกตเข้ากับสัญญาณ SPKR DATA ซึ่งมาจากการแลตช์พอร์ต เพื่อ ON หรือ OFF เสียง การกำเนิดเสียง จึงเกิดจากการส่ง ON หรือ OFF ผ่าน U_{27} คือ ALS175 ส่วนสัญญาณความถี่ของเสียงเกิดจากวงจรตั้งเวลาในตัว 8254 ซึ่งค่าความถี่ของเสียงสามารถโปรแกรมได้

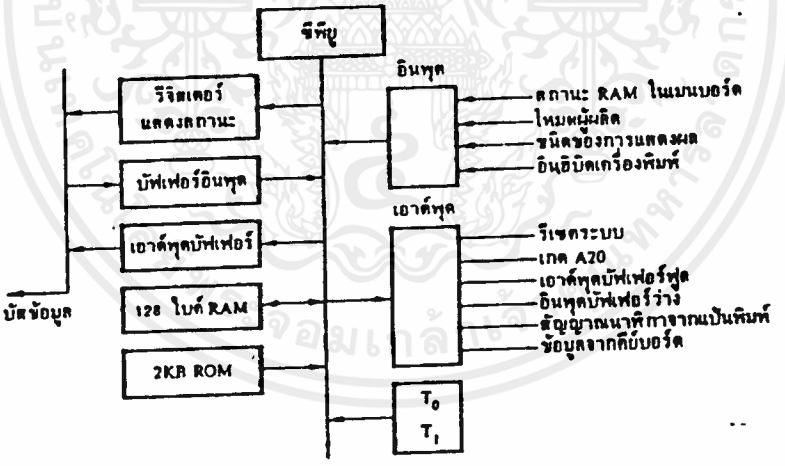
2.13 วงจรควบคุมคีย์บอร์ด

มีสิ่งที่น่าสนใจแตกต่างจากเครื่องเอกซ์ที คือ ใช้ไมโครโปรเซสเซอร์ชิปเดียว คือ 8042 สำหรับแทน 8255 เดิมของเอกซ์ที ในเครื่องที่มีการเชื่อมต่อกับคีย์บอร์ดจะต้องผ่านทาง 8042 นี้ โครงสร้างของไมโครโปรเซสเซอร์ 8042 ที่ต่ออยู่กับระบบ แสดงเป็นบล็อกไดอะแกรม เพื่อความเข้าใจได้ง่ายดังรูปที่ 2.24

ผู้ออกแบบพีซีเอทีเห็นว่า เราใช้ 8255 และดิพสวิตช์ในพีซีเอ็กซ์ทีมีจุดอ่อน เพราะต้องการใช้ฮาร์ดแวร์เพิ่มในการตรวจสอบแป้นพิมพ์ และหากเปลี่ยนแปลงแก้ไขรหัสแกนที่มาจากแป้นพิมพ์ที่จะยุ่งยาก และยังทำให้ฮาร์ดแวร์ส่วนนี้มีมากขึ้น การใช้ 8042 จึงเป็นความคิดที่ดีมาก

8042 นี้มีรอมภายใน 2 กิโลไบต์ และมีแรม 128 ไบต์ มีพอร์ตอินพุต/เอาต์พุตที่สามารถเชื่อมต่อกันภายนอกได้โดยตรง

การรับข้อมูลจากแป้นพิมพ์รับเข้าทางขา T_0 และ T_5 โดยสัญญาณนาฬิกาที่ชิงโครนัส



รูปที่ 2.25 วงจรบล็อกไดอะแกรม 8042 สำหรับเชื่อมต่อกับแป้นพิมพ์

กับข้อมูลเข้าทาง T_0 และตัวข้อมูลคีย์บอร์ดเข้าทาง T_1 จะเห็นจุดเชื่อมต่อสายออกมาสองปลั๊ก ปลั๊กแรกเป็นตัวเชื่อมกับคีย์บอร์ดมี 5 ขา เหมือนขาของพีซีเอ็กซ์ที โดยแต่ละขามีรายละเอียดดังนี้

ขา 1 เป็นสัญญาณนาฬิกาของคีย์บอร์ด

ขา 2 เป็นสัญญาณข้อมูล

ขา 3 ว่าง

ขา 4 กราวนด์

ขา 5 แรงดัน 5 โวลต์

การจัดวางขาแสดงดังรูปที่ 2.26 ส่วนปลั๊กอีกตัวคือ ปลั๊กต่อแสดงสถานะและการล็อกคีย์บอร์ด โดยมีปลั๊กขนาด 5 ขา ดังนี้

ปลั๊กคีย์บอร์ดด้านหน้า ปลั๊กสถานะด้านล่าง

รูปที่ 2.26 ลักษณะปลั๊กที่เชื่อมต่อกับ 8042

ขา 1 ต่อกับ LED แสดงไฟเลี้ยง

ขา 2 ว่าง

ขา 3 กราวนด์

ขา 4 ขาล็อกคีย์บอร์ดจากกุญแจล็อก

ขา 5 กราวนด์

สำหรับไมโครคอมพิวเตอร์ชิปเดียว 8042 นี้ต่อเข้ากับระบบโดยการเลือกแอดเดรสทาง -8041CS ซึ่งเป็นการเลือกพอร์ตเหมือนการเลือก 8255 ของซีพียูซีพีทำการเขียนการอ่านผ่านทาง XIOR และ XIOW ทำให้ซีพียู 80286 สามารถติดต่อผ่านเข้าทางบัสข้อมูลในการส่งสถานะได้ พอร์ตของ 8042 พอร์ตแรกคือ พอร์ต P10-P17 จะเห็นว่า P10-P13 ไม่ได้ใช้ ส่วน P14-P17 ใช้สำหรับอินพุต P14 เป็นตัวตรวจสอบแรม P16 เป็นตัวอ่านค่าดิพสวิตช์ เพื่อเลือกว่า ขณะนี้ระบบต่ออยู่กับจอสีหรือจอมโนโครม ส่วน P17 เป็นตัวตรวจสอบสถานะอินฮิบิทคีย์บอร์ด เพื่อล็อกคีย์บอร์ด

การรับข้อมูลจะอ่านเข้ามาทางขา TEST0 และ TEST1 อย่างไรก็ตามมีพอร์ต P26 และ P27 เป็นเอาต์พุตรูปเพื่อการทดสอบคีย์บอร์ด สำหรับการอ่านค่าคีย์บอร์ดจะได้ค่ารหัสสแกนซึ่งต้องแปลงค่าเป็นรหัสเอสซีกภาษาในโปรแกรมแอสเซมบลี

2.14 รีจิสเตอร์แสดงสถานะ

แต่เดิมบนพีซีเอ็กซ์ทีใช้สวิตช์เล็ก ๆ เป็นตัวแสดงสถานะ แต่การ์ดของพีซีเอทีนี้นำใช้รีจิสเตอร์ภายในของ 8042 ในส่วนพอร์ตหมายเลข 64H (ดูจากซีพียู 80286) ซึ่งจะอ่านข้อมูลจาก8042 ได้ 8 บิต เรียกว่ารีจิสเตอร์แสดงสถานะ โดยที่แต่ละบิตมีความหมายแตกต่างกันออกไปตามโปรแกรมที่ตั้งไว้ โดยมีรายละเอียดดังนี้

บิต 0 บอกบัฟเฟอร์ของแป้นพิมพ์เต็ม ถ้ามีค่า "0" หมายถึงบัฟเฟอร์ของแป้นพิมพ์ยังว่างอยู่ ส่วนของบัฟเฟอร์เอาต์พุตอยู่ที่พอร์ตหมายเลข 60H ดังนั้นถ้าบิตนี้มีค่าเป็น "1" และถ้าหากมีการอ่านพอร์ต 60H จะทำให้บิตนี้กลายเป็น "0"

บิต 1 บัฟเฟอร์อินพุตเต็ม บอกว่าบัฟเฟอร์การควบคุมเป็นพิมพ์พอร์ตแอดเดรส 60 และ 64 ว่าว่างหรือเต็ม ถ้าบิตนี้เป็น "0" จะบอกว่าว่าง ถ้าเป็น "1" จะบอกว่าข้อมูลได้รับการเขียนลงไปในบัฟเฟอร์แล้ว และคอนโทรลเลอร์ยังไม่ได้อ่านไป

บิต 2 แฟลกของระบบ บิตนี้จะได้รับการเขียนให้เป็น "0" เมื่อเริ่มต้นรีเซตระบบ บิตนี้จะได้รับการเซตค่าด้วยการส่งคำสั่งเข้ามาควบคุมแป้นพิมพ์

บิต 3 บอกว่าเป็นคำสั่งหรือข้อมูล โดยบัฟเฟอร์ของคอนโทรลเลอร์พอร์ต 60H หรือ 64Hอาจจะเป็นข้อมูลหรือคำสั่ง โดยแอดเดรสพอร์ต 60H กำหนดให้เป็นพอร์ตข้อมูล และ 64H ให้เป็นคำสั่ง หากต้องการเขียนไปยังพอร์ต 64H ต้องเซตบิตนี้ให้เป็น "1" และถ้าเซตเป็น "0" หมายถึงต้องการติดต่อกับพอร์ตแอดเดรส 60H

บิต 4 สถานะการอินฮิบิตคีย์บอร์ด ถ้าเป็น "0" แสดงว่าอินฮิบิตคีย์บอร์ดคือล็อคคีย์บอร์ดนั่นเอง

บิต 5 หมดเวลาส่ง ถ้าเป็น "1" หมายถึงการส่งข้อมูลจากการส่งข้อมูลคอนโทรลเลอร์ไม่สมบูรณ์ เป็นบิตเพื่อบอกสถานะข้อผิดพลาด

บิต 6 หมดเวลารับ เป็น "1" แสดงว่าการส่งข้อมูลมาแล้วโปรแกรมไม่สามารถรับไปในเวลาที่กำหนด

บิต 7 ข้อผิดพลาดจากพาริตี้ เป็น "0" แสดงว่าข้อมูลที่ได้รับเป็นพาริตี้คี่ เป็น "1" แสดงว่าเป็นพาริตี้คู่

เอาต์พุตบัฟเฟอร์ ที่กล่าวถึงคือบัฟเฟอร์แบบอ่านได้อย่างเดียว อยู่ที่พอร์ตหมายเลข 60H ตัวคีย์บอร์ดคอนโทรลเลอร์รับข้อมูลมาจากคีย์บอร์ดแล้วเก็บไว้ในบัฟเฟอร์ตัวนี้

อินพุตบัฟเฟอร์ เป็นรีจิสเตอร์ที่เขียนได้อย่างเดียว อยู่ที่พอร์ต 60H หรือ 64H ถ้าเขียนที่แอดเดรส 60H จะเซตแฟลกเพื่อบอกเป็นการเขียนข้อมูล ถ้าเขียนที่แอดเดรส 64H จะได้รับการเซตแฟลกเป็นการเขียนคำสั่ง ข้อมูลที่เขียนลงในพอร์ต 60H จะส่งต่อให้กับคีย์บอร์ด

พอร์ต 64H พอร์ตคำสั่งโปรแกรมคอนโทรลเลอร์สามารถส่งรหัสเพื่อควบคุมการทำงานของคีย์บอร์ดคอนโทรลเลอร์ผ่านเข้าทางพอร์ตนี้ คำสั่งที่เขียนมีรหัสดังนี้

20 อ่านไบต์คำสั่งของคอนโทรลเลอร์ เพื่อบอกให้คีย์บอร์ดคอนโทรลเลอร์ส่งรหัสคำสั่งขณะนั้นมาที่เอาต์พุตบัฟเฟอร์

60 เขียนไบต์คำสั่งของคอนโทรลเลอร์ เป็นการบอกว่าต้องการเขียนคำสั่งโดยไบต์ที่ตามมาในพอร์ต 60H จะเป็นคำสั่งโดยแต่ละบิตมีความหมายดังนี้

บิต 7 สงวนไว้ (โดยปกติให้เป็น "0")

บิต 6 โหมดไอบีเอ็มหมายถึงการกำหนดรหัสสแกนจะตามที่บริษัทไอบีเอ็มกำหนดไว้

บิต 5 โหมดไอบีเอ็ม เป็นการบอกว่าแป้นพิมพ์เป็นของไอบีเอ็ม ให้บิตนี้เป็น "1" คอนโทรลเลอร์ จะไม่ตรวจสอบพาริตี้หรือแปลงรหัสสแกน

บิต 4 ดิสเอเบิลคีย์บอร์ด ถ้าเขียนด้วย "1" จะเป็นการดิสเอเบิล โดยการทำให้สายสัญญาณนาฬิกาเป็น "0" ข้อมูลจะรับส่งกันไม่ได้

บิต 3 ถ้าเป็น "1" จะทำให้ฟังก์ชันการอินฮิบิตคีย์บอร์ดทำงานไม่ได้

บิต 2 แฟลกระบบ ถ้าเขียนอะไรไปจะอ่านได้ที่รีจิสเตอร์สถานะ

บิต 1 สงวนไว้ ปกติให้ค่า "0"

บิต 0 อีนาเบิลอินเตอร์รัพต์ของบัฟเฟอร์เอาต์พุต ถ้าเขียน "1" หมายถึงอีนาเบิลให้อินเตอร์รัพต์เกิดขึ้นได้

AA ทดสอบตัวเอง เพื่อหาข้อผิดพลาด โดยได้รหัส 55H ที่บัฟเฟอร์เอาต์พุต หมายถึงไม่มีข้อผิดพลาด

AB ทดสอบอินเตอร์เฟส โดยทดสอบสายข้อมูลและสัญญาณนาฬิกา ผลลัพธ์ของการทดสอบจะปรากฏที่เอาต์พุตบัฟเฟอร์

AC คัมพ์ชื่อตรวจสอบระบบคีย์บอร์ด เป็นการส่งข้อมูล 16 ไบต์จากหน่วยความจำภายใน 8042 สถานะของพอร์ตอินพุต สถานะปัจจุบันของพอร์ตเอาต์พุต สถานะของคอนโทรลเลอร์ข้อมูลทั้งหมดจะส่งมาในเฟอร์แมตของรหัสสแกน

AD ดิสเอเบิลคีย์บอร์ด คำสั่งนี้จะเซตบิต 4 ของคำสั่งที่สั่งคอนโทรลเลอร์เพื่อดิสเอเบิลการรับส่งข้อมูล

AE อีนาเบิลการเชื่อมต่อคีย์บอร์ด คำสั่งนี้จะเคลียร์บิต 4 ของไบต์คำสั่งเพื่อให้รับส่งข้อมูลได้

C0 อ่านพอร์ตอินพุต คำสั่งนี้ทำให้คอนโทรลเลอร์อ่านพอร์ตอินพุต แล้วนำข้อมูลใส่ไว้ในบัฟเฟอร์เอาต์พุต คำสั่งนี้จะใช้ได้ต่อเมื่อบัฟเฟอร์เอาต์พุตว่างอยู่

D0 อ่านพอร์ตเอาต์พุต คำสั่งนี้ทำให้คอนโทรลเลอร์อ่านพอร์ตเอาต์พุต แล้วเก็บข้อมูลไว้ในบัฟเฟอร์เอาต์พุต คำสั่งนี้ใช้ได้ต่อเมื่อบัฟเฟอร์เอาต์พุตว่างอยู่

D1 เขียนพอร์ตเอาต์พุต ข้อมูลไบต์ต่อมาที่เขียนในพอร์ต 60H จะส่งลงไปในพอร์ตเอาต์พุตของคอนโทรลเลอร์ (8042)

อนึ่งบิต 0 ของพอร์ตเอาต์พุตของคอนโทรลเลอร์ต่อกับรีเซตของระบบ

E0 อ่านอินพุตทดสอบคำสั่งนี้ทำให้คอนโทรลเลอร์อ่าน T_0 และ T_1 และข้อมูลที่ได้จะเก็บไว้ที่บัฟเฟอร์เอาต์พุต บิตข้อมูลเป็น 0 แทน T_0 บิตข้อมูลเป็น 1 แทน T_1

F0-FF พัลส์ของพอร์ตเอาต์พุต บิต 0 ถึงบิต 3 ของเอาต์พุตของพอร์ตคอนโทรลเลอร์ส่งเป็นพัลส์ออกไปโดยมีพัลส์ลอจิก "0" ประมาณ 6 ไมโครวินาที บิต 0-3 ของคำสั่งนี้แสดงว่าจะให้พัลส์ไปปรากฏที่บิตใดของเอาต์พุต ถ้าเป็น 0 แสดงว่าจะมีพัลส์

อนึ่งบิต 0 ของพอร์ตเอาต์พุตเป็นการต่อรีเซตถ้าส่งพัลส์ออกทางบิตนี้จะทำให้รีเซต

2.15 วงจรกำเนิดเวลาจริงและซีมอสแรม

พีซีเอทีมีข้อแตกต่างกับพีซีเอ็กซ์ทีคือ มีนาฬิกาจริงอยู่ที่สามารถปรับปรุงเวลาเองอย่างอัตโนมัติ มีแบตเตอรี่สำรองอยู่ ทำให้นาฬิกาดำเนินได้ตลอดเวลา ชิพที่ใช้ในที่นี้คือชิพมอโตโรล่า MC146818 โดยมี RAM อยู่ภายในเพื่อเก็บข้อมูลสถานะและเวลา การเลือกชิพซีมอสเพราะต้องการให้ใช้กำลังงานต่ำมาก จึงใช้กับแบตเตอรี่แบบลิเทียมก้อนเล็กๆได้เป็นปี

วงจร MC146818 นี้ต่อเข้ากับไฟฟ้าสำรองแบตเตอรี่ โดยมี MC14089 เป็นวงจรกำเนิดสัญญาณฐานเวลา 32.768 กิโลเฮิรตซ์ ดังนั้นวงจรสำรองไฟฟ้าจึงต้องคงต่อชิพทั้งสองนี้ได้ตลอดเวลา แม้จะเลิกจ่ายไฟฟ้าให้กับระบบแล้ว

MC146818 นี้จะส่งสัญญาณอินเตอร์รัพต์ทาง IRQ8 ให้กับระบบอย่างต่อเนื่อง เพื่อให้ซีพียูอ่านข้อมูลเวลาจากชิพ เพื่อนำไปปรับปรุงเวลาของระบบ

การอ่านซีมอสเรม (CMOS RAM) จะทำได้ด้วยการส่งค่าแอดเดรสของซีมอสเรมออกไปทางพอร์ต 70H เสียก่อน ค่าของแอดเดรสจะได้รับการแจ้งไว้ภายในหลัง

ข้อมูลในซีมอสเรมนี้มีความสำคัญมาก ดังนั้นการเขียนข้อมูลหรือปรับปรุงข้อมูลบางไบต์จำเป็นต้องแก้ไขไบต์ตรวจสอบผลรวมด้วย การเก็บข้อมูลทั้ง 64 ไบต์เป็นดังนี้

ข้อมูลเกี่ยวกับนาฬิกา ไบต์ 00-0D จำนวน 14 ไบต์

ไบต์ 00 เวลาหน่วยวินาที

ไบต์ 01 หลักวินาทีที่ตั้งเวลาไว้

ไบต์ 02 เวลาหน่วยนาที

ไบต์ 03 หลักนาทีที่ตั้งเวลาไว้

ไบต์ 04 หลักชั่วโมง

ไบต์ 05 หลักชั่วโมงที่ตั้งเวลาไว้

ไบต์ 06 วันที่ในสัปดาห์

ไบต์ 07 วันที่ของเดือน

ไบต์ 08 เดือน

ไบต์ 09 ปี

ไบต์ 0A รีจิสเตอร์แสดงสถานะ A

ไบต์ 0B รีจิสเตอร์แสดงสถานะ B

ไบต์ 0C รีจิสเตอร์แสดงสถานะ C

ไบต์ 0D รีจิสเตอร์แสดงสถานะ D

ไบต์ 0E เป็นไบต์แสดงสถานะการตรวจสอบ

ไบต์ 0F แสดงสถานะของไฟเลี้ยง

ไบต์ 10 แสดงสถานะการติดตั้งฟลอปปีดิสก์แบบ 1.2 MB หรือ 360 KB หรืออื่นๆ

ไบต์ 11 สงวนไว้

ไบต์ 12 แสดงการติดตั้งฮาร์ดดิสก์โดยบอกชนิดของฮาร์ดดิสก์

ไบต์ 13 สงวนไว้

ไบต์ 14 บอกสถานะของคอนฟิกูเรชั่นของระบบ

ไบต์ 15,16 บอกขนาดของหน่วยความจำบนเมนบอร์ด

ไบต์ 17,18 บอกขนาดหน่วยความจำที่ขยาย

ไบต์ 19 ถึง 2D สงวนไว้

ไบต์ 2E และ 2F ใช้สำหรับตรวจสอบ check sum

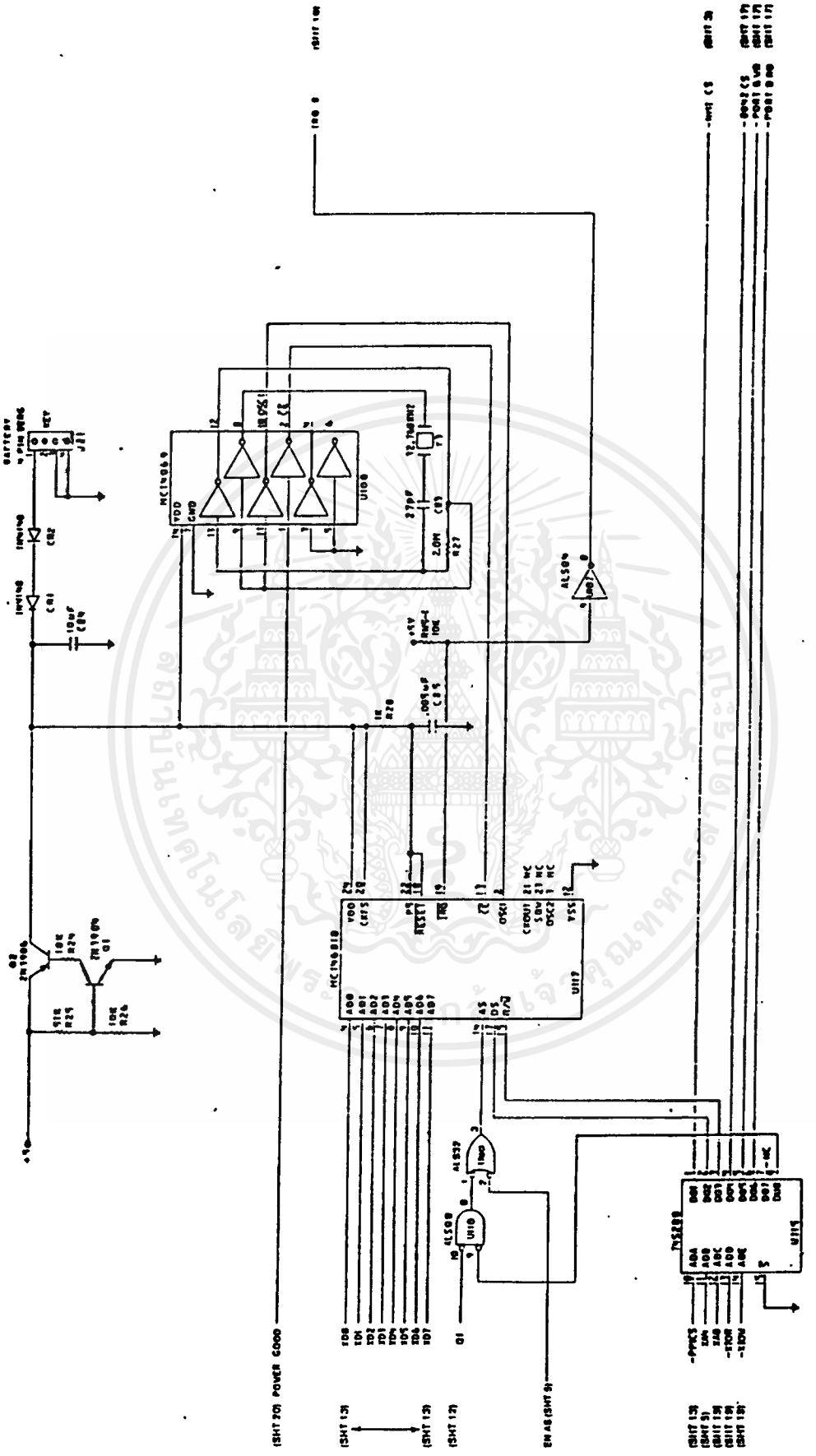
ไบต์ 30 และ 31 บอกขนาดหน่วยความจำที่ขยาย

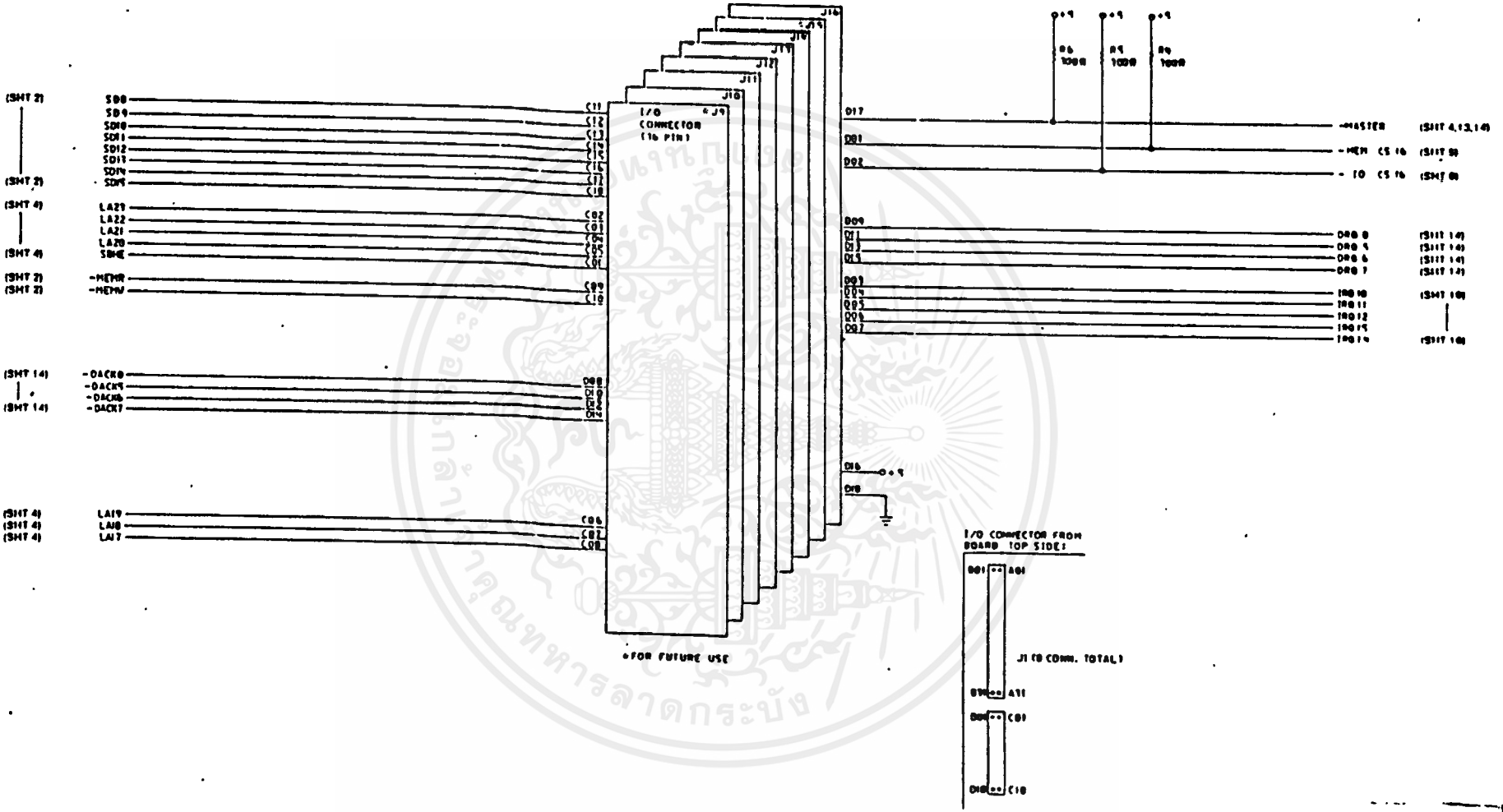
ไบต์ 32 บอกตัวเลข BCD เป็นศตวรรษให้ไบออสใช้

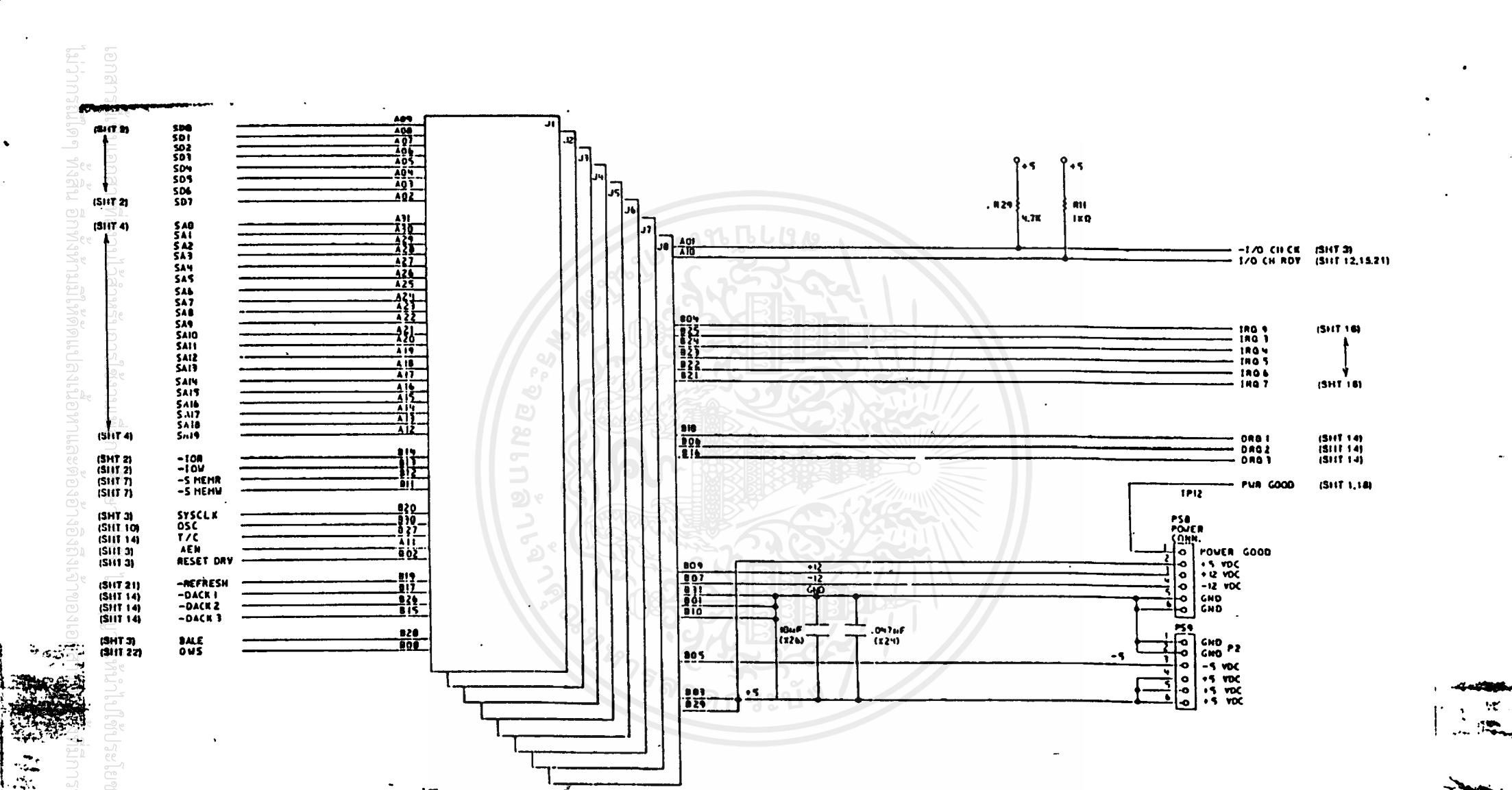
ไบต์ 33 แฟลกข่าวสาร

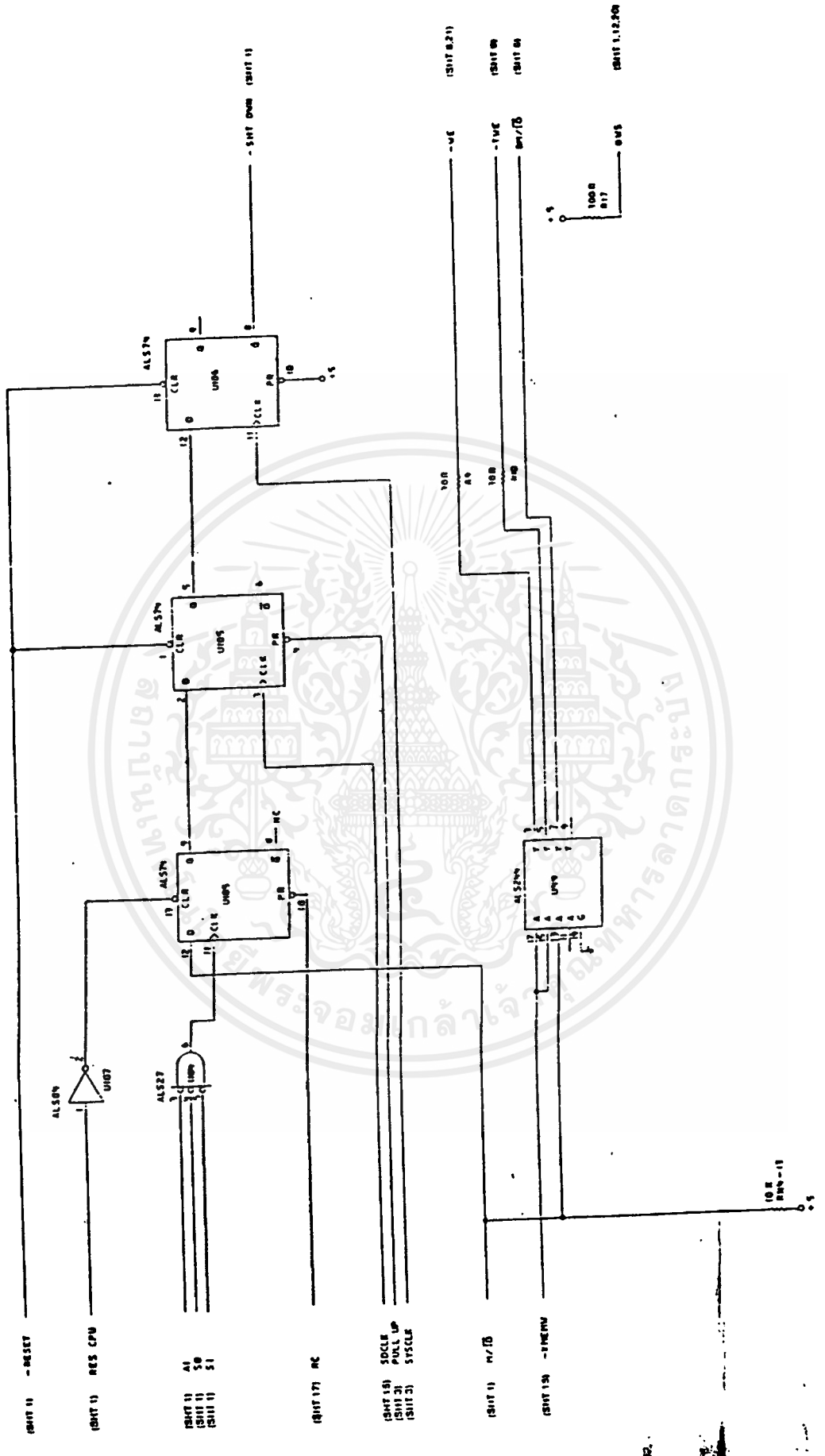
ในการใช้งาน พีซีเอที ไบออสจะตรวจสอบ หน่วยความจำนี้ทุกครั้งที่เริ่มต้นระบบ หากแบตเตอรี่หายไปทำให้ข้อมูลในซีมอสเรมหายไป จะทำให้ระบบไม่สามารถทำงานได้

การทำงานใด ๆ ของระบบจะได้รับการติดตั้งค่าเริ่มต้นไว้ที่ไบออส โปรแกรมไบออสจะกำหนดค่าต่าง ๆ เหล่านี้เอาไว้ การศึกษาให้เข้าใจระบบได้ดีควรจะต้องรู้เรื่องฮาร์ดแวร์คือ วงจรของระบบและคูโหมดการทำงานต่าง ๆ ในโปรแกรมไบออส




I/O CONNECTOR (16 PIN)
*FOR FUTURE USE
I/O CONNECTOR FROM BOARD TOP SIDE:
J1 (8 CONN. TOTAL)
-MASTER
-MEM CS 16
-I/O CS 16


SD0
SD1
SD2
SD3
SD4
SD5
SD6
SD7
SA0
SA1
SA2
SA3
SA4
SA5
SA6
SA7
SA8
SA9
SA10
SA11
SA12
SA13
SA14
SA15
SA16
SA17
SA18
SA19
-IOR
-IOW
-S MEMR
-S MEMW
SYSCLK
OSC
T/C
AEN
RESET DRV
-REFRESH
-DACK 1
-DACK 2
-DACK 3
BALE
OWS
J1
J2
J3
J4
J5
J6
J7
J8
+5
-I/O CH CK
I/O CH RDY
IRQ 9
IRQ 3
IRQ 4
IRQ 5
IRQ 6
IRQ 7
DRQ 1
DRQ 2
DRQ 3
PWR GOOD
TP12
PS8
POWER CONN.
POWER GOOD
+5 VDC
+12 VDC
-12 VDC
GND
GND
PS9
GND
GND P2
-5 VDC
+5 VDC
+5 VDC
+5 VDC
+12
-12
GND
-5
.047μF (E24)



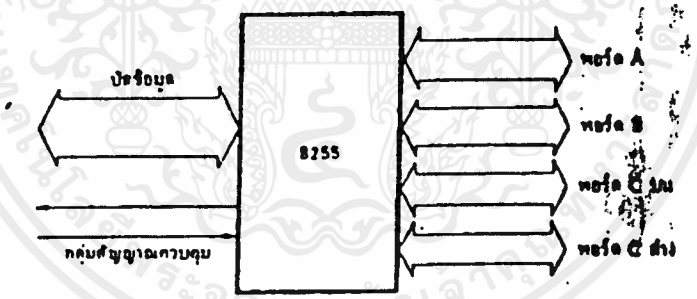
บทที่ 3

อินเตอร์เฟสการ์ด

3.1 8255 PPI

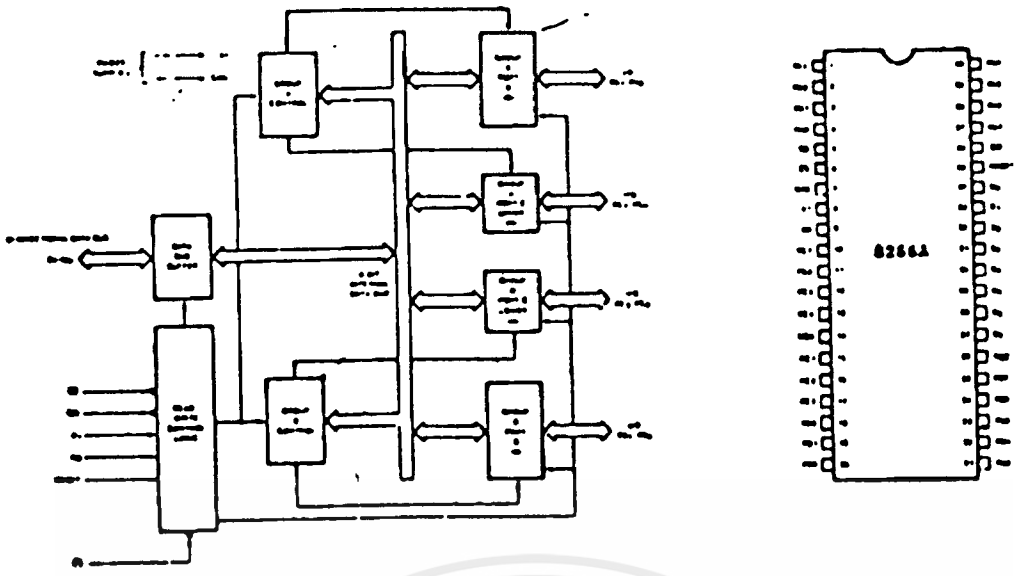
3.1.1 หลักการเบื้องต้นของ 8255

เป็นไอซี 40 ขา ตัวไอซี 8255ได้รับการออกแบบมาเพื่อให้มีสัญญาณเชื่อมโยงกับ 8080 แต่สัญญาณนี้เหมาะที่จะใช้กับ IBM PC ได้ดีเช่นเดียวกัน 8255เป็นไอซีที่ต่อเป็นพอร์ตให้กับไมโครโปรเซสเซอร์ได้ 3 พอร์ตโดยมีโครงสร้างพื้นฐานแสดงด้วยบล็อกไดอะแกรมได้ดังรูปที่ 3.1



รูปที่ 3.1 ไดอะแกรมของไอซี 8255

การเรียกพอร์ตของ 8255 จะเรียกพอร์ตว่าพอร์ต A,B และ C โดยพอร์ต C แยกเป็น 2 ส่วนคือ PC_0 -PC_3 เรียกว่าพอร์ต C ล่างจำนวน 4 บิตและพอร์ต C บนคือ PC_4 - PC_7 ที่พิเศษคือ พอร์ตทุกพอร์ตเป็นได้ทั้ง พอร์ตอินพุต และเอาต์พุต



รูปที่ 3.2 ไดอะแกรมการทำงานของ 8255

3.1.2 ขาต่าง ๆ ของ 8255

เพื่อให้เข้าใจวิธีต่อใช้งานต่าง ๆ ระหว่าง IBM PC/AT กับ 8255 จึงจำเป็นต้องเข้าใจความหมายและตำแหน่งของขาต่าง ๆ เสียก่อน ขาทั้ง 40 ขา ของไอซีประกอบด้วย

D_0 - D_7 เป็นขาข้อมูลของอินพุต-เอาต์พุต ที่จะต้องผ่านเข้าออกจากส่วนนี้ D_0 - D_7 จึงต้องต่อเข้ากับระบบบัสของไมโครโปรเซสเซอร์เพื่อให้ไมโครโปรเซสเซอร์สามารถอ่านหรือเขียนข้อมูลออกทางพอร์ตผ่านทางบัสนี้

CS (เลือกชิพ) ขานี้เป็นขาอินพุตที่จะรับสัญญาณจากภายนอก เพื่อเลือกชิพ 8255 นี้โดย เมื่อขานี้เป็นลอจิก "0" จะทำให้ตัว 8255 ต่อเข้ากับระบบของบัสของไมโครโปรเซสเซอร์เพื่อให้ไมโครโปรเซสเซอร์เขียนหรืออ่านข้อมูลออกจากพอร์ตได้

RD ขาสัญญาณการอ่านเป็นสัญญาณอินพุตที่จะส่งมาจาก CPU เมื่อสัญญาณที่ขานี้เป็น "0" และ CS เป็น "0" ตัว 8255 จะทำให้ตัวชิพนี้อ่านข้อมูลจากบัส ในขณะเป็นอินพุต

WR ขาสัญญาณการเขียน จะแอคทีฟเมื่อสัญญาณ WR เป็น "0" และ CS เป็น "0" สัญญาณนี้จะมาจาก CPU เมื่อต้องการเขียนข้อมูลลงบนพอร์ตที่กำหนด

A_0 - A_1 ขาแอดเดรสลอจิกของขาทั้งสองข้างจะถอดรหัสเป็น 4 เพื่อกำหนดรีจิสเตอร์ภายในที่เชื่อมต่อกับพอร์ตอินพุต-เอาต์พุตของ 8255

RESET ขารีเซต เป็นสัญญาณที่ส่งจากภายนอกเข้ามาทำการรีเซต 8255 เพื่อเคลียร์สถานะต่าง ๆ ของ 8255 เมื่อ 8255 ได้รับการรีเซต มันจะกลับเข้าสู่โหมดอินพุตหรือทุกพอร์ตเป็นพอร์ตอินพุต

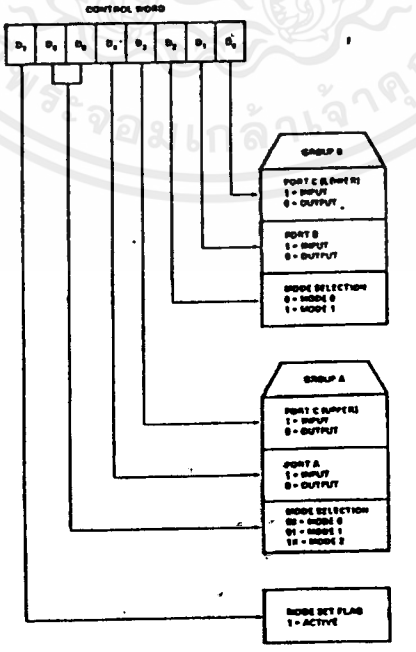
PA_0 - PA_7 เป็นสายสัญญาณที่พอร์ตของ 8255 ที่ชื่อพอร์ต Aการเลือกพอร์ตโดยขาแอดเดรส A_0 - A_1

PC_0 - PC_1 เป็นสัญญาณที่พอร์ต Cของ 8255การกำหนดพอร์ตนี้จะได้รับการกำหนดโดยขาแอดเดรส A_0 - A_1 พอร์ต C นี้แบ่งเป็น 2 กลุ่ม คือกลุ่ม PC_0 - PC_3 และกลุ่ม PC_4-PC_7

รูปที่ 3.3 แสดงตำแหน่งการวางขาของ 8255

3.1.3 MODE การทำงานของ 8255

การเลือกการทำงานของ 8255 สามารถให้ทำงานได้หลายรูปแบบโดยเรียก แต่ละแบบว่าโหมด (MODE) ต่างๆ มีอยู่ 3 โหมด คือ โหมด 0 โหมด 1 และโหมด 2



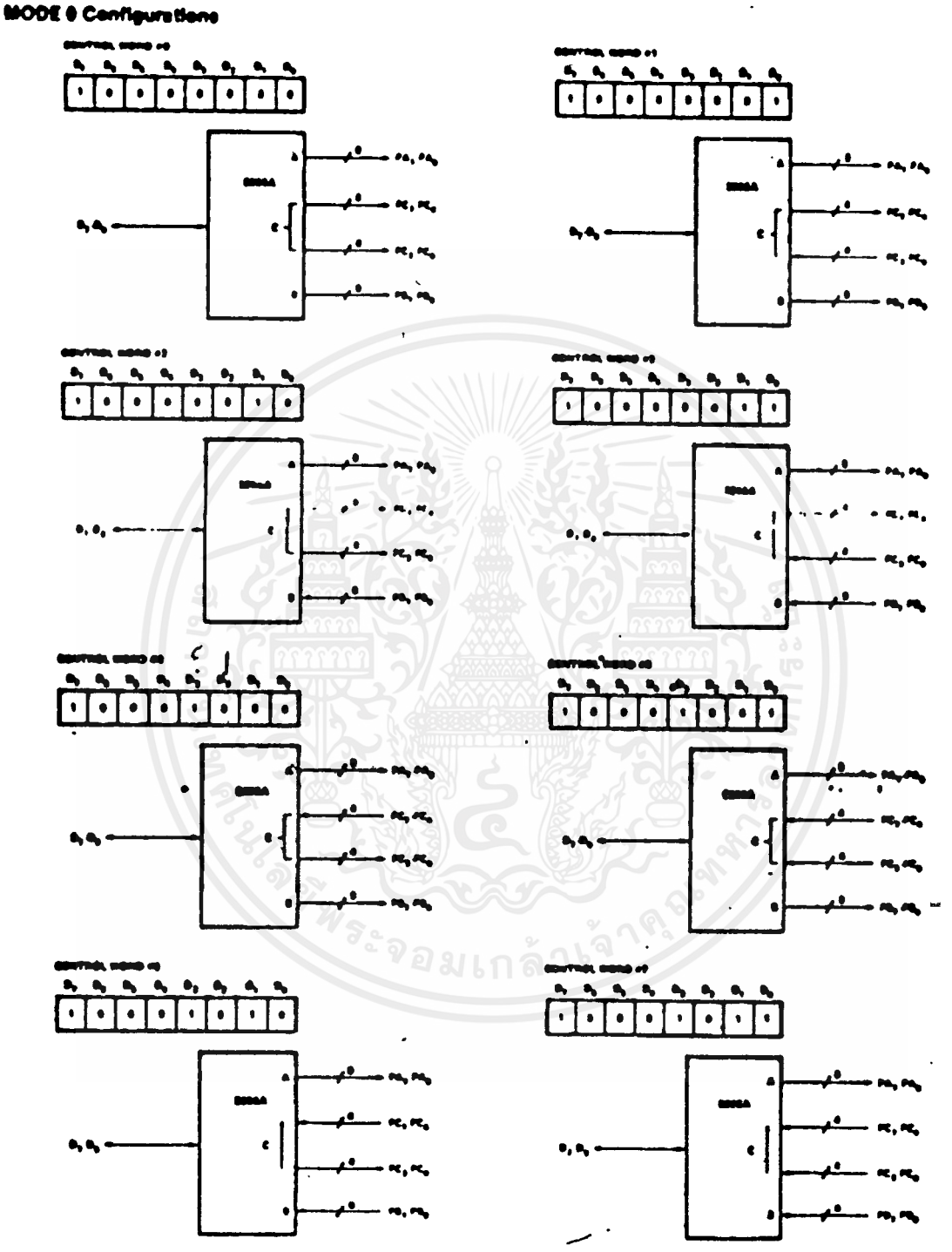
รูปที่ 3.4 ความหมายและโหมดต่าง ๆ ของรหัสควบคุม

