ปีการศึกษา 2532

เครื่องสแกนภาพ (IMAGE SCANNER)

โดย

นาย ทวีศักดิ์ นววิธรัตนา 291067
นาย ธีรพัฒน์ เอื้ออารักษ์ 291074
นาย ธีรภัทร เชื่อนทา 291077

อาจารย์ที่ปรึกษา
ดร. รัตติกร วรากูลศิริพันธุ์

ปริญญานิพนธ์ปีการศึกษา 2532

เรื่อง เครื่องสแกนภาพ (IMAGE SCANNER)

ผู้จัดทำ

1. นาย ทวีศักดิ์ นววิธรัตนา 291067
2. นาย ธีรพัฒน์ เอื้ออารักษ์ 291074
3. นาย ธีรภัทร เขื่อนทา 291077

อาจารย์ที่ปรึกษา

(ดร. รัตติกร วรากูลศิริพันธุ์)

IMAGE SCANNER

Taweesak Nawawitrattana 291067
Tirapat Ua-arak 291074
Teerapattara Kuanta 291077
Dr.Ratttikorn Warakulsiripan Advisor
1989

Abstract

The Image Scanner is an instrument for scanning pictuure and then converting it to digital signal which can be displayed or proceeded to any process. The instrument is very useful to which make use of the picture of the image processing , or the data base which picture is a important factor. Since the present , the image scanner is a very expensive instrument. So this project is determined to generate a low-price , convenient and efficient image scanner.

The instrument which go under the experimentary is very useful to takes about 36 minutes to read a picture of A4 paper. A resolution is about 200 dots per inch. The problem for this image scanner are a low resolution of scan head and a low speed of stepping motor used.

IMAGE SCANNER

Taweesak Nawáwitrattana 291067
Tirapat Ua-arak 291074
Teerapattara Kuanta 291077
Dr.Ratttikorn Warakulsiripan Advisor
1989

Abstract

The Image Scanner is an instrument for scanning pictuure and then converting it to digital signal which can be displayed or proceeded to any process. The instrument is very useful to which make use of the picture of the image processing , or the data base which picture is a important factor. Since the present , the image scanner is a very expensive instrument. So this project is determined to generate a low-price , convenient and efficient image scanner.

The instrument which go under the experimentary is very useful to takes about 36 minutes to read a picture of A4 paper. A resolution is about 200 dots per inch. The problem for this image scanner are a low resolution of scan head and a low speed of stepping motor used.

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

คอมพิวเตอร์ในปัจจุบันนับวันจะมีราคาถูกลง และมีขีดความสามารถสูงขึ้น จึงมีการนำมาใช้งานอย่างแพร่หลายในหลายด้าน โดยเฉพาะในยุคนี้ซึ่งเป็นยุคของ ข่าวสารข้อมูล (Information) ข้อมูลจะมีความสำคัญมาก ซึ่งคอมพิวเตอร์จะมีบทบาทในการเป็นตัวประมวลข้อมูล (Data Processing) ซึ่งแต่เดิมจะมีเพียงข้อมูลที่เป็นตัวเลขหรือตัวอักษร หรือที่เรียกว่า Text Data แต่ในปัจจุบันฐานข้อมูลบางอย่างต้องการข้อมูลที่มีภาพประกอบ มิฉะนั้นจะขาดความสมบูรณ์และไม่สะดวกในการใช้งาน เช่น ระบบทะเบียนประวัติของพนักงาน ถ้าไม่มีภาพของพนักงานก็จะไม่ทราบว่าเป็นพนักงานคนใด เครื่องที่ใช้ในการอ่านข้อมูลภาพ หรือ IMAGE SCANNER จึงมีความสำคัญขึ้นมา วิธีในการอ่านภาพมีหลายวิธี วิธีที่มีประสิทธิภาพสูงสุดในขณะนี้คือใช้ CCD (Charge Couple Device) เป็นตัวอ่าน ซึ่งสามารถให้ความละเอียดของภาพถึง 400 จุดต่อนิ้ว (400 DPI) และใช้เวลาในการอ่านภาพขนาดกระดาษ A4 (กว้าง 8.5 นิ้ว ยาว 10.5 นิ้ว) เพียง 1 นาที เท่านั้น แต่อุปกรณ์ชนิดนี้มีราคาสูงและมีความซับซ้อนของระบบมาก อีกวิธีหนึ่งจะใช้โฟโต้ทรานซิสเตอร์ และโฟโต้ไดโอดเป็นตัวอ่านภาพ วิธีนี้มีราคาไม่แพงแต่ความละเอียดของภาพต่ำ และเวลาที่ใช้ในการสแกนต่อภาพจะสูง เพราะต้องอ่านภาพทีละจุดไม่ได้อ่านทีเดียวทั้งแถว (line) เหมือนกับแบบแรก

เครื่องสแกนภาพที่ใช้วิธีหลัง สามารถใช้เครื่องพิมพ์ (printer) เป็นตัวขับหัวอ่านได้ แต่มีข้อเสียที่หลายประการที่จะกล่าวต่อไป ในโครงงานนี้จึงได้สร้างแท่นสแกนขึ้นมาใหม่ ซึ่งสามารถแก้ไขข้อบกพร่องข้างต้นได้

บทที่ 2

ทฤษฎีการทำงานของวงจร

2.1 บล๊อกไดอะแกรม

ประกอบด้วยส่วนที่ทำหน้าที่ควบคุมการหมุนของมอเตอร์ และส่วนที่ทำหน้าที่อ่านข้อมูลจากภาพที่แปลงเป็นสัญญาณดิจิตอลมาเก็บไว้ที่ RAM ซึ่ง 2 ส่วนนี้จะทำงานสัมพันธ์กัน คือ เมื่อมอเตอร์เลื่อนหัวสแกนไป 1 จุด ก็จะมีการเก็บข้อมูลเข้า RAM จนกระทั่งได้ข้อมูลของภาพครบ 1 ภาพ