ตารางที่ 3.5 ตารางแสดงสัญญาณแต่ละ PORT

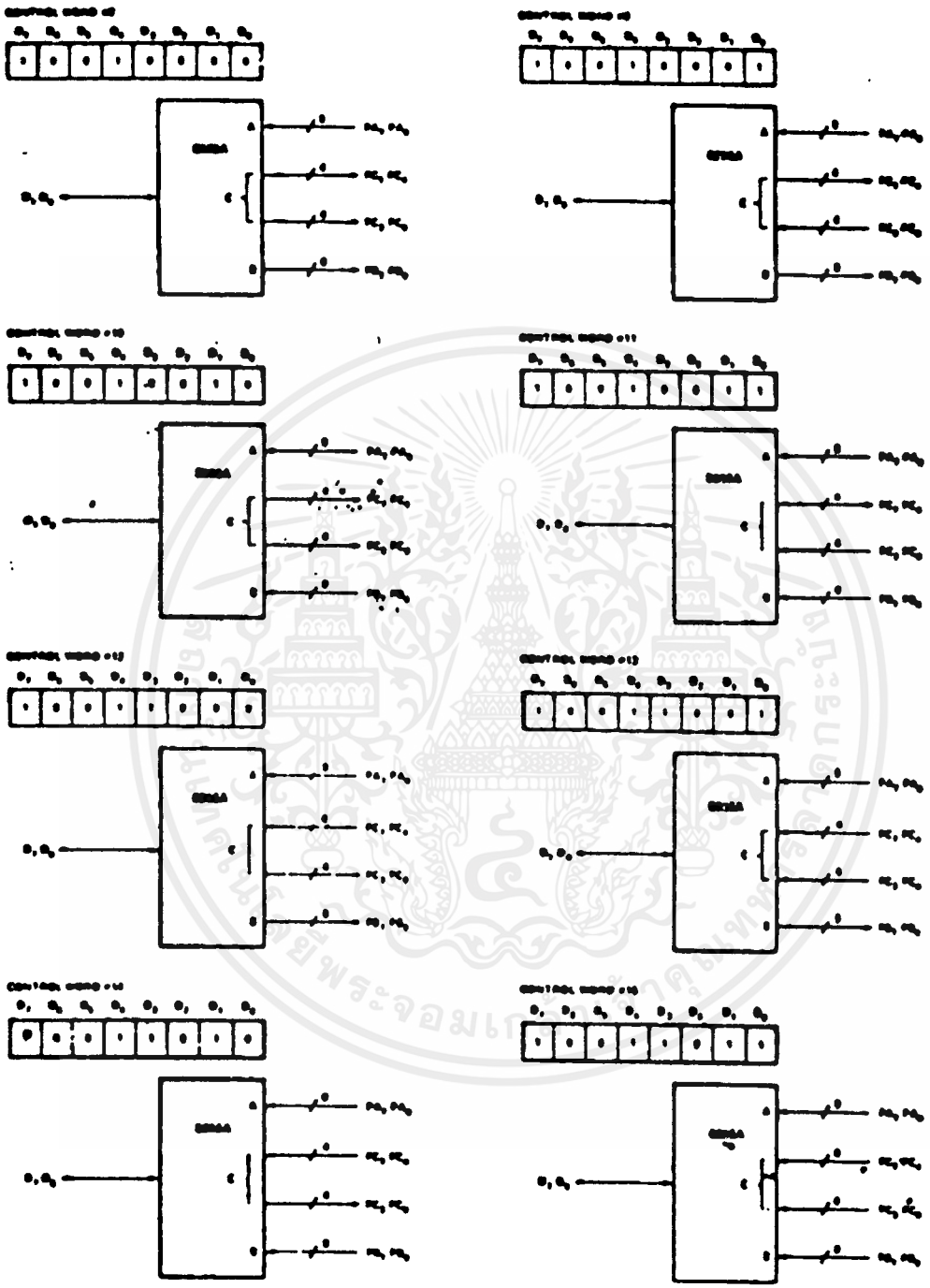
A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

(1) โหมด 0

โหมด 0 เป็นโหมดที่กำหนดให้พอร์ตทุกพอร์ต บนตัว 8255 เป็นพอร์ตอินพุต-เอาต์พุตแบบพื้นฐาน รูปแบบความเป็นไปได้จึงมีทั้งสิ้น 16 รูปแบบตามลักษณะของพอร์ต A,B,C บนและ C ล่างลักษณะควบคุมแต่ละรหัสควบคุมแต่ละแบบ จะเป็นดังรูปที่ 3.5 และ 3.6



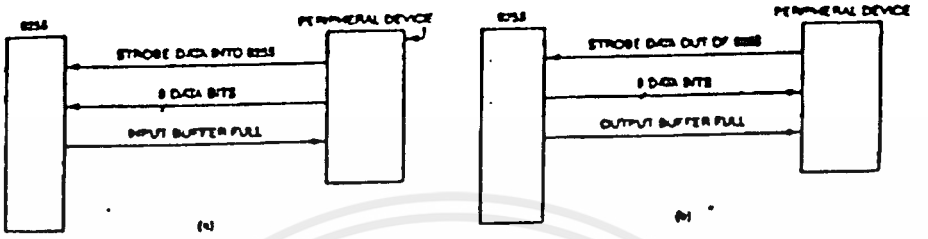
รูปที่ 3.5 แสดงลักษณะการใช้งานของ mode 0



รูปที่ 3.6 แสดงลักษณะการใช้งานของ mode 0 (ต่อ)

(2) โหมด 1

การทำงานของ 8255 ในโหมด 1 เป็นโหมดที่ทำให้อินพุท-เอาต์พุต มีการตรวจสอบสัญญาณ (HANDSHAKING) โดยใช้อินพุท-เอาต์พุตของพอร์ต A และ B เป็นหลัก และใช้พอร์ต C บนเป็นสัญญาณ handshake ของพอร์ต A ส่วนพอร์ต Cล่างเป็นสัญญาณ handshake ของพอร์ตB



รูปที่ 3.7 โครงสร้างของการจัดสัญญาณ HANDSHAKE

ตารางที่ 3.2 หน้าที่ต่างๆ ของพอร์ต C เป็นสัญญาณ handshake เมื่อ 8255 ทำงาน Mode 1

PC	IN	OUT
PC0	$INTR_B$	$INTR_B$
PC_1	IBF_B	OBF_B
PC2	STB_B	ACK_B
PC3	$INTR_A$	$INTR_A$
PC4	STB_A	VO
PC5	IBF_A	VO
PC6	VO	ACK_A
PC7	VO	OBF_A

แนวความคิดของการใช้ อินพุท-เอาต์พุตแบบ handshake ก็เพื่อให้มีการซิงโครไนซ์ระหว่างอุปกรณ์ภายนอกที่ทำงานได้ช้ากับการทำงานของคอมพิวเตอร์ที่ทำงานได้เร็ว เช่นเครื่องพิมพ์ทำงานได้ช้า เมื่อคอมพิวเตอร์ส่งตัวอักษรตัวแรกมาพิมพ์ เครื่องพิมพ์รับตัวอักษร และกำลังจะพิมพ์คอมพิวเตอร์ก็ส่งตัวที่ 2 และตัวที่ 3 ตามมา ทำให้การประมวลผลของอุปกรณ์เครื่องพิมพ์ทำงานไม่ทัน อาจทำให้ข้อมูลเสียหาย ดังนั้น เครื่องพิมพ์จึงส่งสัญญาณบอกคอมพิวเตอร์ว่าอย่าเพิ่งส่งมา เพราะยังไม่พร้อมรับ

ลักษณะของการรับส่งข้อมูล อินพุต-เอาท์พุตแบบมี handshake แสดงได้ดังรูป 3.7และ PA_0 - PA_7 เป็นอินพุตโดยมีพอร์ต C เป็นสัญญาณ handshake ดังแผนผังในรูปที่ 3.8

เมื่อโปรแกรม 8255 เป็นโหมด 1แล้วตัว 8255 จะให้พอร์ต C เป็นสัญญาณควบคุมโดยแต่ละบิตของพอร์ต เป็นไปตามที่กำหนดไว้แล้วดังตารางที่ 3.1 โดยปกติ 8255 ยังให้สัญญาณอินเตอร์รัพต์ของ 8255 จะเกิดขึ้นที่ขา PC_0 และ PC_3 โดยที่บัฟเฟอร์พร้อมแล้วและต้องการให้ซีพียูส่งอินพุตหรือเอาท์พุตมาที่บัฟเฟอร์ สัญญาณอินเตอร์รัพต์ที่เกิดขึ้น สังเกตว่าสัญญาณอินเตอร์รัพต์เป็นสัญญาณเอคทีฟลอจิก "1" ซึ่งตรงกับของ 8080 เมื่อใช้กับ IBM สัญญาณ INT ของ IBM จะรับด้วยลอจิก "0"

โครงสร้างการ handshake ของ 8255 แสดงด้วยสัญญาณทางไฟฟ้า ได้ดังรูปที่ 3.9 สังเกตว่าการทำงานของ 8255เกี่ยวกับสัญญาณ RD และ WR ซึ่งจะทำให้สัญญาณควบคุมเปลี่ยนไป

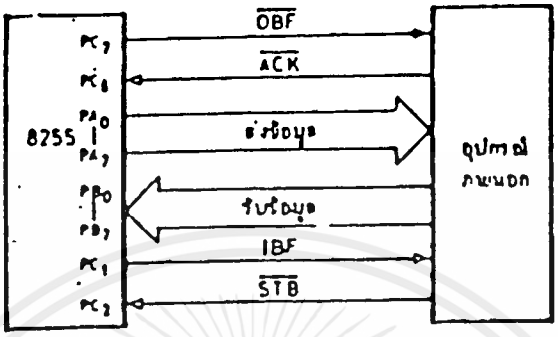
การตรวจสอบสัญญาณซึ่งกันและกันนี้ เป็นวิธีการรับส่งที่มีประสิทธิภาพ เช่น ในการอินพุต เมื่ออุปกรณ์ภายนอกต้องการส่งข้อมูลเข้าซีพียูก็จะส่งข้อมูลแบบขนานเข้ามา พร้อมทั้งสเตรม (STB) บอก 8255 ตัว 8255 จะนำข้อมูลนั้นไปเก็บไว้ในรีจิสเตอร์ภายในก่อน แล้วส่งสัญญาณตอบบอกว่าบัฟเฟอร์ยังเต็มอยู่นะ (IBF) อย่าเพิ่งส่งมาอีก ครั้นเมื่อซีพียูอ่านข้อความจากรีจิสเตอร์ไปแล้วส่วนของสัญญาณบัฟเฟอร์อินพุต (IBF)ก็จะบอกว่าว่างแล้วส่งมาให้อุปกรณ์ภายนอกก็จะส่งมาอีก

ในทำนองเดียวกัน สำหรับพอร์ตเอาท์พุต เมื่อซีพียูส่งข้อมูลออกมาทาง พอร์ตเอาท์พุตให้กับ 8255 ตัว 8255 ก็จะรับไว้ในรีจิสเตอร์ภายใน พร้อมทั้งส่งสัญญาณออกไปบอกอุปกรณ์ภายนอกว่าเอาท์พุตบัฟเฟอร์ของฉันมีข้อมูลอยู่นะ (OBF) มาอ่านเอาไปสิ อุปกรณ์ภายนอกเมื่อทราบและพร้อมจะอ่านก็จะส่งสัญญาณตอบรับ (ACK) พร้อมกับอ่านข้อมูลไปโดยสัญญาณ ACK จะมีความหมายว่าฉันอ่านข้อมูลของเธอไปแล้วนะ ตัว 8255 ก็ตอบกลับว่า บัฟเฟอร์ฉันว่างแล้วเธอรอก่อนนะจะมีข้อมูลใหม่ส่งมาให้อีก

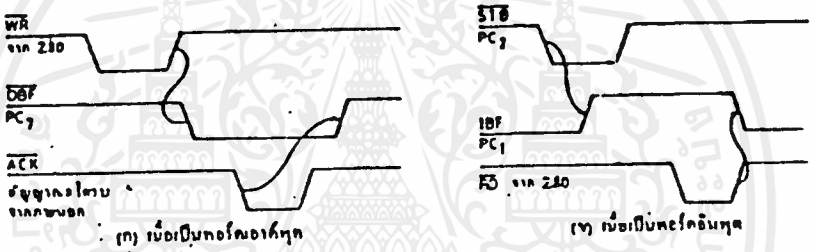
ในการที่จะมีโปรแกรมโหมด 1 นี้นั้นเราจะให้รหัสควบคุมเป็น 101 (I/0) 01(I/0)0 ในส่วน (I/0) หมายถึงถ้าเป็นอินพุตก็เป็น "1" เอาท์พุตก็เป็น "0" โดย I/0 ตัวแรกเป็นของพอร์ต A ตัวที่ 2 พอร์ต B เช่น ถ้าต้องการให้พอร์ต A เป็นเอาท์พุต และ พอร์ต B เป็นอินพุต เราจะใช้รหัสควบคุมเป็น 10100110 หรือ A6H

หากจะพิจารณาการทำงานของซีพียูจะเห็นว่า ทำอย่างไรจึงเขียน หรืออ่านพอร์ตได้ถูกต้องวิธีง่ายวิธีหนึ่งคือซีพียูจะคอยตรวจสอบสัญญาณของ 8255เช่นการที่เอาท์พุตซีพียูจะคอยอ่านพอร์ต C แล้วตรวจสอบบิต 7(OBF) หลังจากที่ส่งข้อมูลไปแล้ว ถ้าบิต 7 ยังเป็น "0" แสดงว่ายังไม่

ได้รับการสโตรบแล้วก็เป็น "1" แล้ว แสดงว่าอุปกรณ์ภายนอกรับข้อมูลไปแล้ว สำหรับการอินพุต คอยตรวจสอบจากสัญญาณ IBF ได้เช่นกันว่า มีข้อมูลใหม่เข้ามาหรือยังก็ตรวจสอบบิต PC_1 ของพอร์ต C



รูปที่ 3.8 การต่อ 8255 ในโหมด



รูปที่ 3.9 แผนผังเวลาการรับ/ส่งข้อมูลแบบ handshake

สรุปโหมด 1

(1) แบ่งออกเป็น 2 กลุ่มคือ กลุ่ม A และกลุ่ม B

(2) แต่ละกลุ่มประกอบด้วย DATA PORT 8 BIT และ 4 BIT HAND SHAKING

(3) สำหรับ 8 BIT DATA จะเป็นได้ทั้ง INPUT/OUTPUT แบบ LATCH ได้ทั้ง 2 กรณี

(4) ใช้ 4 BIT เป็น CONTROL,SATAUS ของ DATA 8 BIT

สัญญาณควบคุม ในการอินพุต (INPUT SIGNAL CONTROL DEFINTION)

- STD (STORBE INPUT)เป็นขาสัญญาณอินพุต ซึ่งเมื่อเป็น"LOW"ก็จะแสดงแสดงว่าข้อมูลได้ถูกส่งเข้าไปใน LATCH เอาไว้
- IBF (INPUT BUFFER FULL F/F) เป็นขาเอาต์พุต เมื่อเป็น "HIGH"ก็แสดงว่าข้อมูลได้ถูกส่งเข้าไปใน INPUT แล้วโดยที่ IBF จะถูกเซ็ทขาลงของสัญญาณ STB (FALLING EDGE OF RD INPUT) และถูกรีเซ็ทโดยสัญญาณขาขึ้นของสัญญาณ RD (RISING OF RD INPUT)

- INTR (INTERRUPT REQUEST) เป็นสัญญาณเอาต์พุตซึ่งใช้ INTERRUPT CPU เมื่ออุปกรณ์ I/O ต้องการขอทำการ INTERRUPT กับ CPU ขา INTR จะถูกเซ็ทโดยสัญญาณขาขึ้นของ STB ถ้า IBF เป็น "1" และ INTR เป็น "1" และ INTR จะถูกรีเซ็ทโดยสัญญาณ RD ซึ่งแสดงว่า CPU ยอมรับการ INTERRUPT แล้ว
- INTE A ถูกควบคุมด้วย BIT SET/RESET ของ PORT C4
- INTE E ถูกควบคุมด้วย BIT SET/RESET ของ PORT C2

สัญญาณควบคุมในการดีเอาต์พุต (OUTPUT CONTROL SIGNAL DEFINITION)

- OBF (OUTPUT BUFFER FULL F/F) เป็นขาสัญญาณเอาต์พุตจะเป็น "LOW" เพื่อแสดงว่าให้มีการ WRITE ข้อมูลออกมาที่พอร์ตที่กำหนดของ CPU มายัง 8255 ได้ OBF F/F จะเซ็ทเป็น "LOW" โดย RISING EDGE ของ WR INPUT SIGNAL
- ACK (ACKNOWLEDGE INPUT) เป็นสัญญาณอินพุต จะเป็น "LOW" เพื่อแสดงว่าข้อมูลที่พอร์ต A หรือพอร์ต B ของ 8255 ได้ถูกรับไปแล้วซึ่งเป็นสัญญาณตอบรับจากอุปกรณ์ภายนอกที่พอร์ต
- INTR (INTERRUPT) เป็นสัญญาณเอาต์พุตซึ่งจะเป็น "HIGH" เพื่อ INTERRUPT CPU เมื่ออุปกรณ์เอาต์พุตได้รับข้อมูลจาก CPU แล้วขา INTR จะเซ็ทที่ RISING EDGE ของ ACK ถ้า OBF เป็น "HIGH" INTE จะ

  รีเซ็ทที่ FALLING EDGE ของสัญญาณ WR

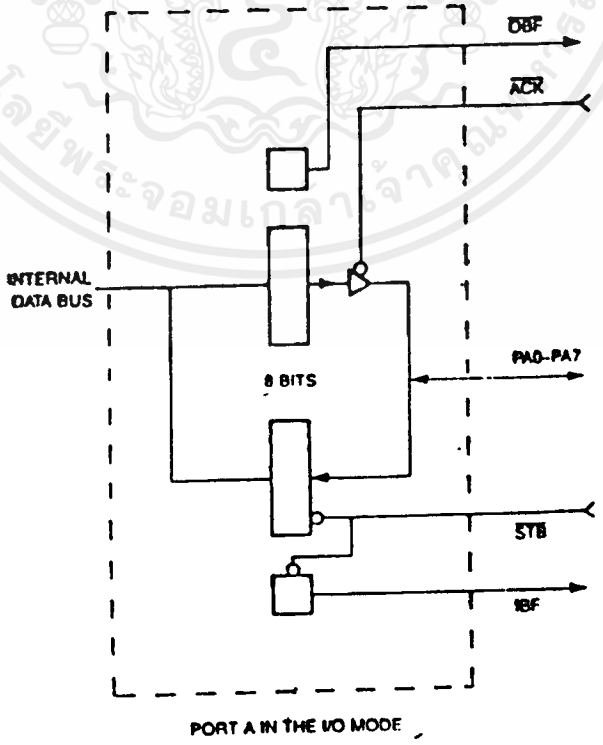
(3) โหมด 2

8255 ยังมีโหมดการทำงานอีกโหมดหนึ่งคือโหมด 2 ซึ่งทำได้เฉพาะพอร์ต A ในโหมดนี้ 8255 จะใช้พอร์ต A ทำหน้าที่เป็นพอร์ตแบบสองทิศทางคือสามารถเป็นได้ทั้งอินพุตและเอาต์พุตโดยโครงสร้างของพอร์ต A ทั้งอินพุตมี handshake ทั้งคู่ส่วนพอร์ต C ที่เหลือจะทำหน้าที่เป็นสัญญาณตรวจสอบแต่ละขาดังตารางที่ 3.3

ตารางที่ 3.3 หน้าที่ของพอร์ต C ในโหมด 2

PORT C LINE	DEFINTION
PC0	I/O
PC1	I/O
PC2	I/O
PC3	ITRa
PC4	STBa
PC5	IBFa
PC6	ACKa
PC7	OBFa

โครงสร้างของพอร์ต A ที่ทำงานแบบสองทิศทาง เป็นดังรูปที่ 3.10



รูปที่ 3.10 โครงสร้างของพอร์ต A ที่ทำงานแบบพอร์ต 2 ทิศทาง

สังเกตว่า เมื่อโปรแกรมพอร์ต A เป็นโหมด 2 แล้ว พอร์ต B จะต้องโปรแกรมเป็นโหมด 0 หรือโหมด 1 ก็ได้ ซึ่งการทำงานได้เยกอิสระอีก ในการใช้งานพอร์ตแบบ 2 นี้ทางนี้มาใช้ได้กับงานบางประเภท เช่น ใช้ในการรับส่งข้อมูลของพอร์ตมาตรฐานบางประเภทเช่น IEEE 488 หรือใช้เชื่อมโยงระหว่างคอมพิวเตอร์ในการรับส่งข้อมูลสลับกันไปมาได้

สรุปโหมด 2 BASIC FUNCTION DEFFINITIONS

(1) ใช้ GROUP A เพียง GROUP เดียว

(2) มี 8 BIT PORT ใช้งานเพียงพอร์ตเดียวเป็น BI-DIRECT ION BUS PORT (PORT A) และมีพอร์ตสำหรับควบคุมอีก 5 BIT คือ พอร์ต C

(3) ทั้งอินพุตและเอาต์พุตเป็นแบบ LATCH SIGANAL ได้

(4) มีบิทสำหรับควบคุม 5 BIT(พอร์ต C ใช้สำหรับ CONTROL และ STATUS สำหรับ 8 BIT BI-DIRECTION BUS PORT (PORT A)

สัญญาณควบคุมในกรณี MODE 2BI-DIRECTION BUS I/O CONTROL SIGNAL DEFINTION

INTR (INTERRUPT REQUEST) เป็นขาสัญญาณ OUTPUT "HIGH" เมื่อต้องการทำงานเป็นอินพุตและเอาต์พุต

กรณีที่ทำงานเป็น OUTPUT

- OBF (OUTPUT BUFFER FULL) OUTPUT SIGNAL"LOW" เมื่อขานี้เป็น LOW แสดงว่าขณะนี้ 8255 ได้รับข้อมูลจากพอร์ต A แล้ว
- ACK (ACKNOWLEDGE) INPUT SIGNAL จะเป็น "LOW" เพื่อบอกให้รู้ว่า ขณะนี้ได้มีข้อมูลมาอยู่ที่พอร์ต A แล้ว พร้อมที่จะส่งข้อมูลออกไปทางพอร์ต A และจะเซ็ทเอาต์พุตเป็นสภาวะ "HIGH IMPLEDANCE" เมื่อเป็น "HIGH"
- INTE1 (THE INTERRUPT F/F ASSOCIATED WITH OBF) เป็นฟลิบฟลอภายในซึ่งจะสร้างสัญญาณ INTR A โดยทำงานร่วมกับขา OBF ซึ่งสามารถเซ็ทหรือรีเซ็ทได้ที่ PC6

ในกรณี INPUT OPERATION

- STB (STOBE INPUT) เป็นสัญญาณอินพุตเป็น "LOW" เพื่อมีการ LATCH ข้อมูลเอาไว้ใน INPUT LATCH
- IBF (INPUT BUFFER FULL F/F) เป็นขาสัญญาณเอาต์พุต จะเป็น "HIGH" เพื่อแสดงให้รู้ว่าได้กระทำการโหลดข้อมูลไว้ใน INPUT LATCH เรียบร้อยแล้ว

- INTE 2 (THE INTE F/F ASSOCIATED WITH IBF)เป็นฟลิปฟลอบภายในซึ่งจะสร้างสัญญาณ INTE A โดยทำงานร่วมกับขา IBFโดยสามารถเซ็ทหรือรีเซ็ทที่ขาPC4
- INTR (INTERRUPT REQUEST) เป็นขาสัญญาณเอาต์พุตจะเป็น HIGH เมื่อต้องการทำการอินเตอร์รัพต์กับ CPU

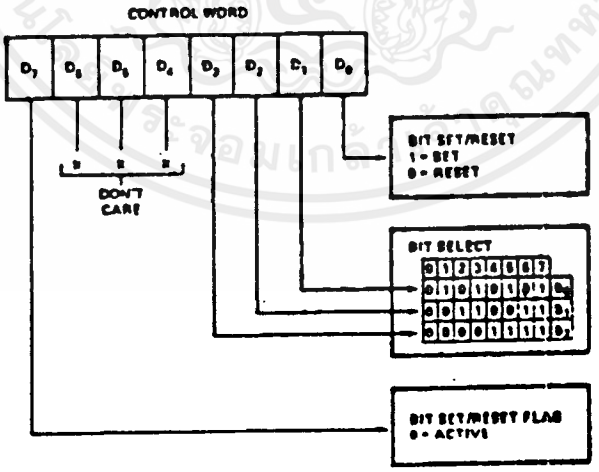
3.1.4 การเซ็ท/รีเซ็ทบิต

บิตทั้งสามของพอร์ต C สามารถที่จะเซ็ทหรือรีเซ็ทได้โดย CONTROL WORD ที่ส่งมาจาก CPU เพื่อทำการควบคุมให้พอร์ต C ทำตาม CONTROL WORD ที่ส่งมาจากพอร์ต C จะใช้เป็นฟังก์ชันการควบคุมอินเตอร์รัพต์ (STATUS/CONTROL INTERRUPT CONTEROL FUNCTIONS)

ในการทำงานของโหมด 1 หรือโหมด 2 สัญญาณควบคุม สามารถส่งผ่าน INTERRUPT REQUEST เข้าไปยัง CPU โดยสัญญาณนี้สร้างขึ้นโดยพอร์ต C เพื่อจะไปทำการ INHIBTTED หรือ ENABLED โดยการเซ็ทหรือการรีเซ็ทขา INTE F/F ของ CPU โดยสัญญาณ BIT SET/RESET ของ พอร์ต C

(BIT SET) ตอบสนองการอินเตอร์รัพต์ (INTERRUPT ENABLE)

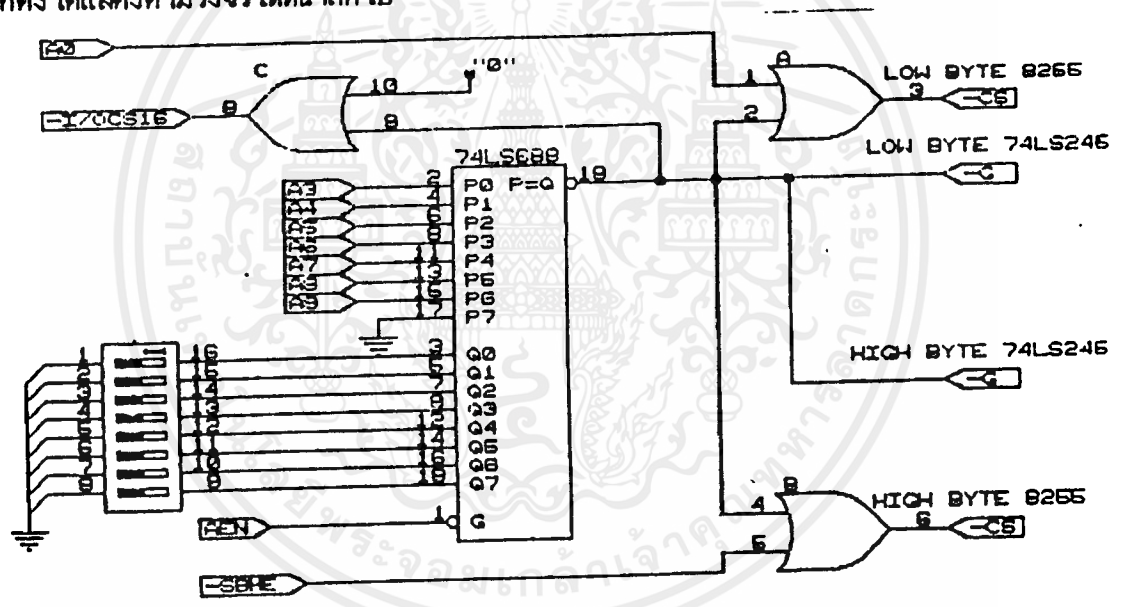
(BIT RRESSET)ไม่ตอบสนองการอินเตอร์รัพต์ (INTERRUPT DISABLE)



รูปที่ 3.11 แสดงการเซ็ท-รีเซ็ทบิต

3.2 ดีโค้ดเดอร์

IC ที่เรานำมาทำการดีโค้ด เป็น IC74LS688 การดีโค้ดสัญญาณต่างๆ จาก SLOP ของ IBM PC/AT มาใช้เลือกแอดเดรสให้แก่ 8255 สัญญาณที่จะนำมาทำการดีโค้ดก็มี A0-A9 แต่ 8255 ต้องการสัญญาณจาก A0 และ A1 ไปควบคุมรีจิสเตอร์ภายในตัวมันดังนั้นสัญญาณแอดเดรสที่เรานำมาใช้จึงมี A3-A9 ส่วน A1 และ A2 เราจะนำไปต่อเข้ากับ 8255 โดยตรงโดยที่เราได้นำสัญญาณ A1 จาก SLOT ต่อเข้ากับ A0 และสัญญาณ A2 จาก SLOT ต่อเข้ากับขา A1 ของ 8255 ที่เราไม่นำสัญญาณ A0 จาก SLOT ต่อเข้าโดยตรงกับ 8255 ก็เพราะว่าเราต้องนำสัญญาณ SBHE และ สัญญาณจาก SLOT ที่เราได้นำมาการทำดีโค้ดยังมีสัญญาณ AEN และ IOS16 อีกดังได้แสดงตามวงจรให้หน้าถัดไป



รูปที่ 3.12 การทำงานของวงจรดีโค้ดเดอร์

ก่อนที่จะอธิบายถึงการทำงานของวงจร ดีโค้ดเดอร์ เราจะต้องรู้ก่อนว่าการดีโค้ดของ IBM PC/AT แยกออกเป็น 2 แบบ คือการดีโค้ดหน่วยความจำและการดีโค้ดพอร์ตการดีโค้ดหน่วยความจำเราจะไม่ขอกล่าวถึง เพราะอินเตอร์เฟสการ์ดของเราต้องการติดต่อกับ I/O พอร์ตเท่านั้นและการและการดีโค้ดพอร์ตซึ่งจะใช้ในการเลือกแอดเดรสที่จะใช้กับแผ่นวงจรของเราดูได้ในช่วงแอดเดรส 300H-31FH จากตำแหน่งแอดเดรสที่จะใช้การดีโค้ดนั้นซึ่งในที่นี้จะใช้แอดเดรส 10 ตำแหน่งคือ A0-A9 เข้ามาทำการดีโค้ด ส่วนตำแหน่ง A10-A15 จะไม่ได้นำมาใช้งานแต่ค่าแอดเดรสเหล่านี้ ยังคงเปลี่ยนแปลงตามค่าแอดเดรสที่กำหนดไว้ใน คำสั่ง OUT หรือ IN เพียงแต่ไม่ได้ถูกนำมาใช้ดีโค้ดร่วมกับ A0-A9 เท่านั้น

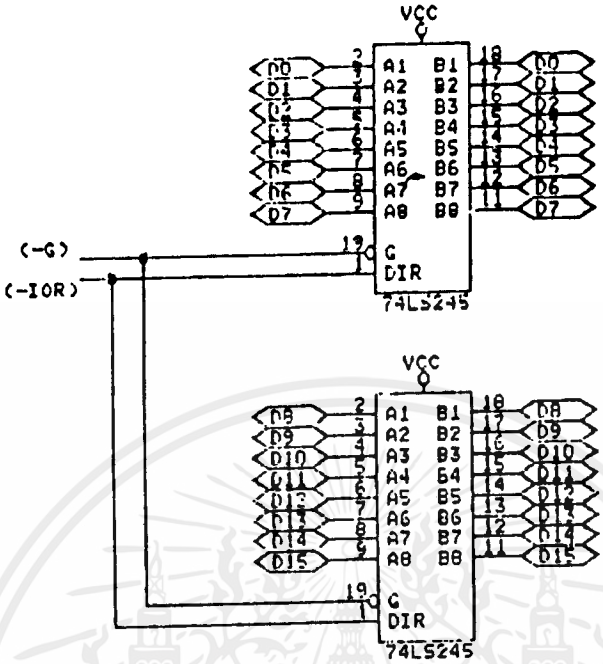
จากรูปที่ 3.12 เป็นการแสดงวงจรดีโคดของ การ์ดอินเตอร์เฟส 8255 จะเห็นได้ว่านอกจากได้เอาสัญญาณแอดเดรส A3-A9 ,AEN แล้วยังเอาสัญญาณ IOCS16 ,A0 และ SBHE เข้ามาดีโคดด้วย คือ สัญญาณ A0 จะใช้ร่วมกับ SBHE ซึ่ง A0 จะเป็นตัวกำหนดการรับและส่งข้อมูล 8 บิตล่างคือ (D0-D7) ส่วนสัญญาณ SBHE จะเป็นตัวกำหนดการรับส่งข้อมูล 8 บิตบน คือ (D8-D15) ส่วนสัญญาณ IOCS16 เป็นตัวบอกให้มีการรับและส่งข้อมูลแบบ 16 บิต

การทำงานของ IC74LS688 นั้นจะเป็นลักษณะของวงจรเปรียบเทียบซึ่งรายละเอียดต่าง ๆ ได้อธิบายไว้ก่อนหน้านี้แล้วการทำงานของ IC ตัวนี้จะเปรียบเทียบสัญญาณ อินพุตซึ่งแยกเป็น 2 ส่วนคือ P0-P7 และ Q0-Q7 สัญญาณเอาท์พุตจะเป็น "0" อินพุต Q เราจะต่อเข้ากับ DIP SW และเราจะเซ็ทค่าที่ DIP SW ไว้ที่พอร์ต 306 เมื่อมีสัญญาณแอดเดรส 306H มาที่อินพุต P ของ IC 688 ก็จะมีสัญญาณ LOW ออกมาที่ขา 19(CS)และสัญญาณนี้จะไปเป็นสัญญาณ CS ให้กับวงจรบัฟเฟอร์

ตารางที่ 3.4 แสดงแอดเดรส ของ พอร์ต

แอดเดรส	
# 300	พอร์ต A LOW BYTE
# 301	พอร์ต A HIGH BYTE
# 302	พอร์ต B LOW BYTE
# 303	พอร์ต B HIGH BYTE
# 304	พอร์ต C LOW BYTE
# 305	พอร์ต C HIGH BYTE
# 306	CONTROL
# 307	CONTROL

3.3 บัฟเฟอร์



รูปที่ 3.13 วงจรบัฟเฟอร์

ในวงจรการอินเตอร์เฟส 8255 นี้เราจะใช้ IC 74LS245 เป็นบัฟเฟอร์ 2 ทิศทาง สัญญาณ DATA ของ D0-D7 และ D8-D15 ซึ่งต่อออกมาจาก SLOT ของ IBMซึ่งการทำงานของ IC จะอธิบายดังต่อไปนี้

วงจรบัฟเฟอร์ จะทำหน้าที่เหมือนกับสวิทช์ ปิด-เปิด ให้ต่อสัญญาณถึงกันดังแสดงในรูปที่ 3.13 จะเห็นได้ว่า IC 74LS245 จะทำงาน 2 ทิศทาง เป็นทั้งรับและส่งข้อมูลโดยมีสัญญาณควบคุม 2 สัญญาณอีนาเบิล (-G) และสัญญาณควบคุมทิศทาง (DIR)

จากวงจรที่แสดงในรูปบัสข้อมูล D0-D7 จาก SLOT ต่อเข้ากับ A1-A8 ของ 74LS245 ตัวที่เป็น LOW BYTE และขา B1-B8 ต่อเข้ากับขาสัญญาณ D0-D7 ของ 8255 ตัวที่เป็น LOW BYTE และบัสข้อมูล D8-D15 จาก SLOT ต่อเข้ากับขา A1-A8 ของ 74LS245 ตัวที่เป็น HIGH BYTE ขาสัญญาณ B1-B8 ต่อเข้ากับสัญญาณ D0-D7 ของ 8255 ตัวที่เป็น HIGH BYTEขาสัญญาณอีนาเบิล (-G) ของ IC 74LS245 ตัวที่เป็น LOW BYTE และขาสัญญาณอีนาเบิล (-G) ของ 74LS245 ตัวที่เป็น HIGH BYTE จะถูกต่อกับสัญญาณ (-G) ขา 19 ของ 688 ส่วนขาสัญญาณ DIR ขา 1 ของ 74LS245 ตัวที่เป็น HIGH BYTE และ LOW BYTE จะต่อเข้าด้วยกันและต่อเข้ากับขาสัญญาณ IOR จาก SLOT การทำงานของวงจรบัฟเฟอร์ นี้เกิดขึ้นเมื่อขาสัญญาณอีนาเบิลทำงานนั่นคือเมื่อมีสัญญาณแอดเดรส ที่ 306 วงจรถอดรหัสจะทำงาน และจะมีสัญญาณมาชักให้บัฟ

74LS245 ทำงานด้วย ส่วนทิศทางของข้อมูลจะส่งข้อมูลหรือผ่านขึ้นอยู่กับ สัญญาณ DIR ดังนี้ คือ

1 สัญญาณ DIR เป็นลอจิก "1" ข้อมูล D0-D15 จาก SLOT จะไปปรากฏที่ขาสัญญาณ D0-D7 ของ 8255 ทั้ง 2 ตัว คือทั้ง LOW BYTE และ HIGH BYTE

2 สัญญาณ DIR เป็นลอจิก "0" ข้อมูลที่ขาสัญญาณ D0-D7 จาก 8255 ทั้ง 2 ตัวจะไปปรากฏที่บัสข้อมูล D0-D15 ของ SLOT

3.4 เอดีซี

การแปลง สัญญาณอานาล็อกเป็นสัญญาณดิจิตอลเป็นส่วนประกอบที่มีความสำคัญในระบบข้อมูลในส่วนนี้ จะพิจารณาการแปลงสัญญาณอานาล็อกเป็นสัญญาณดิจิตอล วงจรการแปลงสัญญาณ, วงจรเลือกและการเชื่อมโยงข้อมูลของ ADCs สุดท้ายจะพิจารณาการออกแบบ 16 ช่องของระบบข้อมูลจาก IBM PC

3.4.1 ADC Essentials

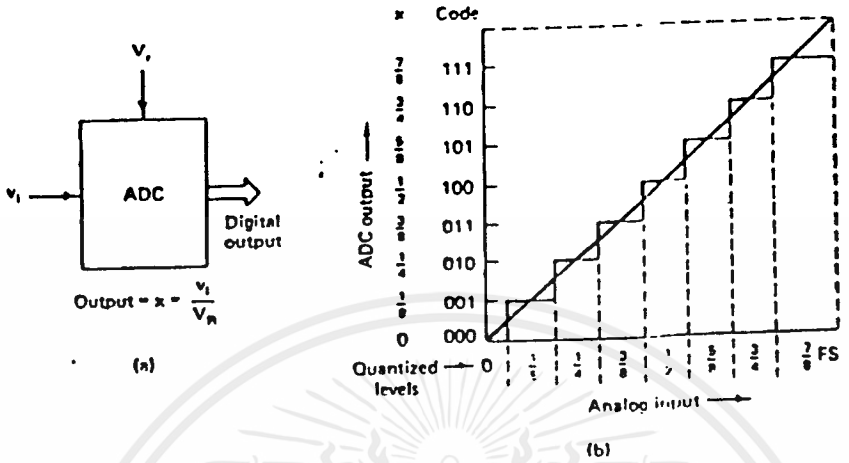
ในส่วนนี้จะแนะนำถึงหลักการพื้นฐาน การพิจารณาอินพุตและเอาต์พุตของ ADC Basic Input/Output Relationship การแปลงอานาล็อกเป็นดิจิตอลเป็นการทำงานที่มีอัตราส่วนที่มีความสำคัญ สัญญาณอานาล็อกที่อินพุต (V_1) จะแปลงในรูปเศษส่วน X โดยการเปรียบเทียบกับสัญญาณอ้างอิง Vr จากรูปที่ 4.1(a) แสดงภาพประกอบที่มีความสัมพันธ์กัน ถ้าการเปลี่ยนทางเอาต์พุตรหัส ซึ่งประกอบไปด้วย n บิต ในการพิจารณาจำนวนเอาต์พุตที่ระดับคงที่ 2^n จากหนึ่งถึงหนึ่งที่ตรงกันข้ามอินพุตที่เหมือนกันระดับที่ต่ำที่สุดในระดับค่าของสัญญาณอานาล็อกกับ 2ระดับของรหัสความแตกต่าง ดังนั้นเราเรียกว่า บิตที่ต่ำที่สุดว่า LSB ดังนั้น

$$Q = LSB = \frac{FS}{2^n}$$

ขณะที่ Q คือจำนวนของ LSB ซึ่งหมายถึง จำนวนค่าของสัญญาณอานาล็อกของ LSB และ FS คือค่าฟูลสเกลของระดับสัญญาณอานาล็อกที่อินพุต

ค่าของสัญญาณอานาล็อกจะแสดงให้เห็นโดยเปรียบเหมือนกับรหัสดิจิตอลโดยทั่วไปจะมีค่าอยู่ตรงกลางช่วงสัญญาณระดับความแตกต่างซึ่งมีค่าเท่ากับ + (1/2)LSB และยังคงแสดงให้เห็นได้โดยรหัสทางเอาต์พุตจากรูป 3.14(b) แสดงให้เห็นถึงความสัมพันธ์ทางการเปลี่ยนแปลงทางอุดมคติของ ADC ที่มีขนาด 3 บิต ขนาดของ LSB มีค่าเท่ากับ (1/8)ร และช่วงอินพุตซึ่งมีจำนวน

ที่เปลี่ยนในระดับ 0 ถึง (7/8)FS ขนาดสูงสุดของเอาท์พุตจะมีเลขไบนารี่เท่ากับ 111 ซึ่งไม่เต็มสเกลเต่จะมีค่าเท่ากับ(7/8)FS



รูปที่ 3.14 Anaiog-to-digtal converter

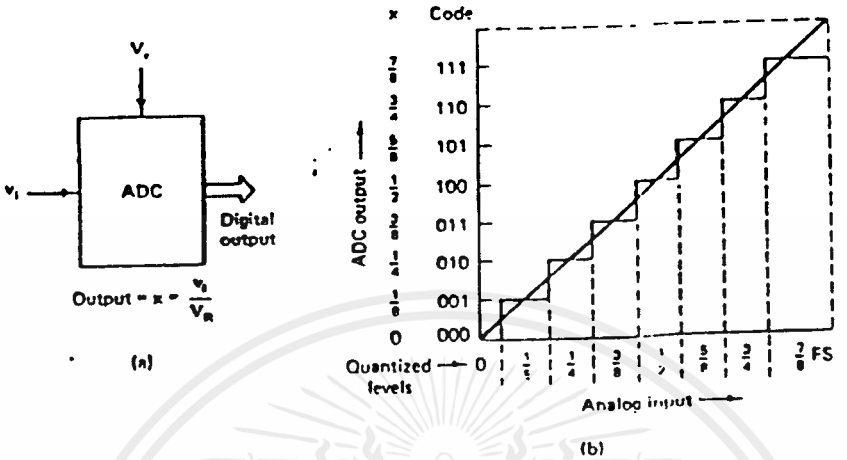
(a) Basic relationship

(b) ideal characteristics of a 3-bit ADC

Conveerter Errors

ในทางปฏิบัติการแปลงสัญญาณอาจจะผิดเพี้ยนจากทางอุดมคติจากรูป 3.14(b) offset หรือ zero error หาได้โดยขณะที่ค่าสัญญาณอนาล็อกซึ่งไม่เพียงพอที่จะผ่านศูนย์ สเกลของวงจรสมมูลย์อาจจะแตกต่างจากทางอุดมคติโดยทั่วไปสเกลหรือเกนที่คลาดเคลื่อนรูปที่ 3.15(b)จาก ADCs จะมีอยู่ 2 ชนิดที่ไม่เป็นเชิงเส้น Integral linearity error หาได้จากค่าสูงสุดของการผิดเพี้ยนของวงจรสมมูลย์จากเส้นทางในอุดมคติเมื่อ offset และ error gain คือ ศูนย์จากรูป 3.15(c) ความแตกต่างของ linearity error คือการเฉลี่ยของปริมาณจากทางอุดมคติของค่าสัญญาณอนาล็อก หมายเหตุถ้าความแตกต่างของการไม่เป็นเชิงเส้นเกินกว่า 1 LSBเอาท์พุตอาจจะมีความผิดพลาดซึ่งแสดงในรูปที่ 3.15(d)

ที่เปนอนาในระดับ 0 ถึง (7/8)FS ขนาดสูงสุดของเอาต์พุตจะมีเลขไบนารี่เท่ากับ 111 ซึ่งไม่เต็มสเกลเต่จะมีค่าเท่ากับ(7/8)FS



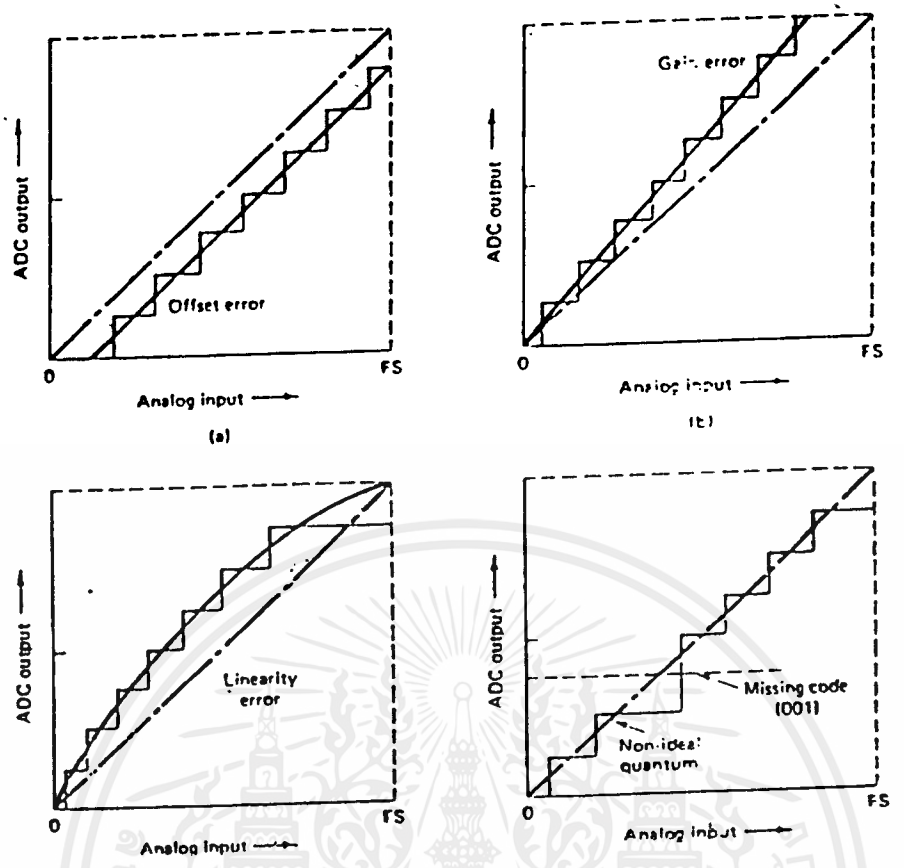
รูปที่ 3.14 Anaiog-to-digtal converter

(a) Basic relationship

(b) ideal characteristics of a 3-bit ADC

Conveerter Errors

ในทางปฏิบัติการแปลงสัญญาณอาจจะผิดเหจากทางอุดมคติจากรูป 3.14(b) offsetหรือ zero error หาได้โดยขณะที่ค่าสัญญาณอานาล็อกซึ่งไม่เพียงพอที่นำจะผ่านศูนย์ สเกลของวงจรสมมูลย์อาจจะแตกต่างจากทางอุดมคติโดยทั่วไปสเกลหรือเกนที่คลาดเคลื่อนรูปที่ 3.15(b)จาก ADCs จะมีอยู่ 2 ชนิดที่นำเป็นเชิงเส้น Integral linearity error หาได้จากค่าสูงสุดของการผิดเหของวงจรสมมูลย์จากเส้นทางในอุดมคติเมื่อ offset และ error gain คือ ศูนย์จากรูป 3.15(c) ความแตกต่างของ linearity error คือการเฉของปริมาณจากทางอุดมคติของค่าสัญญาณอานาล็อก หมายเหตุถ้าความแตกต่างของการไม่เป็นเชิงเส้นเกินกว่า 1 LSBเอาต์พุตอาจจะมีความผิดพลาดซึ่งแสดงในรูปที่ 3.15(d)



รูปที่ 3.15 Conerter errors

(a) offset error

(b) Gain error

(c) Integral linearity error

(d) Differential linearity error

Converter Reslution

ความหมายของส่วนของการเปลี่ยนหาได้จาก ขณะที่เปลี่ยนขนาดเล็กน้อยต้องการในสัญญาณอนาล็อกที่อินพุตของ ADC เอาต์พุตที่ได้อยู่ในรูปของรหัสซึ่งมีค่าๆ หนึ่งโดยทั่วไปผลที่ได้จากการเปลี่ยนทางอุดมคติ และผลของการเปลี่ยนแปลงของความจุ จะต้องมากกว่าค่าความจริงในทางปฏิบัติมันสามารถเสตง เบอร์เซนต์ของฟูลสเกลในช่วงอินพุตที่มีค่าเป็นมิลลิวอลท์ และทั้งหมดพร้อมกันในขณะการเปลี่ยนของจำนวนบิต

Converter Accuracy

ค่าความเป็นจริงของ ADCsหาได้โดยความแตกต่างระหว่างค่าของแรงดันอินพุตจริงและการลงมุติ ป้ายหักฟูลสเกลของรหัสเบนารี่ทางเอาท์พุต เรียกว่า ค่าที่มีความแน่นอน เมื่อแรงดันมีค่าจริงมันมากกว่าความเกี่ยวพันธ์ของสัญญาณอานาล็อกขนาดบิตของ LSB เรียกว่าความสัมพันธ์ของความจริงในส่วนนี้ผลรวมของค่าผิดพลาดจากการเปลี่ยนทั้งหมด ประกอบด้วยความผิดพลาดของ quantization error การแปลงที่ผิดพลาดโดยทั่วไปจะมีขนาดความผิดพลาดเท่ากับ LSB ในการแปลงค่าสัญญาณต้องการค่าของ offset หรือการปรับเกณฑ์ด้วยตัวเอง ADC0816 แต่ละตัวผลรวมของค่าผิดพลาดขึ้นอยู่กับคุณลักษณะของตัว เมื่อการคิดค่าความผิดพลาดแยกออกไป จะได้ง่ายต่อการคำนวน

Conversion Time and Conerter Throughput Rate

หลักการสั่งให้ทำงาน ADC ต้องการเวลาๆ หนึ่งในการเปลี่ยน t_c ก่อน ก่อนการแปลงต้องมีการเตรียมค่าข้อมูลของเอาท์พุต แรงดันอินพุตจะมีการแปลงระหว่างกระบวนการเปลี่ยนที่ไม่แน่นอนให้เกิดเอาท์พุต ความเที่ยงตรงในการเปลี่ยนแปลงเป็นค่าจริงเท่านั้นดังนั้นการแปลงขนาด n บิตจะใช้เวลาในการแปลงคือ t_c

$$\left[\frac{dv}{dt}\right]_{max} < \frac{FS}{2^n t_c}$$

จากตัวอย่างในการพิจารณา สัญญาณอินพุตรูปซายน์ที่มีแอมปลิจูด A และความถี่ f จะถูกเปลี่ยนโดยขนาด 8 บิตซึ่งเปลี่ยนจากค่าของเวลาที่ 100 μS:

$$V_1 = A \sin (2\pi ft)$$

อัตราการเปลี่ยนของสัญญาณอินพุตหาได้จาก

$$\frac{dv_1}{dt} = 2\pi fA \cos (2\pi ft)$$

และอัตราการเปลี่ยนสูงสุด หาได้จาก

$$\left[\frac{dv_1}{dt}\right]_{max} = 2\pi fA$$

ถ้าค่าฟูลสเกล FS เท่ากับ 2A ค่าพีคทูพีคของรูปซายน์ เราได้

$$2\pi fA < \frac{2A}{2^n t_c \pi}$$

$$f < \frac{1}{2^n t_c \pi}$$

$$f_{max} = \frac{1}{2^n t_c \pi} = 12.4 \text{ Hz}$$

ดังนั้นความสัมพันธ์ของสัญญาณซายน์เราจะกำหนดที่ความถี่ค่าสุดเท่ากับ 12.4 Hz เราให้ค่าความแตกต่างโดยใช้ วงจรแซมปลิ้งแอนด์โฮล (S/H) ระหว่างอินพุทและ ADC วงจร S/H คือ วงจรการสุ่มตัวอย่างสัญญาณอานาล็อกอินพุทอย่างรวดเร็ว และความเที่ยวพันธ์คงที่ ขณะที่ ADC แปลงสัญญาณในการคำนวนของอัตราส่วนของการแปลงที่ยอมรับได้ของแรงดันอินพุตคือ เวลาที่หน่วงไว้ เรียกว่า t_m เวลาหน่วงที่เกิดขึ้นในวงจร S/H ระหว่าง เวลาเก็บค่าชั่วขณะซึ่งใช้เวลาเพียง เล็กน้อยเป็นนาโนวินาที ถ้าใช้วงจร S/H เวลาที่ใช้จะมีค่า 20 นาโนวินาที และสามารถคำนวนความถี่ที่ยอมรับได้มีค่า

$$f_{max} = \frac{1}{2^8 t_m \pi} = 62.17 \text{ Hz}$$

ค่าของการแปลงสัญญาณที่ดีที่สุด คือ 100 ns ค่าของ f_{max} สามารถเพิ่มขึ้นจาก 100 ns ถ้า S/H เพื่อที่จะเก็บค่าไว้ชั่วขณะค่าที่ไม่แน่นอนในการคำนวนของ f_{max} ช่วง เวลาเก็บค่าชั่วขณะ

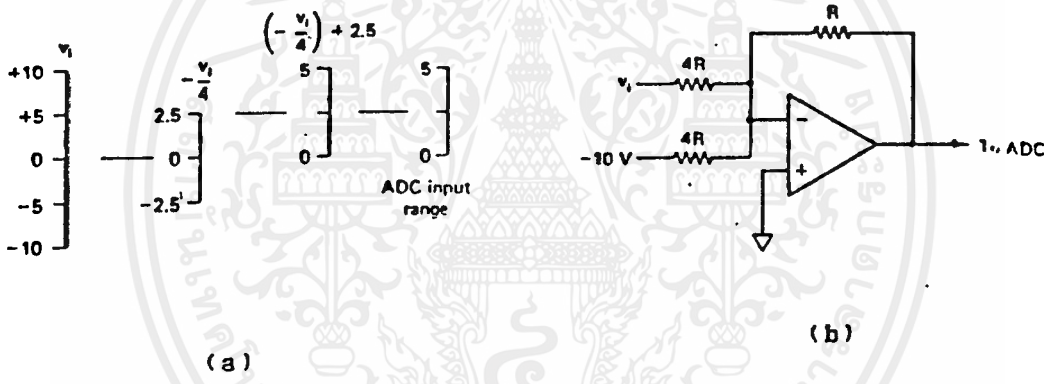
อัตราส่วนของการเปลี่ยนแปลง เป็นอีกสิ่งหนึ่งที่มีความสำคัญซึ่งหาได้จากจำนวนของวงจรเวลาเมื่อต้องการในการแปลง จะมีค่าเท่ากับการแปลงกลับของ เวลาทั้งหมดถ้าไม่มีวงจร S/H

Converter Input and Outputs

Analog input signal จากการแปลงสัญญาณส่วนใหญ่ ในการออกแบบจะยอมรับความแตกต่างหรือสัญญาณอินพุตทางด้านเดียวสัญญาณจะต้องอยู่ในย่านของอินพุตสัญญาณอินพุตทั้งหมดนี้อยู่ในย่าน 0 ถึง 5 โวลท์ ถ้าสัญญาณอินพุทไม่อยู่ในย่านที่กำหนดรหัสที่ได้จากการแปลงจะไม่ใช่ค่าจริงค่าของความผิดพลาดในการแปลงในย่าน ผลลัพธ์ที่ได้ของค่าความผิดพลาดจะปรากฏที่เอาท์พุตในการแก้ไขปัญหาที่ดีที่สุด เลือกย่านของอินพุตที่ใช้ในการแปลงโดย ใช้วงจรขยายโดยทั่วไปสัญญาณอินพุตทั้งหมด ในระบบต้องการกระบวนการบางอย่าง และสามารถทำงานได้เหมือนกันใน

เรื่อนไขของวงจรส่วนสุดท้าย บางครั้งสัญญาณอินพุทที่มีระยะความกว้างของสัญญาณลดลง เนื่องจากสัดส่วนสเกลที่ลดลงของแรงดันอ้างอิงซึ่งในการออกแบบการทำงานของการเปลี่ยนแปลงสัญญาณขึ้นอยู่กับ การปรับค่าของสัญญาณอ้างอิง

ถ้าสัญญาณอินพุทเป็นแบบ bipolar เรายังคงใช้การแปลงสัญญาณแบบด้านเดียวโดยขั้นแรกย่านของอินพุทจะลดลงและผลบวกของ offset แสดงให้เห็นดังรูปที่ 3.16 เราสามารถเปลี่ยนสัญญาณอินพุทแบบ bipolar ในการแปลงสัญญาณได้การแปลงสัญญาณของอินพุทจะเป็นผลรวมของ 0 ถึง + 5 V และรหัสทาง เอาต์พุทที่เป็นสัญญาณดิจิตอลสองชั้วได้ การลบเลขแบบทูคอมพลีเม้นท์,ออปเซ็ทไบนารี่ขนาดของ เครื่องหมายและการพิจารณาอื่น ๆ



รูปที่ 3.16 Adapting a bipolar signal to a unipolar ADC

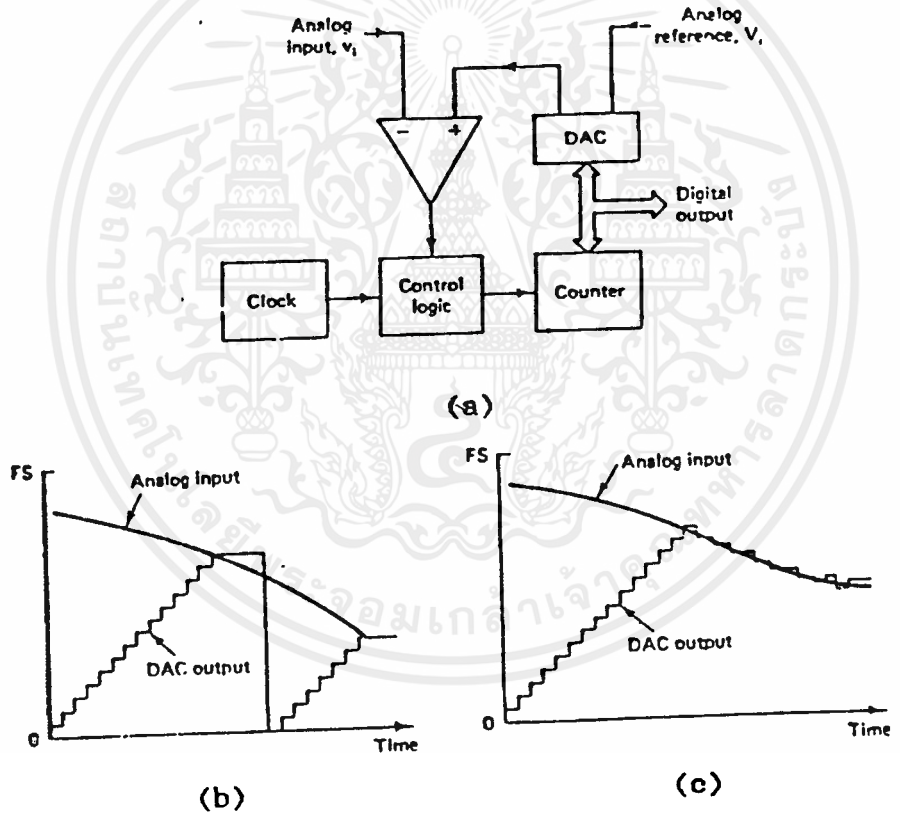
(a) Input signal is scaled and an offset is added

(b) Circuit that implements this function

Analog reference signal จากรูป 3.17 (a) แสดงถึงอินพุทและเอาต์พุทของ ADC ADC ทุกๆ ตัวต้องการสัญญาณอานาล็อกอ้างอิงเพื่อที่จะเป็นอัตราส่วนในการแปลงสัญญาณทางอินพุทค่าความผิดพลาดของสัญญาณอ้างอิงสามารถที่จะปรับได้ ในขั้นแรกการเกิดกระแส เวลาและอุณหภูมิเกิดขึ้นการขยายความผิดพลาด ในส่วนของคุณลักษณะของ ADC ค่าความเที่ยงตรงและเสถียรภาพของแรงดันอ้างอิงจะต้องเป็นไปตามความจริงของ ADC IC regular จะเป็นแหล่งจ่ายให้กับแรงดันอ้างอิงซึ่งในการแปลงของอุณหภูมิ จะต้องไม่มีผลต่อแหล่งจ่าย

Counter or Tracking ADC

จากรูป 3.17 (a) แสดงถึงบล็อกไดอะแกรมการนับของ ADC มันใช้ในการนับจากภายในตัวเอาต์พุต DAC จนกระทั่งมันมีค่าเท่ากับหรือ มีค่ามากกว่าแรงดันของสัญญาณอินพุต การนับจะเริ่มจากการแปลงสัญญาณ clock เอาต์พุตของ DAC จะเริ่มจาก LSB ที่เวลาหนึ่ง ดังแสดงในรูป 5.5 (b) การเปรียบเทียบจะหยุดก็ต่อเมื่อ แรงดัน DAC จะสะสมถึงระดับแรงดันอินพุตการนับสุดท้าย คือเอาต์พุตของสัญญาณดิจิตอลการแปลงของ ADC จะเปลี่ยนแปลงไปตามระดับ ของสัญญาณอินพุตจะต้องมากกว่า 2^n ของเวลา clock การแปลงขนาด n บิต จากระดับสัญญาณอินพุตจะมีค่ามากสัญญาณสเกล



จากรูป 3.17 Counter or tracking-type ADC and circuit waveforms

(a) Simplified block diagram

(b) Waveforms for counter type

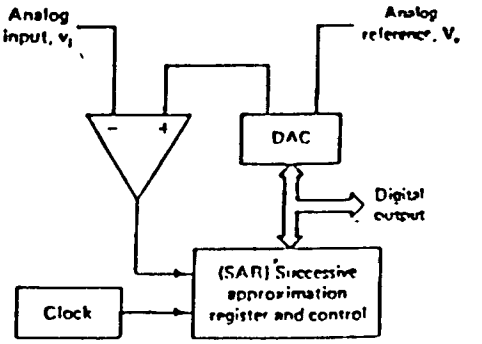
(c) Waveforms for tracking type

สิ่งที่ใช้ในการแปลง เรียกว่า tracking หรือ servo ใช้ในการนับขึ้นหรือลง ของระดับแรงดัน DAC ไปยังแนวทางสัญญาณอินพุตที่ต่อเนื่องในขณะที่สัญญาณอินพุตที่เปลี่ยนมีขนาดไม่เท่ากันจากรูป 3.18 (c) แสดงแรงดัน DAC ของชนิด tracking-type ADC จากภายนอก การหยุดนับได้เราจะใช้วงจร S/H สัญญาณดิจิตอลทางเอาต์พุตและเก็บค่าของเวลาการแปลงสัญญาณทำให้สัญญาณดิจิตอลทางเอาต์พุตที่มีค่าเหมือนกับค่าสูงสุดและต่ำสุดของสัญญาณอินพุตให้ค่าเกินกว่าที่เรียก

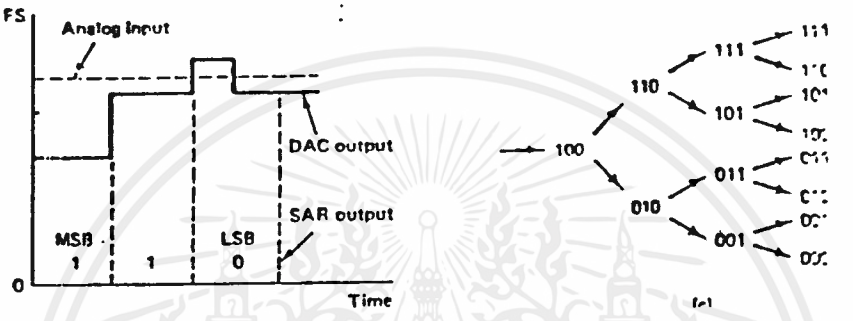
Successive-Approximation ADC

Successive approximation เป็นเทคนิคในการแปลงสัญญาณระดับกลางจนถึงสูงมันอยู่ภายใน DAC แต่อย่างไรก็ตามมันไม่เหมือนกับ ADC การประมาณค่าต่างๆ ของ ADCในระดับสัญญาณอินพุต ขณะเวลา ciock ที่ n จาก ADC ขนาด n บิต เวลาในการแปลงของกระบวนการนี้ไม่สั้นและไม่ขึ้นอยู่กับแรงดันอินพุตเทคนิคนี้เป็นพื้นฐานในการประมาณค่า ของสัญญาณอินพุตกับรหัสไบนารี่และการเดาค่า ในการพิจารณาการประมาณค่านี้ จากแต่ละบิตในรหัสจนกระทั่งได้ค่าของการประมาณที่ถูกต้องที่สุดในแต่ละขั้นตอนของกระบวนการนี้ ค่าของกระแสไบนารี่ของการประมาณค่า คือค่า Successive-approximation register (SAR)

จากรูป 3.19 แสดงถึงโครงสร้างพื้นฐานและการทำงานของ ADC ขนาด 3 บิต เริ่มแรกการเปลี่ยนสัญญาณรหัส MSB ของ SARทำงานก่อนในขั้นแรกจะประมาณค่าเท่ากับครึ่งหนึ่งของฟูลสเกลของสัญญาณอินพุต การเปรียบเทียบการให้ค่าเอาต์พุตและการควบคุมการบิตของ MSB ถ้าการประมาณในขั้นแรกมีค่ามากกว่าสัญญาณอินพุต อีกอย่างหนึ่ง MSB ทางซ้ายจะเปิดในส่วนนี้สัญญาณ clock จะควบคุมการบิตเปิด MSB ต่อไปการเปรียบเทียบในการตัดสินจากระดับของสัญญาณอินพุต บิตจะปิดหรือเก็บไว้การแปลงสัญญาณจะดำเนินต่อไปในวิธีดังกล่าวจนกระทั่ง LSB ถูกทดสอบ ที่ค่านี้จะถูกเก็บไว้ใน SARและรีจิสเตอร์เอาต์พุตจะเก็บการประมาณค่าไบนารี่ที่ถูกต้องที่สุดของสัญญาณอินพุตจนกระทั่งได้ค่าเป็นดิจิตอลเอาต์พุต บิตได้มาจากผลรวมของการต่อบิตจนกระทั่งถึงบิตสุดท้ายเพื่อที่จะประมาณค่าให้ได้เอาต์พุตในส่วนการแปลง



(a)



(b) (c)

จากรูป 3.18 Successive-approximation ADC converter

(a) Block diagram

(b) Circuit waveforms

(c) Logic flow diagram

Dual-slope Integrating ADC

จากรูป 3.19 แสดงเทคนิคของ dual-slope แรงดันอินพุต คือ การอินติเกรตจากค่าคงที่ของระยะเวลา T_1 โดยทั่วไปจะมีลักษณะเหมือนกับการนับสูงสุดของการนับภายในสุดท้ายระยะเวลาของการรีเซ็ทและการอินติเกรตอินพุต สวิทช์จะเคลื่อนที่ไปยังแรงดันอ้างอิงลบการอินติเกรตเอาท์พุตจะลดลงเป็นเชิงเส้นจน กระทั่งมันค่าเท่ากับศูนย์เมื่อการหยุด และการอินติเกรตถูกรีเซ็ทการชาร์จจะเริ่มต้น โดยค่าความจุของการอินติเกรต ระหว่างระยะเวลาเริ่มแรกต้องเท่ากับการดิสชาร์จในระยะเวลาที่สอง ดังนั้น

$$T_1 V_{1(avg)} = t_2 V_r$$

ดังนั้น

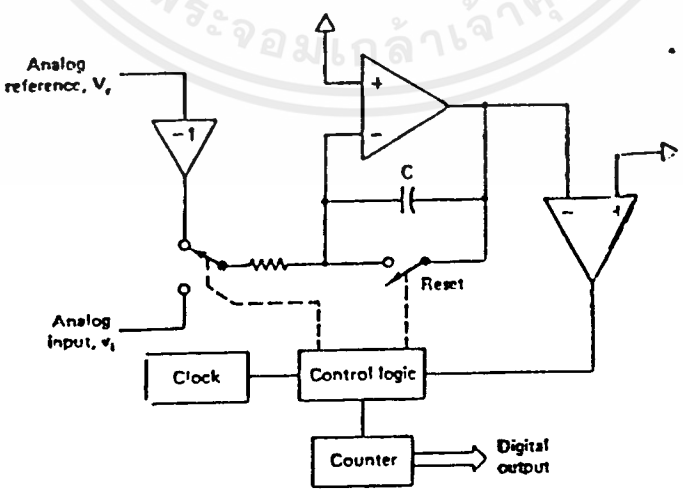
$$\frac{t_2}{V_1} = \frac{V_1\ (avg)}{V_r} = x$$

อัตราส่วนของระยะเวลา ดังนั้นการนับของไบนารี่จะมีความเกี่ยวพันธ์กับการนับทั้งหมดของการนับการนับในระยะหลัง t_2 คือค่าของไบนารี่เอาท์พุทของ ADC

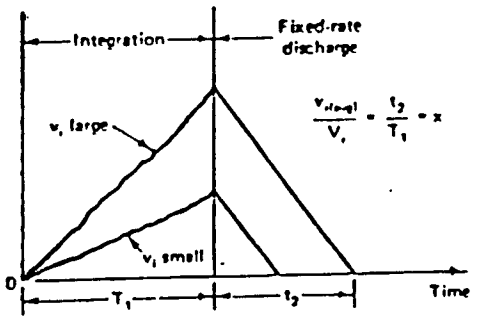
เทคนิค dual-slope มีประโยชน์มาก เริ่มจากคุณลักษณะของ noise-rejection ตั้งแต่ค่าแรงดันอินพุท คือการอินติเกรตเกินระยะเวลา ค่าของสัญญาณรบกวนความถี่สูงจะบนอยู่ในสัญญาณอินพุทาให้ยกเลิก ค่าเฉลี่ยของเวลา T_1 สามารถที่จะเลือกช่วงระยะกาจัดของ $1/T_1$ ค่าของความถี่ในการเลือกจึงเป็นสิ่งสำคัญ

หมายเหตุ การปรับค่าของความถี่ของสัญญาณ clock จะไม่มีผลต่อการกระจายค่าการแปลงสัญญาณจะขึ้นอยู่กับค่าของความจุของวงจรอานาล็อกเท่านั้น และ ไม่ขึ้นกับการแปลงสัญญาณความแตกต่างที่ไม่เป็นเชิงเส้นตั้งแต่ค่าเอาท์พุทของการอินติเกรตจะอิสระและได้ค่ารหัสทางเอาท์พุทด้วยเหตุนี้การกระจายที่ดีจะเกี่ยวพันธ์ และแปรเปลี่ยนไปตามการปรับขนาดของการนับภายนอกและความถี่ของclock

การแปลงสัญญาณของ dual-slope จะทำงานช้า จากตัวอย่างถ้า T_1 มีค่า 60 Hz และเป็นฮาร์โมนิค ค่าของผลลัพธ์จะได้ 16,67 mS เวลาที่เข้าในการแปลงสัญญาณจะมีค่าเป็น 2 เท่าของค่านี้ การเปลี่ยนแปลงสัญญาณจะต้องไม่น้อยกว่าการสุ่ม 30 ค่าต่อวินาที ซึ่งสามารถเห็นได้ใน digital panel meters (DPMs), ดิจิตอลมัลติมิเตอร์ (DMMS), เครื่องวัดอุณหภูมิ



(a)



(b)

รูปที่ 3.19 Dual-slope ADC

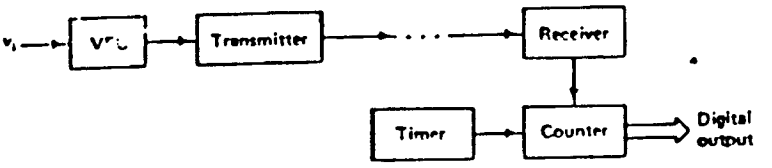
(a) Block diagram

(b) Circuit waveforms

Voltage-to-Freguency ADC

จากรูป 3.20 แสดงถึง Voltage-to-frequency ของ ADC แรงดันอานาล็อกของอินพุต จะถูก เปลี่ยนโดยการแปลงสัญญาณ Voltage-to-frequency ที่มีความเที่ยงตรง (VFC) ใน จำนวนพัลส์ของความถี่จะ เป็นสัดส่วนกับแรงดันอินพุต การนับของสัญญาณดิจิตอลทาง เอาต์พุตได้มา จากการนับของจำนวนพัลส์จะมากกว่าระยะที่คงที่ของ เวลาจากการสัง เกตุ เห็นสัญญาณอินพุตผลของ การอินติเกรต เกินกว่าระยะ เวลา เทคนิคdual-slope จะมีคุณลักษณะคือความเร็วช้าแต่ไม่มี สัญญาณรบกวน

ถ้าระยะการแปลงสัญญาณที่สามารถยอมรับได้ เทคนิคการแปลงสัญญาณ v-f ให้การ กระจายของการชาร์จสัญญาณอย่างช้าๆ จากตัวอย่างที่ความถี่ 10 KHz VFC จะนับระยะ เวลา 1 วินาที มีความแม่นยำ 1 ใน 10000 จากตัวอย่างมีค่าการกระจาย 13 บิต ความเที่ยงตรง จะอยู่นอกสัญญาณอินพุต ประโยชน์สูงสุดของการแปลงสัญญาณ v-f จะเห็นได้จากการส่งข้อมูล ระยะไกลในสิ่งแวดล้อมที่มีสัญญาณรบกวนการประยุกต์ใช้การของ VFC จะอยู่ในบริเวณของ เครื่อง ส่งระยะไกลพัลส์ที่ทำ เนิดมาจาก VFC คือ สัญญาณดิจิตอลที่ส่ง เป็นระยะทางไปยังส่วนที่รับการรวม การนับที่ตัวรับได้จากการ เปลี่ยนจะแสดงในรูปสัญญาณดิจิตอลที่เอาต์พุต สิ่งที่หลีก เลี่ยงของสัญญาณ อานาล็อกจากสัญญาณอานาล็อกภายนอกและมัน เป็นไปได้ทำให้ค่าของsignal-to-noise-ratio ลดลง ข้อมูลทางดิจิตอลที่ถูกส่งออกจะถูกจัดในสัญญาณรบกวน

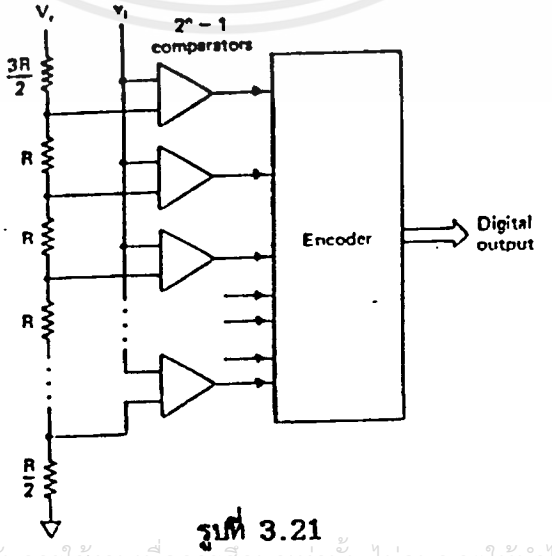


รูปที่ 3.20 V-F type of A/D conversion

ในทางปฏิบัติการแปลงสัญญาณ A/D ขึ้นอยู่กับ VFCs ซึ่งจะต้องมีความเป็นเชิงเส้นและมีความเสถียรภาพสิ่งนี้จะใช้วิธีของการแปลง v-fส่วนสำคัญของการแปลงสัญญาณนี้คือความสมดุลย์ของการชาร์จ

Parallel or Flash ADC

จากรูป 3.21 แสดงเทคนิคการแปลงแบบ flash หรือ parallel ซึ่งมีการแปลงสัญญาณที่มีความเร็วมาก เช่นการประยุกต์ใช้กับ วีดีโอ, เรดาร์และดิจิตอลออสซิลโลสโคปในส่วนนี้ใช้สัญญาณอินพุตมาเปรียบเทียบโดยที่คอมพาราเตอร์แต่ทั้งหมดจะแตกต่างกัน 1 LSB หลักการแปลงสัญญาณนี้ใช้หลักการพื้นฐานโดยใช้แรงดันอ้างอิงและความต้านทานภายใน network คอมพาราเตอร์มีค่ามากกว่าระดับสัญญาณอินพุตจะปิดขณะที่ตัวอื่นยังเปิดอยู่คอมพาราเตอร์ทั้งหมดนี้จะเปลี่ยนสเตทในกระบวนการ quantization จะเปรียบเทียบทีละครั้ง เอาต์พุตที่ได้จากคอมพารามิเตอร์จะต้องนำเข้าไปการถอดรหัส ADC ใช้การแปลงสัญญาณแบบ parallel จะมีการแปลงในอัตราที่ความถี่ 100 MHz ที่มีขนาด 8 บิตแต่อย่างไรก็ตามการแปลงสัญญาณแบบนี้จะถูกจำกัดด้วยจำนวนของตัวคอมพาราเตอร์ ซึ่ง ADC ที่มีขนาด 8 บิตจะมีตัวคอมพาราเตอร์ 255 ตัว



รูปที่ 3.21

Software Implementations

ฟังก์ชั่นของการนับการเลื่อนการกลับการเปลี่ยนรหัสและค่าอื่น ๆ ใช้โปรแกรมในการทำงานของไมโครโปรเซสเซอร์ โปรแกรมเป็นส่วนหนึ่งของการทำงานในการแปลงสัญญาณ A/D เป็นบางครั้งแต่อย่างไรก็ตามยังคงมีความจำกัดต่อเนื่องจากค่าในทางปฏิบัติ เพราะว่าค่าของ ADCs จะต้องทำงานได้ดี

Shaft Encoders

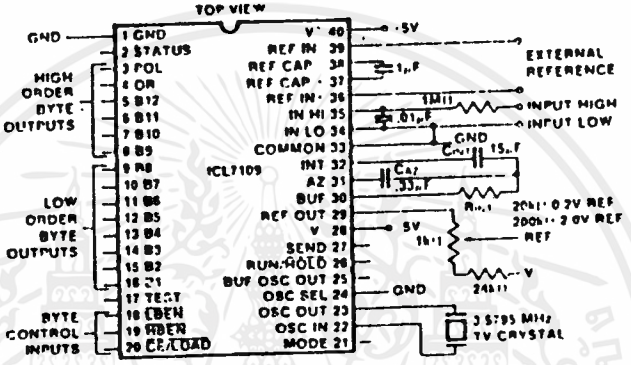
Shaft encoders จะใช้วิธีการเปลี่ยนแปลงสัญญาณแบบ electromechanicalของการเปลี่ยน A/D ซึ่งการเปลี่ยนจะมีลักษณะเดียวกับดิจิตอลทางเอาต์พุตโดยทั่วไปการเข้ารหัสจะใช้ optical หรือ magnetic sensors รหัสทาง mechanism จากเอาต์พุตที่แสดงไบนารี่ 3 บิต ข้อเสียของรหัสไบนารี่ คือ ค่าของแสงที่ส่งผ่าน mechanism จะมีผลทำให้เกิดความผิดพลาดทางเอาต์พุตมาก จากตัวอย่างรหัส 001 สามารถหมุนไป 111 ถ้า MSB คือmisaligned โดยทั่วไปให้ 180-องศา ความผิดพลาดปัญหานี้สามารถแก้ไขได้โดยใช้ graycode Shaft Encoder เป็นอุปกรณ์ที่รัดกุม มันสามารถที่จะเพิ่มหรือวัดมุมที่สมบูรณ์

3.4.3 ลักษณะของ ICL 7109

ICL 7109 เป็น chip ที่ใช้แปลงสัญญาณ analog เป็น digital ขนาด 12 bit ที่มีคุณสมบัติที่สำคัญดังต่อไปนี้ คือ

- เป็น ADC แบบ Dual Slop Integrating ขนาด 12 bit binary มีสัญญาณแสดงขั้ว polarity สัญญาณเกินย่าน overrange
- Three state output compatible TTL สามารถทำ handshake ในการ Interface แบบ parallel หรือ Serial กับ Microprocessor ได้ง่าย
- สามารถควบคุมเวลาในการแปลงสัญญาณได้ที่ขา 26 (RUN / HOLD) และสามารถดูเวลาในการแปลงสัญญาณได้ที่ขา 2 (STATUS)

- Differential Input และ Differential Reference มีความถูกต้องและเป็นค่าจริง
- สัญญาณรบกวนต่ำประมาณ 15 V_{p-p}
- Input current ขณะที่ยังไม่ใช้งานมีค่าประมาณ 1 pA
- สามารถแปลงสัญญาณได้มากกว่า 30 ครั้งต่อวินาที
- สัญญาณ clock จะใช้ cystal TV 3.58 MHz ซึ่งจะทำให้เกิดการแปลงสัญญาณ 7.5 ครั้งต่อวินาที หรือใช้ RC Network ก็ได้



รูปที่ 3.22 Pin Configuration and test Circuit

3.4.4 ความหน้าที่การทำงานของ ICL 7109 CPL

GND (1) : Digital Ground , 0 V , Ground return for all digital logic

STATUS (2) : "HIGH" ในช่วงระหว่าง integrate และ deintegrate จนกระทั่งข้อมูล latched

: "LOW" เมื่อ analog section เป็น auto zero

POL (3) : "HIGH" เมื่อสัญญาณอินพุตเป็น บวก

OR (4) : "HIGH" เมื่อเกิดการ overranged

B1-B12 (5-16) : All three state output data bits

TEST (17) : "HIGH" ถ้าต้องการทำงานตามปรกติ

: "LOW" เมื่อ test พุทปิดเป็น high

LBEN (18) : "LOW" เมื่อขา mode (21) เป็น LOW และขา ce/load (20) เป็น low ทำให้

output b1-b8 active แต่ถ้าขา mode เป็น high (ทำ handshake) เพื่อให้ขาเป็น LOW จะทำให้สัญญาณออกที่ขา b1-b8

HEBN (19) : "LOW" เมื่อขา mode (21)เป็น low และขา ce/koad (20)เป็น low ทำให้ output b9-b12 ,pol,or "active" ถ้าขา mode (21)เป็น high(ทำ handshake) จะทำให้ high byte active

CE/LOAD (20) : "LOW" เมื่อขา mode (21)เป็น low ทำให้ขานี้เป็น low ด้วยจะทำให้ b1-b12 or,pol "enable"

: "HIGH" จะทำงานในโหมด handshake

MODE (21) : "LOW" ทำงานใน mode direct

: "HIGH" ทำงานร่วมกับขา cd/load (20) lebn (18),hebn (19) ทำงานใน mode handshake

OSC IN (22) : oscillator input

OSC OUT (23) : oscillator output

OSC SEL (24) : "HIGH" จะให้สัญญาณ osc โดยใช้ rc network สัญญาณ clock จะมี phase และ duty cycle เหมือน buf osc out

: "LOW" จะให้สัญญาณ oscillatorโดยใช้ crystal ความถี่ clock จะเป็น 1/38 ของความถี่ที่ buf osc out

BUF OSC OUT (25) : BUFFERED OSCILLATOR OUTPUT

RUN/HOLD (26) : "HIGH" การแปลงค่าจะเป็นไปอย่างต่อเนื่องทุก ๆ 8192 clock pulses

: "LOW"เมื่อแปลงสัญญาณเสร็จแล้วจะอยู่ในช่วง auto zero ก่อนการแปลงสัญญาณใหม่

SEND (27) : เป็นสัญญาณ input จะใช้ใน handshake mode เพื่อบอกให้รู้ว่าอุปกรณ์ภายนอกพร้อมจะรับข้อมูล ถ้าไม่มีการใช้งานต้องต่อไว้กับ +vcc

V- (28) : ต่อกับ - 5 v เมื่อเทียบกับ gnd

REF OUT (29) : "Reference Voltage Output"ปกติจะเป็น.8 v เมื่อเทียบ +vcc(40)

BUFFER (30) : Buffer Amplifier Output

AUTO-ZERO (31) : Auto Zero Node ด้านในต่อกับ Ca2

INTEGRATOR (32) : Integrator Output ด้านนอกต่อกับ Cint

COMMON (33) : Analog Common ระบบจะเป็น auto zeroed เมื่อต่อกับ common

INPUT LO (34) : Input Low ปรกติมักต่อกับ gnd

INPUT HI (35) : Input High ปรกติมักต่อกับ +5v

REF IN + (36) : แรงดันอ้างอิง input เป็น +

REF CAP+ (37) : Reference Capacitor Positive

REF CAP- (38) : Reference Capacitor Negative

REF IN - (39) : แรงดันอ้างอิงอินพุตเป็น -

V + (40) : ต่อกับ +5v เมื่อเทียบกับ gnd

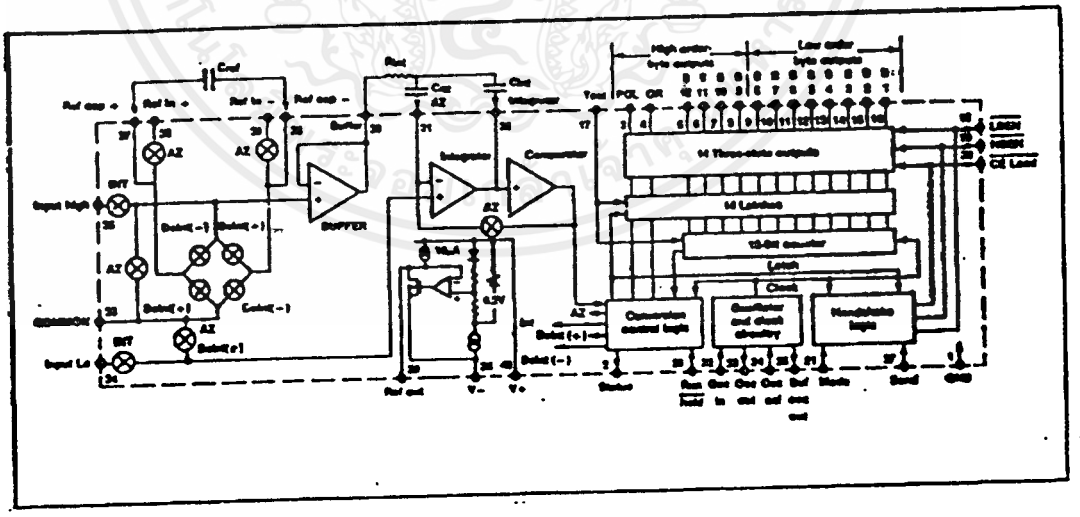
3.4.5 การทำงานของ A/D ICL 7109

การอธิบายการทำงานของ A/D ในส่วนนี้จะอธิบายเพียงเป็นที่พอเข้าใจสำหรับรายละเอียดเพิ่มเติมสามารถหาอ่านได้ใน Data Sheet ของบริษัทผู้ผลิตการทำงานของ IC 7109 ซึ่งเป็น Dual Slope Integration A/D มีดังนี้

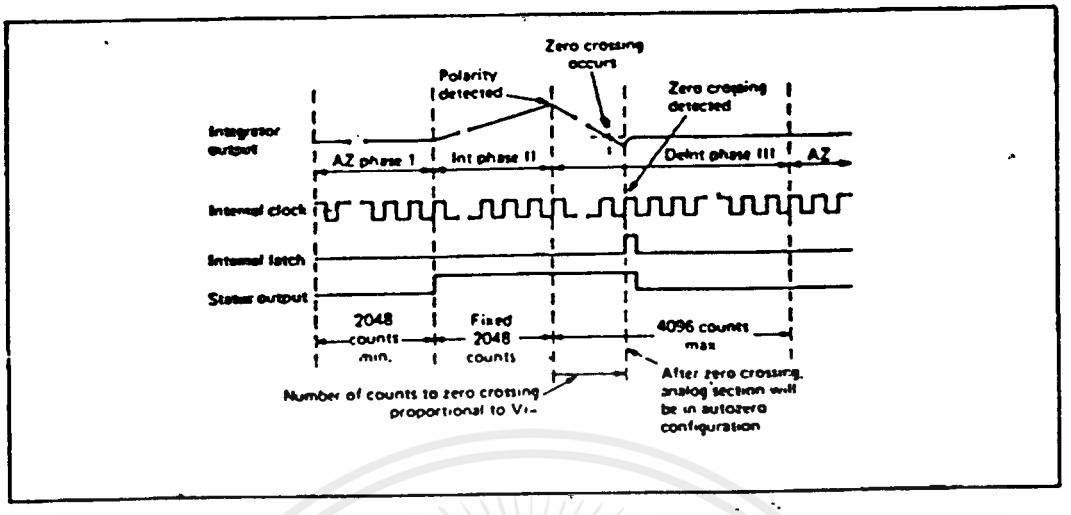
Auto-Zero phase เป็นขั้นตอนของการแก้ไขค่า offset error ที่เกิดขึ้นจากอุปกรณ์ภายในของ IC และทำให้ IC อยู่ในสภาวะเริ่มต้น ในระหว่างช่วง Auto Zero มีเหตุการณ์ 3 อย่างเกิดขึ้น อย่างแรก Input high และ low จะตัดวงจรออกจากขาของ IC แล้วถูกนำมาต่อวงจรเข้ากับ Analog common และอย่างที่ 2 Reference capacitor C_{ref} จะถูกชาร์จประจุด้วยแรงดัน Reference voltage อย่างที่ 3 Feedback loop ของ AZ จะทำให้ Auto-Zero capacitor C_{az} ถูกชาร์จประจุเพื่อไปชดเชย Offset voltage ของวงจร Buffer amplifier, Inte grator และ Comparator เนื่องจากวาน Feedback loop มีวงจร Comparator อยู่ใน loop ด้วย ทำให้ค่าความถูกต้องของขบวนการ Auto-Zero ถูกจำกัดด้วย noise ของระบบ อย่างไรก็ตาม offset voltage ทางด้าน Input ก็อยู่ในระดับที่ต่ำกว่า 10 uV

Singnal Integrate phase ในระหว่างขั้นตอนนี้ Auto-Zero Loop ทั้งหมดจะเปิดออก Input high และ low จะถูกปลดออกจาก Analog common แล้วต่อเข้ากับขาของ IC A/D จะ integrate ผลต่างแรงดันระหว่างขา Input high และ low ด้วยเวลาคงที่เป็นจำนวน 2048 สัญญาณ clock ที่จุดสุดท้ายของขั้นตอนนี้ขั้วของแรงดันสัญญาณ Input จะถูกตรวจสอบ

De-integrate phase นี้เป็นขั้นตอนสุดท้ายของการแปลงสัญญาณ input low จะถูกวงจรออกจากขาของ IC แล้วต่อเข้ากับ Analag common และ Input high จะต่อเข้ากับ Reference capacitor C_{ref} ซึ่งมีแรงดันเท่ากับ Reference voltage จากการชาร์จประจุในขั้นตอน Auto-Zero วงจรภายในตัว IC จะต่อขั้วแรงดันของ C_{ref} ได้อย่างถูกต้องด้วยผลการตรวจสอบขั้วแรงดันในตอนท้ายของขั้นตอน Auto-Zero เพื่อให้ De-integrate อย่างถูกทิศทางการ De-integrateจะทำด้วย Slope คงที่จนแรงดันของการ De-integrate มีค่าผ่านจุดศูนย์ (Zero Crossing) เวลาที่เข้าในการ De-integrate จะเป็นผลของการแปลงสัญญาณออกในทาง 12 bit counter และ Tri-state Outputขั้นตอนนี้อาจจะเรียกอีกอย่างหนึ่งว่า Reference Integrate



รูปที่ 3.23 แสดงวงจรสมมูล (Equivalent circuit) ของ ICL 7109



รูปที่ 3.24 แสดง Timing dirgram ของการ Conversion

3.4.6 ข้อกำหนดการใช้งานของ ICL 7109

Differential Input ผลต่างของแรงดัน Input จะมีค่าเท่าไรก็ได้ในช่วงแรงดันร่วมของทั้ง 2 input (Common.mode voltage) ซึ่งมีค่าต่ำกว่าแรงดันของแหล่งจ่ายด้านบวกอยู่ 1 V ค่าแรงดันในช่วงนี้จะทำให้ระบบมี CMRR.(Common Mode Rejection Ratio) ประมาณ 86 db อย่างไรก็ตาม เนื่องจาก Integrator มีการเปลี่ยนแปลง Output ตาม Common mode voltage ในการใช้งานจึงต้องออกแบบไม่ให้วงจร Integrator เกิดสภาวะ Saturate ในกรณีที่เลวร้ายที่สุดเมื่อมี Common mode voltage สูง ๆ ใกล้เคียงกับแรงดัน Full scale voltage และแรงดันผลต่าง (Differential voltage) เป็นลบทำให้ Integrator ให้ Output เป็นบวก ขณะเดียวกันกับ Common mode voltage ทำให้เกิดการ swing ด้านบวกด้วยเช่นกัน โอกาสที่จะทำให้เกิดการ saturate จึงเกิดขึ้นได้ง่ายในการใช้งานเพื่อจะหลีกเลี่ยงการณีนี้ เราทำได้โดยการลด Full scale voltage ให้ต่ำกว่า 4 V เพื่อที่จะให้ Integrator output swing ได้ในช่วง 0.3V จากแหล่งจ่ายทั้งสองด้าน

ICL 7109 จะทำงานดีที่สุดเมื่อ Analog common มีค่าใกล้เคียงกับ Digital ground สำหรับแหล่งจ่าย +5 V และ -5 V Full scale voltage 4 V เป็นจุดที่ทำงานดีที่สุด

Differential Reference ICL 7109 มี Reference voltage ในตัวขา 29 Reference output สามารถต่อวงจรแบ่งแรงดันเพื่อสร้าง Reference voltage ค่าต่างๆ ด้วย Sink current 20 mA โดยจะไม่มีผลต่อแรงดัน Refer ence outputขา 29 นี้ต้องต่อ