ส่วนที่ทำหน้าที่ควบคุมมอเตอร์ประกอบด้วย บล๊อก INTERFACE , CONTROL CCT , SEQENCER และ MOTOR DRIVER

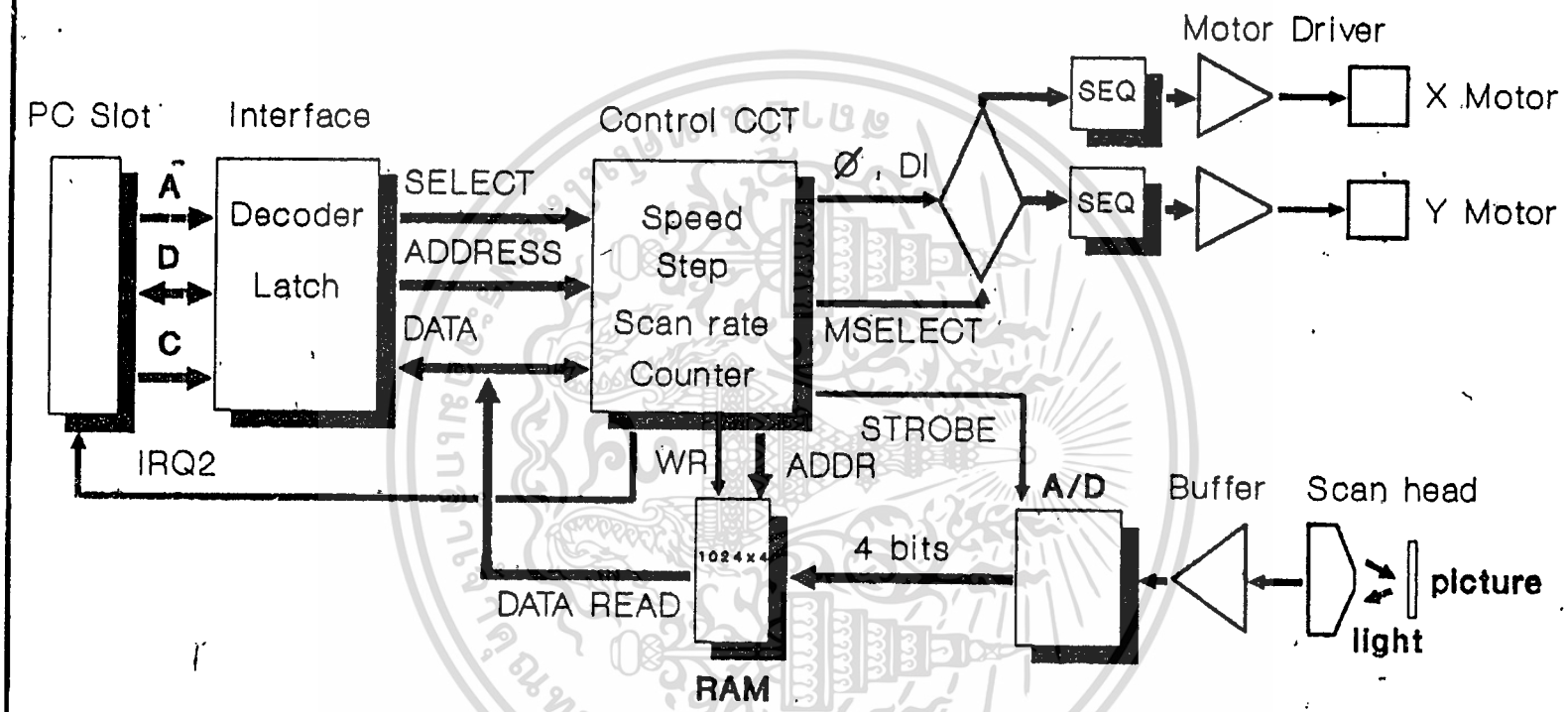
บล๊อก INTERFACE ประกอบด้วยส่วนที่ทำหน้าที่ถอดรหัส (ส่วน DECODER) โดยจะถอดรหัสแอดแดรส 10 เส้น ออกเป็นพอร์ต 8 พอร์ต และส่วนที่คงค่าสัญญาณควบคุมที่ส่งมาเพื่อใช้ในการควบคุมการสแกน (ส่วน LATCH) สัญญาณเหล่านี้จะส่งมาที่ CONTROL CIRCUIT ซึ่งส่วนใหญ่จะประกอบด้วยวงจรนับ โดยวงจรนับตัวแรก (ใช้ควบคุม SPEED หรือความเร็วของมอเตอร์) จะเป็นวงจรหารความถี่ออกเป็นสัญญาณ 0 ที่จะส่งให้วงจรนับตัวที่สอง (ควบคุม STEP) ที่ทำหน้าที่นับจำนวนพัลส์ ซึ่งจะหมายถึงจำนวนสเตปที่เราต้องการให้มอเตอร์เคลื่อน สัญญาณเดียวกันนี้จะส่งไปยังวงจรนับตัวที่สาม (SCAN RATE) ซึ่งเป็นตัวกำหนดความละเอียดในการสแกนภาพ

การอ่านภาพใช้หัวสแกนที่เป็นอุปกรณ์ประเภท ออพโตคัปเปเลอร์ ซึ่งประกอบด้วยโฟโต้ไดโอดทำหน้าที่ส่งแสงอินฟราเรดไปยังภาพ และโฟโต้ทรานซิสเตอร์เป็นตัวรับแสงที่สะท้อนจากภาพ ซึ่งปริมาณแสงที่สะท้อนจากภาพจะขึ้นกับ

1. พื้นผิวของกระดาษ ถ้ากระดาษขรุขระมากปริมาณแสงที่สะท้อนจะน้อย เปรียบเทียบกับกระดาษที่มีผิวเรียบ

2. ระยะระหว่างภาพกับออพโตคัปเปเลอร์ ปริมาณแสงจะสูงสุดเมื่อภาพอยู่ห่างจากออพโตคัปเปเลอร์เท่ากับระยะโฟกัส

3. สีของภาพ พื้นผิวสีขาวจะสะท้อนแสงได้มากกว่าพื้นผิวสีดำ



Scanner block diagram

รูปที่ 2.1 บล็อคไดอะแกรมของเครื่องสแกน

2.2 การอินเตอร์เฟสกับ IBM PC

ในการที่จะใช้ประโยชน์จากเครื่องคอมพิวเตอร์ให้ควบคุมอุปกรณ์ภายนอกจำเป็นต้องมีอุปกรณ์ชนิดหนึ่งที่ช่วยติดต่อระหว่างเครื่องคอมพิวเตอร์กับอุปกรณ์อินพุท/เอาท์พุท เรียกว่าวงจรอินเตอร์เฟส (Interfacing Circuit) ซึ่งคอมพิวเตอร์แต่ละแบบจะวิธีการที่คล้ายกันดังนั้นจะกล่าวเพียงเครื่อง IBM PC เท่านั้น

สัญญาณที่เกี่ยวข้องต่าง ๆ ที่สำคัญมีดังนี้

AEN Address Enable ใช้บอกสถานะของแอดเดรสที่ซีพียูส่งมา โดยถ้ามีค่าเป็น "0" จะหมายถึงการทำ DMA (คำอธิบายสามารถหาได้จากหนังสืออ้างอิงหมายเลข 2) ถ้ามีค่าเป็น "1" จะหมายถึงแอดเดรสที่ไม่ใช่การทำ DMA ซึ่งจะเป็นช่วงของการ อ่าน/เขียน กับหน่วยความจำหรือ อุปกรณ์อินพุท/เอาท์พุท

A0-A15 สัญญาณ A10-A15 จะไม่ใช้ในการติดต่อกับ อุปกรณ์อินพุท/เอาท์พุท ดังนั้นจำนวน อุปกรณ์อินพุท/เอาท์พุทที่สามารถมีได้สูงสุดจะถูกกำหนดโดย A0-A9 คือ 2^{10} = 1024 พอร์ท มีข้อน่าสังเกตุคือพอร์ทที่สำหรับส่งค่าออก (เอาท์พุทพอร์ต) จะเป็นเบอร์ 100H - 3FFH เท่านั้น ส่วนพอร์ตสำหรับรับค่าเข้า (อินพุทพอร์ต) จะเป็นได้ทุกเบอร์คือตั้งแต่ 000H - 3FFH

$\overline{IOR}$, $\overline{IOW}$ เป็นสัญญาณที่จะบอกให้รู้ว่ามีการอ่านหรือเขียนข้อมูลกับ อุปกรณ์อินพุท/เอาท์พุท ตามลำดับจะใช้เป็นสัญญาณสโตรบ (strobe) ให้รับหรือส่งข้อมูล

D0-D7 เป็นขาข้อมูลที่ปรากฏที่บัสข้อมูล (Data bus)

เมื่อต้องการให้คอมพิวเตอร์ส่งค่าใด ๆ ไปยังพอร์ตที่ต้องการคำสั่งที่ง่ายที่สุดคือ

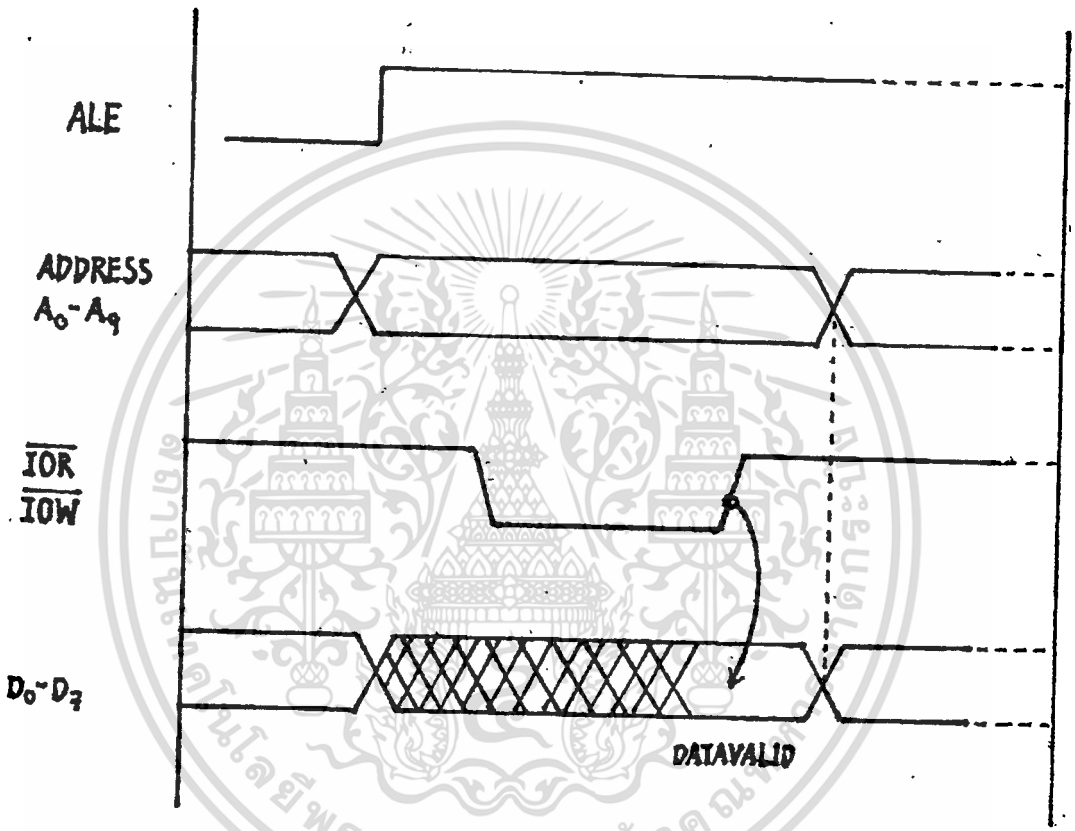
OUT พอร์ตที่ต้องการ,ค่าที่ต้องการส่ง

เมื่อคอมพิวเตอร์ได้รับคำสั่งนี้จะอธิบายขั้นตอนได้โดยแผนภูมิเวลา (Timing diagram) ดังภาพที่ 2.2 ซึ่งจุดที่อุปกรณ์อินพุท/เอาท์พุท ควรจะรับข้อมูลคือขณะที่สัญญาณ IOW เปลี่ยนเป็นสัญญาณของขาขึ้น เพราะในขณะนั้นข้อมูลจะถูกส่งออกมาอย่างแน่นอนแล้ว