R pullup สำหรับ Bias อุปกรณ์ภายในด้วยกระแสราว ๆ 10 uA แรงดัน Reference output ปกติจะมีค่าต่ำกว่าแหล่งจ่ายด้านบวก (V+) อยู่ 2.8 V และมีค่าสัมประสิทธิ์อุณหภูมิ 80 ppm ต่อองศา ความมีเสถียรภาพของ Reference voltage เป็นส่วนสำคัญของค่าความถูกต้องของ A/D ICL 7109 เป็น A/D 12 bit ความละเอียด 1 ใน 4096 หรือ 244 ppm ถ้าใช้ Reference voitage 80 ppm/^{o}C จะทำให้เกิดความผิดพลาดไป 1 LSB เมื่ออุณหภูมิเปลี่ยนไป 3 ^{o}C ดังนั้นในงานที่ต้องการความละเอียดถูกต้องมาก ๆ เราจึงจำเป็นต้องใช้ Reference voltage จากภายนอกที่มีค่าสัมประสิทธิ์อุณหภูมิต่ำ วสำหรับโครงงานนี้เลือกใช้ค่า Full scale voltage 4.096 v ที่ +5 V และ -5 V supply และใช้ Reference voltage ในตัว ICL 7109 เพราะ Reference voltage ที่มีค่าสัมประสิทธิ์อุณหภูมิต่ำกว่า 80 ppm/^{o}C หาซื้อได้ยากในท้องตลาดและมีราคาแพง

การเลือก Integrating Resistorวงจร Buffer amplifier และวงจร Integrator ถูกจัดวงจรแบบ class A ต้องการกระแสเลี้ยงวงจรในสภาวะสงบ 100 uA แต่เราสามารถป้อน bias 20 uA ได้โดยที่สามารถยอมรับความผิดเพี้ยนเป็นเชิงเส้นได้ Integrating resistor R_{int} ต้องมีขนาดใหญ่พอที่จะทำให้วงจรมีความเป็นเชิงเส้นตลอดช่วงแรงดัน Input แต่ต้องมีขนาดไม่มากเกินไปจนมีกระแสไหล biasวงจรไม่พอสำหรับค่า Full scale voltage 4.096 V R_{int} 200 K เป็นค่าที่ดีที่สุด ค่า R_{int} คำนวนได้จาก

$$R_{int} = \frac{\text{FULL scale voltage}}{20 \text{ uA}}$$

การเลือก Integrating CapacitorและAuto-Zero Capacitor Integrating capacitor C_{int} ต้องเลือกให้ค่า Output ของ Intergrator สามารถ swing ได้สูงสุดโดยไม่เกิดอาการ saturate ของ Integrator(ประมาณ 0.3 V จากแหล่งจ่ายแต่ละด้าน สำหรับแหล่งจ่าย +5Vและ -5Vและ Analog Common ต่อเข้ากับ GND ของ IC Integrator output ควร swing อยู่ในช่วง 3.5 V ถึง 4 V ค่า C_{int} คำนวนได้จาก

$$C_{int} = \frac{(2048 * \text{clock period}) * (20 \text{ uA})}{\text{Integrator output voltage swing}}$$

ที่ clock 61.72 KHz (crystal 3.58 MHz/58 ตามวงจรที่ผู้ผลิตแนะนำมาใน Data Sheet เพื่อลดผลของสัญญาณ line 60 Hz) ที่ 4.096 V Full scale vol tage C_{int} และ C_{az} ควรจะมีค่า 0.15 uF และ 0.33 uF ตามลำดับ Capacitor ที่นำมาใช้ควร

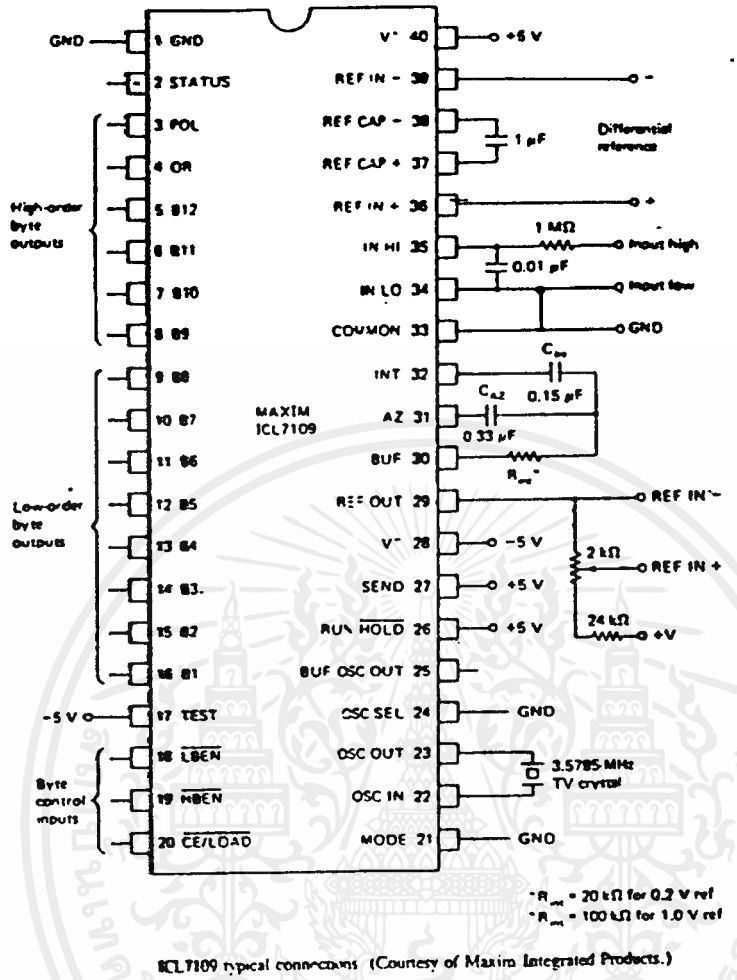
เป็นชนิดที่มี Dielectric loss ต่ำเพื่อป้องกันค่าผิดพลาดในการ integrate Capacitor ชนิด Polypropylene เหมาะสมที่จะใช้งานได้ในช่วงอุณหภูมิ 85 ^{o}c ถ้าต้องการใช้งานย่านอุณหภูมิพิเศษ Capacitor ชนิด Teflon จะเหมาะสมที่สุดขนาดของ Auto-Zero capacitor C_{az} มีผลต่อ noise ระบบขนาดของ C_{az} ที่ใหญ่จะให้ noise ต่ำ แต่อย่างไรก็ตาม C_{az} ไม่สามารถมีค่ามากได้เนื่องจากต่อขนานอยู่กับ C_{int} ค่า R-C time constantจะเป็นตัวกำหนดความเร็วของการแก้ไขการ Error ในช่วงท้าย ๆของขั้นตอน Auto-Zero ที่ 4.096 mV Full scale voltage noise จะมีความสำคัญมากเพราะที่ Full scale voltage ต่ำ noise จะเข้ามาได้ง่ายกว่า C_{az} ผู้ผลิตแนะนำให้มีค่าเป็น 2 เท่าของ C_{int} และในทางกลับกัน ที่ 4.096 V Full scale voltage ค่า C_{az} ควรมีค่าเป็น 1/2 เท่าของ C_{int}

Reference Capacitor Reference capacitor C_{ref} ขนาด 1 uF ก็เหมาะสมกับการใช้งานเกือบทุกชนิดแต่ถ้าจัดวางจรใช้งานที่ Common monde voltage สูงและ Full scale voltage ต่ำ ๆ ควรให้ C_{ref} มีค่ามากๆ สำหรับชนิดของ C_{ref} ก็เช่นเดียวกับ C_{az} และ C_{int} ที่ต้องการ loss ต่ำ ๆ

Digital Section วงจรส่วนของ Digital ประกอบด้วย clock oscil lator, 12 bit Binary counter with output latches และ วงจร TTL compatible three-state dirvers อีกทั้งส่วนของ UART handshake logic (UART Universal Asynchronous Receiver Transmitter) ซึ่งเป็นส่วนของการต่อ ICL 7109 ให้เป็นระบบ Remote data acquisition ที่จะส่งข้อมูลของการแปลงสัญญาณโดยระบบการสื่อสารแบบอนุกรมซึ่งไม่ได้ใช้ในโครงงานนี้

Mode ของ A/D ขา MODE Input มีไว้สำหรับเลือก Output mode ของ A/D ถ้าขา MODE เป็น low A/D จะอยู่ใน Direct output ซึ่ง Output ของการแปลงสัญญาณสามารถอ่านได้โดยการควบคุมผ่าน Control input และ Bytena ble input และถ้าขา MODE เป็น high A/D จะอยู่ใน UART handshake mode A/D จะส่งผลการแปลงสัญญาณทุก ๆ Conersion cycle ซึ่งใน mode นี้จะไม่ขอกล่าวถึงในโครงงานนี้

ใน Direct mode เราสามารถจัดรูปแบบของ Output ให้อยู่ในรูป 8 bit, 12 bit หรือ 16 bit ได้ ทั้งนี้เพื่อความสะดวกในการต่อวงจรอุปกรณ์อื่น ๆ สำหรับในโครงการนี้จัดรูปแบบ Output เป็นแบบ 16 bit หรือ 12 bit โดยการต่อวงจรให้ขา LBEN HBEN และขา CE/LOAD เป็น low เพื่อให้มี Output ออกจาก A/D ไปยัง IC 74LS373



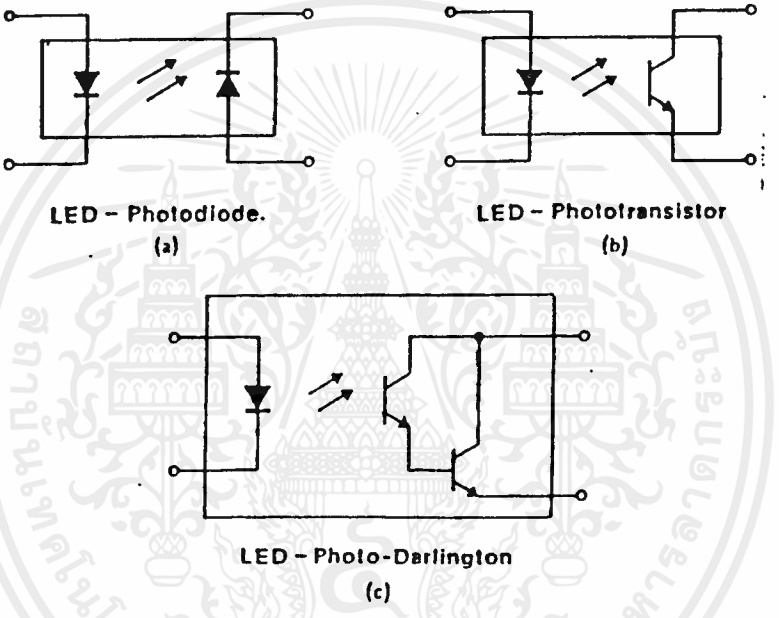
ICL7109 typical connections (Courtesy of Maxim Integrated Products.)

รูปที่ 3.25 การต่อวงจรอย่างง่ายของ ICL 7109

3.5 OPTO-ISOLATOR

ออพโต้ไอโซเลเตอร์ เป็นอุปกรณ์ซึ่งตัวนำ ที่ประกอบด้วยตัวกำเนิดแสงและตัวรับแสง ซึ่งต่อเข้าด้วยกัน ซึ่งในการส่งผ่านจะมีการแยกกันระหว่าง อินพุทและเอาท์พุททางไฟฟ้าอย่างเด็ดขาด ซึ่งสามารถนำมาใช้แทน รีเลย์, ไอโซเลย์ชั่นทรานฟอร์เมอร์และบล็อกกิ้งคาปาซิเตอร์ ออพโต้ไอโซเลเตอร์ ยังมีความเชื่อมั่นและหน้าที่ที่ทำเช้าได้ดีกว่าด้วย ชนิดของออพโต้ไอโซเล้เตอร์โดยทั่ว ๆ ไป แสดงไว้ในรูปที่ 3.26 ทั้งหมดเป็น LED โฟโต้ไดรอด, LED โฟโต้ทรานซิสเตอร์ และ LED โฟโต้ดาร์ลิงตันคามลาคับ ซึ่งในแต่ละตัวนี้ใช้ LEDเป็นตัวกำเนิดแสงและใช้ไดรอดหรือทรานซิสเตอร์เป็นตัวรับแสง ด้านอินพุทได้รับแหล่งจ่ายจากแหล่งจ่ายกระแส (current source) ขณะที่ด้านเอาท์พุทจะไปต่อกับวงจรอิเลคทรอนิคที่ต้องการนำมาต่อ

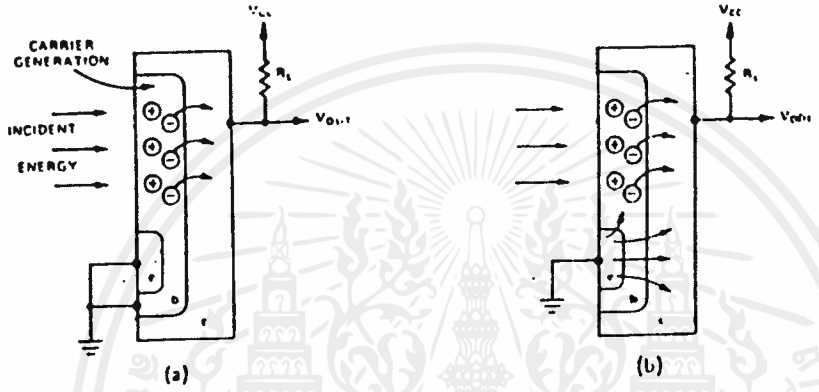
ในส่วนที่ได้กล่าวถึงคุณลักษณะทางแสงภายนอกของออพโต้ไอโซเลเตอร์แล้ว คุณลักษณะของออพโต้ไอโซเลเตอร์ ก็คืออุปกรณ์ทางไฟฟ้าตัวหนึ่งนั่นเอง โดยมีตัวแปรสองตัวเป็นตัวชี้บ่งกำหนด คุณลักษณะทางไฟฟ้าของออพโต้ไอโซเลเตอร์ คือ อย่างไรจึงจะส่งสัญญาณจากอินพุทไปยังเอาท์พุทได้ ซึ่งประสิทธิภาพการส่งผ่าน คือ อัตราส่วนของกระแสด้านเอาท์พุท ส่วนด้วยกระแสด้านอินพุท อัตราส่วนของการส่งผ่านของกระแส [Current transfer ratio (CRT)]นี้จะแปรตามอุณหภูมิและกระแสด้านอินพุท



รูปที่ 3.26 ชนิดของออพโต้ไอโซเลเตอร์

Opto-Isolator โดยทั่วไปประกอบด้วย แกลเลี่ยมอาร์เซไนด์(GaAs),อินฟาเรดไดโอด(IR LED)และโฟโต้ทรานซิสเตอร์ โดยบรรจุอยู่ในตัวถัง เดียวกันโดยทั่วไปตัวถังจะใช้แบบ DIP (Dual in-Line Package) เมื่ออินฟาเรดไดโอดได้รับกระแสไบแอสตรงจะเปล่งแสงอินฟาเรดออกมา ซึ่งจะมีความยาวคลื่นประมาณ 900 นาโนเมตร พลังงานที่เกิดจากรังสีนี้จะถูกส่งผ่านไปยังตัวกลางแสงและตกกระทบบนพื้นผิวหน้าของ Base ของโฟโต้ทรานซิสเตอร์ โดยที่โฟโต้ทรานซิสเตอร์จะถูกออกแบบ ให้ย่านของ Base มีขนาดใหญ่ ดังนั้นจึงทำให้รอยต่อระหว่าง Base กับ Collector มีพื้นที่มากด้วยเหตุว่าทางด้าน Emitter จะมีขนาดเล็ก พลังงานบางส่วนที่เกิดจากการตกกระทบของแสงซึ่งอยู่ในรูปของโฟตอน ซึ่งจะทำให้เกิดคู่ electron-hole ในย่านของ Base

จากรูป 3.27A Emitter และ Base ถูกต่อลงกราวด์และจ่ายไฟบวกให้แก่ Collector ซึ่งจะทำให้อุปกรณ์สารกึ่งตัวนำนี้เป็นโฟโต้ไดรอด สนามไฟฟ้าที่ตกคร่อมCollector -base จะไปดึง electron ให้ข้ามไปยังด้านของ collector อย่างรวดเร็ว จะทำให้ hole เลื่อนไปยังขั้วของ base และดึงอิเลคตรอนจากขั้วซึ่งจะทำให้เกิดกระแสอิเลคตรอนไหลระหว่าง base กับ collector ทำให้เกิดแรงดันตกคร่อม ความต้านทานโหลด (R_L) Time Constant

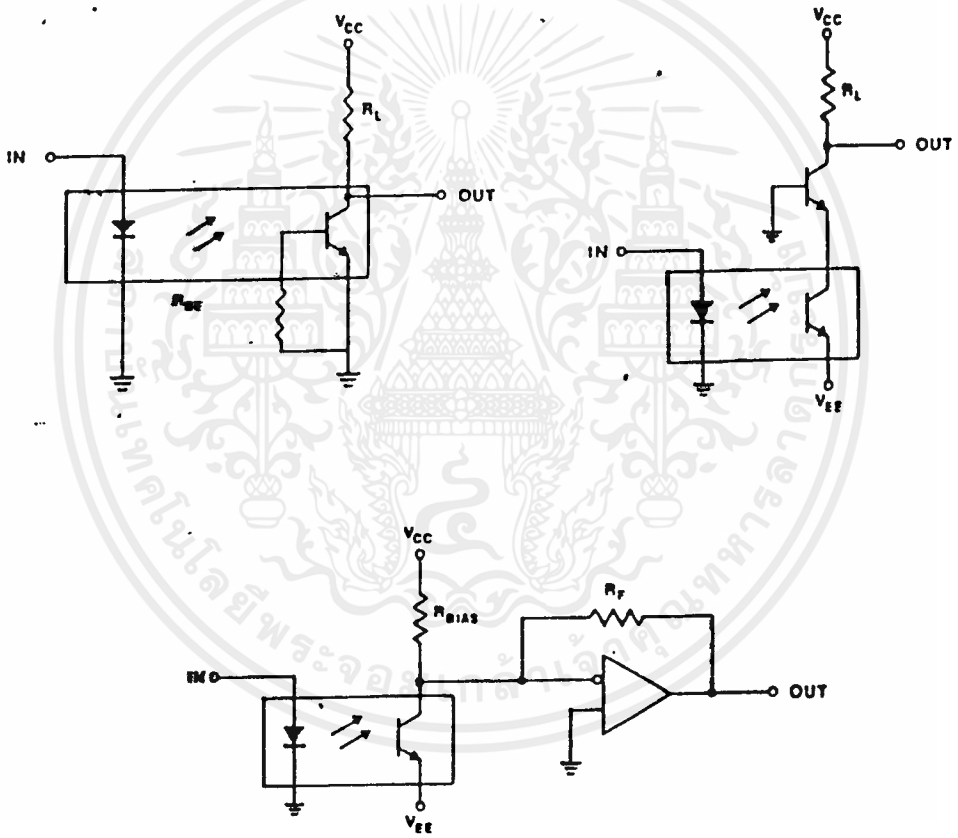


รูปที่ 3.27 ภาคตัดขวางของออพโต้ไอโซเลเตอร์

จะขึ้นอยู่กับค่าคาปาซิแตนท์ ของรอยต่อ(C_{cb}) และความต้านโหลด กระแสทางด้านเอาท์พุทในรูปนี้จึงนิยมใช้ โดยทั่วไปเราสามารถต่อวงจรได้อีกแบบดังรูปที่ 3.27B ซึ่งในรูปนี้ baseจะถูกปล่อยลอยไว้และ emitterจะถูกต่อลงกราวด์ Hole ที่เกิดขึ้นในย่านของ base จะทำให้เกิดแรงดันที่ base เพิ่มขึ้นหรือเป็นการไบแอสตรงแก่รอยต่อของ Base-Emitter อิเลคตรอนจาก Emitter จะถูกดูดเข้าไปยัง Base ซึ่งเป็นธรรมชาติของ Hole เนื่องจาก Collector นั้นอคติกับ Base มากอิเลคตรอนที่เหลือจากการรวมตัวกับ Hole จะถูกผลักให้เข้าไปยังย่านของ Collector จะทำให้กระแส Collector เพิ่มขึ้นซึ่งจะเป็นการไบแอสตรงให้แก่ทรานซิสเตอร์ ซึ่งกระแสนี้จะขึ้นอยู่กับ อัตราการขยายกระแส(Current gain) ของตัวทรานซิสเตอร์เอง อัตราการขยายกระแส ของรูป B จะสูงกว่ารูป A เป็นร้อยเท่า

การเพิ่มความเร็วให้แก่ออพโต้ไอโซเลเตอร์ ความเร็วของออพโต้ไอโซเลเตอร์ คือผลตอบสนองต่อสัญญาณของตัวรับ(Sensor)เวลาในการสวิตสขึ้นอยู่กับเวลาของ Base Storage และ RC Time constan ของเอาท์พุทย่านของ Collector-base ที่มีขนาดใหญ่จะมีความเร็ว

สูงกว่า ในรูป 3.28A นั้น Base emitter จะต่ออยู่กับความต้านทาน ความต้านทานตัวนี้จะเป็นตัวแบ่ง Storage charge ซึ่งเป็นการลดเวลา Turn-of ของโฟโต้ทรานซิสเตอร์ แต่จะทำให้อัตราส่วนของการส่งผ่าน (CTR) ลดลงในรูป 3.28B จะเป็นการต่อ Cascade กับ Amplifier ในการนี้นั้นโฟโต้ทรานซิสเตอร์จะเป็นแหล่งจ่ายกระแส และความต้านทานโหลด(R_L) จะทำให้ความต้านทานมี Emitter มีค่าน้อย ในรูปที่ 3.28C ต่อออปแอมป์เป็นเอาท์พุทของวงจรโฟโต้ทรานซิสเตอร์จะเป็นแหล่งกำเนิดกระแสขับออปแอมป์ที่จุด Summing วิธีนี้จะทำให้ R_L มีค่าต่ำเป็นผลให้ความเร็วเพิ่มขึ้น

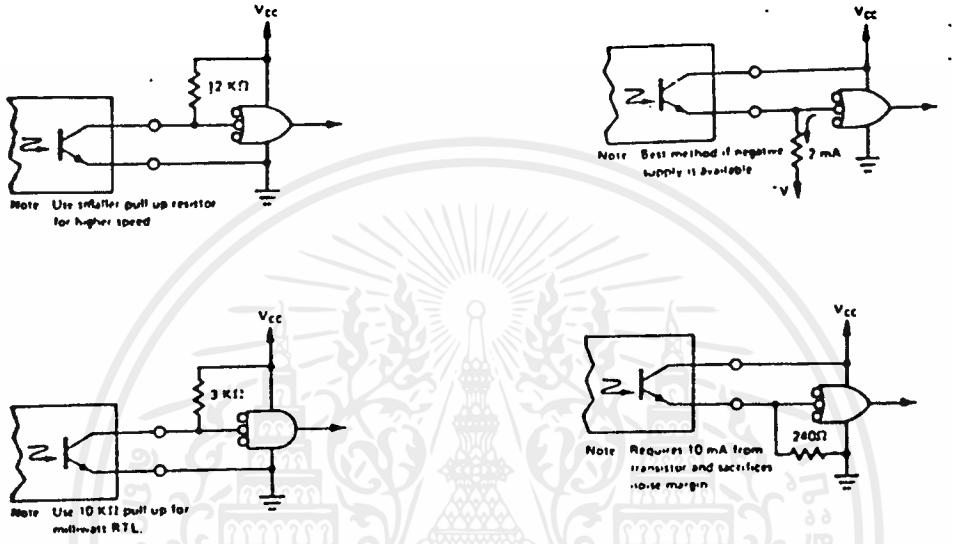


รูปที่ 3.28 วงจรการเพิ่มความเร็วให้แก่ออปโต้ไอโซเลเตอร์

การประยุกต์ใช้ออพติคัลไอโซเลเตอร์

ตัวอย่างแรกคือ เครื่องแสดงภาพการเต้นของหัวใจโดยใช้กระแสไฟฟ้าผ่านมือทั้งสองข้าง เราต้องมีการแยกระหว่างส่วนที่สัมผัสกับมือของผู้ป่วย กับอุปกรณ์ทางไฟฟ้าเพื่อ

ความปลอดภัยซึ่งเราสามารถใช้ออพเค้าเอรีเลเตอร์เป็นตัวแยกได้ ตัวอย่างที่สองคือจะใช้ ออพเค้าเอรีเลเตอร์ เป็นตัวแยกแรงดันที่ควบคุม potentiometer และในการควบคุมระยะไกลเราสามารถใช้ ออพเค้าเอรีเลเตอร์เป็นสวิทช์ปิดเปิดได้ ออพเค้าเอรีเลเตอร์สามารถใช้ในวงจรขยายคลาส B ได้

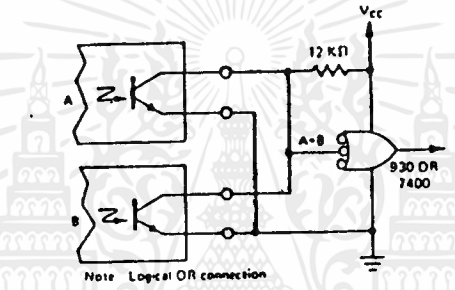


รูปที่ 3.29 การใช้ออพเค้าเอรีเลเตอร์เป็นตัวขับ TTL

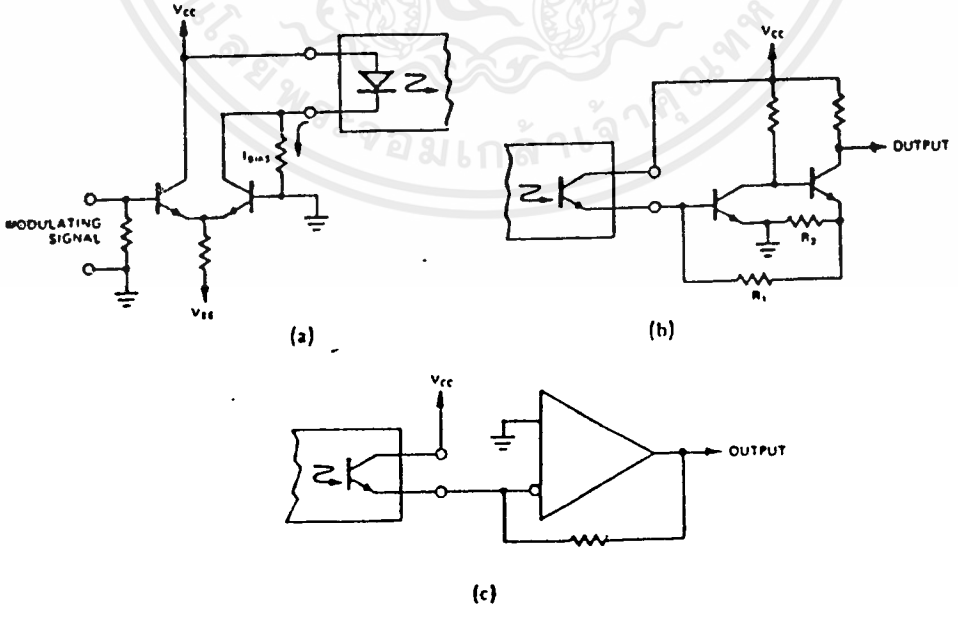
ออพเค้าเอรีเลเตอร์ สามารถขับด้วย TTL ได้ และสามารถใช้เป็นตัวขับให้ TTL ทำงานก็ได้ รูปที่ 3.29 เป็นการใช้ออพเค้าเอรีเลเตอร์เป็นเอาท์พุทเพื่อขับวงจรที่นำมาต่อซึ่งในรูป A นั้น TTL ทั้ง 2 ตัวจะทำงานที่ LOW ในรูปด้านบนวงจรเกทถูกขับด้วยออพเค้าเอรีเลเตอร์ด้วยความต้านทาน Pull-up 12 K ในวงจรรูปล่างเกทถูกขับด้วยออพเค้าเอรีเลเตอร์ด้วยความต้านทาน Pull-up 3 K ส่วนในรูป B TTL ทั้งสองตัวจะทำงานที่ High ซึ่งวงจรทั้งสองใช้ OR Gate วงจรรูปบนได้รับแหล่งจ่ายไฟบวกและลบ ส่วนด้านล่างได้รับเฉพาะไฟบวก

ในรูป 3.30 แสดงถึงการใช้ออพเค้าเอรีเลเตอร์ ในวงจรลอจิก ในรูปนี้ใช้ออพเค้าเอรีเลเตอร์สองตัวเพื่อเป็นการกระทำทางลอจิกกับ ORGate วงจรนี้สามารถใช้ได้ทั้ง เอชีกระกูล TTL และ DTL ส่วนการต่อให้ TTL เป็นตัวขับออพเค้าเอรีเลเตอร์ แสดงไว้ดังรูปที่ 3.32 ซึ่งมีด้วยกันสองแบบ ในรูป A TTL จะทำงานที่ High หรือเป็นแหล่งจ่ายกระแส (Current

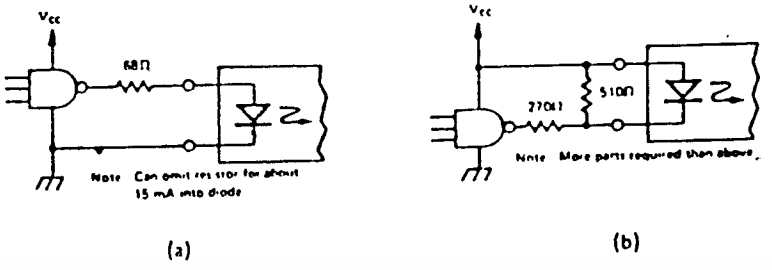
Source) โดยเอาท์พุทของ TTL จะถูกต่ออยู่กับขา Anode ของ LED ส่วนในรูป B TTL จะทำงานที่ LOW หรือเป็นกระแสซิงค์ (Current sink) และเอาท์พุทของ TTL จะต่ออยู่กับขา Cathode ของ LED วงจรที่ต่อเกทเป็นแหล่งจ่ายกระแสเมื่อ เอาท์พุทเป็น High จะทำให้ LED ทำงานส่วนที่เกทที่เป็นกระแสซิงค์จะทำให้ LED ไม่ทำงาน รูป 3.31 เป็นตัวอย่างการประยุกต์ใช้กับวงจรลิเนียร์ ซึ่งในรูปแรกเป็นดิฟเฟอร์เรนเชียลเอมป์ลิไฟเออร์ (Diff.-amp) จากวงจรในรูป A นี้เมื่อ LED ได้รับการไบแอสตรงจะทำให้เกิดกระแสเพสในวงจร ในรูปวงจร B เป็น Feedback amplifier ซึ่งคุณสมบัติคือ มีอินพุทอิมพีแดนซ์ต่ำ ในรูปวงจร C เป็นวงจรขยายที่มีความเร็วสูง



รูปที่ 3.30 การใช้ออพติคัลไอโซเลเตอร์ในฟังก์ชันลอจิก



รูปที่ 3.31 การประยุกต์ใช้ออพติคัลไอโซเลเตอร์ในวงจร ลิเนียร์



รูปที่ 3.32 วงจรการใช้ TTL ขับออพโต้ไอโซเลเตอร์

บทที่ 4

ซอฟแวร์

4.1 ภาษา C

4.1.1 ประวัติภาษา C

ในปีพ.ศ.2515 เคนนิธ ริทชี่ (Dennie Ritchie) ก็ได้พัฒนาภาษา C ขึ้นที่ห้องปฏิบัติการเบล ที่เมืองรีอิล มลรัฐนิวเจอร์ซี่ได้นำเอาหลักการของภาษา B ที่พิจารณาโดย เคน ทอมสัน (Ken Thomson)โดยที่ทอมสันให้ชื่อเป็นBCPT(Basic combined Programming Language) การพัฒนาของริทชี่จึงให้ชื่อภาษาว่า C เห็นจะมาจาก C ซึ่งเป็นตัวอักษรต่อจาก B และ C ก็เป็นตัวที่สองต่อมาจาก B ในคำ BCPL เช่นกัน

ภาษา Cและโปรแกรมจัดระบบงานยูนิกซ์ เป็นเสมือนของที่เกิดขึ้นร่วมกันในคอนแรกจากการพัฒนาของทอมสัน และริทชี่ ผู้ร่วมกันบุกเบิกยูนิกซ์ จนในปัจจุบันยูนิกซ์ได้แพร่หลายและพัฒนาไปอย่างมากมายภาษา C เป็นคอมไพเลอร์ภาษาที่มีอยู่บนเกือบทุกโปรแกรมจัดระบบงาน และมีบนเครื่องแม้กระทั่งไมโครคอมพิวเตอร์ขนาดเล็ก 8 บิต ไปจนถึงมินิเมนเฟรมและแม้กระทั่งเครื่องคอมพิวเตอร์ซูเปอร์เมนเฟรม เช่น เคลย์วัน (Cray-1) ก็ยังมีภาษา C

ภาษา Cมีชื่อเล่นในเรื่องการพัฒนาจนทำให้ภาษา Cเป็นภาษาสำหรับนักพัฒนาโดยเฉพาะ\

ข้อดีของภาษา C

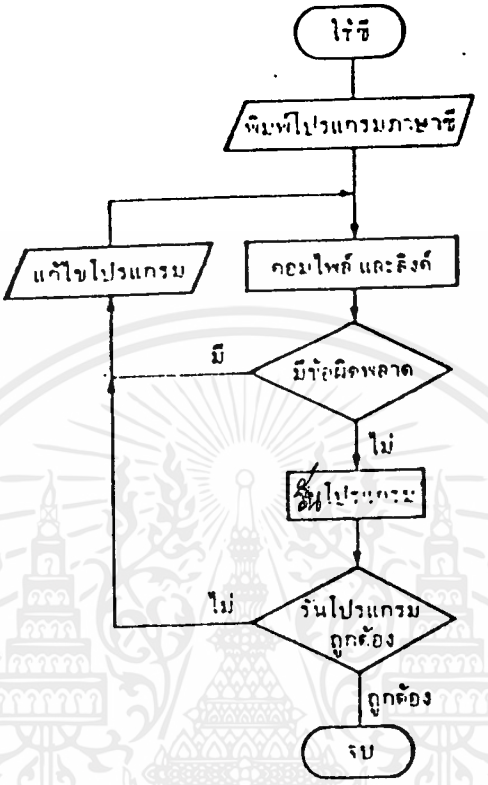
1. เป็นภาษาคอมพิวเตอร์ที่มีการพัฒนาขึ้นเพื่อใช้งานเพื่อให้เป็นภาษามาตรฐานที่ไม่ขึ้นกับโปรแกรมจัดระบบงานหรือขึ้นกับฮาร์ดแวร์ ภาษา C จึงเป็นภาษาคอมพิวเตอร์ตามอุดมการณ์ของนักคอมพิวเตอร์
2. เป็นภาษาคอมพิวเตอร์ที่อาศัยหลักการที่เรียกว่าโปรแกรมโครงสร้างจึงเป็นภาษาที่เหมาะสมสำหรับงานพัฒนา

3. เป็นคอมไพเลอร์ที่มีประสิทธิภาพสูง ให้รหัสออบเจคสั้น ทำงานได้รวดเร็ว เหมาะกับงานพัฒนาที่ต้องการความรวดเร็วเป็นสำคัญ
4. มีความอ่อนตัวของภาษาคล้ายแอสเซมบลี้ ภาษา C สามารถเขียนแทนภาษาแอสเซมบลี้ได้ ดีกันหาที่ผิดหรือแก้ไขตัวโปรแกรมได้ง่ายกว่าภาษา C จึงเป็นภาษาระดับสูงที่ใช้งานได้เหมือนกับภาษาระดับต่ำ
5. มีความคล่องตัวที่จะประยุกต์เข้ากับงานต่าง ๆ ได้เป็นอย่างดี การพัฒนาโปรแกรม เช่น เวิร์ดโปรเซสซิ่ง สเปรดชีต ดาต้าเบส ฯลฯ มักใช้ซีเป็นภาษาสำหรับการพัฒนา
6. เป็นภาษาที่มีอยู่เกือบทุกโปรแกรมจัดระบบงาน มีบนเครื่องไมโครคอมพิวเตอร์ตั้งแต่ขนาด 8 บิต ไปจนถึง มินิเมนเฟรมคอมพิวเตอร์ แม้แต่เครื่องซูเปอร์คอมพิวเตอร์ เช่น เคลย์วัน ก็ยังมีภาษา C จึงเป็นภาษาที่รวมข้อดีเด่นในเรื่องการพัฒนา จนทำให้ภาษา C เป็นภาษาเหมาะกับผู้สนใจที่อยากจะเรียนรู้ หลักการของภาษา วิธีการโปรแกรมและการพัฒนางาน

ภาษา C ในไมโครคอมพิวเตอร์

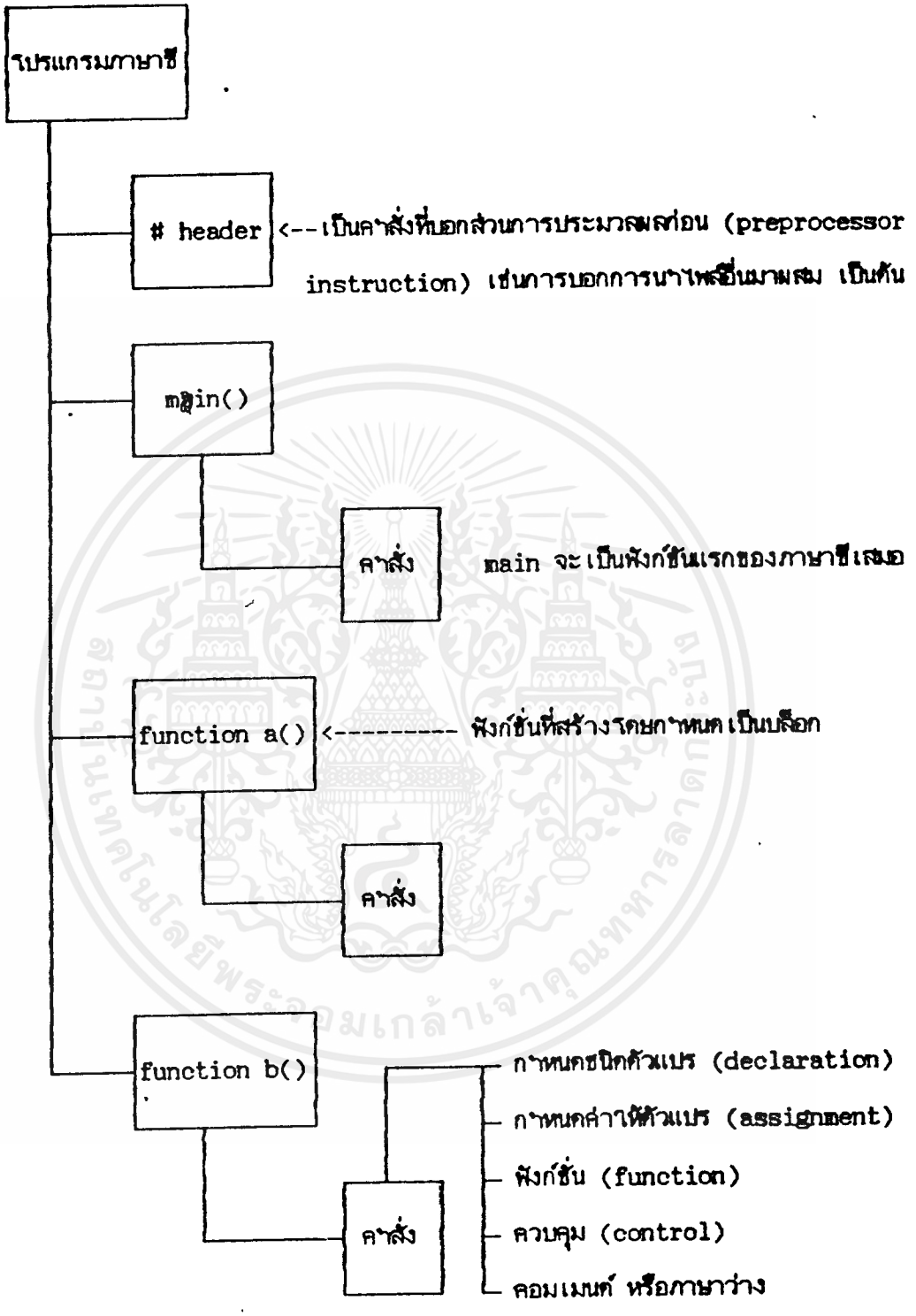
ปัจจุบันทั้งเครื่องไมโครคอมพิวเตอร์ 8 บิตและ 16 บิต ส่วนใหญ่มีคอมไพเลอร์ของภาษา C ให้ใช้ทั้งสิ้นคอมไพเลอร์ที่มีชื่อเสียงในเครื่อง 8 บิตได้แก่ Aztec CBDS C, C/80 Telecom C และ Whitesmiths C ซึ่งความสามารถของภาษา C ดังที่กล่าวแตกต่างกันไปแล้วแต่ผู้พัฒนาได้สร้างขึ้นสำหรับในเครื่อง 16 บิต มีคอมไพเลอร์ ที่มีชื่อเสียงหลายตัวเช่น Lattice หรือไมโครซอฟท์และซีคอมไพเลอร์ของบริษัทดิจิตอลรีเสิร์ช และ Turbo C เป็นต้นภาษา C เหล่านี้อาจมีวิธีการใช้ที่แตกต่างกันบ้างโดยเฉพาะวิธีการคอมไพเลอร์แต่หลักการโดยทั่วไปจะคล้ายคลึงกัน

4.1.2 โครงสร้างภาษา C



รูปที่ 4.1 ผังการใช้คอมไพเลอร์

ขั้นตอนการใช้ภาษา C อาจจะยุ่งยากกว่าภาษาเบสิกในส่วนที่ต้องมีการคอมไพล์และลิงค์ ซึ่งหากโปรแกรมที่เขียนมีขนาดใหญ่ก็จะใช้เวลาในการคอมไพล์มาก เมื่อคอมไพล์เสร็จจะได้ไฟล์ .OBJ คือเป็นไฟล์ออบเจคที่เป็นรหัสและเชื่อมโยงทางการลิงค์กับไฟล์ห้องสมุด(LIB)ก็จะได้ไฟล์ .EXE หรือที่เราเรียกว่า เอ็กซิคิวต์ไฟล์ที่พร้อมสำหรับการใช้งานได้ต่อไป



รูปที่ 4.2 ไดอะแกรมโครงสร้างของภาษาซี

โครงสร้างโปรแกรมภาษาซี มีส่วนสำคัญอยู่ 4 ส่วนคือ

(1) Prepocessor(การประมวลผลเบื้องต้น)คือก่อนที่จะมีการ Compile คอมพิวเตอร์จะกระทำคำสั่งนี้ก่อน

รูปแบบ ต้องขึ้นต้นด้วย เครื่องหมาย "#" เสมอ #.....

define : ใช้สำหรับกำหนดค่าคงที่ให้กับตัวแปรนั้นสามารถเขียนที่บรรทัดไหนของโปรแกรมก็ได้ตัวแปรที่ถูกกำหนดไว้นั้นมีผลบังคับใช้ตั้งแต่คำสั่งนี้ จนสิ้นสุด File # define

x y

ประกอบด้วย 3 ส่วน ส่วนแรก # Define - เป็นคำสั่ง Prepocessor

ส่วนที่ 2 X - เป็นชื่อ ตัวแปร Macro

Y - เป็นค่าหรือสิ่งที่กำหนดให้อาจ

ตัวอย่าง # difine Beep "/007"

difine prase "hello, How are you !"

difine PI 3.1415926

include : เนื่องจากในบางโปรแกรมที่มี # difine จำนวนมากเพื่อเรียกใช้ในหลายโปรแกรม ดังนั้นเพื่อความสะดวกไม่ต้องเรียกมากจึงนำ # difine รวมกันเป็น File หนึ่งเป็น .H แล้วเรียกมาด้วย # include

การใช้งานมีอยู่ 2 ลักษณะ

include (stdio.h) ค้นหา File ใน Directory ของระบบควบคุม

include "Hot.h" ค้นหา File ใน Directory ของผู้ใช้เองก่อน

undef : คำสั่ง เงื่อนไขพิเศษสำหรับโปรแกรมใหญ่ ๆ จะทำการยกเลิกการกำหนดค่าตัวแปรครั้งสุดท้ายของตัวแปรที่กำหนด

ifdef : คำสั่งสำหรับตรวจสอบตัวแปรค่าที่ต้องการตรวจสอบเช่น # ifdef Big ก็จะตรวจสอบสอบค่าของ Bigว่ามีการกำหนดเอาไว้ก่อนด้วยคำสั่ง Prepocessor หรือไม่ ถ้ามีก็จะทำสั่งต่าง ๆ ที่อยู่ถัดมาก่อนทันที ซึ่งอยู่ก่อน # else

else : แต่ถ้าไม่มีการกำหนดค่าไว้ก่อน ก็จะทำตามคำสั่งที่อยู่หลัง # else จนถึง # endif ทันทีเช่นกัน

(2) Main() เป็นฟังก์ชั่นที่ต้องมีในทุกโปรแกรมภาษา (คือการแสดงว่ามี Argument บอกค่าตัวแปรที่ผ่านค่าเข้า-ออกมีส่วนประกอบคือ

{ } กำหนดบล็อกของฟังก์ชั่น สามารถมีหลายบล็อกได้โดยจำนวนเครื่องหมาย {ต้องเท่ากับ}

Heading Function name and argument (ชื่อฟังก์ชั่น) ตั้งตามหลักการตั้งชื่อตัวแปร และฟังก์ชั่นท้ายชื่อต้องมีคู่วงเล็บ () เสมอแม้จะไม่มี argument ในวงเล็บและต้องไม่ปิดท้ายด้วย ; เพื่อแสดงให้ Compiler ทราบว่า เป็นการสร้างฟังก์ชั่นจะมีหรือไม่ก็ได้ใช้สำหรับการผ่านค่าระหว่างฟังก์ชั่น ทำให้เกิดฟังก์ชั่นแบบต่าง ๆ ชื่อที่อยู่ในวงเล็บคือ ชื่อตัวแปรจะมีกี่ตัวก็ได้ให้เยกกันด้วยเครื่องหมาย , การกำหนดให้ตัวใดอยู่หลังขึ้นอยู่กับผู้เขียนโปรแกรม จะกำหนด ส่วนมากเขียนตามลำดับที่นำตัวแปรไปใช้งาน Argument Declaration (การกำหนดตัวแปร) หากมี Argument คือได้กำหนดชื่อตัวแปรไว้ในวงเล็บจะต้องกำหนดแบบข้อมูลให้ตัวแปรให้ส่วนนี้

ตัวอย่าง Charcount (Ch, count)

Char Ch ;

int Count ;

หรือ Starbal () ;

(3) Function Body (ตัวฟังก์ชั่น) คือส่วนที่อยู่ภายในของเครื่องหมาย {} มี 2 ส่วนย่อย คือ

- Declar Local Variable (การกำหนดตัวแปร) เป็นการกำหนดแบบข้อมูลให้แก่แต่ตัวแปรเพื่อใช้ภายในตัวฟังก์ชั่นนั้นฟังก์ชั่นอื่น จะไม่รู้จักตัวแปรที่อยู่ต่างฟังก์ชั่นกัน ทำให้สามารถใช้ชื่อตัวแปรซ้ำกันได้ ในต่างฟังก์ชั่น แม้จะชื่อเดียวกันแต่ต่างฟังก์ชั่นจะเป็นตัวแปรคนละตัวกัน การที่ให้เรากำหนดตัวแปรก่อนใช้งาน ทำให้ขจัดข้อผิดพลาดจากการเขียนชื่อตัวแปรผิดคือเมื่อผิด Compiler จะตรวจพบและแจ้งให้ทราบในช่อง Message

(4) / * Comment * / -(ถ้อยแถลง) หลังจากการกำหนดตัวแปรแล้วจะเป็นการเขียน Statement ซึ่งจะมีเท่าไรก็ได้จนจบโปรแกรมของฟังก์ชั่น

ตัวอย่าง {

```
int num ;
pritf ("enter any key ./ n") ;
}
```

จากแผนภูมิโครงสร้าง Turbo C จะมี Function a () และ Function b () แสดงให้เห็นว่าจะมีฟังก์ชั่นได้หลายฟังก์ชั่น โดยฟังก์ชั่นหลักก่อนแล้วที่เหลือเป็นฟังก์ชั่นย่อยๆ ซึ่งเราสามารถจะทำการเรียกฟังก์ชั่นย่อยๆ ได้โดยมีชื่ออ้างอิง เหมือนการตั้งชื่อ main ()

ตัวอย่าง main ()

```
{
dosub () ;
}
/ * hear is subprogram name dosub * /
dosub ()
{
printf (" subprogram demonstration /n");
}
```

4.1.3 ชนิดของข้อมูลและคำสั่ง

ชื่อตัวแปร-ควรใช้สื่อความหมายที่เข้าใจในตัวมันเอง จะยาวเท่าไรก็ได้แต่ Compiter จะแยกความแตกต่างได้ใน 8 ตัวอักษรแรกเท่านั้น ต้องขึ้นต้นด้วยตัวอักษรก่อนและต้องไม่มีเครื่องคณิตศาสตร์เข้ามาเกี่ยวข้อง

ชนิดของตัวแปร (Declaration) ในการใช้งานต้องกำหนดชนิดของตัวแปรก่อนมีดังนี้

Int	เลขจำนวนเต็มปกติ
short	เลขจำนวนเต็มไม่เกิน 2 Byte
Long	เลขจำนวนเต็มขนาด 4 Byte
Unsigned	เลขไม่คิดเครื่องหมาย

Char ชนิดตัวอักษร

Float เลขมีจุดทศนิยม

Double เลขมีจุดทศนิยมมีความยาวมากขึ้น

ฟังก์ชั่น printf () นำค่าจากโปรแกรมส่งมาให้ผู้ใช้โดยผ่านทาง OUTPUT มาตรฐาน การทำงานจะพิมพ์อักขระทุกตัวในเครื่องหมายคำพูด ยกเว้นที่ตำแหน่ง Format นั้นจะนำค่าคงที่ หรือ ค่าของตัวแปรที่เขียนหลังจุลภาค (,) ถัดจากเครื่องหมายคำพูดปิด มาพิมพ์ตาม Format ที่กำหนด

รูปแบบ printf (ข้อความและFormat การพิมพ์ข้อมูล,....); Format การพิมพ์มี

% d เลขฐานสิบ

% o เลขฐานแปด

% x เลขฐานสิบหก

% u เลขไม่คิดเครื่องหมาย

% e เลขวิทยาศาสตร์ เช่น 1.3 e 17

% f เลขมีจุดทศนิยม

การใช้สตริง==> % c ตัวอักษร

% s ข้อความ

ตัวอย่าง ==>

```
/ * print  function * /
main()

{
int num = 13 ;
float PI = 3.1415926
printf(" hello ,word \n");
printf(" conts : % 4 d = hex :\n",31,31);
printf(" The lveky number is % d \n ",13);
printf(" The valve of pi is % f \n ",PI);
}
```

Run แล้วได้ ==> hello,word

Conts : ....31 = hex % ax \n ",31,31);

The lucky number is 13

The value of pi is 3.141593

ฟังก์ชั่น scanf() ==> รับค่าจาก keyboard แล้วนำใส่ตัวแปรให้โปรแกรมเมื่อจะให้ค่าเป็นตัวเลขนำหน้าตัวแปรด้วย ใช้เครื่องหมาย & นำหน้าตัวแปร เมื่อใส่ค่าเป็นข้อความเก็บในตัวแปร ไม่ต้องมี &

รูปแบบ ==> scanf (ข้อความและformat การพิมพ์ข้อมูล....); มีลักษณะคล้ายกับ printf แต่ scanf ต้องกำหนดขนาดของข้อมูลด้วย คือมี Short (h) และ long (l)ไว้หน้า d, o, x

ตัวอย่าง ==>
```
main ()
{
int age ;
float salary ;
char name [30] ;
scanf ("%s, name");
printf ("Enter your age ,salary .\n");
scanf ("% d %f ", & age, & salary);
printf ("%s is %d years old and earn %f\n name,
        age, salary);
}
```

Run ได้ ==>
```
somchai
Enter your age, salary .
24    5000.20
somchai is 24 years old and earn 5000.200195
```

สังเกต : scanf () ใช้ช่องว่าง (space),Tap, Blank เป็นตัวแยกข้อมูลระหว่างตัวแปร

4.1.4 เครื่องหมายและนิพจน์เปรียบเทียบเงื่อนไข (Operator)

Operator & นิพจน์	ความหมาย
<	น้อยกว่า
<==	น้อยกว่า หรือ เท่ากับ
==	เท่ากับ
>	มากกว่า
>=	มากกว่า หรือ เท่ากับ
!=	ไม่เท่ากับ
=	กำหนดค่าให้เท่ากับตัวแปร
x = ++ n	เพิ่มค่า n อีก 1 แล้วค่อยส่งเป็นตัวแปร x
-- n	ลดค่า n ลง 1
x = n ++	นำค่า n มาใส่ใน x ก่อนแล้วเพิ่มค่า n อีก 1
&&	and (และ)
!!	or (หรือ)
!	not (ไม่)

4.1.5 ฟังก์ชั่นควบคุมการทำงาน

ประโยคเงื่อนไข หรือประโยคควบคุมการไหล (Control) สำหรับควบคุมทิศทางการทำงานของโปรแกรมอย่างเป็นระเบียบและมีโครงร่างที่ชัดเจน มี 3 แบบ

(1) ประโยคคำสั่งที่ใช้เงื่อนไขในการตัดสินใจทำงานมี if , switch

(2) ประโยคคำสั่งที่ใช้ควบคุม การทำงานซ้ำ (Loop Control) มี

(3) ประโยคคำสั่งที่ใช้ในการกระโดดไปยังส่วนต่าง ๆ ของโปรแกรมมี goto.

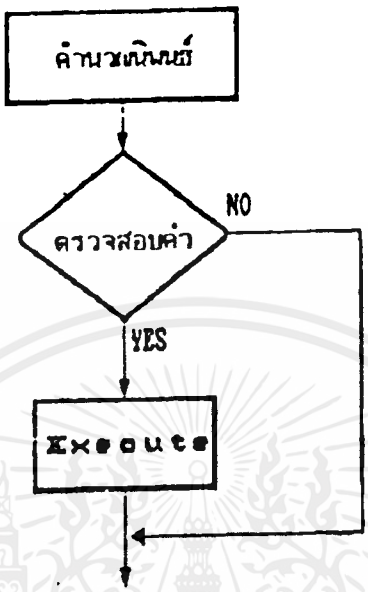
if ประโยคคำสั่งที่ใช้เงื่อนไข แบบง่ายที่สุด

รูปแบบ if (expression)

staement (ทำเมื่อเงื่อนไขถูกต้อง)

การทำงาน ถ้านิพจน์ใน expression มีค่าจริง ==> Execute ประโยคที่ตามมา

ถ้านิพจน์ใน expression มีค่าเท็จ ==> ข้ามประโยคนั้น



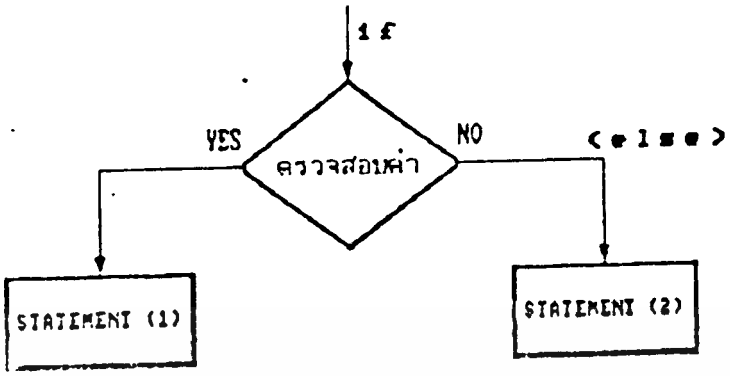
รูปที่ 4.3 โฟลค์ชาร์ตการทำงานของ if

if - else คล้าย if แต่เพิ่มส่วน Execute ในกรณีเงื่อนไขเป็นเท็จ (else)

รูปแบบ if (expression)
statement(1)
else
statement(2)

การทำงาน ถ้า expression เป็นจริง staement ที่อยู่ตามหลังจะถูก Executec และข้าม statement หลัง else ไป

ถ้า expression เป็นเท็จ statement หลัง expression จะถูกข้ามไปและจะ Execute statement หลัง else



รูปที่ 4.4 แสดงโฟลว์ชาร์ตการทำงานของ if - else

เมื่อมีการใช้ประโยค if ซ้อนในประโยค if อีกครั้งให้ส่วนเริ่มปีกกาคั่นแต่ละส่วน

(1) ประโยค if ซ้อนอยู่ภายในของประโยค if - else เรียก "nested if"

ตัวอย่าง ==> if (n > 0)

```
{
 if (n % 2 == 0)
      printf ("postive and even \n ");
}
```

(2) ประโยค if - else ซ้อนประโยค if เรียก "nested if - else"

ตัวอย่าง ==> (n > 0)

```
{
 if (n % 2 == 0)
     printf ("positive and even \n");
 else
     printf ("positive and odd \n");
}
```

Switch การตรวจสอบเงื่อนไขที่มี การเลือกหลาย ๆ ทาง

รูปแบบ switch (expression) ค่าคงที่ (Contant)

```
{
 case 1 cont : statement 1 ;
 case 2 cont : statement 2 ;
               break ;
 case 3 cont : statement 3 ;
               break ;
 defaulf     : statement 4 ;
}
```

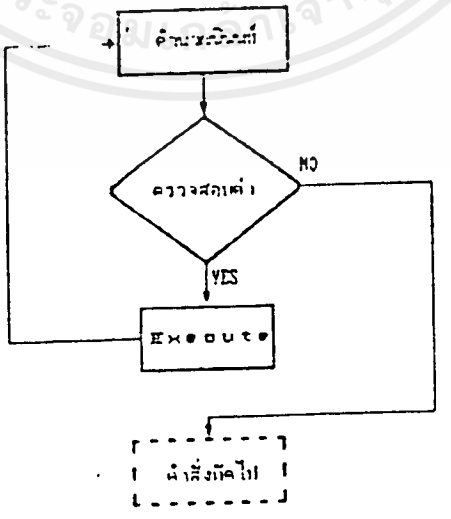
การทำงาน ตรวจสอบค่า expression กับค่าคงที่ (Cont)หลัง case แต่ละตัวถ้า statement ใน case ใด เท่ากันก็จะทำงาน และจบการทำงานสังเกต ในตอนท้ายของแต่ละ case จะมีประโยค break ; ใช้จบการทำงานของคำสั่ง

* กรณีที่เปรียบเทียบ หรือไม่มีเงื่อนไขใน caseใดเป็นจริงเลยจะ Execute ตามคำสั่งหลัง default

While ประโยคคำสั่งควบคุมการทำงานซ้ำ (Loop Control) ใช้ในงานที่ต้องการ Execute ประโยคใด ประโยคหนึ่งซ้ำหลาย ๆ ครั้ง

รูปแบบ While (expression)

```
     statement ;
```



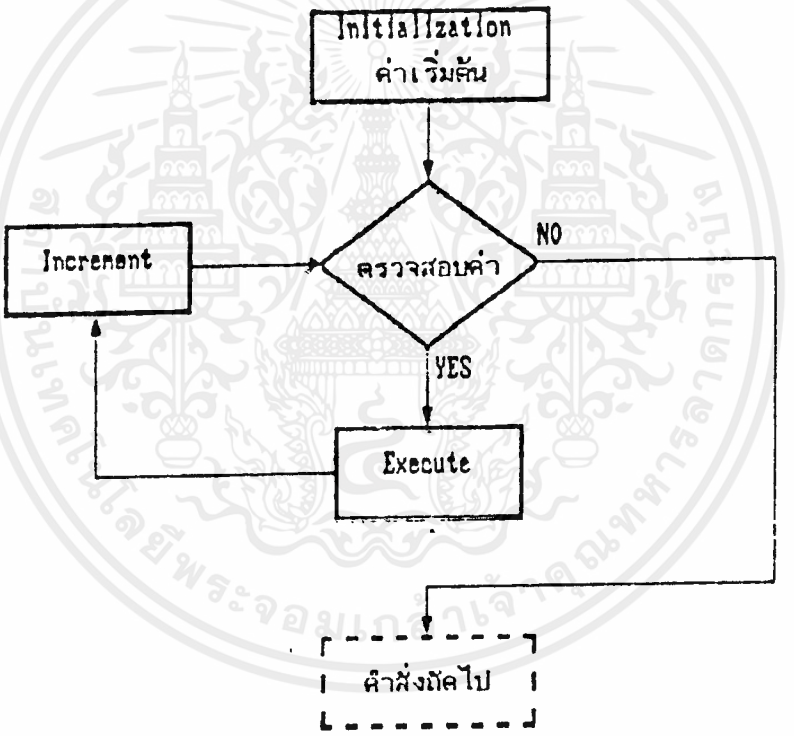
รูปที่ 4.5 แสดงโฟลว์ชาร์ตการทำงานของ while

การทำงาน ตรวจสอบเงื่อนไข expression ถ้าจริงจะทำงาน statement แล้วกลับไปคำนวณค่าและตรวจสอบค่า expression ใหม่วนไปเรื่อย ๆ จนเงื่อนไข expression เป็นเท็จ จึงจะจบ Loop การทำงานแล้วไปทำตามคำสั่งถัดไปต่อ

For ประโยคควบคุมการทำงานซ้ำ (Loop Control)คล้าย while แต่ต่างกันที่ For มีการกำหนด (ตัวแปรเริ่มต้น; คำสั่งเงื่อนไข; ตัวแปรที่เปลี่ยนในแต่ละรอบ) ลักษณะเด่นคือมีรูปแบบที่อ่านง่าย

รูปแบบ For (Initialization; Condition; Increment)

Statement ;



รูปที่ 4.6 แสดงโฟลว์ชาร์ตการทำงานของ For

การทำงาน เมื่อกำหนดค่าตัวแปรเริ่มต้น (Initialization) และตรวจสอบค่าเงื่อนไขถ้าเป็นจริงจะทำงานคำ Statement แล้ววนต่อไปที่ Increment เพื่อเปลี่ยนแปลงค่าของตัวแปร เสร็จแล้วเข้า Loop ตรวจสอบค่าอีก

- ถ้าการตรวจสอบเงื่อนไข ยังเป็นจริงอยู่ก็จะวน Loop เหมือนเดิม
- แต่ถ้าการตรวจสอบเงื่อนไขเป็นเท็จ ก็จะจบการรวน Loop

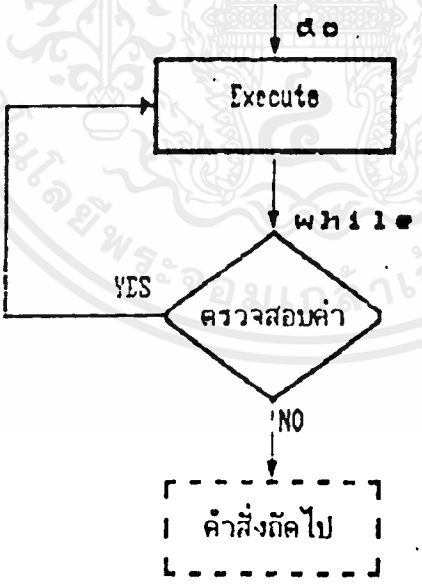
ตัวอย่าง ==> main ()

```
{
int x ;
for(x=1; x<=100; x++) printf ("%d",x);
}
```

Do While ประโยคคำสั่งเงื่อนไข ควบคุมการทำงานซ้ำ (Loop Control) เช่นเดียวกันกับ For และ While แต่แตกต่างกันที่คำสั่ง For,While จะทำงานตรวจสอบเงื่อนไขก่อนที่จะเข้า Loop ส่วนคำสั่ง Do while จะเข้า Loop ก่อน แล้วจึงทำการตรวจสอบเงื่อนไข

รูปแบบ Do

```
{
  Statement
}
  While (expression);
```



รูปที่ 4.7 แสดงโฟลว์ชาร์ตการทำงานของ Do while

การทำงาน จะทำการ Execute ตาม Statement ก่อนเป็นการเริ่มต้นเข้าสู่ Loop แล้วค่อยตรวจสอบค่า

- ถ้าเป็นจริงจะกลับไปทำตาม Statement ใหม่
- ถ้าเป็นเท็จก็จะออกจาก Loop ไปทำคำสั่งถัดไป

Goto เป็นประโยคคำสั่งให้มี การกระโดดไปยังจุดใดจุดหนึ่ง ในโปรแกรมทันที

รูปแบบ Sub : .

.....

.....

Goto sub ;

ปกติ Sub เป็น identifier ซึ่งมีความหมาย ; อยู่ท้ายชื่อในส่วนของโปรแกรมที่ต้องการให้คำสั่ง goto โดดไป Execute ที่ประโยคถัดจาก Label

Break เป็นคำสั่งให้จบการทำงานหรือออกจาก Loop ของ คำสั่ง for, while, do-while และจะกระโดดไป execute คำสั่งถัดไปนอก Loop

Continue คำสั่งให้กลับไปวนรอบ (Loop) ต่ออีก เมื่อการทำงานพบ continue เครื่องจะไม่ Execute คำสั่งหลัง continue แต่จะกลับวน Loop ต่อไปใหม่

* คำสั่ง continue จะไม่มีความหมายเมื่อใช้อยู่ในคำสั่ง switch เนื่องจากประโยค Switch ไม่มีการทำงานซ้ำ

4.1.6 อินพุต/เอาต์พุตในเครื่องคอมพิวเตอร์

การทำงานที่เกี่ยวกับข้อมูลที่เป็นบิต

บางโปรแกรมจำเป็นต้องกระทำกับข้อมูลเป็นบิตในไบท์ หรือเป็นเวิร์ดภาษาซีมีวิธีการจัดการกับข้อมูลเป็นบิตได้ โอเปอเรเตอร์เกี่ยวกับบิต ในการประมวลผลด้วยตัวเลขไบนารี่ ภาษาซีใช้ตัวแปร หรือค่าคงตัวที่เป็นตัวเลขจำนวนเต็มเช่นใช้ตัวเลขไบนารี่ (00011001)เราจะใช้ 25 เป็นตัวเลขฐานสิบ และเลขฐานแปดเป็น 031 หรือ เลขฐานสิบหก เป็น 0*19 แต่เราจะใช้ตัวเลขเพียง 8 บิต หรือ 1 ไบท์ เป็นตัวอย่างตัวโอเปอเรเตอร์ที่ใช้เป็นบิตมีหลายตัวเช่น

~ : one's complement

โอเปอเรเตอร์นี้เป็นแบบที่ใช้กับโอเปอเรเตอร์ เพียงตัวเดียว เช่น

~ (10011010) == (01100101)

& : Bitwise AND

โอเปอเรเตอร์นี้จะกระทำการ AND กับบิตต่อบิต ขององตัวเลข ดังตัวอย่าง เช่น

(10010011) & (00111101) == (00010001)

: : Bitwise OR

โอเปอเรเตอร์นี้จะกระทำการ OR กับบิตต่อบิตของตัวเลข ในโอเปอเรเตอร์ทั้งสอง เช่น

(10010011) : (00111101) == (10111111)

^ : Bitwise EXCLUSIVE OR

โอเปอเรเตอร์นี้จะกระทำการ EX OR กันบิตต่อบิต เช่น

(10010011) ^ (00111101) == (10101110)

ตัวอย่างการใช้ในภาษาซี

สมมติว่าต้องการตรวจสอบว่า flags ซึ่งเป็นตัวแสดงสถานะโดยใช้บิต 1 (เมื่อขวาสุดเป็นบิต 0) เราใช้การ AND เพื่อนำมาตรวจสอบได้ โดยตั้ง MASK มีค่าเป็น 00000010 เมื่อมา AND กับ flags ถ้าได้ 0 ก็แสดงว่า flags บิตนั้นเป็น 0 ถ้าผลลัพธ์ไม่ใช่ 0 ก็แสดงว่าเป็น flags เป็น 1

```
#define MAKS 2
:
:
flags = flags & MASK
```

โอเปอเรเตอร์การเลื่อนบิต

<< : left Shift

โอเปอเรเตอร์นี้จะนำค่าทางซ้ายมือมาเลื่อนจำนวนบิตไปทางซ้ายจำนวนบิตที่เลื่อนจะกำหนดด้วยค่าที่อยู่ทางขวามือของโอเปอเรเตอร์เช่น (10001010) <<2 == (00101000) ผลลัพธ์เลื่อนบิตไปทางซ้าย 2 บิต

>> : Right Shift

เป็นการเลื่อนโตไปทางขวาเช่นเดียวกับการเลื่อนมาทางซ้าย ดังตัวอย่างเช่น

(10001010) >>2 == (00100010)

00100010

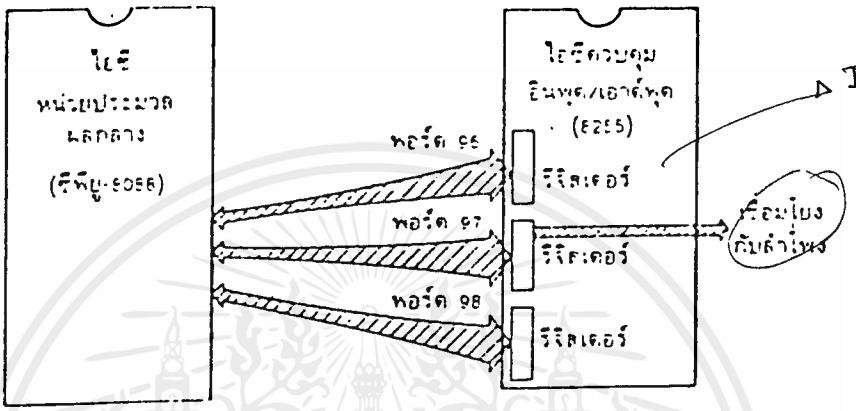
ไมโครคอมพิวเตอร์หลายเครื่องที่ใช้กันอยู่โดยทั่วไป จะมีหน่วยประมวลผลกลาง (ซีพียู-Central Processing Unit)โดยใช้ไอซีเบอร์ 8088 หรือ 8086 เครื่องไมโครคอมพิวเตอร์ที่รู้จักกันดีโดยทั่วไปคือ เครื่อง IBM-PC มีซีพียู 8088 ซึ่งทำหน้าที่เป็นสมองหลักของเครื่อง โครงสร้าง หรือฮาร์ดแวร์ส่วนใหญ่ของเครื่องคอมพิวเตอร์เครื่องใดเครื่องหนึ่ง มักจะเกี่ยวโยงกับซีพียูของเครื่องนั้นเป็นสำคัญในตอนนี้จึงขอกล่าวถึง ลักษณะของการส่งผ่านข้อมูลอุปกรณ์อินพุต / เอาต์พุต สำหรับระบบที่ใช้ซีพียูในตระกูล 8088/8086 และจะยกตัวอย่างเป็นกรณีศึกษาโดยเฉพาะบนเครื่องพีซี IBM-PC นี้

ซีพียูในเครื่องคอมพิวเตอร์โดยทั่วไปจะต้องมีการทำงานร่วมกับอุปกรณ์รับ/ส่งข้อมูลหลาย ๆ ตัว เช่น คีย์บอร์ด ลำโพง หรืออาจเป็นเทปคาสเซ็ท ดิสก์ จอภาพ หน่วยความจำภายในวงจรควบคุมเวลา หรืออาจเป็นซีพียูตัวอื่นที่นำไปใช้ควบคุมการทำงาน หรือการส่งผ่านข้อมูลเราสามารถที่จะแบ่งชนิดของข้อมูลที่ติดต่อสัมพันธ์กับซีพียูเป็น 2 ใหญ่ ๆ กล่าวคือกลุ่มแรกเป็นกลุ่มของหน่วยความจำ โดยซีพียูจะกำหนดตำแหน่ง (address) ของหน่วยความจำในตำแหน่งที่ต้องการเองในกลุ่มที่ 2 เป็นกลุ่มของอุปกรณ์อินพุต/เอาต์พุตโดยที่ซีพียูจะกำหนดตำแหน่งของอุปกรณ์เองว่าซีพียูจะติดต่อกับอุปกรณ์อินพุตหรือเอาต์พุตตัวไหนตำแหน่งของอุปกรณ์อินพุต/เอาต์พุตนี้จะถูกเรียกว่าพอร์ต (port)สำหรับซีพียู 8088 นี้จะกำหนดพอร์ตได้ถึง 65536 พอร์ต ยกตัวอย่างเช่นในระบบคอมพิวเตอร์เครื่องหนึ่งอาจจะมีการกำหนดวงจรฮาร์ดแวร์ ทำให้พอร์ตหมายเลข 992,993 และ 1000-1004 เป็นพอร์ตสำหรับการติดต่อกับวงจรในการควบคุมการทำพิกัดสีของจอภาพ ส่วนพอร์ตที่ 97 จะติดอยู่กับลำโพง (ตัวอย่างของเครื่อง IBM-PC) เป็นต้น

พอร์ตหมายเลข 97 นี้จะมีวงจรอิเล็กทรอนิกส์ที่ต่อกับลำโพง โดยผ่านทางไอซีควบคุมตัวหนึ่งเบอร์ 8255 ไอซีตัวนี้เป็นไอซีพิเศษที่ทำหน้าที่ในการรับคำสั่งหรือข้อมูลเพื่อให้มีการเชื่อมโยงระหว่างตัวซีพียูและอุปกรณ์อินพุต/เอาต์พุตภายนอกได้ ภายในตัวไอซี 8255 นี้มีหน่วยความจำย่อยสำหรับเป็นที่เก็บคำสั่งข้อมูลหรือสถานะต่าง ๆ ที่อาจจะถูกส่งมาจากซีพียูได้หน่วยความจำย่อยนี้เรียกว่า รีจิสเตอร์ (register)

ไอซี 8255 นี้จะมีรีจิสเตอร์สำคัญภายในอยู่ 3ตัวข้อมูลหรือตัวเลขที่อยู่ในรีจิสเตอร์เหล่า

นั้นจะบอกหรือควบคุมสถานะของการทำงาน รีจิสเตอร์สำคัญตัวหนึ่งที่ต่อเชื่อมโยงกับซีพียูเป็นพอร์ต หมายเลข 97 ของซีพียู รีจิสเตอร์พอร์ต 97 นี้เหละที่จะถูกนำไปเชื่อมโยงกับลำโพงอีกทีหนึ่ง ดังนั้นถ้าเราสามารถเขียนโปรแกรมในภาษาซี เพื่อจะทำการส่งข้อมูลมายังพอร์ตที่ 97 นี้เรา สามารถส่งสัญญาณนี้ไปยังลำโพงได้โดยการควบคุมให้เกิดเสียงตามโปรแกรมได้เช่นกัน



รูปที่ 4.8 แสดงลักษณะเชื่อมโยงของซีพียูกับอินพุต/เอาต์พุต โดยผ่านทางไอซีควบคุม อินพุต / เอาต์พุตเบอร์ 8255 ซีพียูกำหนดตำแหน่งของอินพุต/เอาต์พุตโดยกำหนดเป็นหมาย เลขของพอร์ต (port)

คำสั่งการรับ/ส่งข้อมูลทางพอร์ตในภาษาซี

(1) outportb (int port,char byte); ซีพียูจะส่งข้อมูลไปยังพอร์ตเอาต์พุต input คือค่าตัวเลขจำนวนเต็มของเบอร์ port ที่ต้องการส่ง (Char byte)เอาต์ออกไปการ ใช้งานต้องเรียกจาก # include <dos.h>

(2) inportb (int port); ซีพียูจะรับข้อมูลจากพอร์ตอินพุต input port คือตัวเลขจำ นวนเต็มของเบอร์พอร์ตที่ ต้องการให้ข้อมูลที่พอร์ตเบอร์ นั้นผ่านเข้ามาการใช้งานต้อง เรียกจาก # include <dos.h>

ตัวอย่างโปรแกรมควบคุมลำโพงให้มีเสียงดัง

```
# include <dos.h>
main ()
```

```
{
   int x ;
   x = inportb (97); /* อ่านข้อมูลจาก port 97 เก็บใน x * /
   printf ("port 97 = % \n",x); / * พิมพ์ผลลัพธ์ * /
   outportb (97,79); / * ส่งข้อมูล 79 ไป port 97, ลำโพงดัง * /
   outportb (97,x);  / * ส่งข้อมูลเดิมกลับไป port 97, ลำโพงหยุดดัง * /
  }
```

นอกจากนี้ยังมีคำสั่งอื่น ๆ ที่เกี่ยวข้อง เช่น

```
sleep (unsigned seconds);
```

เมื่อ โปรแกรมทำงานจนพบคำสั่งนี้จะทำการหยุดโปรแกรมชั่วขณะหนึ่ง เป็นเวลาตามที่ตั้งไว้ในวงเล็บคือค่าเวลาที่เป็นวินาที

4.2 Turbo C Version 1.5

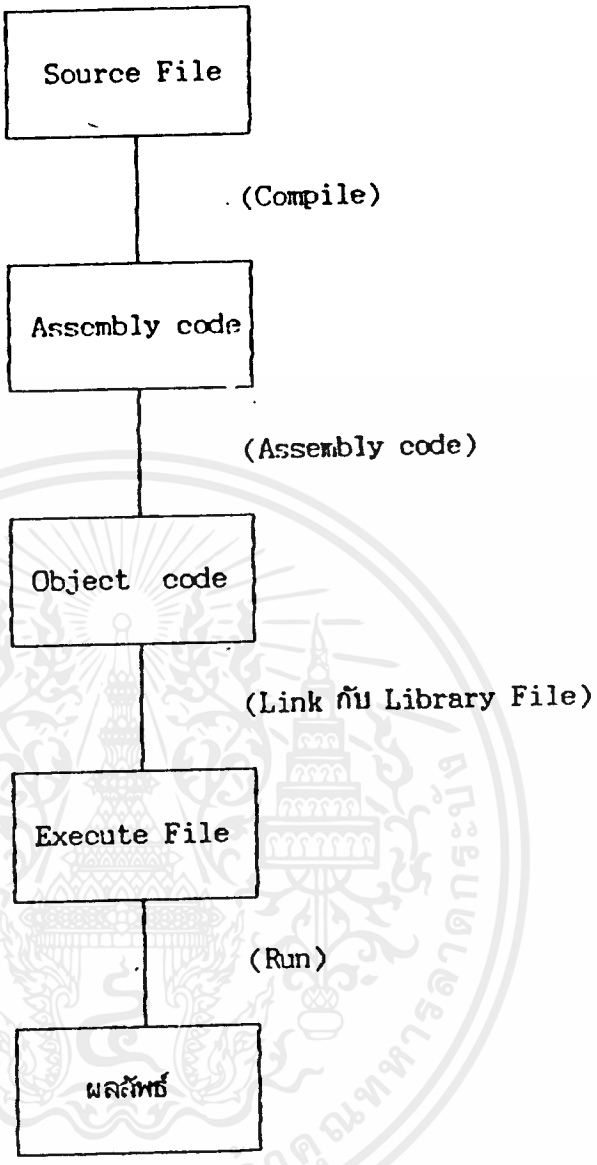
Turbo C Ver 1.5 ผลิตโดย Borland มี Turbo เป็นตัว Editor เช่นเดียวกับ Turbo Pascal และ Turbo Prolog ปัจจุบัน Turbo C ผลิตถึง Version 2.0 แล้ว Turbo C เป็น C ตามมาตราฐาน ANSI (Ammerican National Standaed Institude) สามารถนำ Source code ไปทำการ Compiler ด้วย compiler ตัวอื่นบนเครื่องอื่นหรือเป็นแบบ Turbo ซึ่งมีฟังก์ชั่นสำหรับใช้งานเครื่อง IBM PC เป็นพิเศษ

Type of Compiler

โดยทั่วไป สามารถทำการ Compile ได้ 2 ลักษณะ คือ

(1) Command Line Version เป็นแบบทั่ว ๆไป คือ ต้องสั่งให้ Compile และ Line แยกกันเหมือน C Compiler ตัวอื่น ๆ

(2) Integrated Environment Version เป็นลักษณะของ Turbo คือจะทำการ Compile และจัดการ Link เพื่อทำให้ได้ Execute Program (โปรแกรม : EXE ซึ่งสามารถนำไป Rum ได้เลย)



รูปที่ 4.9 แสดงการ Compile แบบ Integrated Environment Version

ข้อกำหนด

เครื่อง Microcomputer ที่สามารถ Run Turbo C ได้

- IBM PC ,XT ,AT รวมถึงเครื่อง IBM Compatibles
- DOS Version 2.0 หรือสูงกว่า
- Ram อย่างต่ำ 384 K byte
- 80 Column Momitos

โปรแกรม Turbo C บรรจุ Disk 4 แผ่น มี

Disk 1 : The Integrated Environment

Disk 2 : Command - Line Version & Utilities.

Disk 3 : Include Files & Libs I

Disk 4 : Libs II

แบบของการ Compile แบ่งตามขนาดของโปรแกรมสามารถเลือกได้ 5 แบบ

1. Tiny - โปรแกรมขนาดจิ๋ว
2. Small - ใช้งานกับโปรแกรมที่มีขนาดไม่เกิน 64 K bytes.
3. Mediam - โปรแกรมด้านการคำนวน ซึ่งมีตัวโปรแกรมใหญ่
4. Compact - เหมาะกับโปรแกรมด้านธุรกิจ ที่โปรแกรมเล็กแต่ข้อมูลมาก
5. Large - เหมาะกับโปรแกรมขนาดใหญ่
6. Hyge - โปรแกรมขนาดใหญ่โตมาก ๆ

การใช้งาน แบบ Integratead Enviromment แบบ Small Type < 64 Kbyte แผ่นโปรแกรม 4 แผ่นที่ได้มาเราจะเก็บไว้เป็นต้นฉบับและจะเตรียม Diskette อีก 2 แผ่น เพื่อไว้ใช้งาน

แผ่นที่ 1. เป็น Program disk แผ่นนี้ควรจะมี System ของ DOS ด้วย

แผ่นที่ 2. เป็น Work disk

เตรียม Diskette แผ่นที่ 1 (Program disk) หลังจากเปิดเครื่องเข้าสู่ DOS V 3.1 แล้ว ใส่แผ่น ที่ 1 ใน Drive B:

A> Format b: /s (format แผ่นที่ 1 ให้มี System of Dos ด้วย) Copy ไฟล์ที่ต้องการลงใน Program disk

- ใส่แผ่นโปรแกรม 1 (Integrated Environment) ลงใน drive A:
- ใส่แผ่นใช้งาน 1 (Program disk) ใน drive B:

A> Copy TC.EXE B:

A> Copy TCHELP.TCH B:

เตรียม Diskette แผ่นที่ 2 (Work disk)

A> Format B:

- ใส่แผ่นโปรแกรม 2 (Include files & Lid 1) ใน drive A:
- ใส่แผ่นใช้งาน 2 (Work disk) ใน drive B:

```
B> md INCLVED
B> cd INCLVED
B> copy A: *.h B:
B> copy A: / sys/STAT.h B:
B> cd\
B> md LIB
B> cd LIB
B> copy A: COS.OBJ B:
B> copy A: EMU.LIB B:
B> copy A: FP 87 .lIB B:
B> copy A: MATHS.LIB B:
B> copy A: CS.LIB B:
B> cd \.
```

เท่านี้ก็เสร็จสิ้นการเตรียม disk ใช้งาน 2 แผ่นเข้าสู่ระบบใส่แฟ้น Program ใน drive A: รัน Turbo C โดย

A> TC.

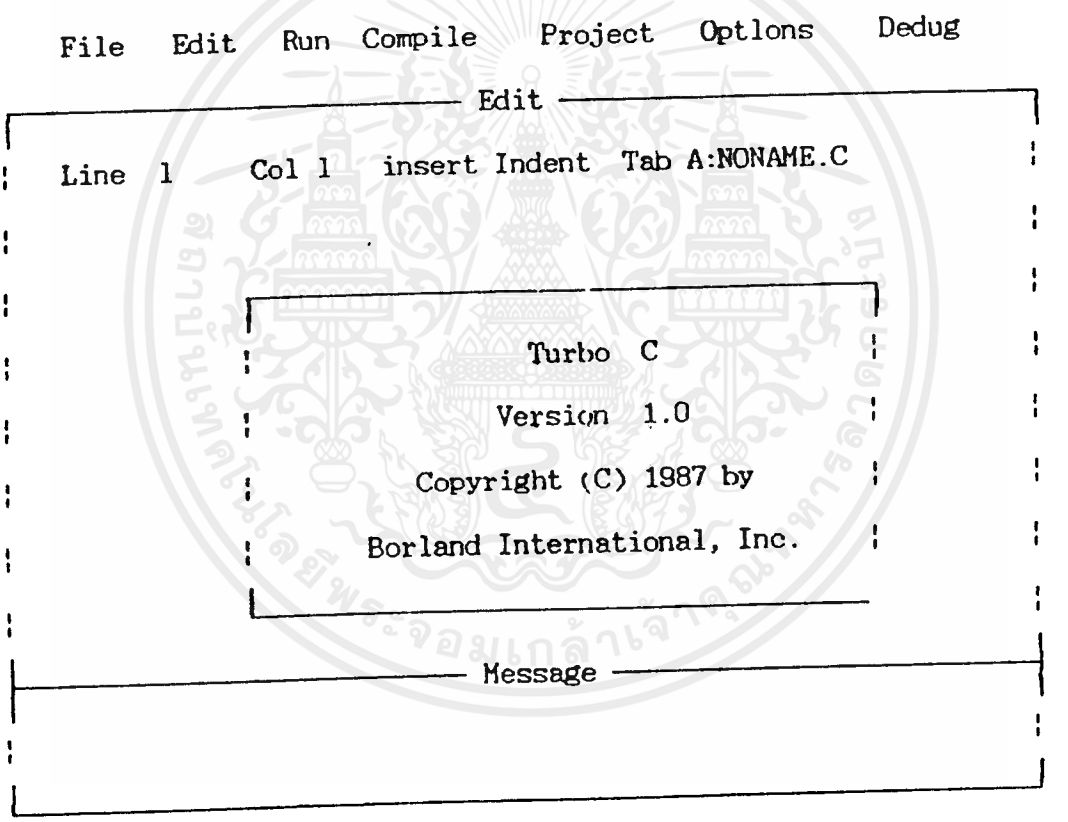
เครื่องจะ Load โปรแกรม TC.EXE นานพอสมควร จะได้ Menu แสดงอักษรผู้ผลิตบริษัทบอร์แลนด์หลังจากกดคีย์ใดก็ได้ อักษรโชร์บริษัทจะหายไปจะได้จอภาพมี 4 ส่วนหลัก ได้แก่

- Main Menu บรรทัดบนสุดแสดงคำสั่งหลักในการใช้งาน 7 ชนิดคือ File, Edit,

Run, Compile, Project, Option, Debug

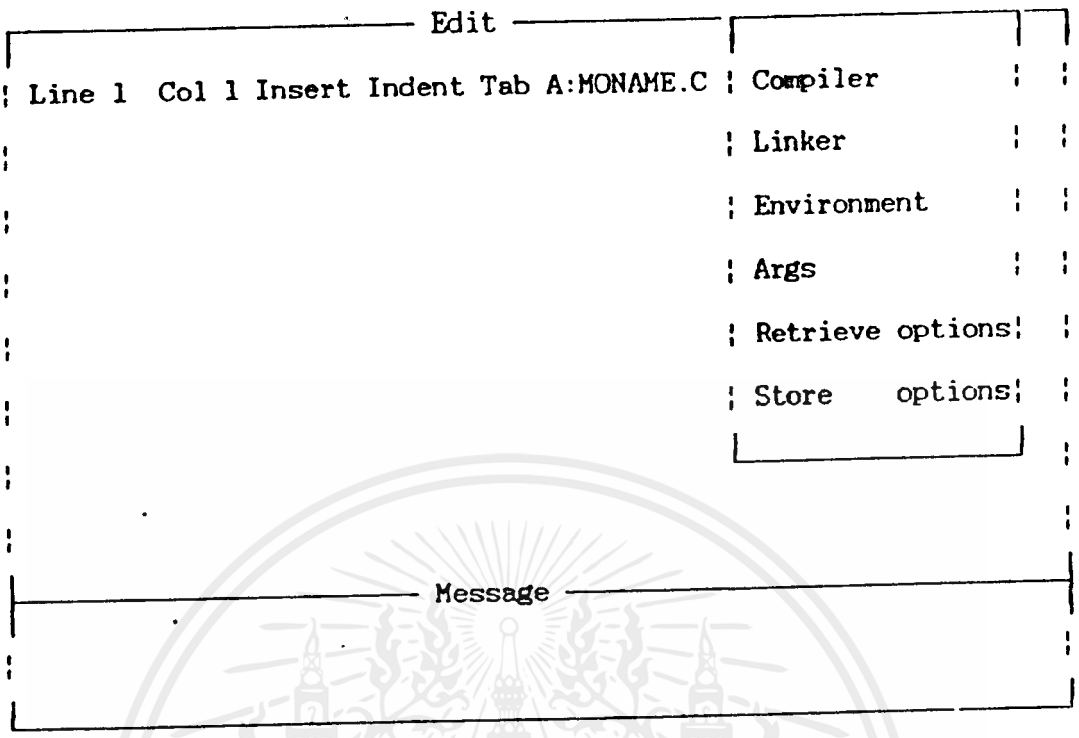
มีวิธีการเลือกใช้งาน 3 วิธีคือ

1. กดตัวอักษรตัวแรกของงาน เช่นเมื่อจะ Run ให้กด R
2. กด Key Movement <== ==> เลื่อนแถบ High Light ไปยังชื่อของงานแล้วกด Enter <==:
3. กดคีย์ ALT พร้อม ๆ กับกดคีย์ตัวอักษรแรกของงานที่ต้องการสำหรับวิธีนี้สามารถเลือกสั่งในระหว่างทำงานอื่นอยู่โดยตรง โดยไม่ต้องผ่าน Main Menu



รูปที่ 4.10 แสดงหน้าจอ Main Menu ของ Turbo C

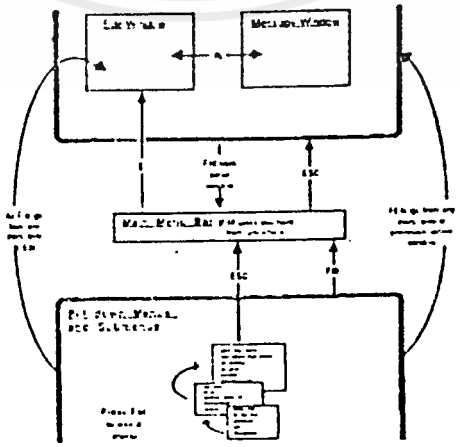
ในแต่ละคำสั่งหลักของ Main Menu จะมีคำสั่งย่อย ๆ ลงไปอีกเรื่อย ๆ ดังรูป



รูปที่ 4.11 แสดง Main Menu ของคำสั่งย่อย ๆ

เมื่อต้องการกลับมาที่ Menu หลักให้กด ESC

- Edit windors แสดงข้อมูลของโปรแกรมที่เขียนรวมถึง Editor จะบอกตำแหน่งต่าง ๆ เช่น Line..... Col ... Insert Indent Tab A: Noname. C.
- Message window ช่องข้อมูลอื่น ๆ เช่น บอกตำแหน่งการ Error
- Quick Refference คีย์ที่ใช้อยู่บ่อย ๆ



รูปที่ 4.12 แสดงตัวอย่างการใช้ Hot Keys

ตารางที่ 4.1 แสดงรายการต่าง ๆ ของ Hot Keys

Key (s)	Function
F1	Brings a Help Window With info about your current Position
F2	Saves the currently in the Editor
F3	Lets you load a file (an input box will appear)
F5	Zooms and unzooms the active window
F6	Switches to the active window
F7	Takes you to the previous error
F8	Takes you to the next error
F9	Preformsa "make"
F10	Invokes the main menu
Alt-F1	Brings up the last Help screen you referenced
Alt-F3	Lets you pick a File to load
Alt-F9	Compiles. OBJ (the file loaded in the Editor)
Alt-F10	Displays the version screen
Alt-C	Takes you to the Compile menu
Alt-D	Takes you to the Debug menu
Alt-E	Puts you in the Editor
Alt-F	Takes you to the File menu
Alt-O	Takes you to the Options menu
Alt-P	Takes you to the Project menu
Alt-R	Runs your program
Alt-X	Quits Turbo Cand takes you to DOS

เมื่อสร้างโปรแกรมแล้วทำการ Run เมื่อต้องการ Error จะโชว์ส่วนที่ผิดพลาด ในแต่ละจุดและ

เมื่อแก้ไขแล้วจึงสามารถ Run ได้ ก็จะได้ ไฟล์เพิ่มอีก 2 ไฟล์คือ

file.obj , file.exe

file.obj คือไฟล์ที่ถูกแปลงเป็นภาษา Assembly

file.exe คือไฟล์ที่ถูกแปลงให้เป็น Machine Language. ซึ่งสามารถนำไป Rum โปรแกรมทาง DOS ได้เลย โดยไม่ต้องผ่านเข้าไป Compile ใหม่

4.3 โหมดกราฟิก

4.3.1 โหมดและชุดสี

ก่อนที่โปรแกรมทางกราฟิกจะถูกเรียกมาใช้งาน เครื่องคอมพิวเตอร์จะต้องถูกปรับให้อยู่ในสภาวะแสดงผลหรือโหมดที่ถูกต้อง และสำหรับ IBM PC แล้ว นั่นหมายถึงการใช้โหมดของจอภาพและชุดสี (mode and palette) ที่เหมาะสม

ตารางที่ 4.2

โหมด	แบบ	ขนาด	อะแดปเตอร์
0	text, b/w	40×25	CGA, EGA
1	text, 16 colors	40×25	CGA, EGA
2	text, b/w	80×25	CGA, EGA
3	text, 16 colors	80×25	CGA, EGA
4	graphics, 4 colors	320×200	CGA, EGA
5	graphics, 4 grey tones	320×200	CGA, EGA
6	graphics, b/w	640×200	CGA, EGA
7	text, b/w	80×25	monochrome
8	graphics, 16 colors	160×200	PCjr
9	graphics, 16 colors	320×200	PCjr
10	graphics, 4 colors PCjr, 16 colors EGA	640×200	PCjr, EGA
13	graphics, 16 colors	320×200	EGA
14	graphics, 16 colors	640×200	EGA
15	graphics, 4 colors	640×350	EGA

วิดีโอโหมดที่สามารถใช้ได้บน IBM PC ดังตารางที่ 4.2 สิ่งสำคัญที่ต้อง ไม่ลืมคือมุมบนซ้ายสุดเป็นตำแหน่ง (0,0) โดยมีแกน X อยู่แนวดิ่งและแกน Y อยู่ในแนวระดับ BOIS อินเทอร์รัพท์หมายเลข 16 ฟังก์ 0 ใช้ในการจัดวีดีโอโหมด และถูกใช้งานในฟังก์ชั่นชื่อ mode ( ) ดังที่แสดง

```
/* กำหนดวีดีโอโหมด */
void mode (int mode_code)
{
        union REGS r;
        r.h.al = mode_code;
        r.h.ah = 0;
        int86(0x10,&r,&r);
}
```

4.3.2 การเขียนจุดสี

ในรูทีนทางกราฟิกพื้นฐานคือการเขียนจุดสี ซึ่งก็คือ การเขียนข้อมูลที่เล็กที่สุดที่สามารถอ้างตำแหน่งได้บนจอภาพสำหรับ IBM PC และ compatible มี 2 วิธีที่สามารถเขียนข้อมูลในจุดสีที่ต้องการวิธีแรกคือการใช้อินเตอร์รัพท์ ROM BIOS ซึ่งเป็นวิธีที่สะดวกที่สุด แต่ความช้าของวิธีนี้ก็เป็นข้อด้อยที่ทำให้คำนึงถึงวิธีใหม่ วิธีที่สองซึ่งเป็นการติดต่อลงไปที่ VIDEO RAM ที่ใช้แสดงผล ซึ่งวิธีนี้จะเป็นวิธีที่ใช้เวลาน้อยกว่าวิธีแรก

มาทำความเข้าใจระบบ CGA/EGA ในกราฟิกโหมด

ใน VIDEO RAM ของ CGA จะเริ่มต้นที่ตำแหน่ง B800:0000H ในหน่วยความจำและ EGA ก็ใช้ตำแหน่งเดียวกันสำหรับทุก ๆ โหมด ที่ compatible กับ CGA (สำหรับข้อมูลโดยละเอียดสามารถหาได้จาก IBM Tecnical Reference) ในโหมด 4