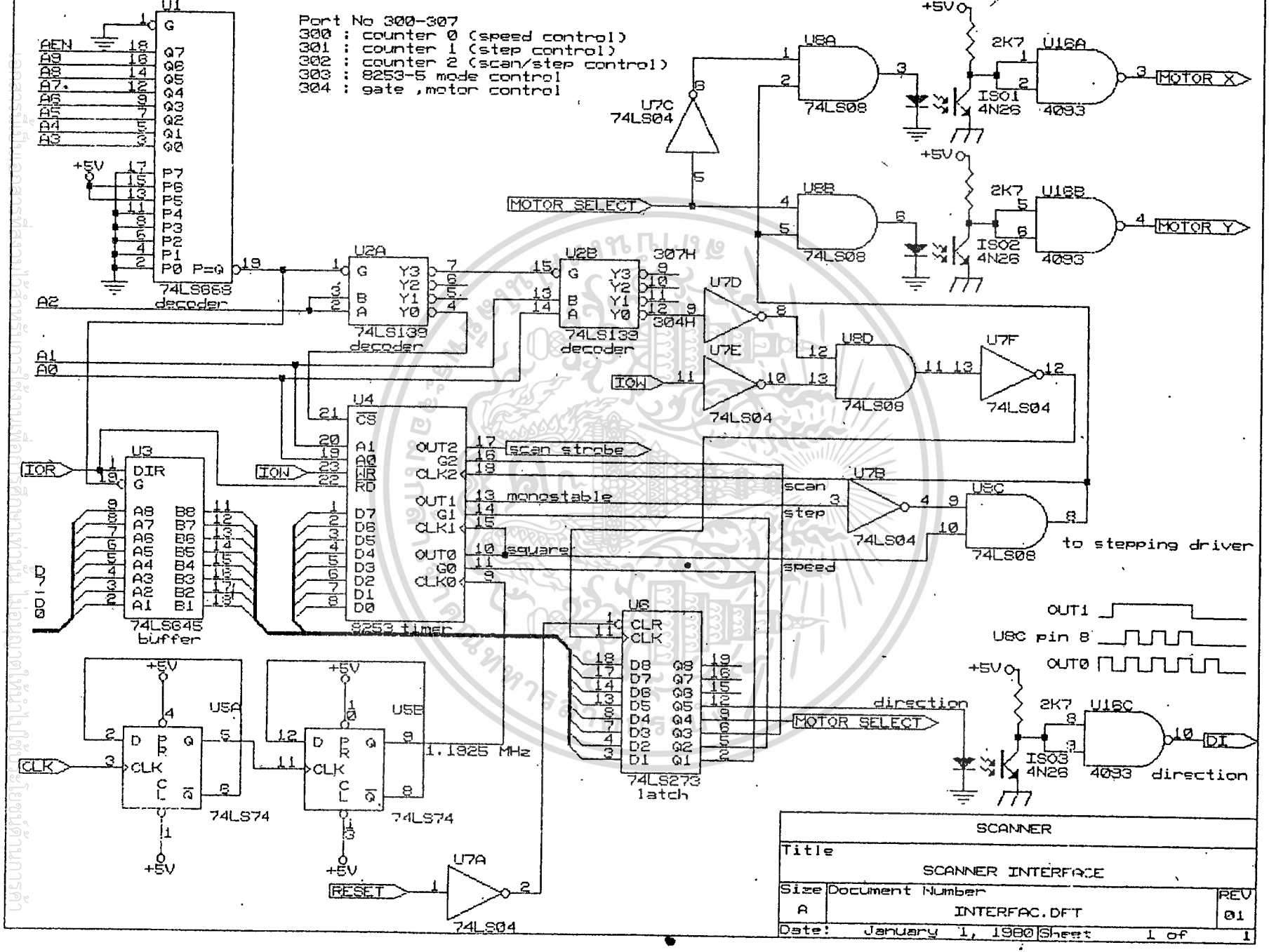
ในทำนองเดียวกัน เมื่อต้องการให้คอมพิวตอร์รับค่าจากอุปกรณ์อินพุท/เอาท์พุท จะใช้คำสั่ง

IN รีจิสเตอร์ที่รับค่า,พอร์ตที่ต้องการ

อุปกรณ์อินพุท/เอาท์พุท ต้องให้ข้อมูลที่ถูกต้อง (Data valid) แก่บัสข้อมูลก่อนที่สัญญาณ $\overline{IOR}$ จะเปลี่ยนจากลอจิกต่ำไปเป็นลอจิกสูง



รูปที่ 2.1 แผนภูมิเวลาแสดงการรับ-ส่งข้อมูล
ของเครื่องคอมพิวเตอร์



SCANNER		
Title	SCANNER INTERFACE	
Size A	Document Number INTERFAC.DFT	REV 01
Date: January 1, 1980	Sheet 1 of 1	