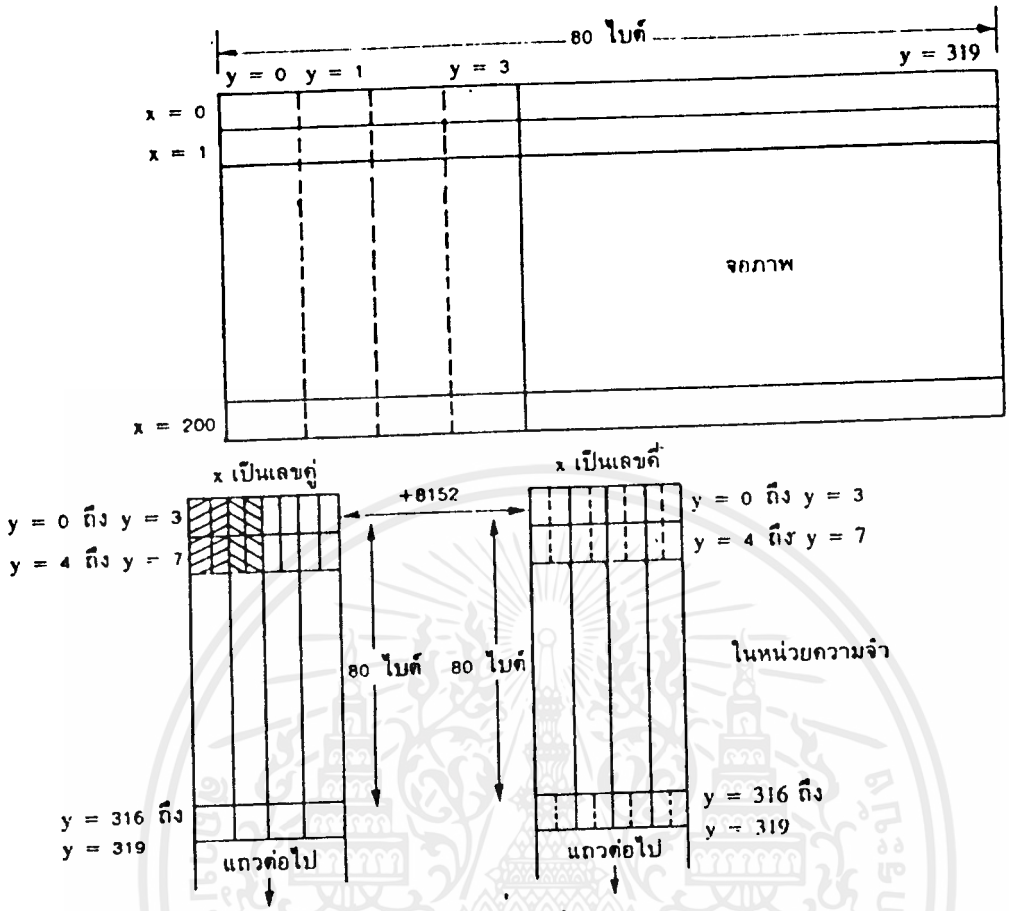
ในแต่ละไบท์ของของหน่วยความจำจะใช้อ้างจุดสี (pixel) 4 จุดโดยแต่ละจุดจะใช้ 2 บิตดังนั้นความละเอียด 320 x 200 ต้องใช้หน่วยความจำขนาด 16K เนื่องจากว่า 2 บิตจะสามารถเก็บข้อมูลได้ 4 รูปแบบดังนั้นจึงมีเพียงแค่สีสีในโหมด 4 ข้อมูลของแต่ละสีจะถูกแสดงดังตารางที่ 4.2

ตารางที่ 4.3

ค่า	สีในชุดสีที่ 0	สีในชุดสีที่ 1
0	blackground	background
1	green	blue
2	red	purple
3	yellow	white

ในจอ CGA มีเรื่องที่น่าสนใจในการจัดแบ่งหน่วยความจำคือ จุดสีที่มีตำแหน่งหมายเลขคู่จะเริ่มต้นที่ตำแหน่ง B800:0000H แต่จุดสีที่มีตำแหน่งหมายเลขคี่จะถูกเก็บที่ตำแหน่ง 8152 ไบท์ที่เหนือขึ้นไปคือประมาณที่ B800:2000H

จากจอภาพในโหมด 4 นี้แต่ละแถวบนหน้าจอจะมี 320 จุดดังนั้นจึงต้องการหน่วยความจำ 80 ไบท์(ไบท์ละสี่จุด) โดยแบ่ง 80 ไบท์สำหรับจุดสีที่มีค่า x เป็นเลขคู่ และสำหรับ x เป็นเลขคี่อีกแถวละ 80 ภายในแต่ละไบท์ จุดสีจะถูกเก็บเรียงตามลำดับจากซ้ายไปขวา ตามที่ปรากฏบนจอภาพนั่นหมายความว่าจุดสีหมายเลข 0 จะอยู่ในบิตที่ 6 และ 7 และจุดสีที่ 3 จะใช้บิต 0 และ 1 ดังรูปที่ 4.1ในการเขียนโปรแกรมที่ต้องติดต่อกับ VIDEO RAMจะสนับสนุนการทำงานของคุณสมบัติที่น่าสนใจของฟังก์ชันที่เขียนจุดสีใน ROM BIOS ในกรณีที่มีการเรียกอินเทอร์รัพท์ของการเขียนจุดสีของ ROM BIOS ด้วยการที่บิตที่ 7 (บิตซ้ายสุด) ของรหัสสี (color code) มีค่าเป็น 1 ต่อจากนั้นสีที่อยู่ในตัวแปรรหัสสี จะถูก XOR กับสีที่มีอยู่เดิมในตำแหน่งนั้นแทนที่จะเป็นการเขียนทับตามแบบทั่วไป ข้อดีของวิธีนี้คือ จุดสีนั้นจะสามารถกลับมาเป็นสีเดิมได้โดยจะอธิบายให้เห็นประโยชน์ของวิธีนี้ในรายละเอียดเมื่อได้ศึกษามากขึ้นในตอนท้ายของบทนี้คุณสมบัติคือ ทุก ๆ บิตจะมีค่าเป็น 1 ยกเว้นที่ตำแหน่งของจุดที่จะต้องถูกเปลี่ยนจะมีค่าในบิตนั้นเป็น 0 ค่าของ bit mask จะถูก AND กับค่าของไบท์เดิม และค่าที่ได้นี้จะถูก OR ด้วยข้อมูลใหม่แต่อาจมีวิธีที่แตกต่างกันออกไป เช่น ถ้าต้องการ XOR ค่าใหม่กับค่าเก่าในกรณีนี้เพียงแค่ OR ค่าของไบท์เดิมด้วย 0 และ XOR กับค่าของข้อมูลใหม่เข้าไปผลที่เกิดขึ้นก็คือ จะให้สีใหม่ที่เป็นผลมาจากการ XOR



รูปที่ 4.13

ตำแหน่ง (address) ของไบต์ที่ถูกต้องคือ การนำค่า x ในคู่ลำดับคูณกับ 40 และบวกผลที่ได้กับค่าของ Y ในคู่ลำดับที่ถูกหารด้วย 4 (index = X*40+(Y>>2) โดย Y>>2 = Y/4) และในการกำหนดว่าจุดสีนั้นจะอยู่ในกลุ่มคู่หรือกลุ่มคี่จะถูกกำหนดโดยค่าของ x mod ด้วย 2 (x%2) ถ้าผลลัพธ์ที่ได้มีค่า 0 ซึ่งบ่งว่าค่า x เป็นเลขคู่ดังนั้นจุดนั้นจะอยู่ในหน่วยความจำกลุ่มคู่ ถ้ามีเศษเป็น 1จะถูกจัดอยู่ในกลุ่มคี่และในการหาตำแหน่งของบิตในไบต์นั้น ๆ จะถูกกำหนดโดยค่า Y modulus 4 (%4) ซึ่งจะเป็นค่าที่นำไปคำนวนการเลื่อนบิต โดยสามารถดูรายละเอียดได้จากฟังก์ชัน mempoint()

การเลื่อนบิต (bit shift operation) ใช้ในการจัดค่าของตัวแปรรหัสสี (color code) และ bit mask เพื่อจัดรูปแบบของข้อมูลที่เหมาะสม

4.3.3.การลากเส้น

ฟังก์ชั่นพื้นฐานของการใช้งานทางกราฟิกคือ การลากเส้นโดยการกำหนดจุดเริ่มต้นและ

จุดสุดท้ายของคู่ลำดับของเส้น แน่นอนการลากเส้นในแนวตั้งและแนวนอนจะไม่ยากหรือเป็นปัญหา แต่ถ้าหากต้องการลากเส้นเฉียงเมื่อใด ปัญหาคือจะวาดเส้นเฉียงด้วยวิธีใดบนจอภาพที่ไม่มีความละเอียดมากนักให้เป็นเส้นตรงเฉียง เช่น เส้นตรงจากจุด (0,0) ไปจุด (80,120)

วิธีการวาดเส้นเฉียง โดยมีฟังก์ชันที่ใช้ค่าอัตราส่วนระหว่างส่วนเปลี่ยนแปลงแกน x และแกน Y ก่อนที่จะมาศึกษาว่าโปรแกรมประเภทนี้มีขั้นตอนอย่างไร ขอให้พิจารณาเส้นจากจุด (0,0) ไปจุด (5,10) จะได้ส่วนเปลี่ยนแกน X คือ 5 และส่วนเปลี่ยนแกน Y คือ 10 ดังนั้นอัตราส่วนคือ 1/2 และค่านี้จะเป็นค่าที่ถูกนำไปกำหนดการเปลี่ยนแปลงของคู่ลำดับ X และ Y เมื่อเกิดการลากเส้นขึ้น ในกรณีนี้หมายความว่า ค่าของคู่ลำดับของ X จะเปลี่ยนแปลงน้อยกว่าความที่ที่ Y เปลี่ยนแปลงไปครั้งหนึ่ง นักเขียนโปรแกรมมือใหม่ส่วนใหญ่ จะใช้วิธีนี้เสมอในการสร้างฟังก์ชันการลากเส้น แม้ว่าวิธีที่ใช้คณิตศาสตร์อย่างง่ายแต่การที่โปรแกรมนี้จะทำงานได้อย่างถูกต้อง ต้องใช้ตัวแปรที่มีจุดทศนิยม (เลขอิงค์ชนี : floating pooint) และต้องระมัดระวังอย่างยิ่งในการบัดเศษ นั่นหมายถึงการเสียเวลาต่อจุดที่ต้องการแสดงผลนั่นหมายถึงความช้าของโปรแกรมถ้าขาด math coprcessor ดังนั้นจึงหันมาใช้กับอีกอัลกอริทึมหนึ่งคือ

อัลกอริทึมของ Bresenham แม้ว่าจะใช้อัตราส่วนการเปลี่ยนแปลงทางแกน X และแกน Y เช่นกันแต่ไม่มีความจำเป็นที่ต้องใช้ตัวแปรที่มีจุดทศนิยม และอัตราส่วนระหว่างส่วนเปลี่ยนทางแกน X และ Y จะถูกจัดการโดยวิธีการบวกและลบกัน ดังนั้นความคิดพื้นฐานที่ซ่อนอยู่ในอัลกอริทึมของ Bresenham ก็คือการทำบันทึกหรือจำค่าผิดพลาด (ในตัวแปร Xerr,Yerr)ซึ่งเกิดจากค่าระหว่างค่าในอุดมคติของแต่ละจุดและค่าที่ถูกแสดงผลออกมาจริง ๆ ค่าที่ผิดพลาดระหว่างค่าในอุดมคติและค่าจริงของด้านหนึ่งจะเป็นข้อจำกัดของฮาร์ดแวร์ (จากความจริงที่ว่าจะไม่มีการแสดงผลแบบใดที่มีการแสดงผลความละเอียดเป็นอนันต์) ดังนั้นจุดที่เหมาะจริงในการแสดงผลก็คือค่าประมาณที่ใกล้เคียงที่สุดในโปรแกรมนั้นจะสังเกตได้ว่าแต่ละรอบของการทำซ้ำในการวาดเส้นตัวแปรสองตัวคือ Xerr และ Yerr จะถูกเพิ่มขึ้นโดยการเปลี่ยนเป็นค่าของคู่ลำดับ X และ Y และเมื่อค่าผิดพลาดที่ได้เข้าหาค่าจำกัดที่กำหนดจะทำการรีเซ็ตเป็น 0 และตัวนับคู่ลำดับจะมีค่าเพิ่มขึ้น กระบวนการนี้จะมีการทำต่อเนื่องกันไปเรื่อย ๆ จนลากเส้นนั้น ๆ เสร็จฟังก์ชัน line() จะทำงานตามที่ได้อธิบายและควรปรึกษาโปรแกรมชนิดนี้เข้าใจ

หมายเหตุ ฟังก์ชัน `mempoint()` ที่เคยถูกพัฒนาจากตอนที่แล้วถูกใช้ในการเขียนจุดบนจอภาพให้เป็นเส้น

4.3.4 การวาดและการระบายสีรูปสี่เหลี่ยม

เมื่อได้ศึกษาฟังก์ชันในการวาดเส้นแล้วการวาดรูปทรงสี่เหลี่ยมดังที่แสดงในโปรแกรมที่จะเป็นการง่ายขึ้นโปรแกรมนี้เป็นโปรแกรมที่เขียนโครงร่างของรูปทรงสี่เหลี่ยมในสีที่กำหนด และให้ค่าคู่ลำดับของมุมสองมุมที่ตรงข้ามกันและสีเป็นค่าพารามิเตอร์

```
/* วาดสี่เหลี่ยม */
void box(int startx,int endx,int endy,int color_code)
{
line (startx,starty,endx,starty,color_code);
line (startx,stafty,startx,endy,color_code);
line (startx,endy,endx,endy,color_code);
line (endx,starty,endz,endy,color_code);
}
```

ในการระบายสีรูปสี่เหลี่ยมจะใช้ฟังก์ชัน fill_box () โดยจะใช้วงรอบ เรียกฟังก์ชัน line () มาระบายสีให้กับรูป

```
/* ระบายสีสี่เหลี่ยมตามสีที่กำหนด */
void fill_box(int startx,int starty, int endx int endy, int
color_code)
{
        register int i , begin ,end;
        begin= startx<endx ? startx : endx;
        end  = startx>endx ? startx : endx;
        for (i=begin;i<end;i++)
        line(i,starty,i,endy,color_code) ;
 }
```

4.3.5 การวาดวงกลม

การ จะวาดรูปวงกลมให้ง่ายและรวดเร็วที่สุดนั้น จะใช้ . Bresenham's circle -drawing algorithm (ซึ่งใช้หลักการเดียวกับ line drawing algorithm) คือการไม่ใช้ตัวแปรทศนิยม (floating point) ยกเว้นตัวแปร asp_ratio ที่เป็น globalประเภท double ดังนั้นอัลกอริทึมนี้จึงมีความเร็วสูง

วิธีการทำงานหลักการของอัลกอริทึมนี้ที่ใช้ในการคำนวนตำแหน่งของจุด X และ Yโดยใช้วิธีการหามาจากค่าของความแตกต่างระหว่าง X และ Y และปรับปรุงค่านี้ให้เก็บในตัวแปรชื่อ delta และมีฟังก์ชันสนับสนุนการทำงานของฟังก์ชัน circle() คือฟังก์ชัน plot_circle() และ ฟังก์ชัน circle() นี้ยังไม่สามารถทำให้โปรแกรมสั้นลงได้(ตามคุณสมบัติของภาษาซี) ข้อดีคือจะได้รหัสที่เร็วขึ้น แต่ข้อเสียคืออาจทำให้ยากต่อการทำความเข้าใจและแก้ไข (debug) โปรแกรมในภายหลัง

ค่าของตัวแปร asp_ratio จะต้องถูกกำหนดตัวแปรไว้นอกฟังก์ชัน เพราะฟังก์ชัน circle () และฟังก์ชัน plot_circle()ต้องการใช้ตัวแปรนี้ร่วมกันนอกจากนี้ยังสามารถสร้างวงรีได้จาก การเปลี่ยนค่าของ asp_ratio การเรียกใช้ฟังก์ชัน circle() ต้องมีค่าพารามิเตอร์ที่เป็นพิกัดของจุดศูนย์กลางความยาวของรัศมี (หน่วยเป็นจุด) และสีของวงกลม

4.3.5 การวาดวงกลม

การ จะวาดรูปวงกลมให้ง่ายและรวดเร็วที่สุดนั้น จะใช้ . Bresenham's circle -drawing algorithm (ซึ่งใช้หลักการเดียวกับ line drawing algorithm) คือการไม่ใช้ตัวแปรทศนิยม (floating point) ยกเว้นตัวแปร asp_ratio ที่เป็น globalประเภท double ดังนั้นอัลกอริทึมนี้จึงมีความเร็วสูง

วิธีการทางานหลักการของอัลกอริทึมที่ที่ใช้ในการคานวนตาแหน่งของจุด X และ Yโดยใช้วิธีการหามาจากค่าของความแตกต่างระหว่าง X และ Y และปรับปรุงค่านี้ให้เก็บในตัวแปรชื่อ delta และมีฟังก์ชันสนับสนุนการทางานของฟังก์ชัน circle() คือฟังก์ชัน plot_circle() และ ฟังก์ชัน circle() นี้ยังไม่สามารถทาให้โปรแกรมสั้นลงได้(ตามคุณสมบัติของภาษาซี) ข้อดีคือจะได้รหัสที่เร็วขึ้น แต่ข้อเสียคืออาจทาให้ยากต่อการทาความเข้าใจและแก้ไข (debug) โปรแกรมในภายหลัง

ค่าของตัวแปร asp_ratio จะต้องถูกกาหนดตัวแปรไว้นอกฟังก์ชัน เพราะฟังก์ชัน circle () และฟังก์ชัน plot_circle()ต้องการใช้ตัวแปรนี้ร่วมกันนอกจากนี้ยังสามารถสร้างวงรีได้จาก การเปลี่ยนค่าของ asp_ratio การเรียกใช้ฟังก์ชัน circle() ต้องมีค่าพารามิเตอร์ที่เป็นพิกัดของจุดศูนย์กลางความยาวของรัศมี (หน่วยเป็นจุด) และสีของวงกลม

บทที่ 5

PLC

5.1 พื้นฐานของ PLC

ระบบควบคุมแบบซีเควน (sequence control)แต่เดิมจะประกอบด้วยอุปกรณ์ไฟฟ้าเชิงกล(electromechanical device)ซึ่งได้แก่รีเลย์ (relay)ตัวตั้งเวลา (timer) ตัวนับ (counter)และอื่นๆ ลักษณะการทำงานของระบบควบคุมนี้จะมีอยู่ด้วยกันสองสภาวะคือ เปิด กับ ปิด หรืออาจจะพูดได้ว่าเป็นแบบ"ON" กับ "OFF" นั่นเองระบบควบคุมแบบซีเควนซ์ ที่ประกอบกันขึ้นจากอุปกรณ์ดังกล่าวข้างต้นนี้ จะมีข้อเสียหลายประการ เช่น มีขนาดใหญ่ มีกำลังงานสูงและเมื่อมีการเปลี่ยนแปลงการทำงานก็ต้องทำการสร้างวงจรควบคุมใหม่ ซึ่งไม่สามารถประยุกต์ใช้งานกับระบบควบคุมที่มีความยุ่งยากและซับซ้อนเหล่านี้เป็นต้น

เนื่องจากเทคโนโลยีทางด้านไมโครโปรเซสเซอร์ (microprocessor) ในปัจจุบันได้เจริญก้าวหน้าไปมากจึงได้นำมาประยุกต์ใช้งานหลาย ๆ ด้าน โดยเฉพาะอย่างยิ่งในด้านของการควบคุมแบบต่อเนื่อง ทำให้มีการค้นคว้า ทดลอง และวิจัย เครื่องควบคุมชนิดโปรแกรม (programmable logic controller) หรือเรียกสั้น ๆ ว่า PLC ขึ้นมาใช้กับการควบคุมสำหรับงานด้านอุตสาหกรรมโดยเฉพาะ

PLC จะมีส่วนที่เป็น อินพุตที่สามารถต่อใช้งานเข้ากับตัวตรวจจับต่าง ๆ (sensor) และส่วนเอาต์พุตจะควบคุมการทำงานของอุปกรณ์หรือเครื่องจักรโดยสามารถสร้างวงจร หรือเงื่อนไขการทำงาน ของเครื่องจักรเหล่านี้ ได้จากการป้อนเป็นโปรแกรมสั่งงาน ที่เรียกว่า ladder diagram เข้าไปโปรแกรมนี้จะทำหน้าที่เหมือนกับวงจรรีเลย์ ตัวตั้งเวลา ตัวนับ และอื่นๆ ของวงจรรีเลย์อีก ภายหลังจากที่มีการเลือกสวิทช์บังคับให้ PLC เริ่มทำงานหรือรันโปรแกรม PLC ก็จะทำงานตามขั้นตอนเหมือนกับโปรแกรมที่ผู้ใช้ป้อนเข้าไปทุกประการ โดยที่ PLC จะสร้างอุปกรณ์ควบคุมต่าง ๆ ภายในเอง เช่น รีเลย์ตัวตั้งเวลา ตัวนับ ได้ด้วยซอฟแวร์ โดยปรากฏอยู่ในรูปของฟังก์ชันการทำงานที่ตรงกับสภาพความเป็นจริงนอกจากนี้ เงื่อนไขต่าง ๆ

ที่เขียนเป็นโปรแกรม ก็จะมีลักษณะคล้ายกับการต่อสายของอุปกรณ์เหล่านี้กันเป็นวงจรขึ้นมา แต่เนื่องจากว่า เป็นการปฏิบัติการทางซอฟท์แวร์ จึงทำให้ผู้ใช้สามารถแก้ไขและเพิ่มเติมวงจรได้โดยการเปลี่ยนแปลงโปรแกรมของวงจรนั้น ๆ จากแป้นพิมพ์ของ PLC ซึ่งทำได้สะดวก แน่นอน และง่ายกว่าการเดินสายสายไฟที่เป็นวงจรรีเลย์เช่นแต่ก่อนอยู่มาก ดังนั้นจึงพอสรุปข้อดีต่าง ๆ ของ PLCไว้ดังนี้

- สิ้นเปลือง เนื้อที่น้อย เพราะมีขนาดเล็ก
- สามารถที่จะใช้ควบคุมเครื่องจักรหรือระบบกระบวนการใด ๆ ก็ได้ ถ้าเลือกขนาดของPLC ที่เหมาะสม
- การเปลี่ยนลำดับขั้นตอน หรือ เงื่อนไขของการทำงาน ก็จะได้ตามต้องการ เพราะใช้หลักการทางโปรแกรม
- ตัวตั้งเวลา ตัวนับ จะเป็นซอฟท์แวร์ ทำให้การกำหนดค่าต่าง ๆง่ายและสามารถเปลี่ยนแปลงค่าได้ตลอดเวลา ไม่ต้องมีฮาร์ดแวร์ร่วม และทำให้ราคาถูกลง
- รีเลย์ภายใน (internal relay) ก็เป็นซอฟท์แวร์เช่นเดียวกัน จึงลดค่าใช้จ่ายในการเดินสายลดฮาร์ดแวร์และทำให้ขนาดเล็กลงด้วย
- การติดตั้งทำให้สะดวกและง่าย
- การขยายระบบให้ใหญ่ขึ้นทำได้โดยง่าย
- ราคาถูกกว่าระบบรีเลย์
- ความน่าเชื่อถือ reliability ดีเพราะ เป็นอุปกรณ์สารกึ่งตัวนำ ไม่มีการเคลื่อนสายมากไม่มีปัญหาเกี่ยวกับหน้าสัมผัส (contact) ของรีเลย์
- มีระบบตรวจสอบหาที่ผิดพลาดด้วยตัวเอง การตรวจสอบแก้ไข เมื่อมีปัญหาจึงทำให้เร็ว
- ลดการเดินสายยาว ๆ และลดค่าใช้จ่ายในการเดินสายได้เพราะมีอินพุต / เอาต์พุตแบบรีโมต
- การบำรุงรักษาทำได้ง่าย
- เวลาในการทำงานเร็วกว่าระบบที่ใช้รีเลย์
- มีฟังก์ชันทางคณิตศาสตร์ได้แก่ บวก ลบ คูณ หาร และอื่นๆ
- ต่อเข้ากับคอมพิวเตอร์เพื่อให้โปรแกรมทำได้หลายแบบ เช่น คำสั่งในรูปของแลดเดอร์ไดอะแกรม คำสั่งบูลีน (boolean instruction) ตลอดจนการตรวจสอบค่า

สภาวะในขณะที่ PLC กำลังทำงานอยู่

- ใช้ได้ในทุกสภาพแวดล้อมของงานอุตสาหกรรมจากการที่ได้กล่าวมานี้ จะเห็นได้ว่า PLCเป็นเครื่องควบคุมที่มีประสิทธิภาพ และมีประโยชน์อย่างยิ่งต่องานอุตสาหกรรมในปัจจุบันและในอนาคต

นอกจากนี้ PLC ที่มีให้เลือกใช้อยู่ในปัจจุบัน ทั้งระบบเล็กและระบบใหญ่ มีการออกแบบระบบให้มีความยืดหยุ่นสูงในการที่จะต่อร่วมกับอุปกรณ์สมทบประเภทต่าง ๆ ทั้งนี้เพื่อเป็นการเพิ่มประสิทธิภาพการทำงานของ PLC ให้มีสมรรถนะสูงขึ้น

อุปกรณ์สมทบต่าง ๆ มีดังนี้

(1) I/O UNITS

INPUT,OUTPUT MODULE

OPTICAL TRANSMITING I/O UNIT

(2) SPECIAL I/O UNITS

ANALOG INPUT/OUTPUT UNITS

PID UNNIT

TEMPERATURE SENSOR UNIT

HIGH-SPEED COUNTER UNIT

POSITION CONTROL UNIT

ASCLL UNIT

LADDER PROGRAM I/O UNIT

FILE MEMORY UNIT

MCR UNIT (MAGNETIC CARD READER)

(3) LINK SYSTEM

REMOTE I/O SYSTEM

I/O LINKS

PC LING SYSTEMS

HOST LINK SYSTEMS

SYSMAC NET LINK SYSTEMS

(4) PERIPHERAL TOOLS

PROGRAMMING CONSOLE

EPROM WRITER

PRINTER INTERFACE UNIT

SYSMAC C-SERIES SUPPORT TOOLS

FIT : FACTORY INTELLIGENT TERMINAL

LSS : LADDER SUPPORT SOFTWARE

GPC : GRAPHICS PROGRAMMING CONSOLE

จากที่กล่าวมาทั้งหมดจะเห็นได้ว่าเป็นอุปกรณ์ต่อร่วมเหล่านี้ มีอยู่จำนวนมาก ดังนั้น ในการออกแบบระบบทางด้าน hard ware ผู้ออกแบบสามารถเลือกอุปกรณ์ต่อร่วมได้ตามความเหมาะสมในการที่จะประยุกต์ใช้ตามประเภทลักษณะงาน

5.2 โครงสร้างและการทำงาน

5.2.1 โครงสร้างโดยทั่วไปของ PLC

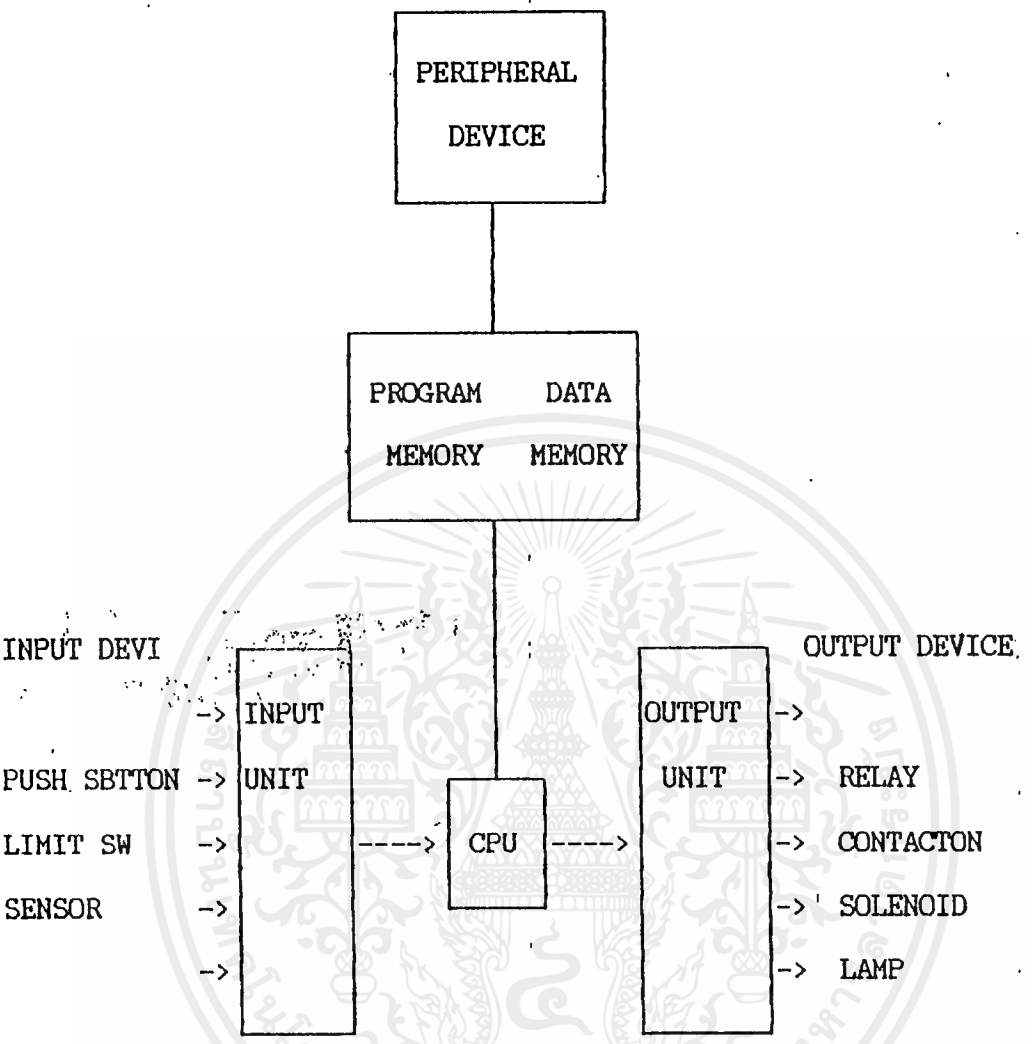
ส่วนประกอบที่สำคัญของ PLC แบ่งออกเป็น 4 ส่วน

(1) หน่วยประมวลผล (CPU UNIT)

(2) หน่วยความจำ (MEMORY UNIT)

(3) หน่วย อินพุต-เอาต์พุต (INPUT-OUTPUT UNIT)

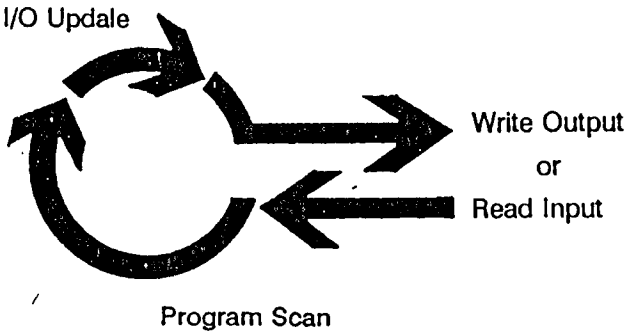
(4) อุปกรณ์ต่อร่วม (PERIPHERAL DEVICES)



รูปที่ 5.1 โครงสร้างของ PLC

หน่วยประมวลผล (CPU UNIT)

หน่วยประมวลผลทำหน้าที่ควบคุมการทำงานของระบบทั้งหมด โดยรับข้อมูลอินพุตเข้ามาทำการประมวลผลแล้วส่งผลที่ได้ออกไป จากนั้นก็จะวนกลับไปรับข้อมูลอินพุตเข้ามาอีกแล้วจะทำซ้ำ ๆ ในลักษณะเช่นนี้ไปเรื่อย ๆ การทำงานของ CPU จะอยู่ภายใต้การควบคุมของโปรแกรมที่ผู้ใช้ป้อนเข้าไปโดยที่การทำงานในแต่ละรอบนี้เรียกว่า การสแกน (scaning)สำหรับเวลาของการสแกนขึ้นอยู่กับ ขนาดของหน่วยความจำ และความเร็วของหน่วยประมวลผล ช่วงเวลาของการสแกนจะทำให้ทราบถึงความสามารถในการตอบสนองต่อการเปลี่ยนแปลงของอินพุต-เอาต์พุต ว่ามีความรวดเร็วเพียงใด



รูปที่ 5.2 การสแกนที่มีผลตอบสนองต่ออินพุต-เอาต์พุต

หน่วยความจำ (MENORY UNIT)

เป็นองค์ประกอบสำคัญของระบบ เพราะใช้เป็นที่เก็บโปรแกรมและข้อมูล และขนาดของหน่วยความจำจะเป็นสิ่งที่กำหนดความสามารถของระบบ ปกติมักจะมีขนาดวัดกันเป็นสเต็ปของคำสั่งในการโปรแกรมระบบที่ขนาดของหน่วยความจำมาก จะทำให้ผู้ใช้สามารถเขียนโปรแกรมที่มีความซับซ้อนได้มากขึ้น

หน่วยความจำ PLC จะแบ่งเป็น

- RAM (Random Acccess Memory)
- EPROM (Erasable Programmable Read Only Memory)

หน่วยความจำ RAM

เป็นหน่วยความจำที่ใช้เก็บโปรแกรมควบคุม ที่ป้อนโดยผู้ใช้ให้กับ PLC ทั้งนี้เพราะโปรแกรมควบคุมนี้อาจต้องมีการเปลี่ยนแปลงแก้ไข ดังนั้นจึงจำเป็นต้องใช้หน่วยความจำที่สามารถลบข้อมูลเดิมและนำโปรแกรมใหม่เข้าไปเก็บไว้ ในการใช้งานจริง ๆ แล้วต้องมีแหล่งจ่ายไฟสำรองต่อ เพื่อป้องกันไม่ให้ข้อมูลสูญหาย เมื่อไฟฟ้าดับ

หน่วยความจำ EPROM

เป็นหน่วยความจำสำหรับเก็บโปรแกรมที่มีการพัฒนาจนใช้งานได้ดีให้เป็นการถาวรและในการอัดโปรแกรมจะทำได้โดยถ่ายข้อมูลจากหน่วยความจำ RAM มาสู่หน่วยความจำ EPROM โดยอาศัยเครื่องอัดโปรแกรมพิเศษ (eprom writer)ต่อร่วมกันชุดของ PLC ซึ่งจะทำให้ได้โปรแกรมถาวรดังกล่าวในหน่วยความจำ EPROM นั้น และพร้อมที่จะนำมาติดตั้ง (install)

ลงใน PLC เพื่อให้ PLC มีการทำงานตามโปรแกรมที่บรรจุ หน่วยความจำประเภทนี้โปรแกรมจะไม่มีการสูญหายเมื่อไฟดับ แต่ถ้ามีความจำเป็นที่จะลบโปรแกรมภายในก็สามารถทำได้โดยใช้เครื่องล้างโปรแกรม

หน่วยอินพุต-เอาต์พุต (INPUT_OUTPUT UNIT)

ในที่นี้จะกล่าวเฉพาะหน่วยอินพุต-เอาต์พุต แบบดิจิตอลเท่านั้น

หน่วยอินพุต (INPUT UNIT)

หน่วยอินพุตทำหน้าที่รับสัญญาณจากอุปกรณ์ภายนอก ที่เป็นสวิทช์และตัวตรวจจับต่าง ๆ แล้วแปลงชนิดของสัญญาณขาเข้าดังกล่าวไม่ว่าจะเป็น AC,DC ให้เป็นสัญญาณที่เหมาะสม เพื่อส่งเข้าไปให้แก่หน่วยประมวลผลกลาง ดังนั้นในการเลือกใช้ประเภทของอินพุตนั้นผู้ใช้ จะต้องเลือกให้เหมาะสม และถูกต้องตามประเภทของการใช้งานด้วยมิเช่นนั้นอาจเกิดความเสียหายขึ้นได้ ปกติอินพุตที่ดีจะต้องมีหน้าที่ดังนี้

- เปลี่ยนแปลงระดับของสัญญาณเข้าให้เป็นระดับสัญญาณที่เหมาะสมกับระบบของ PLC
- การส่งสัญญาณระหว่างหน่วยอินพุตกับหน่วยประมวลผล จะติดต่อกันด้วยสายสัญญาณโดยอาศัยอุปกรณ์ประเภทโฟโตทรานซิสเตอร์ทั้งนี้เพื่อแยกสัญญาณ (isolate) ทางไฟฟ้าให้ออกจากกันเป็นการป้องกันมิให้หน่วยประมวลผลได้รับความเสียหาย เมื่อหน่วยอินพุตเกิดการลัดวงจร

3. ไม่มีการสั่นของหน้าสัมผัส (contact chattering)

หน่วยเอาต์พุต (OUTPUT UNIT)

หน่วยเอาต์พุตทำหน้าที่รับค่าสภาวะที่ได้จากการประมวลผลของหน่วยประมวลผลเพื่อนำค่าสภาวะเหล่านี้ไปควบคุมอุปกรณ์ภายนอก รีเลย์ โซลินอยด์ หลอดไฟเสดงสภาวะ เป็นต้น นอกจากนั้นยังทำหน้าที่ ISOLATE สัญญาณของหน่วยประมวลผล ออกจากอุปกรณ์เอาต์พุต โดยปกติแล้ว หน่วยเอาต์พุตมีความสามารถในการขับโหลดด้วยกระแสประมาณ 1 ถึง 2 แอมแปร์ ในกรณีที่โหลดต้องการกระแสมากกว่านี้ผู้ใช้จะต้องนำไปต่อเข้ากับอุปกรณ์ขับ หรือขยายอีกทีหนึ่ง เช่น รีเลย์ โซลิดสเตทรีเลย์ และคอนแทคเตอร์ เป็นต้น

เอาต์พุตของ PLC จะมีอยู่ด้วยกันหลายแบบผู้ใช้ต้องพิจารณาเลือกใช้งานให้ถูกต้อง คือ

เอาต์พุตที่ใช้กับไฟกระแสสลับ เอาต์พุตประเภทนี้มีอยู่ด้วยกัน 2 ชนิด คือ รีเลย์ และไทรแอค สำหรับไทรแอคนั้น เป็นสารกึ่งตัวนำจะถูกเอามาใช้กับโหลดที่มีการเปิดปิดอยู่บ่อย ๆ เพื่อลดการอาคต์เมื่อมีการต่อวงจรไฟฟ้า เอาต์พุตใช้กับไฟกระแสตรง เอาต์พุตประเภทนี้มีอยู่ 2 ชนิดด้วยกัน คือ รีเลย์ และทรานซิสเตอร์ สำหรับชนิดทรานซิสเตอร์ ใช้งานที่มีการเปิดปิดอยู่บ่อย ๆ เช่นกัน ตลอดจนการนำไปขับโหลดที่เป็นวงจรอิเล็กทรอนิกส์ เช่น เอาต์พุตทรานซิสเตอร์แบบโอเพนคอลเลคเตอร์เป็นตัวขับ เป็นต้น

คำจำกัดความที่เป็นคุณลักษณะของหน่วยอินพุต-เอาต์พุต

(1) แรงดันอินพุต (Input Voltage Rating) หมายถึง ระดับสัญญาณอินพุตที่หน่วยอินพุตใช้แยกความแตกต่างระหว่าง สภาวะ "ON" และสภาวะ "OFF" ของอุปกรณ์ภายนอก ที่สภาวะ "ON" สัญญาณอินพุตสามารถเปลี่ยนแปลง ได้ประมาณ 10% ถึง 15% ของ input voltage ที่กำหนดไว้

(2) กระแสอินพุต (Input Current Rating) หมายถึง จำนวนกระแสไฟฟ้าที่หน่วยอินพุตต้องการจากอุปกรณ์อินพุต เมื่อมีสภาวะ "ON" และ "OFF" ที่สภาวะ "ON" มีค่าประมาณ 7~10 mA

(3) ระดับเทรสโฮล์ (Input-Threshold Voltage) หมายถึง ช่วงการเปลี่ยนแปลงระดับแรงดันอินพุต ที่หน่วยอินพุตสามารถแยกความแตกต่าง ระหว่างสภาวะ "ON" กับสภาวะ "OFF"

(4) ช่วงเวลาอินพุต (Input Delay) หมายถึง ช่วงเวลาที่หน่วยอินพุตรับสภาวะของอุปกรณ์อินพุต เช่น หน่วยอินพุต AC หรือ DC มีช่วงเวลาอินพุตประมาณ 9 ถึง 25 ms

(5) แรงดันเอาต์พุต (Output Voltage Rating)หมายถึง ระดับสัญญาณไฟฟ้าของหน่วยเอาต์พุตเมื่อมีสภาวะ "ON" และ "OFF"

(6) กระแสเอาต์พุต (Output Current Rating)หมายถึงจำนวนของกระแสไฟฟ้าที่สูงสุดที่หน่วยเอาต์พุตสามารถจ่ายให้อุปกรณ์ภายนอก

(7) กำลังเอาต์พุต (Output Power Rating) หมายถึง ขนาดกำลังไฟฟ้าสูงสุด ที่หน่วยเอาต์พุตจ่ายให้อุปกรณ์ภายนอก

(8) อัตราการใช้กระแส (Current Requirement) หมายถึง จำนวนกระแสไฟฟ้าที่

หน่วยอินพุต และเอาต์พุตต้องการจากหน่วยจ่ายกำลังงาน

(9) อัตราการทนกระแสสูงสุด (Maximum Surge Current) หมายถึง จำนวนกระแสไฟฟ้าสูงสุดที่มีค่ามากกว่ากระแสเอาต์พุตและหน่วยเอาต์พุต สามารถจ่ายให้อุปกรณ์ภายนอกได้โดยไม่เกิดความเสียหายชั่วระยะเวลาหนึ่ง เช่น 20A/1S หมายถึงหน่วยเอาต์พุตสามารถจ่ายกระแสเอาต์พุต ได้ 20 A เป็นเวลา 1 วินาที ก่อนที่หน่วยเอาต์พุตจะเสียหาย

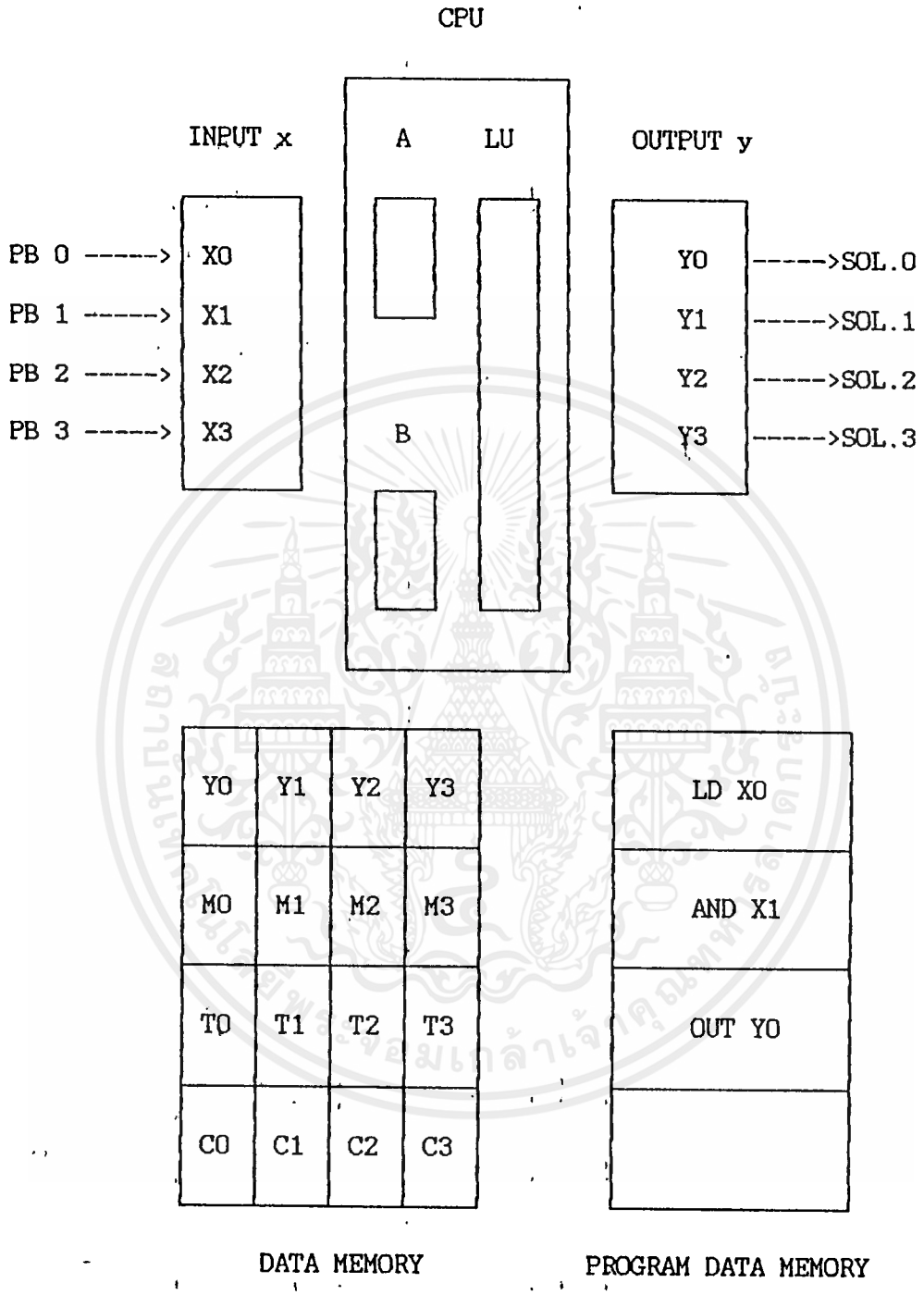
อุปกรณ์ต่อร่วม (PERIPHERAL DEVICES)

อุปกรณ์ที่ใช้ในการต่อร่วมกับ PLC มีอยู่ด้วยกันหลายชนิดหลายประเภท ทั้งนี้เพื่ออำนวยความสะดวกในการพัฒนาโปรแกรม ทำให้การเขียนโปรแกรมเสร็จสมบูรณ์ด้วยเวลาอันสั้น

ประเภทและหน้าที่ของอุปกรณ์ต่อร่วมมีดังนี้

อุปกรณ์ต่อร่วม	หน้าที่การใช้งานเกี่ยวกับโปรแกรม				
	ป้อน	แก้ไข	โหลดใหม่	พิมพ์	สภาวะ
1. PROGRAMMING CONSOLE	*	*			*
2. EPROM WAITER			*		
3. PRINTER				*	
4. GRAPHIC PROGRAMMING	*	*			*
5. CRT MONITOR	*	*			
6. AUDIO CASSETTE			*		
7. LADDER SOFTWARE	*	*	*	*	*

5.2.2 ตัวอย่างการทำงานของ PC.ตามโปรแกรม



รูปที่ 5.3

A : A REGISTER

B : B REGISTER

LU : LOGIC UNIT

X : INPUT MULTIPLEXER

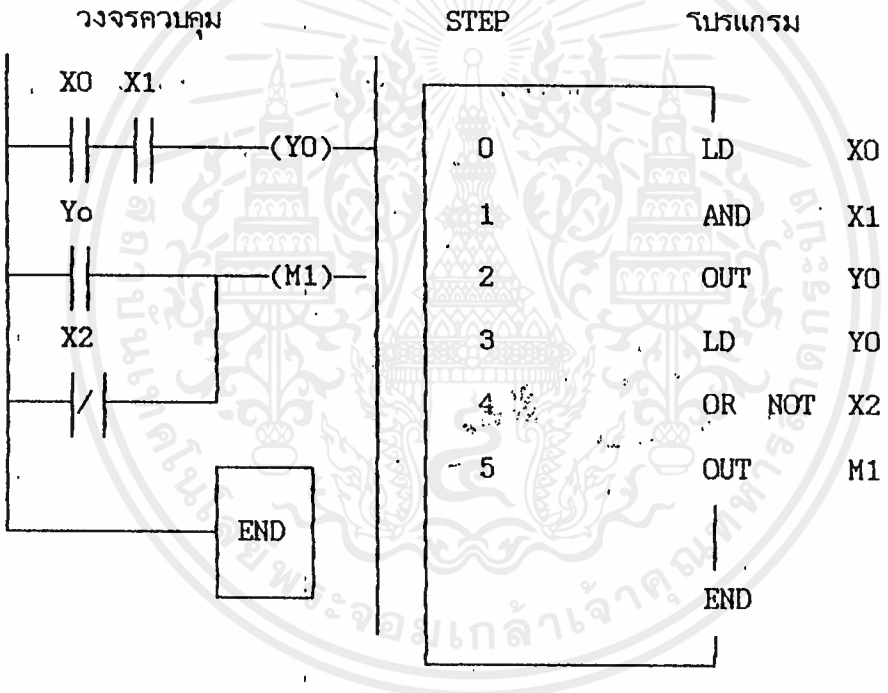
Y : OUTPUT DEMULTIPLEXER

Y0-Y3 : OUT STATUS DATA

M0-M3 : AUXILIARY RELAY STATUS DATA

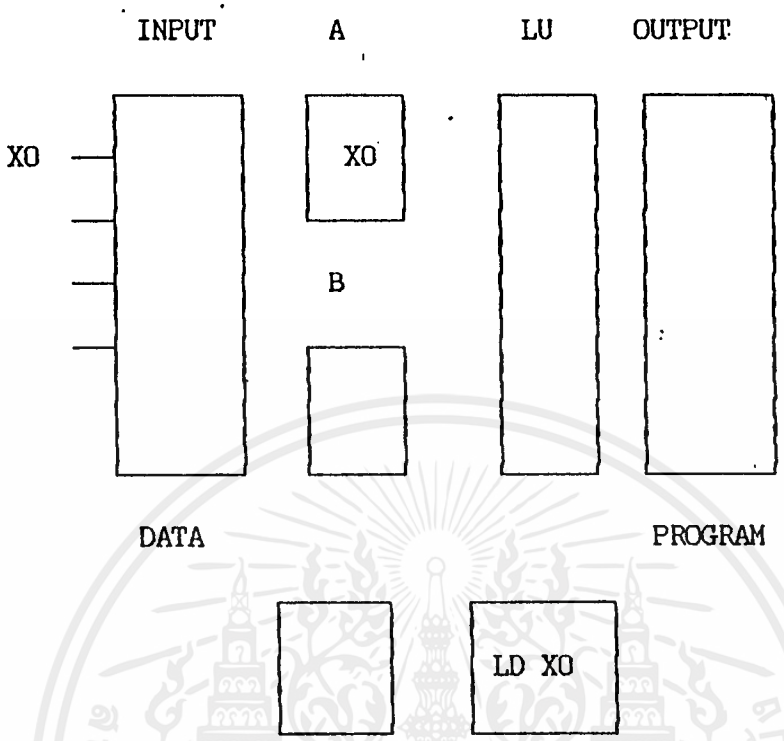
C0-C3 : COUNTER STATUS DATA

T0-T3 : TIMER STATUS DATA



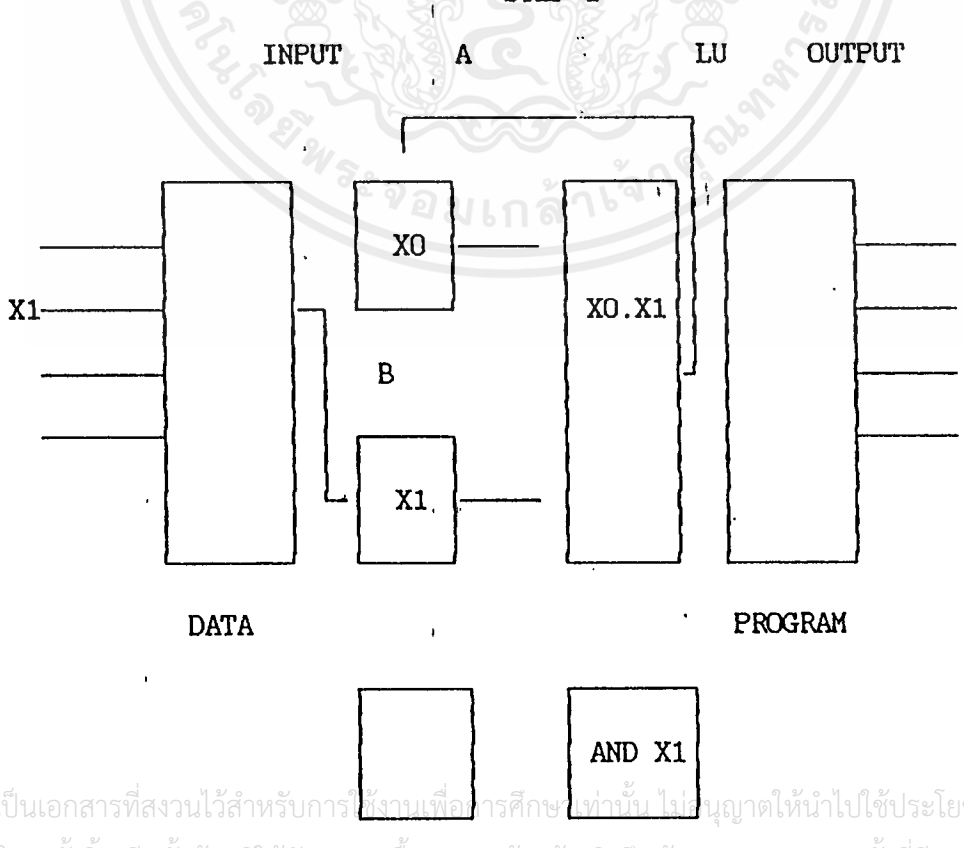
ตัวอย่าง วงจรควบคุม และรายละเอียดโปรแกรมของวงจรตัวอย่าง แต่ละ STEP เราสามารถแสดงการเคลื่อนไหวของข้อมูลใน REGISTER และ MEMORY ภายในต่าง ๆ ได้ดังนี้

STEP 0



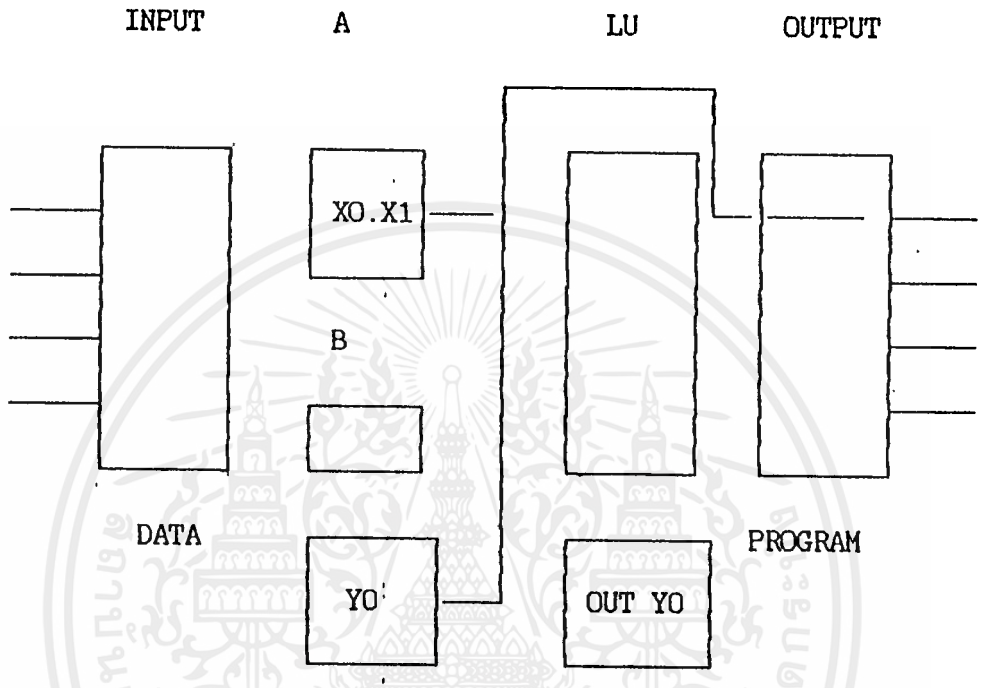
ค่าสภาวะของ X0 จะถูกนำเข้ามาเก็บไว้ใน REGISTER A.

STEP 1



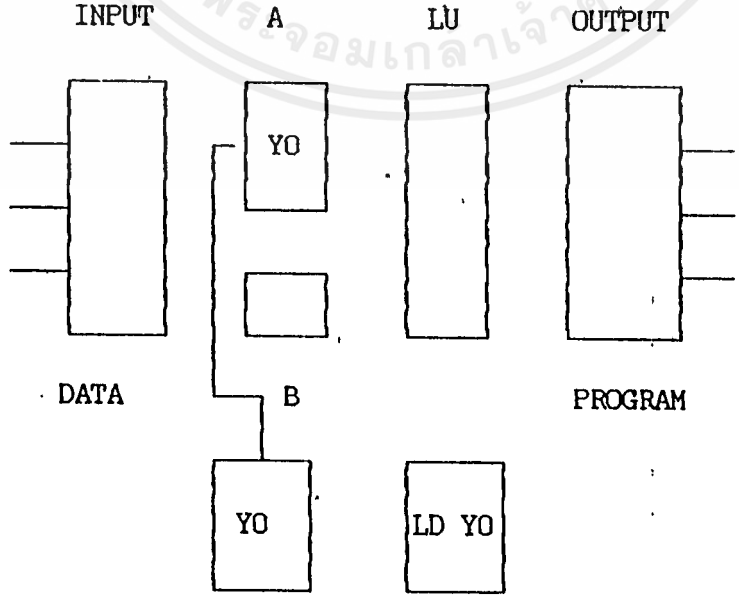
ค่าสภาวะของ X1 ถูกนำเข้าไปเก็บไว้ที่ REGISTER B เสร็จแล้ว LOGIC UNIT (LU.) จะทำหน้าที่นำค่าสภาวะภายใน REGISTER ทั้งสองมาทำเงื่อนไข AND กันแล้วนำผลลัพธ์ที่ได้ไปเก็บไว้ที่ REGISTER A เช่นเดิม

STEP 2



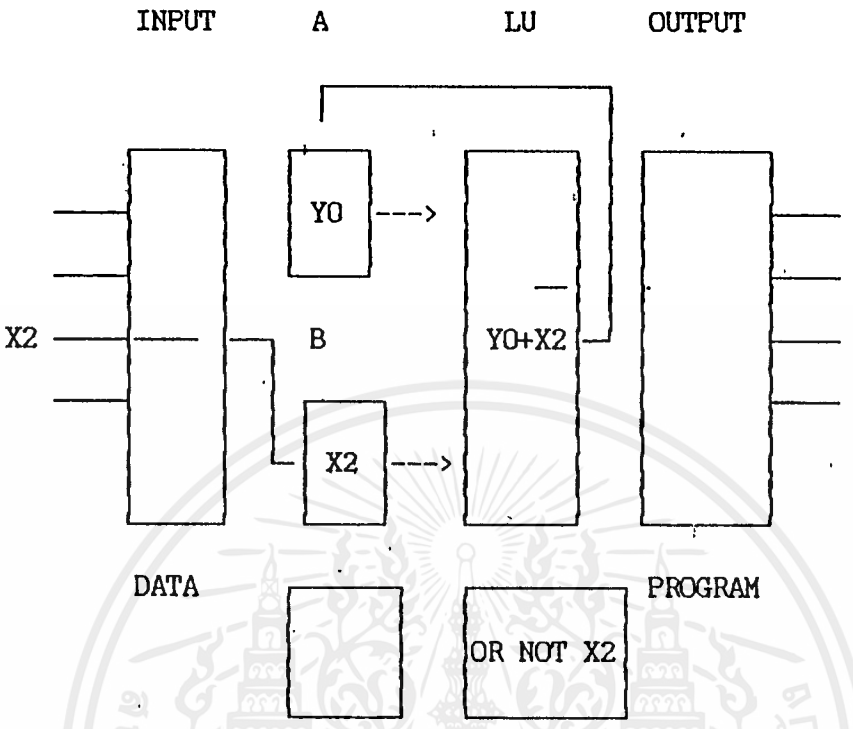
นำค่าสภาวะภายในของ register a ที่เป็นผลมาจาก step ที่ผ่านมา ออกไปยังเอาต์พุตตำแหน่ง YO ในขณะเดียวกันก็นำมาเก็บไว้ที่ data memory ด้วย

STEP 3



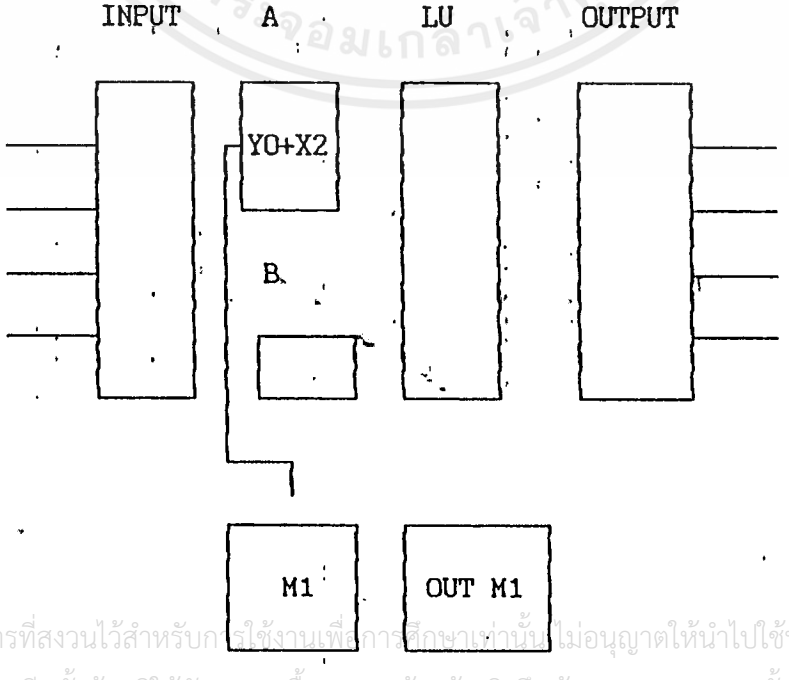
นำค่าสภาวะของ YO ที่อยู่ใน DATA MEMORY ไปเก็บไว้ที่ REGISTER A.

STEP 4



ค่าสภาวะของ X2 ถูกนำไปที่ register b แต่เนื่องจากโปรแกรมที่เขียนให้กับ PC จะเป็นการ or not ดังนั้น logic unitจึงทำการเปลี่ยนค่าสภาวะของ X2 ให้เป็นค่าตรงกันข้าม (X2)เสียก่อนแล้วจึงทำเงื่อนไข OR จากนั้นนำค่าสภาวะผลลัพธ์ที่ได้ไปเก็บไว้ใน register a เช่นเดิม

STEP 5



นำค่าสภาวะภายใน register a ที่ได้จาก stepซึ่งผ่านมาไปเก็บไว้ที่ auxiliary relayหรือรีเลย์ช่วยแต่ในที่นี้หมายถึงรีเลย์ภายในซึ่งเป็นส่วนหนึ่งของ data memoryนี่เป็นตัวอย่างการทำงานไม่ซับซ้อนเท่าใดนัก ในทำนองเดียวกันโปรแกรมที่มีความซับซ้อนยากไปกว่านี้ก็มีลำดับขั้นที่ไม่แตกต่างไปจากตัวอย่างเช่นกัน

5.3 การทดลองใช้ PLC

คำสั่งพื้นฐานสำหรับการเขียนโปรแกรม (basic instruction program) คำสั่งพื้นฐานต่าง ๆของ PLC ที่นำมาใช้งานจะมีดังนี้

INSTRUCTION	SYMBOL	C28P/C40P/C60P FORMAT	
		MNEMONIC	DATA
OUT	——————( )	OUT	POINT NO.
OUT NOT	——————(/)	OUT PUT	POINT NO.
TIMER	——————(TIM)	TIM	TIMER NO. SET VALUE
COUNTER	— CP CNT — — R	CNT	COUNTER NO. SET VALUE

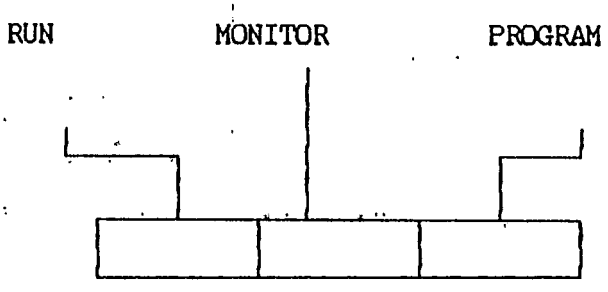
ความหมายของคำสั่งพื้นฐาน

INSTRUCTION	MEANING
LD	เป็นการนำค่าสภาวะปัจจุบันของอุปกรณ์ตำแหน่งต่าง ๆที่กำหนดเข้ามา คำสั่งนี้จะใช้ในการเริ่มต้นของLADDER DIAGRAM หรือเริ่มต้น BLOCK
AND	เป็นการนำค่าสภาวะของอุปกรณ์ต่าง ๆที่กำหนดมากระทำลอจิก ANDกับ ค่าสภาวะปัจจุบัน
OR	เป็นการนำค่าสภาวะของอุปกรณ์ต่าง ๆที่กำหนดมากระทำลอจิก OR กับ ค่าสภาวะปัจจุบัน
OUT	เป็นการให้ค่าสภาวะแก่อุปกรณ์ต่าง ๆที่กำหนด

INSTRUCTION	MEANING
LD NOT	เช่นเดียวกับคำสั่ง LOAD แต่เป็นการนำค่าสภาวะที่เป็นตรงกันข้ามกับสภาวะที่เข้ามา
AND NOT	เช่นเดียวกับคำสั่ง AND แต่เป็นการนำค่าสภาวะที่เป็นตรงกันข้ามกับสภาวะที่เข้ามา
OR NOT	เช่นเดียวกับคำสั่ง OR แต่เป็นการนำค่าสภาวะที่เป็นตรงกันข้ามกับสภาวะที่เข้ามา
OUT NOT	เช่นเดียวกับคำสั่ง OUT แต่เป็นการนำค่าสภาวะที่เป็นตรงกันข้ามกับปัจจุบัน
AND LD	เป็นคำสั่งที่เกี่ยวกับ BLOCK โดยการกระทำลอจิก AND ซึ่งกันและกัน
OR LD	เป็นคำสั่งที่เกี่ยวกับ BLOCK โดยการกระทำลอจิก OR ซึ่งกันและกัน
TIM	เป็นการเรียกใช้ตัวตั้งเวลา
CNT	เป็นการเรียกใช้ตัวนับ
END	เมื่อสิ้นสุดการเขียนโปรแกรม

5.3.1 การใช้งาน PROGRAMMING CONSOLE

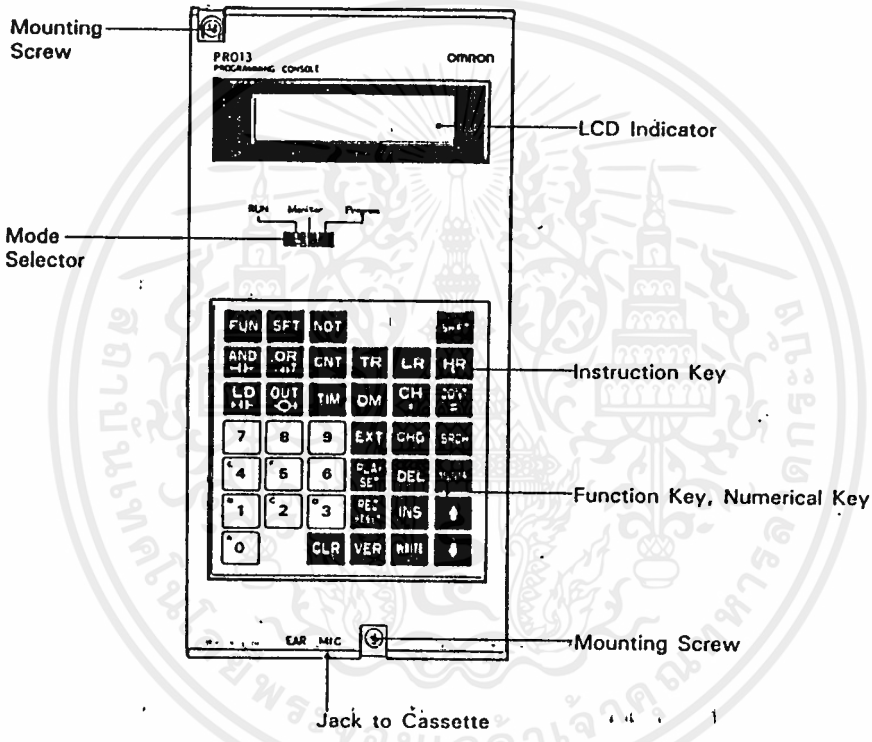
ก) การเลือกโหมดใช้งาน (MODE SELECTOR SWITCH)



PROGRAM MODE เป็น MODE การเขียนโปรแกรมเพื่อใช้ในการสร้าง แก้ไข ลบ แทรกข้อมูล

MONITOR MODE เป็น MODE ที่ PC สามารถ RUN โปรแกรมได้ในขณะเดียวกันสามารถตรวจสอบสภาวะการทำงานต่าง ๆ ของ RELAY, TIMER, COUNTER นอกจากนี้ยังเปลี่ยนแปลงค่าของตัวตั้งเวลาและตัวนับ (SET VALUE)ได้อีกด้วย โดยไม่ต้องหยุดการทำงานใด ๆ ของ PC.

RUN MODE ใช้ในการ RUN โปรแกรมโดยที่ ผู้ใช้ไม่ต้องการที่จะเปลี่ยนแปลง SET VALUE ของ TIMER หรือ COUNTER อีกแล้ว



รูปที่ 5.5 PROGRAMMING CONSOLE

5.3.2 การลบข้อมูลก่อนทำการโปรแกรม (MEMORY AND DATA CLEAR)

(1) จ่ายไฟ (POWER ON SWITCH)ให้กับเครื่อง PC. แล้ว PC.จะแสดงผลยังส่วนแสดงผลดังนี้

จอแสดงผล

```
<PROGRAM >
PASSWORD :
```

(2)เลือก MODE SELECTOR SWITCH ไปยังตำแหน่ง PROGRAM จากนั้นทำการกดที่ KEY ดังนี้

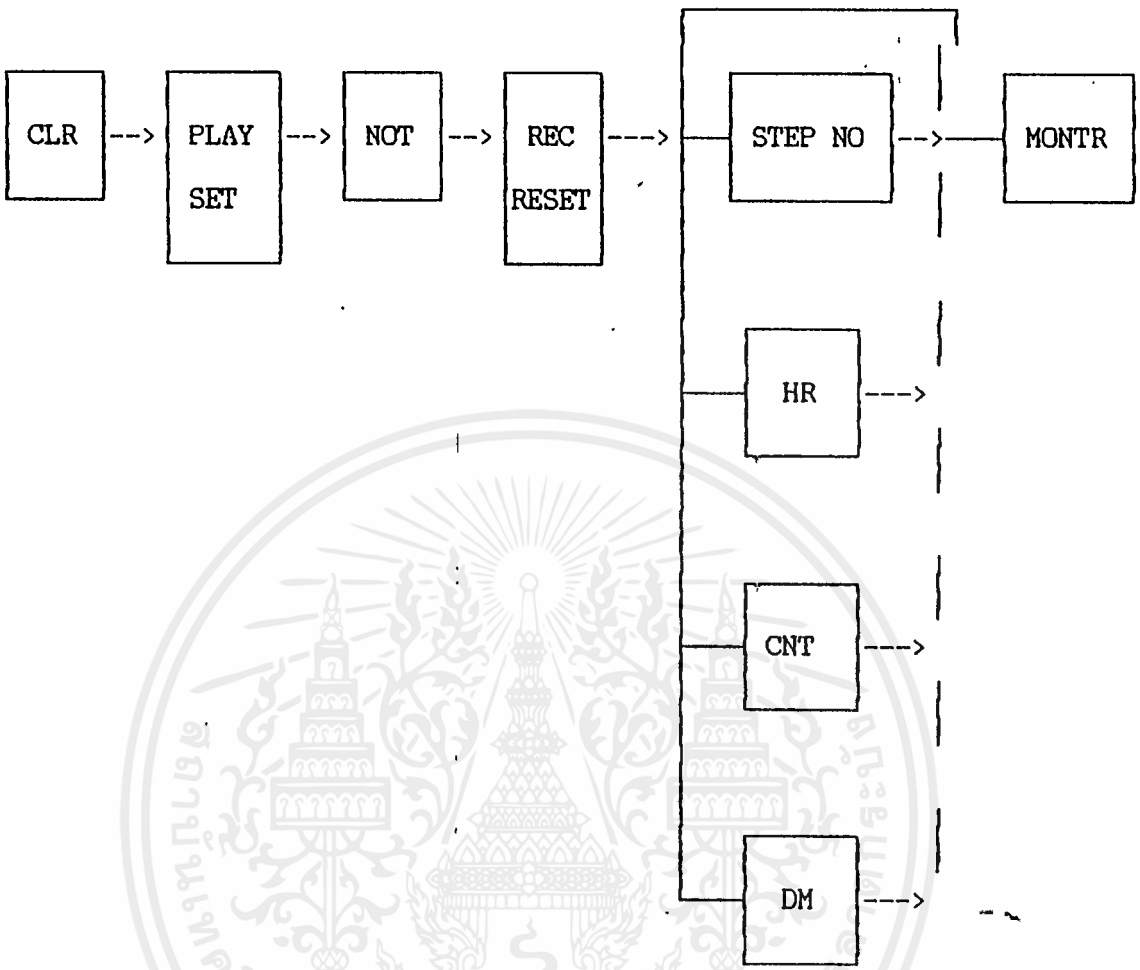
การกดคีย์	จอแสดงผล
CLR --> MONTR --> CLR	0000

วิธีนี้จะเป็นการลบข้อความที่แสดงบนส่วนแสดงผลของ pc. (LCD. INDICATOR) เท่านั้นแต่สำหรับข้อมูลที่อยู่ภายในหน่วยความจำยังคงมีอยู่เช่นเดิม

(3) ทำการลบโปรแกรมทั้งหมดออกจากหน่วยความจำต้องทำการกด KEY ดังนี้

การกดคีย์	จอแสดงผล
SET --> NOT --> RESET	0000 MEMORY CLR?
MON	0000 MEMORY CLR? END
CLR	0000

ภายหลังที่มีการกด KEY ตามขั้นตอนที่กล่าวแล้ว จะทำให้ข้อมูลเดิมถูกลบออกไปทั้งในส่วนของ HR (HOLDING RELAY),CNT (COUNTER)และ DM (DATA MEMORY) แต่ถ้าผู้ใช้ต้องการลบเฉพาะบางส่วนของข้อมูลก็สามารถทำได้ดังนี้

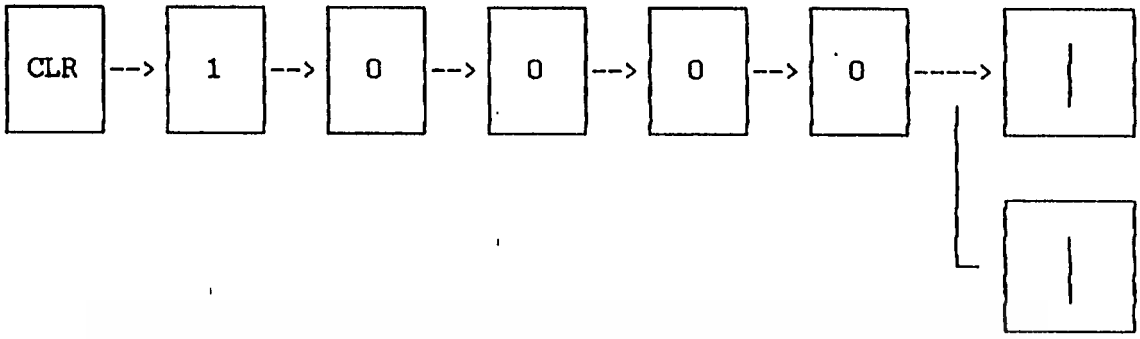


5.3.3 การป้อนโปรแกรม (PROGRAM WRITE)

เมื่อมีการลบข้อมูลเก่าออกหมดแล้วผู้ใช้สามารถทำการเขียนโปรแกรมใหม่เข้าไปได้ทันที และทุกครั้งที่มีการ KEY คำสั่งผู้ใช้จะต้องกด KEY WRITE เสมอ เป็นการเก็บข้อมูลคำสั่งนั้นลงยังหน่วยความจำ

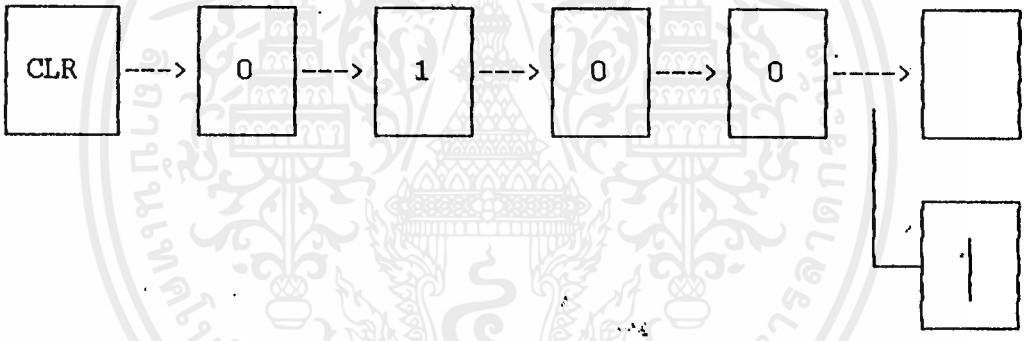
5.3.4 การ SET ADDRESS

เพื่อให้สามารถเลือกไปทำตามแห่งใดของโปรแกรมได้ โดยการข้ามไปที่ ADDRESS นั้น ๆ ทันทีต้องการกด KEY ดังนี้



5.3.5 การอ่านโปรแกรม (PROGRAM RETD)

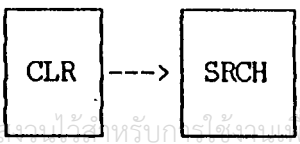
เป็นการอ่านโปรแกรมที่ตำแหน่งใดๆ ซึ่งผู้ใช้สามารถทำการกำหนดได้ ดังเช่น ต้องการอ่านโปรแกรมตำแหน่งที่ 0100 จึงต้องกด KEY ดังนี้

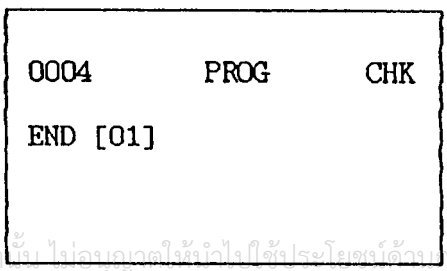


5.3.6 การตรวจสอบโปรแกรม (PROGRAM CHECK)

ภายหลังที่มีการเขียนโปรแกรมเสร็จเรียบร้อยแล้วจะต้องมีการตรวจสอบโปรแกรมอีกครั้ง เพื่อป้องกันความผิดพลาด อันอาจเกิดจากการกำหนดคำสั่งให้กับจุดของ อินพุต เอาต์พุต ผิดพลาดเช่น ใช้คำสั่ง OUT กับตำแหน่งที่ใช้กับเอาต์พุตตำแหน่งเดิม หรือความผิดพลาดจากการเขียน LOGIC ที่นอกเหนือไปจากข้อกำหนดของ PC.

จากการตรวจสอบสามารถกด KEY ได้ดังนี้

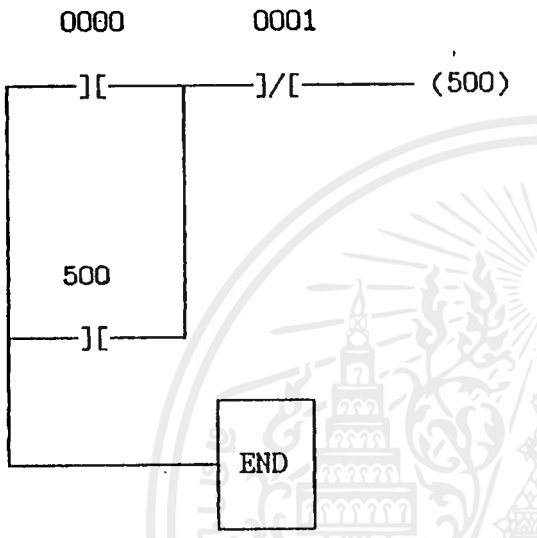




5.3.7 การค้นหาข้อมูลโดยการใช้คำสั่งค้นหา (INSTRUCTION SEARCH)

การค้นหาข้อมูลนี้จะทำให้ผู้ใช้สามารถหาตำแหน่งของ โปรแกรมที่ต้องการแก้ไขหรือต้องการดูค่าสภาวะได้อย่างรวดเร็ว ซึ่งเป็นการค้นหาคำสั่งของโปรแกรมนั่นเอง

ตัวอย่าง

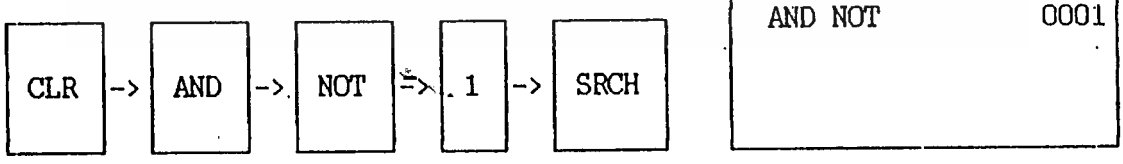


ADDRESS	INSTRUCTION	DATA
0000	LD	0000
0001	OR	500
0002	AND-NOT	0001
0003	OUT	500
0004	END (01)	

รูปที่ 5.6

ถ้าต้องการค้นหาคำสั่ง AND-NOT 0001 ทำได้โดยการกด KEY

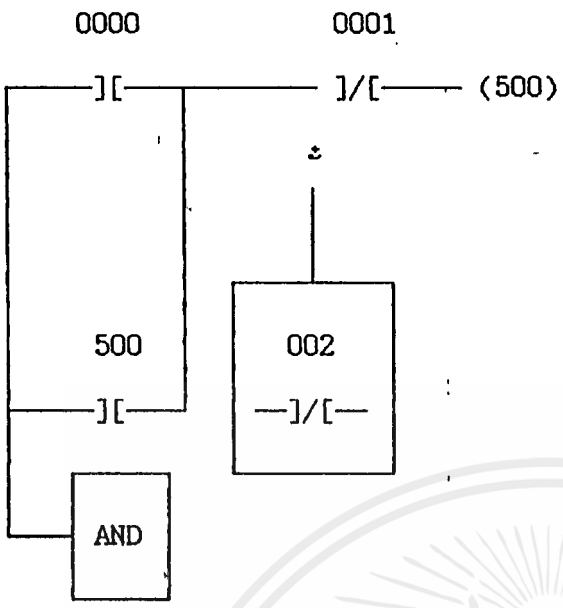
การกดคีย์ จอแสดงผล



0002 SRCH
AND NOT 0001

5.3.8 การแทรกข้อมูลโดยการใช้คำสั่งแทรก (INSTRUCTION INSERT)

ตัวอย่าง

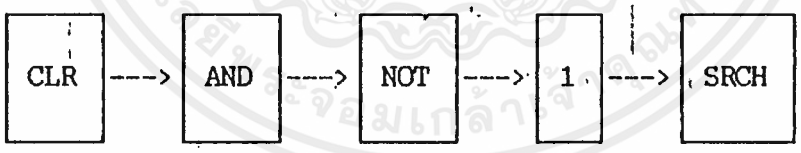


ADDRESS	INSTRUCTION	DATA
0000	LD	0000
0001	OR	500
0002	AND-NOT	0001
0003	OUT	500
0004	END (10)	

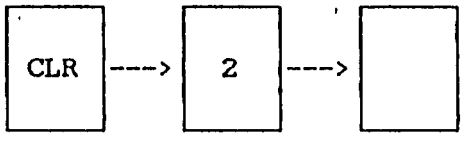
รูปที่ 5.7

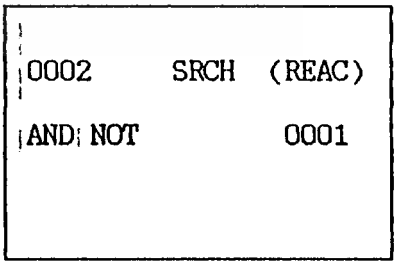
เมื่อต้องการแทรกคำสั่ง AND NOT 0002 เข้าไปหน้าคำสั่ง AND-NOT 0001 ทำได้ตามขั้นตอนดังนี้

(1) ค้นหาคำสั่ง AND-NOT 0001 โดยการกด KEY

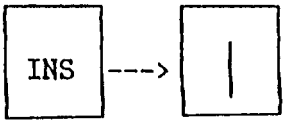


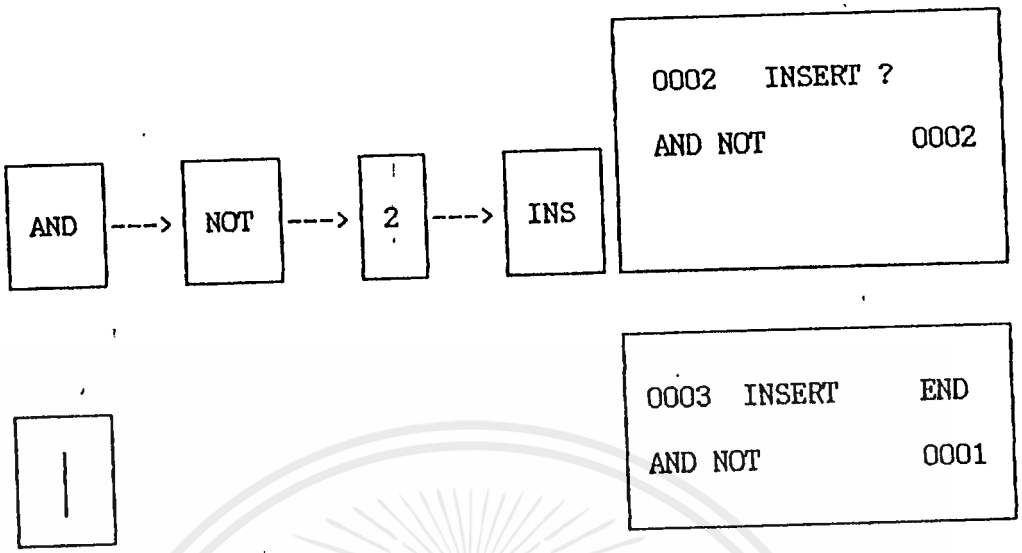
หรือ





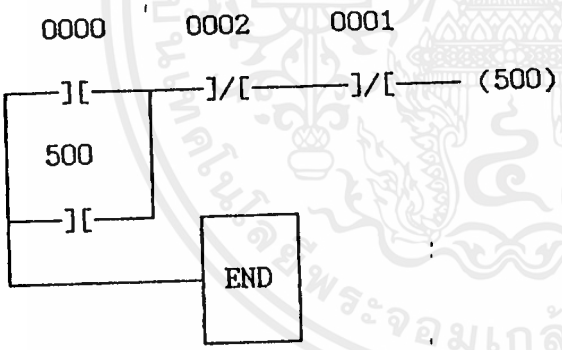
(2) ทำการเขียนคำสั่งที่ต้องแทรกเข้าไปตามแล้วตามด้วย





5.3.9 การลบบางส่วนของโปรแกรมโดยใช้คำสั่งลบ (INSTRUCTION DELETE)

ตัวอย่าง

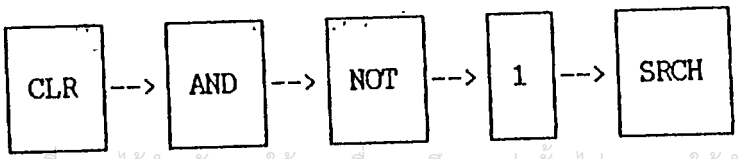


รูปที่ 5.8

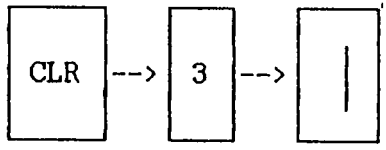
ADDRESS	INSTRUCTION	DATA
0000	LD	0000
0001	OR	500
0002	AND-NOT	0002
0003	AND-NOT	0001
0004	OUT	500
0005	END (01)	

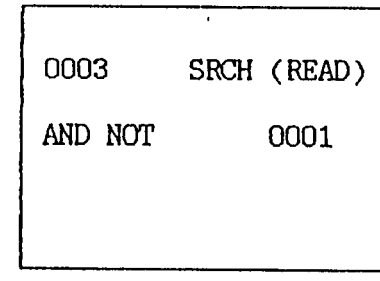
เมื่อต้องการลบบางส่วนของโปรแกรมคือ AND-NOT 0001 ทำได้ดังนี้

(1) ค้นหาคำสั่ง AND-NOT 0001 โดยการกด KEY

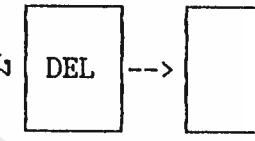


หรือ

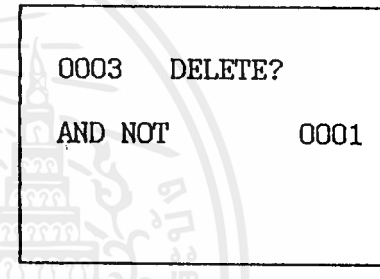


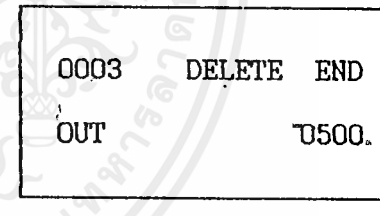


(2) ทำการลบคำสั่งที่ต้องการลบออกจากโปรแกรม โดยคำสั่ง



เมื่อใช้คำสั่งนี้แล้ว จอภาพจะแสดงดังรูป

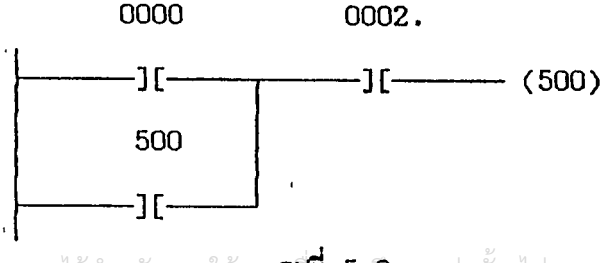




5.3.10 การดูค่าสภาวะเฉพาะจุดหรือหลาย ๆ CHANNEL (DATA MONITOR AND MULTE DATA MONITOR)

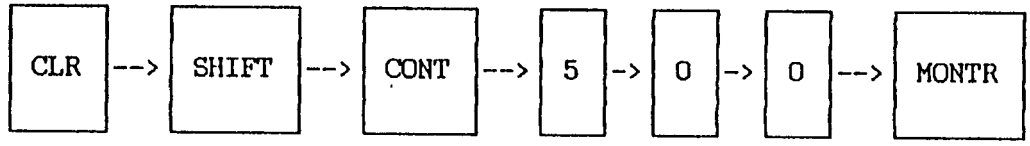
เป็นการที่จะให้ PC. แสดงค่าสภาวะต่าง ๆ ขณะที่กำลังทำงานอยู่ สภาวะที่แสดงได้แก่ I/O RELAY INTERNAL AUXILIARY RELAY, HR RELAY, SPECIAL RELAY, TIM/CNT และ DM CHANNEL

ตัวอย่าง



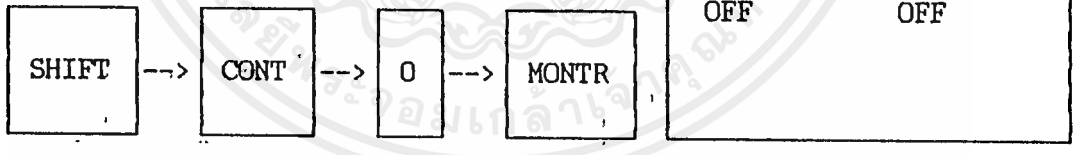
รูปที่ 5.9

(1) ต้องการดูค่าสภาวะ OUTPUT 500 โดยการกด KEY

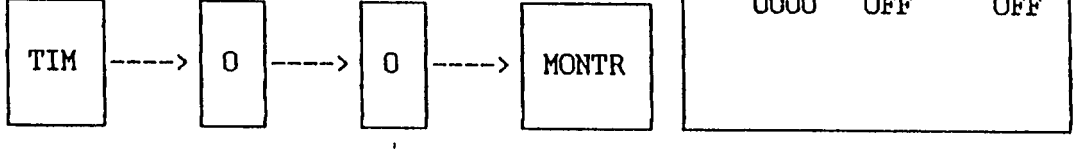


จอแสดงผล

(2) ต้องการดูค่าสภาวะ RELAY NO.0000 ด้วยพร้อมกันทำได้โดยการกด KEY

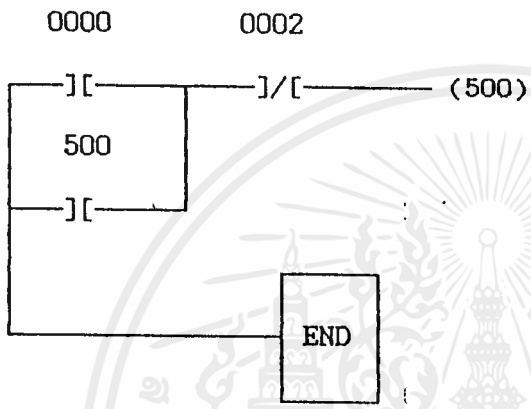


(3) ต้องการกดดูสภาวะที่เป็นค่าของเวลาที่ตัว TIMER 00 โดยการกด KEY



5.3.11 การเซ็ทและรีเซ็ทจาก KEY BOARD (FORCED SET/RESET)

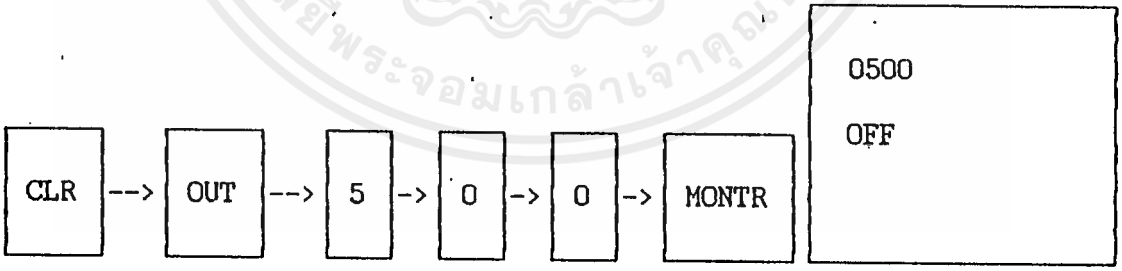
ในส่วนนี้จะเป็นประโยชน์อย่างมากในการที่จะตรวจสอบการทำงานของ OUTPUT โดยที่ไม่ต้องผ่านการทำงานของโปรแกรม ซึ่งผู้ใช้สามารถสั่งการจาก KEY BOARD ของ PC. ได้โดยตรงแต่ MODE SELECTOR SWITCH ต้องไม่อยู่ใน MODE RUN การสั่งการสามารถสั่งได้ทั้ง INPUT, OUTPUT, INTERNAL AUXILIARY RELAYS, COUNTER TIME, HOLDING RELAY



ADDRESS	INSTRUCTION	DATA
0000	LD	0000
0001	OR	500
0002	AND-NOT	0002
0003	OUT	500
0004	END(01)	

รูปที่ 5.10

เมื่อต้องการ FORCE เอาท์พุตตำแหน่งที่ 500 โดยไม่ผ่านการทำงานของโปรแกรมให้เป็น "ON" หรือ "OFF" จาก KEY BOARD โดยตรงให้กด KEY ดังนี้



จากนั้น KEY

RESET

0500 OFF

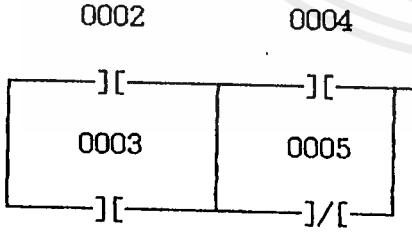
หมายเหตุ รีเลย์พิเศษตำแหน่งที่ 1808 และ 1907 ไม่สามารถทำการ FORCE ได้

5.4 หลักการเขียนโปรแกรม

จากที่ได้กล่าวมาแล้วในตอนต้นของบท ถึงการเขียนโปรแกรมขั้นพื้นฐาน คือการใช้คำสั่ง LD AND OR NOT ผู้ใช้คงจะคุ้นเคยกับคำสั่งเหล่านี้แล้วแต่คำสั่งที่จะแนะนำต่อไปเป็นคำสั่งที่เกี่ยวกับ BLOCK ได้แก่ AND LD และ OR LD