2.3 วงจรส่วนอินเตอร์เฟส

เป็นวงจรที่เชื่อมต่อระหว่างอุปกรณ์อินพุท/เอาท์พุท กับไมโครคอมพิวเตอร์ และเนื่องจากการติดต่อกับ อุปกรณ์อินพุท/เอาท์พุท ของไมโครคอมพิวเตอร์นี้ ใช้ แอดเดรส เพียง 10 เส้น โดยบิท A9 ต้องเป็น 1 เท่านั้น เมื่อต้องการให้เป็น เอาท์พุท พอร์ต (แต่ อินพุท พอร์ต จะไม่ขึ้นกับบิท A9 นี้) ดังนั้นจึงต้องเลือก พอร์ต เบอร์ 300H-307H เพื่อความสะดวกทั้งในการออกแบบวงจรและการเขียนโปรแกรม ซึ่งวงจรนี้ประกอบด้วย U1,U2,U3 โดย U1 (74LS688) จะเป็นตัวกำหนด เบอร์ พอร์ต 2 หลักแรกคือ30 (0110 0000) ส่วนU2,U3จะเป็นตัวกำหนดเบอร์ พอร์ต หลักสุดท้ายคือ 0 ถึง 7 (แต่ในที่นี้ใช้เพียง 0 ถึง 4)

โดยที่ เอาท์พุท (ขา 19) จะเป็น ลอจิก ต่ำ เมื่อขา P_i เหมือนกับ Q_i ทุกขา (i = 0-7) ดังนั้นวงจรในภาพที่ 1 ขา 19 ของ U1 จะเป็น "0" เมื่อ Pi และ Qi เป็นดังตาราง

P_i	0	1	1	0	0	0	0	0
Q_i	AEN	A9	A8	A7	A6	A5	A4	A3

ตารางที่ 2.1 แสดงค่าของ P_i และ Q_i

การ ถอดรหัส แอดเดรส ที่เหลือ (A2-A0) จะใช้ U2 (74LS139) โดย ขา A และ B ของ U2A จะต่ออยู่กับ A2 ซึ่งหมายความว่า เมื่อ A2 เป็น "0" Y0 จะถูกเลือก และเมื่อ A2 เป็น "1" Y3 จะถูกเลือกแทน ซึ่ง Y3 จะไป อินาเบิ้ล U2B ให้ถอด แอดเดรส A1 และ A0 ออกเป็น 4 พอร์ต อีกที โดยได้กำหนดพอร์ทไว้ดังนี้

PORT No.	FUNCTION
300H	วงจรนับเบอร์ 0(SPEED CONTROL)
301H	วงจรนับเบอร์ 1(STEP CONTROL)
302H	วงจรนับเบอร์ 2(SCANRATE CONTROL)
303H	8253-5 MODE CONTROL
304H	MOTOR SELECT, DIRECTION, GATE
305-307H	UNUSED

ตารางที่ 2.2 แสดงการกำหนดพอร์ทต่างๆ

ข้อมูลจากไมโครคอมพิวเตอร์จะต่อกับ U3 ซึ่งเป็น bus tranciver เมื่อถูก อินาเบิ้ล (ขา 19,G เป็นลอจิกต่ำ) ข้อมูลจะผ่านจาก A ไปยัง B หรือ B ไปยัง A โดยขึ้นกับขา DIR ที่ถูกต่อกับขา IOR ซึ่งมีการทำงานเป็นดังนี้

OPERATION	IOR(DIR)	DATA
INPORT	0	B->A
OUTPORT	1	A->B

ตารางที่ 2.3 แสดงการส่งผ่านข้อมูลของ U3

2.4 <u>วงจรส่วนควบคุม</u>

วงจรส่วนนี้ประกอบด้วย U4,U5,U6,U7 และ U8 โดยจะกล่าวถึง U6 ซึ่งเป็น Latch แบบ Positive-edge triggered ก่อน โดยมีแผนภูมิเวลา แสดงถึงการ Latch ข้อมูล ดังรูป 2.4

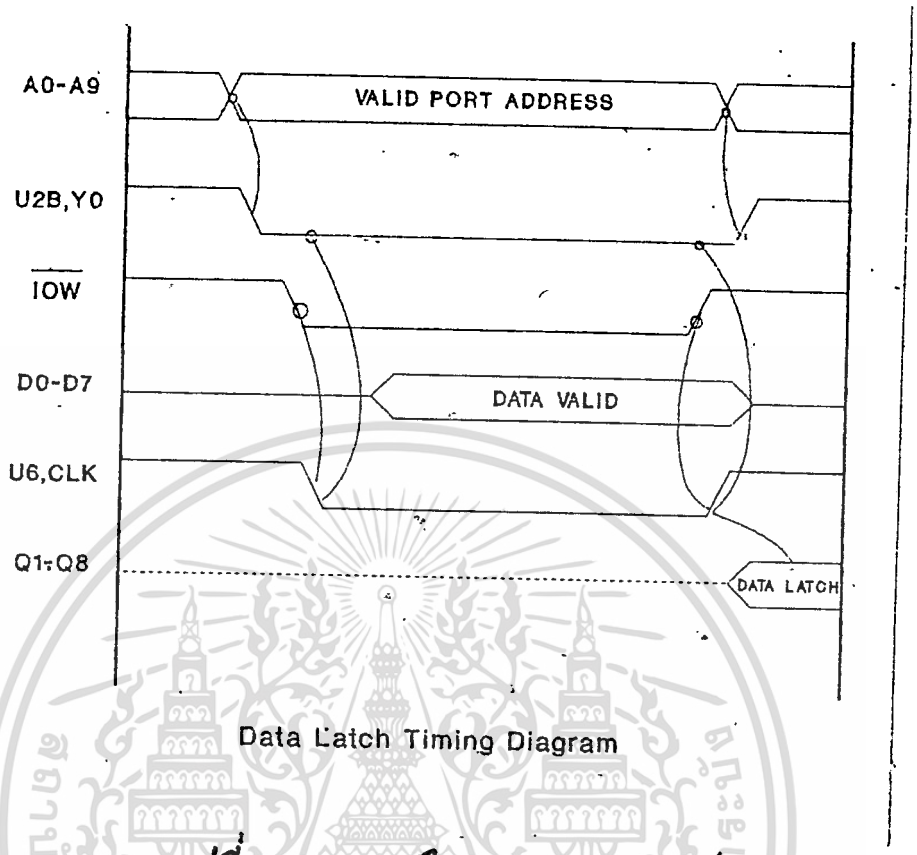
เมื่อมีคำสั่ง OUT มาที่ พอร์ต นี้ (304H) ขา Y0 ของ U2B จะเป็น 0 และเมื่อ IOW เป็น 0 ในเวลาต่อมา จะทำให้ ขา 12 ของ U7F (หรือขา CLK ของ U6) เป็น 1 ซึ่งทำให้เกิดสัญญาณขอบขาขึ้นไปแลทช์ข้อมูลจากบัฟเฟอร์ (U3) ไว้ ซึ่งความ