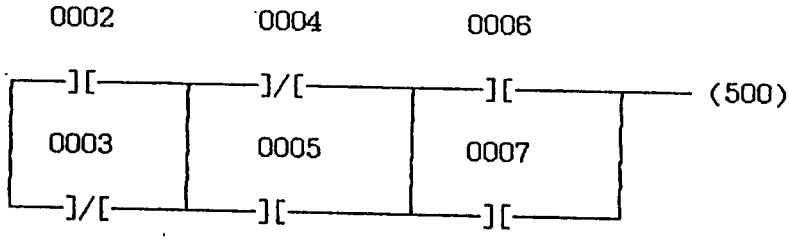
5.4.1 การใช้คำสั่ง AND-LD

เป็นคำสั่ง BLOCK ที่เป็นการนำ BLOCK หลาย ๆ BLOCK มากระทำ LOGIC ANC หรือต่อ แบบ SERIES ซึ่งกันและกัน



ADDRESS	INSTRUCTION	DATA
0000	LD	0002
0001	OR	0003
0002	LD	0004
0003	OR-NOT	0005
0004	AND-LD	--
0005	OUT	500

รูปที่ 5.11 AND-LD



ถ้า LADDER DIAGRAM เป็นตามรูป ผู้ใช้สามารถเขียนโปรแกรม MNEMONICได้ 2 แบบ

วิธีที่ 1

INSTRUCTION	DATA
LD	0002
OR-NOT	0003
LD-NOT	0004
OR	0005
AND-LD	--
LD	0006
OR	0007
AND-LD	--
OUT	0500

วิธีที่ 2

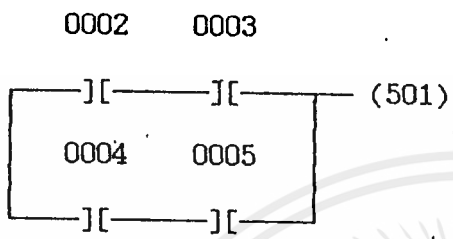
INSTRUCTION	DATA
LD	0002
OR-NOT	0003
LD-NOT	0004
OR	0005
LD	0006
OR	0007
AND-LD	-
AND-LD	-
OUT	0500

รูปที่ 5.12 NUMBER OF BLOCKS FOR AND-LD

จากวิธีที่ 2 การใช้คำสั่ง AND-LD ติดต่อกันนั้นจำนวนคำสั่ง LD และ LD-NOT ก่อนที่จะถึงคำสั่ง AND-LD มีได้มากสุดไม่เกิน 8 ครั้ง แต่ถ้ามีมากกว่าให้ใช้วิธีที่ 1

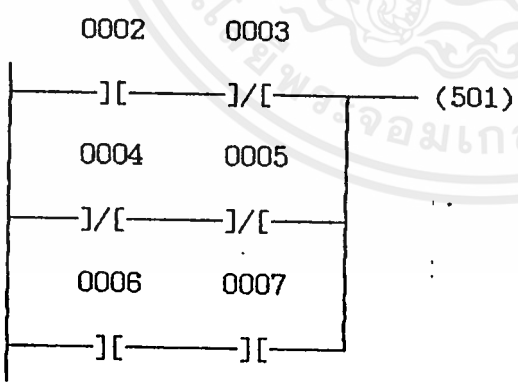
5.4.2 การใช้คำสั่ง OR-LD

คำสั่งนี้ก็เป็นคำสั่ง BLOCK ที่เป็นการนำ BLOCK ต่าง ๆ มากระทำ LOGIC OR หรือ PARALLEL ซึ่งกันและกัน



ADDRESS	INSTRUCTION	DATA
0000	LD	0002
0001	AND-NOT	0003
0002	LD	0004
0003	AND	0005
0004	OR-LD	-
0005	OUT	501

รูปที่ 5.13 OR-LD



เช่นเดียวกับคำสั่ง AND-LD คือสามารถเขียนได้ 2 วิธี

INSTRUCTION	DATA
LD	0002
AND-NOT	0003
LD-NOT	0004
AND-NOT	0005
OR-LD	-
LD	0006
AND	0007
OR-LD	-
OUT	0501

INSTRUCTION	DATA
LD	0002
AND-NOT	0003
LD-NOT	0004
AND-NOT	0005
LD	0006
AND	0007
OR-LD	-
OR-LD	-
OUT	0501

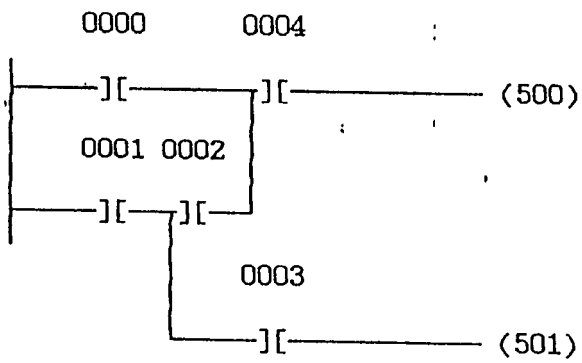
รูปที่ 5.14 NUMBER OF BLOCKS FOR OR-LD

ข้อกำหนดในการเขียนโปรแกรม

(1) จำนวน CONTACT ของ I/O, INTERNAL AUXILIARY RELAY, TIM/CNT จะมีโหลดเพื่อนำมาเขียนโปรแกรมเป็นจำนวนเท่าใดก็ตามความประสงค์ของผู้ใช้ แต่ถึงอย่างไรก็ตามการเขียนโปรแกรมที่ดีจะต้องพยายามประหยัดให้มากเท่าที่สามารถจะทำได้

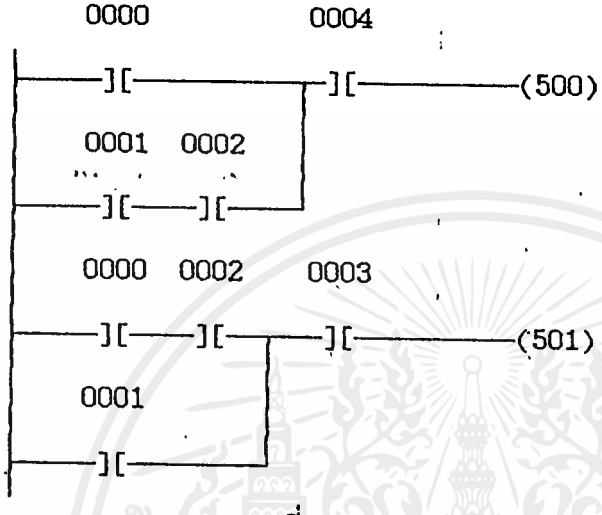
(2) สำหรับ LADDER DIAGRAM การพิจารณาจะกระทำจากซ้ายไปขวาเท่านั้น

ตัวอย่าง



รูปที่ 5.15

จากรูปที่ 5.15 ถ้า CONTACT 0000 0002 และ 0003 มีสภาวะ "ON" ก็ไม่สามารถทำให้ OUTPUT 501 นั้น "ON" ได้เลย ดังนั้นผู้ใช้จะต้องทำการจัดโปรแกรมเสียใหม่เพื่อให้การพิจารณาการกระทำจากซ้ายไปขวา ดังรูปที่ 5.16



รูปที่ 5.16

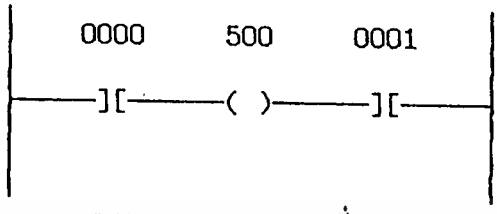
(3) การเขียนโปรแกรมเพื่อต่อ COIL ให้กับ BUS ทางด้านซ้ายโดยตรงไม่สามารถทำได้ ในกรณีที่ต้องการให้ทำงานลักษณะเหมือนกับต่อโดยตรงทำได้ โดยการใช้ SPECIAL - AUXILIARY RELAY 1813 ที่มีสภาวะเป็น "ON" เมื่อ RUN โปรแกรม

LADDER DIAGRAM ที่ผิด LADDER DIAGRAM ที่ถูก

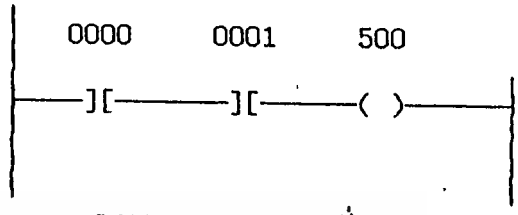
(4) จำนวน CONTACT ที่ใช้ในการต่อ SERIES หรือ PARALLEL ไม่มีขีดจำกัดจะใช้เท่าใดก็ขึ้นอยู่กับความต้องการของผู้ใช้

(5) OUTPUT ทุก ๆ OUTPUT มี AUXILIARY CONTACT เพื่อใช้ในการเขียนโปรแกรมและมีไม่จำกัดจำนวน

(6) ไม่สามารถเขียนโปรแกรมให้ CONTACT อยู่ทางด้านขวาของ COIL ได้

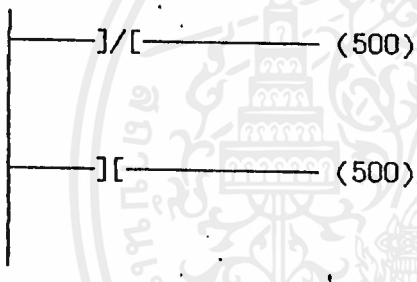


LADDER DIAGRAM ที่ผิด

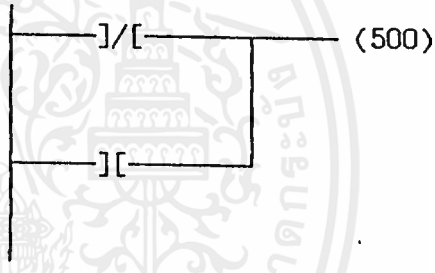


LADDER DIAGRAM ที่ถูก

(7) ไม่สามารถเขียนโปรแกรมให้มี OUTPUT COIL หมายเลขเดียวกันซ้ำหลาย ๆ ครั้ง ได้ต้องจัดรูปเสียใหม่

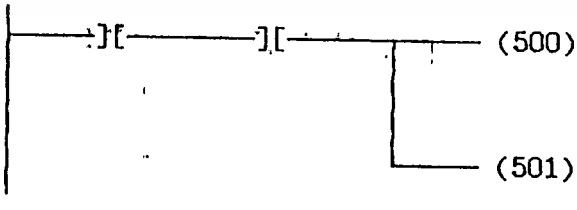


LADDER DIAGRAM ที่ผิด

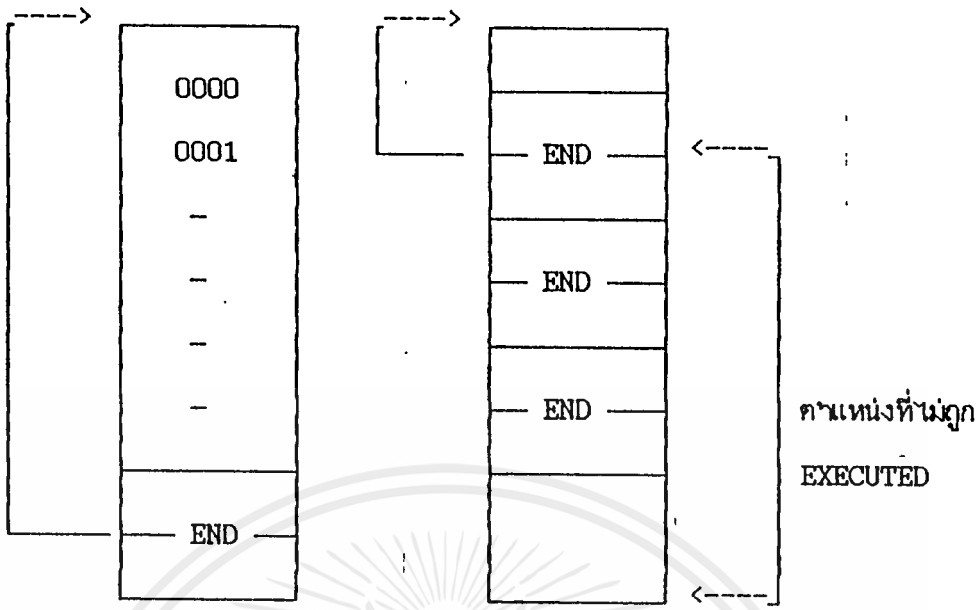


LADDER DIAGRAM

(8) OUTPUT COIL สามารถเขียนโปรแกรมให้ต่อ PARALLEL ได้เพื่อรับเงื่อนไขของ CONTACT ชุดเดียวกัน



(9) โปรแกรมจะถูก EXECUTED จาก ADDRESS แรกจนกระทั่งถึงคำสั่ง END ที่เป็นคำสั่งแรกซึ่งถูกใช้ โดยที่ END อาจจะมีหลายตำแหน่งก็เป็นไปได้ ที่เป็นเช่นนี้เพื่อจุดประสงค์สำหรับการ TEST RUN กรณีแยกโปรแกรมรวมออกเป็นส่วน ๆ และง่ายต่อการตรวจสอบ-แก้ไขโปรแกรม



รูปที่ 5.17 CPU สแกนจาก ADDRESS 0000 ถึง END

ในการใช้คำสั่งนอกเหนือไปจากคำสั่งพื้นฐานแล้ว KEY สำหรับกดบน PROGRAMMING CONSOLE ที่เป็นเฉพาะคำสั่งต่าง ๆ นั้นจะไม่มีให้ใช้ ทั้งนี้เพื่อเป็นการประหยัดและไม่ต้องการให้มี KEY มากจนเกินไปจึงได้เปลี่ยนวิธีการกดจาก DIRECT KEY มาเป็น FUNCTION KEY โดยมีรูปแบบของ FUNCTION KEY ดังนี้

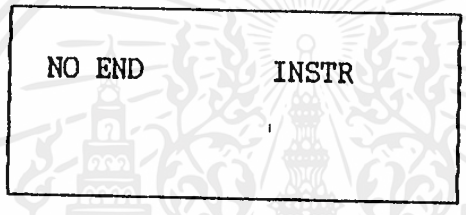
"RUN XX" เมื่อ XX คือ ระหัสที่บ่งบอกถึงคำสั่งต่าง ๆ ของ PC. ตัวอย่างเช่น

FUN 01	หมายถึงคำสั่ง	END (END INSTRUCTION)
FUN 02	"	IL (INTERLOCK)
FUN 03	"	ILC (INTERLOCK CLEAR)
FUN 04	"	JMP (JUMP)
FUN 05	"	JME (JUMP END)
FUN 10	"	SFT (SHIFT REGISTER)
FUN 11	"	KEEP (LATCHING RELAY)
FUN 12	"	CNTR (REVERSIBLE COUNTER)

FUN 13	"	DIFU (DIFFERENTIATION-UP)
FUN 14	"	DIFD (DIFFERENTIATION-DOWN)
FUN 15	"	TIMH (HIGH SPEED TIMER)

5.4.3 การใช้คำสั่ง END (FUN 01)

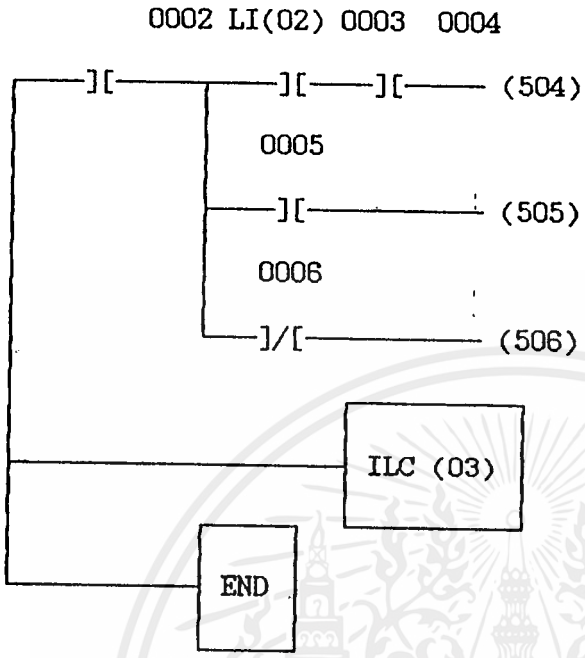
ในการเขียนโปรแกรมทุกครั้งที่สิ้นสุดการเขียนโปรแกรมจะต้องจบด้วยคำสั่ง END แต่ถ้าไม่มีคำสั่ง END แล้ว เมื่อผู้ใช้ทำการ RUN โปรแกรม PC. จะแสดงข้อความ



จากข้อความแสดงว่าไม่มีคำสั่ง END โปรแกรมจะไม่สามารถ RUN ได้ และ INDICATOR ERROR. จะสว่าง

5.4.4 การใช้คำสั่ง IL (FUN 02) ILC (FUN 03) และ TR

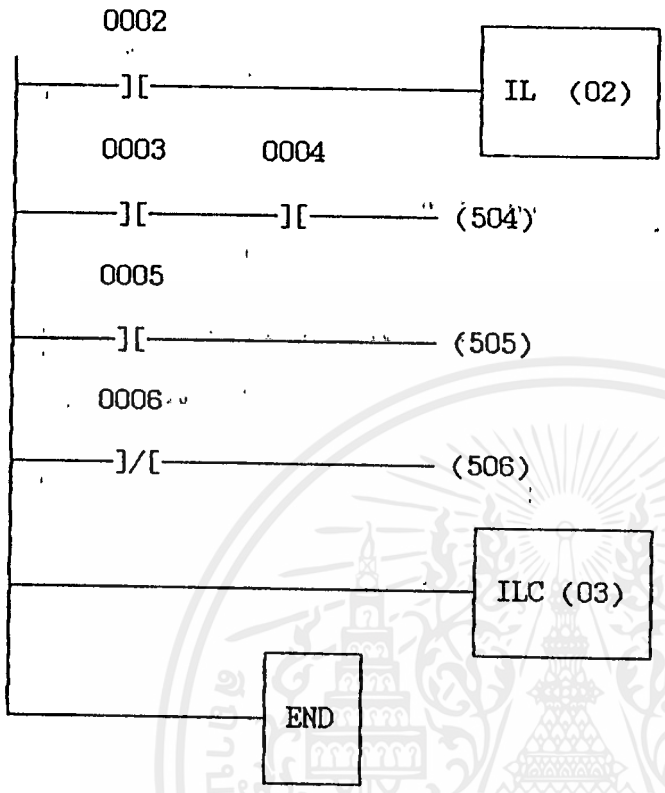
คำสั่ง IL และ ILC จะต้องร่วมกัน คือ ถ้ามีการเริ่มต้นคำสั่งด้วย IL เมื่อใดแล้วในขณะเดียวกัน ต้องการสิ้นสุดการทำงานต้องจบด้วย ILC เงื่อนไขของคำสั่งถ้า CONTACT ตรงส่วนหน้าของ IL มีสภาวะ "ON" จะทำให้โปรแกรมที่อยู่ระหว่าง IL และ ILC ทำงานเป็นปกติ แต่ถ้า CONTACT ตำแหน่งดังกล่าว มีสภาวะ "OFF" ผลจะทำให้การทำงานของโปรแกรมระหว่าง IL และ ILC ไม่ทำงาน ในขณะเดียวกัน OUTPUT COIL ในช่วงนั้นจะต้องมีสภาวะ "OFF" ตามไปด้วยเช่นกัน



ADDRESS	INSTRUCTION	DATA
0000	LD	0002
0001	IL (02)	-
0002	LD	0003
0003	AND	0004
0004	OUT	0504
0005	LD	0005
0006	OUT	0505
0007	LD-NOT	0006
0008	OUT	0506
0009	ILC (03)	-
0010	END (01)	-

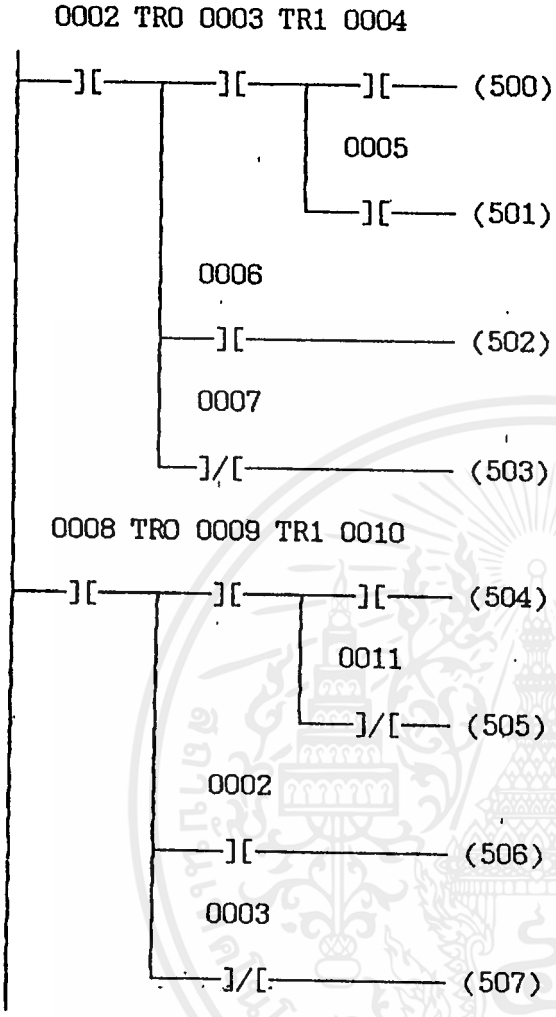
รูปที่ 5.18 IL/ILC

จากรูป 5.18 ถ้าจะเขียน LADDER DIAGRAM ขึ้นมาเสียใหม่เพื่อให้เข้าใจง่ายยิ่งขึ้น จะได้ LADDER DIAGRAM ใหม่ ดังนี้

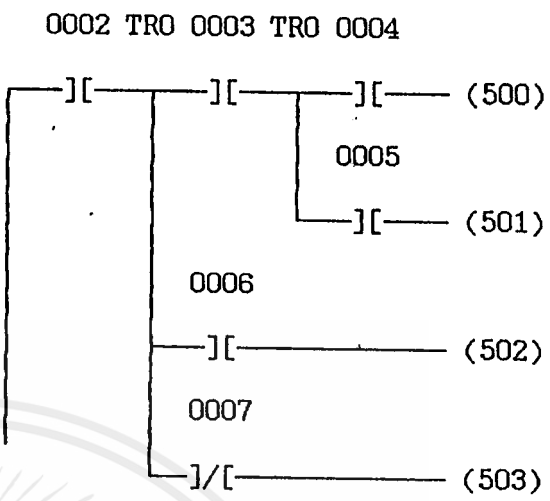


รูปที่ 5.19 REWRITTER PROGRAM FOR IL และ ILC

TR (TEMPORARY MEMORY RELAY) สำหรับคำสั่งนี้จะใช้กับ LADDER ที่มี OUTPUT COIL อยู่หลาย ๆ สาขาโดยที่สาขาหนึ่ง ๆ จะประกอบไปด้วยคำสั่ง LD และ OUT คำสั่ง TR นี้จะมีให้เรียกใช้ตั้งแต่ TR0 จนถึง TR7 ซึ่งในสาขาเดียวกันจะใช้ TR ที่ซ้ำกันไม่ได้แต่ถ้าเป็นสาขาใหญ่หลายสาขาเราจะใช้ TR0-7 ในสาขาใหญ่นั้นใหม่ได้อีก



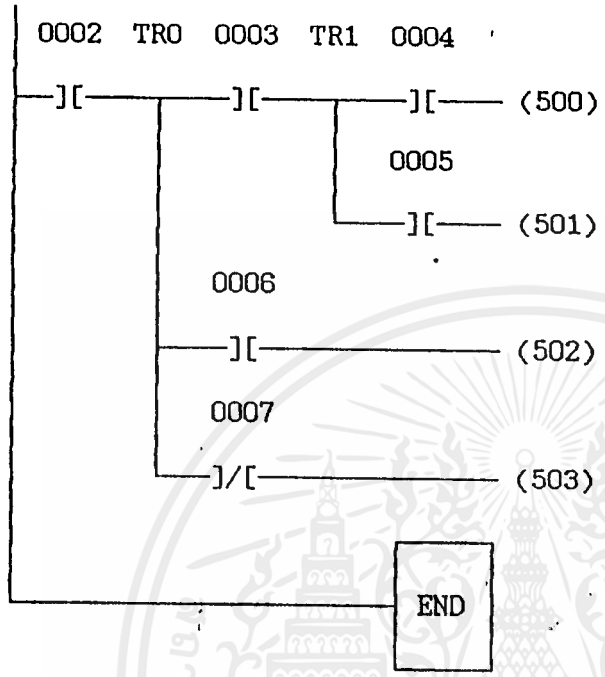
รูปที่ 5.20 LADDER DIAGRAM ที่ใช้งานได้



รูปที่ 5.21 LADDER DIAGRAM ที่ใช้งานไม่ได้

TR จะถูกใช้ก็ต่อเมื่อการเขียนโปรแกรมที่ไม่สามารถใช้คำสั่ง IL และ ILC ได้เช่น โปรแกรมประเภทเอาท์พุตหลาย ๆ สาขาในกรณีเช่นนี้ถ้าใช้ TR จะเหมาะสมมากกว่า

ตัวอย่าง การเขียน LADDER DIAGRAM ด้วย TR



ADDRESS	INSTRUCTION	DATA
0000	LD	0002
0001	OUT	TR0
0002	AND	0003
0003	OUT	TR1
0004	AND	0004
0005	OUT	0500
0006	LD	TR1
0007	AND	0005
0008	OUT	0501
0009	LD	TR0
0010	AND	0006
0011	OUT	0502
0012	LD	TR0
0013	AND-NOT	0007
0014	OUT	0503
0015	END [01]	-

รูปที่ 5.22

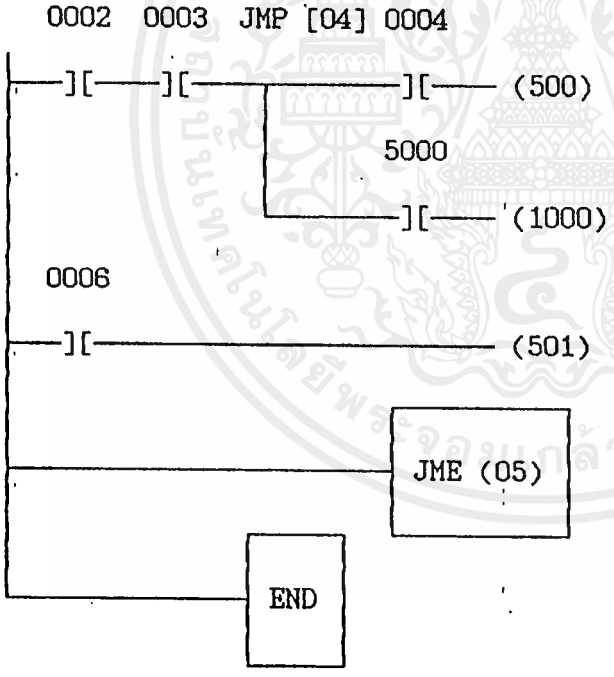
5.4.5 การใช้คำสั่ง JMP (FUN 04) และ JME (FUN 05)

ข้อกำหนดของการใช้คำสั่งนี้

(1) ในการเขียนโปรแกรมนั้นจะสามารถใช้คำสั่งคู่นี้ได้ 8 ครั้ง

(2) ห้ามใช้ HIGH-SPEED COUNTER (FUN 98) ระหว่างคำสั่งคู่นี้

การใช้งานของคำสั่งคู่นี้จะต้องใช้งานคู่กันเงื่อนไขต่าง ๆ ที่อยู่ระหว่างคำสั่ง JMP และ JME จะมีเงื่อนไขการทำงานเป็นปกติในกรณีที่ชุดของ CONTACT ของ JMP มีสภาวะเป็น "ON" แต่ถ้าชุด CONTACT ดังกล่าวมีสภาวะเป็น "OFF" เมื่อใด OUTPUT, TIMER, COUNTER,KEEP ที่อยู่ระหว่างคำสั่งดังกล่าวจะยังคงค้างค่าสภาวะเอาไว้เช่นเดิม CONTACT มีสภาวะ "ON"



ADDRESS	INSTRUCTION	DATA
0000	LD	0002
0001	AND	0003
0002	JMO (04)	–
0003	LD	0004
0004	OUT	0500
0005	LD	0005
0006	OUT	1000
0007	LD	0006
0008	OUT	0501
0009	JME (05)	–
0010	END (01)	

รูปที่ 5.23 JMP/JME

บทที่ 6

บทสรุปการทดลอง

จากโครงงานดังกล่าวเป็นโครงงานที่นำเอาคอมพิวเตอร์ และพีแอลซี มาทำการอินเตอร์เฟสกัน โดยผ่านการ์ดอินเตอร์เฟส 8255 ซึ่งในโครงงาน จะแบ่งการทำงานออกเป็น 3 ส่วนใหญ่ ๆ คือ ส่วนของซอฟแวร์ ส่วนของฮาร์ดแวร์ และส่วนของ พีแอลซี

ในส่วนของซอฟแวร์นั้น จะเป็นการส่งข้อมูลไปควบคุมการทำงาน หรือสั่งการทำงานของเครื่อง พีแอลซี ให้ทำตามคำสั่งที่ต้องการ และในขณะเดียวกัน ก็จะรับข้อมูลจากเครื่อง พีแอลซี มาแสดงผลทางหน้าจอของเครื่องคอมพิวเตอร์ เพื่อแสดงสภาวะการทำงานทั้งหมดของขบวนการทางด้านฮาร์ดแวร์ จะเป็นตัวรับ-ส่งข้อมูล ระหว่างอุปกรณ์ภายนอกกับเครื่องคอมพิวเตอร์ เนื่องจาก 8255 มีพอร์ตสำหรับติดต่อกับอุปกรณ์ภายนอกถึง 3 พอร์ต และยังสามารถเลือกโหมดการทำงานได้อีกด้วย 8255 จะคอยตรวจสอบสัญญาณต่าง ๆ ของอุปกรณ์ภายนอก หรือที่เรียกว่า HAND CHAKE ซึ่งอยู่ในโหมดการทำงานของ 8255 ขณะเดียวกัน ที่การ์ดนี้ จะต้องมีการดีโค้ดสัญญาณต่าง ๆ บนสล๊อตของ IBM PC/AT เพื่อให้ 8255 ทำงานที่แอดเดรสที่เหมาะสม ที่แผ่นการ์ดนี้ ยังมีวงจร เอทูดี ซึ่งจะมีประโยชน์มาก สามารถที่จะรับสัญญาณอนาลอก จากพีแอลซี แล้งแปลงเป็นสัญญาณดิจิตอล เพื่อส่งไปให้คอมพิวเตอร์ส่วนทางด้าน พีแอลซี ก็จะเป็นอุปกรณ์รับ-ส่ง ภายนอกธรรมดา

เอกสารอ้างอิง

[1] ยืน ภู่วรวรรณ "เทคโนโลยีฮาร์ดแวร์ IBM PC" สำนักพิมพ์ ซีเอ็ด ยูเคชั่น, พ.ศ. 2533.

[2] สุพรรณ กุลพาณิชย์ "เทคนิคและการใช้งาน PLC." บริษัทออมร่อนตรีศักดิ์ จำกัด 248 ถนนรัชดาภิเษก ห้วยขวาง กรุงเทพฯ 10310, พ.ศ.2533.

[3] Borland "TURBO C 1.5" Borland International 4585 Scotts Valley Drive Scotts Valley. California 95066.

[4] รศ.มัณฑนา ปราการสมุทร "การเขียนชุดคำสั่งภาษา C" บริษัท ดวงกมลสมัย จำกัด 90/21-25 ถนนราชปรารถ เขตพญาไท กทม. 10400, พ.ศ.2533.

[5] "คู่มือดูชาไอซี" สำนักพิมพ์ซีเอ็ดยูเคชั่น, พ.ศ.2531.

[6] "คู่มือเทียบเบอร์ไอซี TTL" สำนักพิมพ์ซีเอ็ดยูเคชั่น, พ.ศ.2531

กิตติกรรมประกาศ

ขอขอบพระคุณ อาจารย์วิทยา ทิพยสุวรรณกร เป็นอย่างสูงที่ได้ประสิทธิ์ประสาทวิชาความรู้ และคำแนะนำที่เป็นประโยชน์ต่อปริญญานิพนธ์ฉบับนี้จนทำให้ปริญญานิพนธ์ฉบับนี้เสร็จสมบูรณ์ ตามวัตถุประสงค์ที่คณะผู้จัดทำปริญญานิพนธ์ตั้งเป้าหมายไว้ทุกประการ

ภาคผนวก

```
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <graphics.h>
#include <alloc.h>
#include <conio.h>

#define  left   0x4B00
#define  right  0x4D00
#define  null   ''
#define  enter  0x1c0d

char  *comp[] = {"PUMP1","PUMP2","PUMP3" };
char  *ckey[] = {"OFF","OFF","OFF","STOP","RUN","EXIT"} ;
char  *stat[] = {"ON","OFF"};

int xm,ym;
int initial=0,status[3] = {1,1,1};;
int pos = 0,buff=0;

main()
{
void smalltank (int x,int y);
void bigtank (int x,int y);
void spind(int x,int y );
void val (int x,int y,int color);
void pump (int x,int y);
void showpic (void);
void flow1(int color);
void flow2(int color);
void flow3(int color);
void key (int x,int y,char ch[10],int  color);
void pannel (void);
void runplc (void);
int adc_data(void);
void adc_graph(void);

int ErrorCode,GraphDriver,GraphMode ;
int i,opt,kda,buf,z;
unsigned int data;

  GraphDriver = DETECT;                    /* Request auto-detection
  initgraph( &GraphDriver, &GraphMode, "" );
  ErrorCode =    graphresult();  /*                    Read result of ini
  if( ErrorCode != grOk ){                 /* Error occured during ini
    printf(" Graphics System Error: %s\n", grapherrormsg( ErrorCode
    exit( 1 );
}

xm=getmaxx();
ym=getmaxy();
setbkcolor(BLUE );
settextstyle (DEFAULT_FONT,HORIZ_DIR,2);
setcolor(RED);
outtextxy(250,100,"PC & PLC ");
outtextxy(100,220,"APPLICATION PROCESS CONTROL");
settextstyle (DEFAULT_FONT,HORIZ_DIR,1);
outtextxy(230,350,"PRESS ANY KEY TO CONTINUE ");
getch();
pannel();
showpic();
setcolor(1);
setfillstyle (SOLID_FILL,YELLOW);
bar(77,288,123,328);
```

```
setfillstyle (SOLID_FILL,LIGHTMAGENTA);
bar(157,288,203,328);
setfillstyle(SOLID_FILL,YELLOW);
bar(400,50,600,200);
outtextxy(500,170,"0.0000");
outtextxy(500,40,"ADC DATA INPUT");
key(50+pos*80,431,ckey[pos],11);
do {
while(!kbhit()) if(initial==1) { runplc(); adc_graph(); }

   kda=bioskey(0);
       switch(kda) {
case left    :  if(pos<=2) key(50+pos*80,431,stat[status[pos]],14);
                else key(50+pos*80,431,ckey[pos],14);
                pos = (pos==0) ? 5 : --pos;
                if(pos<=2) key(50+pos*80,431,stat[status[pos]],11);
                else key(50+pos*80,431,ckey[pos],11);
                break;
case right   :  if(pos<=2) key(50+pos*80,431,stat[status[pos]],14);
                else key(50+pos*80,431,ckey[pos],14);
                pos = (pos==5) ? 0 : ++pos;
                if(pos<=2) key(50+pos*80,431,stat[status[pos]],11);
                else key(50+pos*80,431,ckey[pos],11);
                break;
case enter   :  if(pos==4) { buff = 0;
                             runplc();
                             initial = 1;
                             }
                if((pos<=2)&&(initial==1)&&(status[pos]==0)) {

                             status[pos] = 1;
                             key(50+pos*80,431,stat[status[pos]],11)

                             if(pos==0) {
                                        outportb(0x300,1);
                                          }
                             else if(pos==1) {
                                        outportb(0x300,2);
                                                }
                             else       outportb(0x300,4);
                                                }
                if (pos==3) {
                               outportb(0x300,5);
                               initial = 0;
                               flow3(15);
                               flow2(15);
                               flow1(15);
                            for(i=0;i<=2;i++){
                              status[i] = 1;
                              key(50+i*80,431,stat[1],14);
                                                  }
                               }

                break;

default:break;
                    }

}while(!((kda==enter)&&(pos==5)));
closegraph();

} /* END MAIN */

void runplc (void)
{
```

```
unsigned int data  ;
int spc:


if (initial==0) {

outportb(0x303,0x82);
outportb(0x300,3);

for(spc=0;spc<=10000;spc++);

outportb(0x300,0);
                  }
data=inportb(0x301);

 switch(data)
              {

   case 0xf8 :  if(buff !=data)
               {
                flow3(15);
                flow1(14);
                setfillstyle (1,1);
                setcolor(1);
                setfillstyle (SOLID_FILL,BLUE );
                bar (285,270,375,230);
                status[0] = 0;
                status[1] = 1;
                status[2] = 2;
                spc = (pos==0) ? 11 : 14;
                key(50-0*80,431,stat[0],spc);
                spc = (pos==2) ? 11 : 14;
                key(50+2*80,431,stat[1],spc);
                buff = data;
                }
                break;
   case 0xfa :  if(buff !=data)
                {
                   flow1(15);
                   flow2(13);

                   status[0] = 1;
                   status[1] = 0;
                   status[2] = 1;
                   setfillstyle(SOLID_FILL,BLUE);
                   bar (285,230,375,190);
                   spc = (pos==1) ? 11 : 14;
                   key(50+1*80,431,stat[0],spc);
                   spc = (pos==0) ? 11 : 14;
                   key(50+0*80,431,stat[1].spc
                   buff = data;
                   }
                break;
   case 0xfc :  if(buff !=data)
                {
                     flow2(15);
                     flow3(1);
                     status[0] = 1;
                     status[1] = 1;
                     status[2] = 0;
                     setfillstyle(SOLID_FILL,BLUE );
                     bar(507,288,553,328);
                     setfillstyle(SOLID_FILL,WHITE);
                     bar (285,270,375,200);
                     spc = (pos ==2) ? 11 : 14;
```

```
                        key(50+2*80,431,stat[0],spc);
                        spc = (pos== 1) ? 11 : 14;
                        key(50+1*80,431,stat[1],spc);
                        buff = data;
                        }
                    break;
        default : break;
                    }
    }

    void smalltank (int x,int y)
    {
    setlinestyle(0,1,3);
    line(x,y,x-25,y);
    line(x,y,x+25,y);
    line(x-25,y,x-25,y-50);
    line(x+25,y,x+25,y-50);
    line(x-25,y-50,x-30,y-52);
    line(x+25,y-50,x+30,y-52);
    }
    void bigtank (int x,int y)
    {
    setcolor(2);
    setlinestyle(0,1,3);
    line(x,y,x-50,y);
    line(x,y,x+50,y);
    line(x-50,y,x-50,y-100);
    line(x+50,y,x+50,y-100);
    line(x-50,y-100,x-55,y-102);
    line(x+50,y-100,x+55,y-102);
    /***************************/
    line(x,y-3,x-47,y-3);
    line(x,y-3,x+47,y-3);
    line(x-47,y,x-47,y-103);
    line(x+47,y,x+47,y-103);
    line(x-47,y-103,x-55,y-103);
    line(x+47,y-103,x+55,y-103);
    /*base*/
    setlinestyle(0,1,3);
    line(x-50,y,x-100,y+40);
    line(x+50,y,x+100,y+40);
    line(x-100,y+40,x-100,y+55);
    line(x+100,y+40,x+100,y+55);
    line(x-100,y+55,x+100,y+55);
    }
    /*************/
    void spind(int x,int y )
    {
    struct PTS {
    int  x, y;  };          /* Structure to hold vertex points */
    struct PTS poly[ 5 ];
    y-=4;
    setfillstyle(SOLID_FILL,7);
    setcolor(1);
    poly[0].x=x-25;
    poly[0].y=y;
    poly[1].x=x+25;
    poly[1].y=y;
    poly[2].x= x;
    poly[2].y= y-25;
    fillpoly( 3, (int far *)poly );     /* Draw the actual polygon */
    setlinestyle(0,1,1);
    line(x,y,x-25,y);
    line(x,y,x+25,y);
    line(x-25,y,x,y-25);
    line(x+25,y,x,y-25);
```

```
setcolor(3);
y+=2;
/*shaft*/
setcolor(1);
line(x-2,y,x-2,y+20);
line(x+2,y,x+2,y+20);
/* motor*/
setfillstyle(SOLID_FILL,7);
setcolor(1);
poly[0].x=x-27;
poly[0].y=y+20;
poly[1].x=x-27;
poly[1].y=y+56;
poly[2].x= x+27;
poly[2].y= y+56;
poly[3].x= x+27;
poly[3].y= y+20;
poly[4].x= x-27;
poly[4].y= y+20;
fillpoly( 4, (int far *)poly );     /* Draw the actual polygon */
setfillstyle(1,1);
bar(x-15,y+6,x+15,y+10);
}
/*****************************************************************/
void flow1(int color)
{
setlinestyle(0,1,3);
setcolor(color);
line(100,288,100,117);
setfillstyle (1,color);
setcolor(4);
pump (100,180);
setlinestyle(0,1,3);
setfillstyle(1,color);
val(160,117,color);
setcolor(color);
line(100,117,143,117);
line(178,117,351,117);
line(350,117,350,175);
}
void flow2(int color)
{
setlinestyle(0,1,3);
setcolor(color);
line(180,288,180,146);
setfillstyle (1,color);
setcolor(4);
pump(180,180);
setlinestyle(0,1,3);
setcolor(color);
line(180,147,213,147);
val (230,147,color);
setcolor(color);
line(247,147,310,147);
line(310,146,310,175);
line(310,146,310,175);
}
void flow3(int color)
{
setlinestyle(0,1,3);
setcolor(color);
line(382,267,398,267);
setfillstyle (1,color);
setcolor(4);
val (415,267,color);
```

```
setlinestyle(0,1,3);
setcolor(color);
line (432,267,530,267);
setfillstyle (1,color);
setcolor(4);
pump(465,267);
setlinestyle(0,1,3);
setcolor(color);
line(530,267,530,280);
line(530,280,530,280);
}
void val (int x,int y,int color)
{
struct PTS {
      int  x, y;  };                    /* Structure to hold vert
struct PTS poly[ 3 ];
    setfillstyle(SOLID_FILL,color);
    setcolor(4);
    setlinestyle(0,1,3);
    poly[0].x=x;
    poly[0].y=y;
    poly[1].x=x-15;
    poly[1].y=y-7;
    poly[2].x= x-15;
    poly[2].y= y+7;
fillpoly( 3, (int far *)poly );     /* Draw the actual polygon */
    poly[0].x=x;
    poly[0].y=y;
    poly[1].x=x+15;
    poly[1].y=y-7;
    poly[2].x=x+15;
    poly[2].y= y+7;
fillpoly( 3, (int far *)poly );     /* Draw the actual polygon */
setlinestyle(0,1,3);
setcolor(4);
line(x,y,x,y-15);
line(x-5,y-15,x+5,y-15);
}
void pump (int x,int y)
{
setlinestyle(0,1,3);
setcolor(4);
fillellipse(x,y,15,15);
}
void showpic(void)
{
setcolor(14);
settextjustify(CENTER_TEXT,CENTER_TEXT );
outtextxy(xm/2,10,"PC & PLC APPLICATION PROCESS CONTROL");
setcolor(2);
smalltank(100,330);
setlinestyle(0,1,1);
setcolor(4);
line(98,288,98,195);
line(102,288,102,195);
setfillstyle(1,15);
setcolor(4);
outtextxy(60,180,"PUMP1");
pump (100,180);
setcolor(4);
line(98,165,98,115);
line(102,165,102,119);
line(98,115,145,115);
line(102,119,145,119);
setfillstyle(1,15);
```

```
val(160,117,15);
setcolor(4);
setlinestyle(0,1,1);
line(176,115,352,115);
line(176,119,348,119);
line(352,115,352,175);
line(348,119,348,175);
/*********************/
setcolor(2);
smalltank(180,330); *
setcolor(4);
setlinestyle(0,1,1);
line(178,288,178,195);
line(182,288,182,195);
setfillstyle(1,15);
setcolor(4);
outtextxy(220,180,"PUMP2");
pump (180,180);   *
setcolor(4);
line(178,165,178,145);
line(182,165,182,149);
line(178,145,215,145);
line(182,149,215,149);
setfillstyle(1,15);
val (230,147,15);  *
setcolor(4);
setlinestyle(0,1,1);
line(246,145,312,145);
line(246,149,308,149);
line(312,145,312,175);
line(308,149,308,175);
/*********************************/
bigtank (330,275);
spind(330,273);
/*********************************/
setcolor(4);
setlinestyle(0,1,1);
line(382,265,400,265);
line(382,269,400,269);
setfillstyle(1,15);
val (415,267,15);   *
setcolor(4);
setlinestyle(0,1,1);
line(430,265,450,265);
line(430,269,450,269);
setcolor(4);
outtextxy (465,240,"PUMP3");
pump (465,267);
setcolor(4);
line(480,265,532,265);
line(480,269,528,269);
line(532,265,532,280 );
line(528,269,528,280);
setcolor(2);
smalltank (530,330); *
}



void key (int x,int y,char ch[10],int  color          )
{
struct PTS {
int  x, y;  };          /* Structure to hold vertex points */
struct PTS poly[ 9 ];
setfillstyle(1,7);
```

```
bar(x-35,y-14,x+35,y+14);

setcolor(color);
setlinestyle(0,1,1);
setfillstyle(SOLID_FILL,color) ;
poly[0].x=x-30;
poly[0].y=y-10;
poly[1].x=x-31;
poly[1].y=y-9;
poly[2].x=x-31;
poly[2].y=y+9;
poly[3].x=x-30;
poly[3].y=y+10;
poly[4].x=x+30;
poly[4].y=y+10;
poly[5].x=x+31;
poly[5].y=y+9;
poly[6].x=x+31;
poly[6].y=y-9;
poly[7].x=x+30;
poly[7].y=y-10;
poly[8].x=x-30;
poly[8].y=y-10;
fillpoly( 8, (int far *)poly );     /* Draw the actual polygon */
setcolor(!11 );
settextstyle  (0,0,1);
settextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(x,y,ch);
}

void background(void)
{
setfillstyle(1,11);
bar (0,0,xm,ym);     /*background*/
setfillstyle(1,15);
bar (0,0,xm,20);     /*top bar*/
setfillstyle(1,15);
bar (0,ym-20,xm,ym);  /*bottom bar*/
}
void pannel (void)
{
int i;
setfillstyle(1,11);
bar (0,0,xm,ym);     /*background*/
setfillstyle(1,1);
bar (0,0,xm,20);      /*top bar*/
setfillstyle(1,1);
bar(0,ym-109,xm,ym-85);
setfillstyle(1,15);
setcolor(14);
outtextxy(50,ym-101,"CONTROL PANNEL");
setcolor(15);
line (0,ym-87,xm,ym-87);
setfillstyle(1,4);

bar(0,ym-86,xm,ym-65);
setcolor(14);
settextstyle  (0,0,1);
settextjustify(CENTER_TEXT,CENTER_TEXT);
for (i=0;i<3;i++)
outtextxy (50+i*80,ym-75,comp[i]);
setcolor(15);
line (0,ym-64,xm,ym-64);
setfillstyle(1,1);
bar(0,ym-63,xm,ym);
```

```
for (i=0;i<6;i++)
key (50+i*80.431,ckey[i],14);
}
int adc_data(void)
{
  int data;

  data = (inport(0x304)&0x0fff);

  return(data);
}

char *s_tr;
void adc_graph(void)
{
   static int round=0,olddat=0;
   int data,scale;
   double vadc;

   data = adc_data();
   scale = (float)data*100/4096;
   vadc =  (float)data*5/4096;
   gcvt(vadc,5,s_tr);

   if(round==200) {
   setfillstyle(SOLID_FILL,YELLOW);
   bar(400,50,600,200);
   round = 0;
                  }
   setcolor(RED);
   line(400+round,150-olddat,401+round,150-scale);
   olddat = scale;
   round++;

   setfillstyle(SOLID_FILL,YELLOW);
   bar(400,160,600,200);
   outtextxy(500,170,s_tr);

}
```