หมายของแต่ละบิทที่ Latch ไว้ มีดังนี้

Q1	ต่อไปยัง G0 ซึ่งจะใช้ควบคุม วงจรนับเบอร์ 0
Q2	"---" G1 "--------" วงจรนับเบอร์ 1
Q3	"---" G2 "--------" วงจรนับเบอร์ 2
Q4	MOTOR SELECT ใช้เลือกมอเตอร์ โดยถ้าเป็น 0 จะเลือกมอเตอร์ X ถ้าเป็น 1 จะเลือก มอเตอร์ Y
Q5	DIRECTION ใช้บอกทิศทางการหมุนของมอเตอร์ ซึ่งได้กำหนดไว้ว่า 0 หมายถึงการหมุนในทิศ CW , 1 หมายถึงการหมุนในทิศ CCW
Q6-Q8	UNUSED

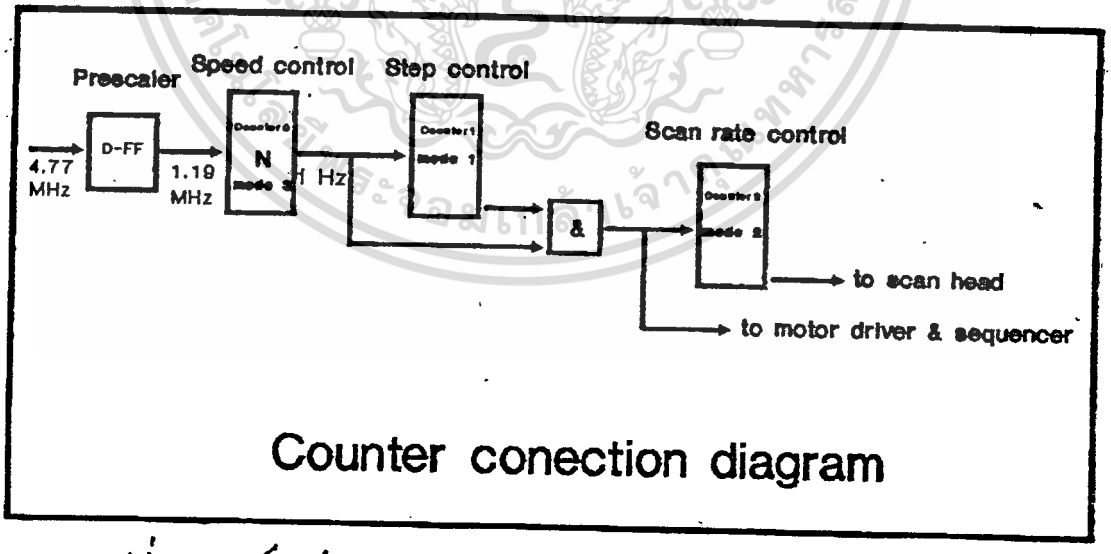
ตารางที่ 2.4 แสดงความหมายของ Q1-Q8 ที่ได้แลทช์ไว้

การทำงานส่วนใหญ่ของส่วนควบคุมนี้จะเกี่ยวกับวงจรนับ จึงใช้ 8253-5 PROGRAMABLE INTERVAL TIMER ซึ่งมีวงจรนับ 16 บิท 3ตัว ที่สามารถโปรแกรมให้ทำงานแต่ละตัวได้อย่างอิสระได้ถึง 6 โหมด และเนื่องจาก 8253-5 สามารถใช้ที่ความถี่สูงสุดเพียง 2.6 MHz แต่สัญญาณ CLK ที่ออกมาจาก PC SLOT มีความถี่สูงสุดถึง 4.77 MHz จึงต้องใช้ D - FLIP FLOP 2 ตัว ต่อเป็นแบบ toggle สองภาค เป็นวงจรหาร 4 ซึ่งจะได้ความถี่ออกมา 1.1925 MHz นำไปต่อกับขา CLK0 เพื่อเป็นความถี่มูลฐานของส่วนวงจรนับ โดย วงจรนับเบอร์ 0 จะถูกโปรแกรมให้เป็น Square Wave Generator (โหมด 3) ให้ความถี่ออกมาที่ขา OUT0 ตามต้องการ (ความถี่สูงสุดที่ออกมาคือ 1.1925 MHz/ 216 = 18 Hz) และขา OUT0 นี้จะถูกต่อไปยังขา CLK1 ซึ่ง วงจรนับเบอร์ 1 ถูกโปรแกรมให้เป็นโมโนสเตเบิ้ล (โหมด1) โดยที่ความยาว(duration) ของ โมโนสเตเบิ้ล นี้สามารถโปรแกรมได้เช่นกัน ดังนั้นจะเสมือนเป็นการ "นับ" จำนวน พัลส์ ที่ส่งมาจาก วงจรนับเบอร์ 0 โดยจะเริ่ม "นับ" ที่ขอบขาขึ้นที่ขา GATE(G1) ซึ่งเอาท์พุทของโมโนสเตเบิ้ล (OUT1) จะนำไป AND กับสัญญาณ OUT0 (หรือ CLK 1) จะได้เป็น พัลส์เทรนส์ ที่มีจำนวนพัลส์ และความถี่ตามต้องการดังภาพ 2.6



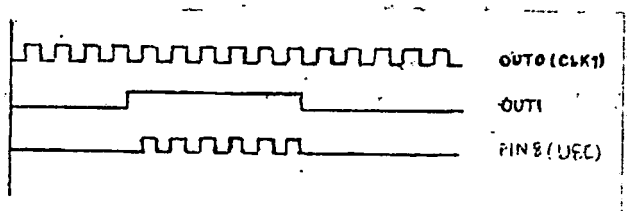
Data Latch Timing Diagram

รูปที่ 2.4 แผนภูมิเวลาแสดงการแลทช์ข้อมูล.



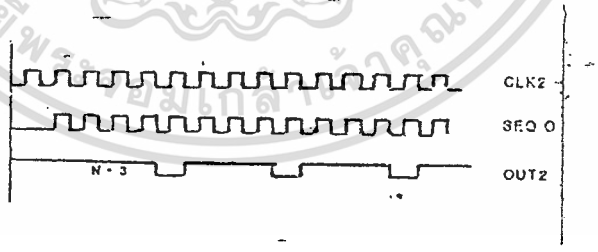
Counter conection diagram

รูปที่ 2.5 บล็อคไดอะแกรมแสดงการต่อวงจรนับของ 8253.



รูปที่ 2.6 แสดงผังภูมิเวลาของเอาท์พุทของ U8C

สัญญาณจาก AND Gate U8Cจะต่อไปยังวงจรวงจรเลือก โดยเมื่อสัญญาน MOTOR SELECT เป็น "0" จะผ่านสัญญาณนี้ไปยัง วงจร ซีเควนเซอร์ ของมอเตอร์ X (รายละเอียดของวงจร ซีเควนเซอร์ จะกล่าวต่อไปภายหลัง) ขณะที่สถานะของ ซีเควนเซอร์ ของ มอเตอร์ Y ไม่มีการเปลี่ยนแปลง และเมื่อ MOTOR SELECT เป็น "1 สัญญานดังกล่าวจะส่งไปให้ ชุด Y แทน อัตราการสแกนต่อการเคลื่อนที่ของมอเตอร์ 1 สเตป จะบอกถึงความ . ละเอียดของภาพที่สแกนได้ สามารถควบคุมโดยการโปรแกรม counter 2 ให้อยู่ใน โหมด 2 (RATE GENERATOR) ซึ่งรับสัญญาณนาฬิกามาจากสัญญาณทริกของวงจรซีเควนเซอร์ โดยมีตารางเวลาดังภาพล่าง จะเห็นว่ามอเตอร์มีการเคลื่อนที่ไปก่อนแล้วจึงมีการเก็บสัญญาณในการสแกน ซึ่งจะมีผลดีคือช่วยให้ขอบของภาพคมขึ้นเพราะมอเตอร์เริ่มมี Torque คงที่ (dynamic torque) เนื่องจากได้ Free running มาช่วงหนึ่ง แล้วจึงค่อยสแกนภาพ

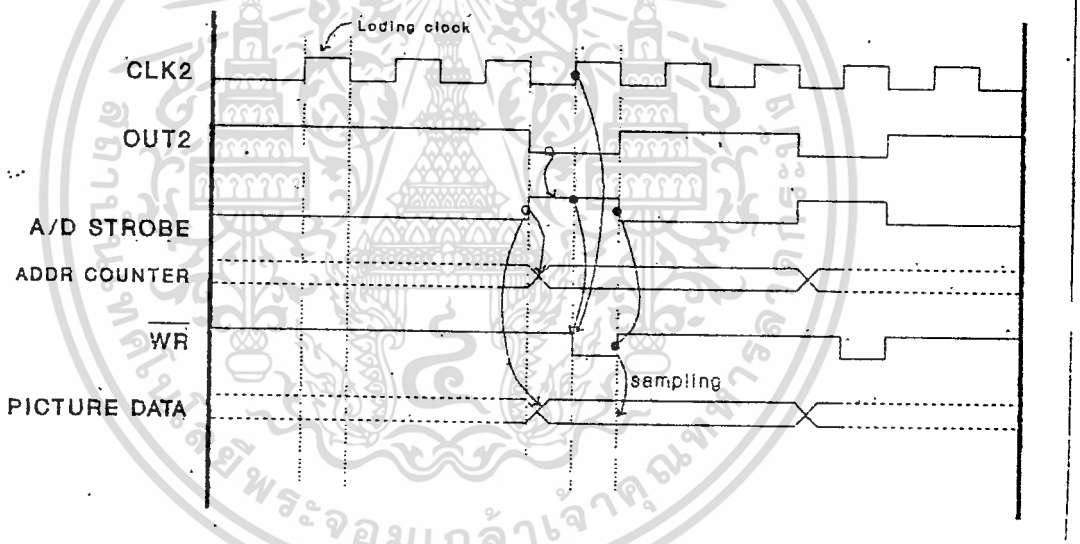


รูปที่ 2.7 แสดงผังภูมิเวลาของ counter 2

วงจรส่วนการเขียนและการแปลงข้อมูลลงสู่ RAM ได้ออกแบบ แผนภูมิเวลา ดังภาพ 2.8 ซึ่งอธิบายได้ดังนี้คือ เมื่อมอเตอร์เริ่มเคลื่อนไปเท่ากับ n-1 step ขา OUT2 จะตกลงเป็นลอจิก 0 ซึ่งขอบขาลงนี้จะไปทริกให้ Address Counter เพิ่มขึ้นอีก 1 และยังไป Strobe A/D ให้ convert ข้อมูลภาพอนาลอกออกเป็นดิจิตอล 4 บิท ออกมาสู่ DATA BUS ขณะที่ OUT2 ยังเป็นลอจิก 0 อยู่นั้นเมื่อสัญญาณนาฬิกาของวงจรนับเบอร์ 2

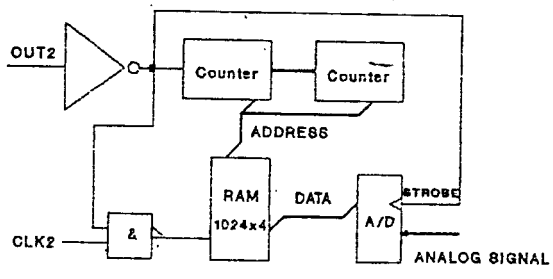
(CLK 2) เกิดขอบขาขึ้น จะไปกระตุ้นให้ WR เป็นลอจิก 0 และ ต่อมากลับเป็นลอจิก 1 เมื่อ CLK กับ OUT 2 กลายเป็น 1 ทำให้เกิดการเขียนข้อมูลสู่ RAM ขึ้นที่ขอบสัญญาณนี้ และเมื่อ OUT2 กลับเป็น 1 การทำงานข้างต้นจะเริ่มขึ้นอีกจนกระทั่งสิ้นสุดการสแกนใน 1 line ซึ่ง ซอร์ฟแวร์ จะทำหน้าที่อ่านข้อมูลจาก RAM มาเก็บไว้ในเครื่องไมโครคอมพิวเตอร์ และจะทำการ Clear Address Counter เพื่อใช้เป็นที่เก็บข้อมูลภาพชั่วคราวของ line ต่อไป

ขนาดความจุของ RAM นี้จะขึ้นกับความละเอียดของภาพใน 1 line เช่นที่ความละเอียด 100 จุดต่อหนึ่งนิ้ว หรือ 8.5 x 100 = 850 จุดต่อ 1 line ดังนั้นจะใช้ RAM ขนาด 1K x 4 bit เป็น Buffer ซึ่งสามารถให้ความละเอียดใน 1 line ได้ 1024 จุด หรือ 1024/8.5 = 120 จุดต่อนิ้ว และถ้าต้องการเพิ่มความละเอียดขึ้นอีก ก็สามารถทำได้ง่าย โดย Cascade Counter และ Cascade RAM ให้มีความจุตามต้องการ



Scan Strobe Timing Diagram

รูปที่ 2.8 แผนภูมิเวลาแสดงการเขียนและแปลงข้อมูลลง RAM



Counter & Buffer Diagram

รูปที่ 2.9 บล็อคไดอะแกรมของวงจรนับและบัฟเฟอร์

2.5 วงจรขับมอเตอร์

วงจรขับแต่ละเฟสจะประกอบด้วยทรานซิสเตอร์ 2 ตัว ทำงานในช่วงไม่นำกระแส (cut off) กับช่วงอิ่มตัว (saturated)

Q1 2N2222 $\beta_1 = 100$

Q2 TIP31 $\beta_2 = 20$

$i_{c2} = \beta_2 i_{b2}$

i_{c2} คือกระแสที่มอเตอร์ต้องการมีค่าประมาณ 1 Amp

$i_{b2} = 1/20 = 50$ mA

$i_{b2} = i_{c1} = \beta_1 i_{b1}$

$i_{b1} = i_{c1} / \beta_1 = 50/100 = 0.5$ mA

i_{b1} คือกระแสสูงสุดที่ได้จากเอาท์พุตของวงจงซีเควนเซอร์

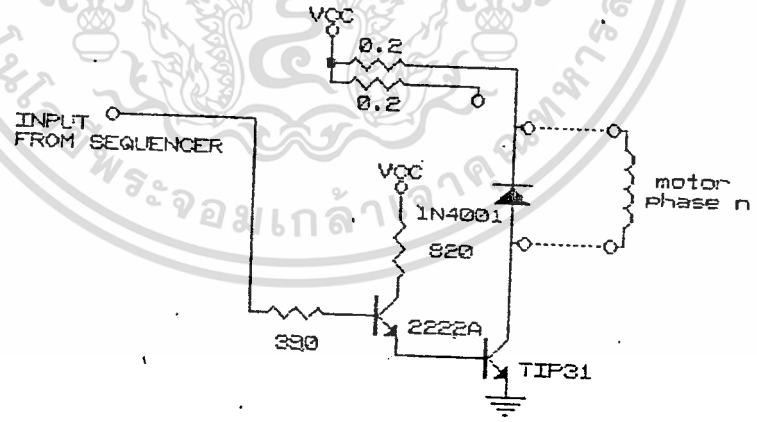
$V_{OH} = i_{b1} R1 + V_{BE1} + V_{BE2}$

V_{OH} คือโวลเตจสูงสุดที่ได้จากวงจรซีเควนเซอร์มีค่าประมาณ 3.4 โวลต์

$3.4 = 0.5\ R1 + 0.6 + 0.6$

$R1 = 440$ โอห์ม

ใช้ค่า 390 โอห์ม



รูปที่ 2.10 วงจรขับมอเตอร์

2.6 วงจรจ่ายไฟเลี้ยง (POWER SUPPLY)

เป็นวงจรเรคกูเลเตอร์แบบขนาน โดยใช้ซีเนอร์ไดโอดเป็นตัวรักษาแรงดันเอาต์พุต ให้คงที่และมีทรานซิสเตอร์ช่วยขยายกระแส โดยจะจ่ายแรงดันได้ 6 และ 4 โวลต์ ให้กับสเตปปิงมอเตอร์ 2 ตัว ที่ใช้ในการขับหัวสแกน

การคำนวณค่า R

I_c = 1 Amp

$I_b = I_c / \beta = 1/20 = 50$ mA

β คืออัตราขยายกระแสของทรานซิสเตอร์ 2N3055 มีค่าประมาณ 20

$I_{R1} = I_b + I_z = 55$ mA

Iz คือ กระแสที่ไหลผ่านซีเนอร์ไดโอดเพื่อให้ซีเนอร์ไดโอดทำงานในช่วง breakdown มีค่าประมาณ 5 mA

$V_{R1} = (2 V_{AC} - V_D) - V_Z$

V_{R1} คือ คักดาคร่อม R1

V_{AC} คือ คักดาขด secondary ของหม้อแปลง

V_D คือ คักดาคร่อมไดโอดเร็คทิฟายด์

V_Z คือ คักดาในช่วงเบรคดาวน์ของซีเนอร์ไดโอด

$V_{R1} = (9\ 2 - 0.7) - 6.8 = 5.2$ โวลต์

$R1 = V_{R1} / I_{R1} = 5.2 / 52 = 95$ โอห์ม

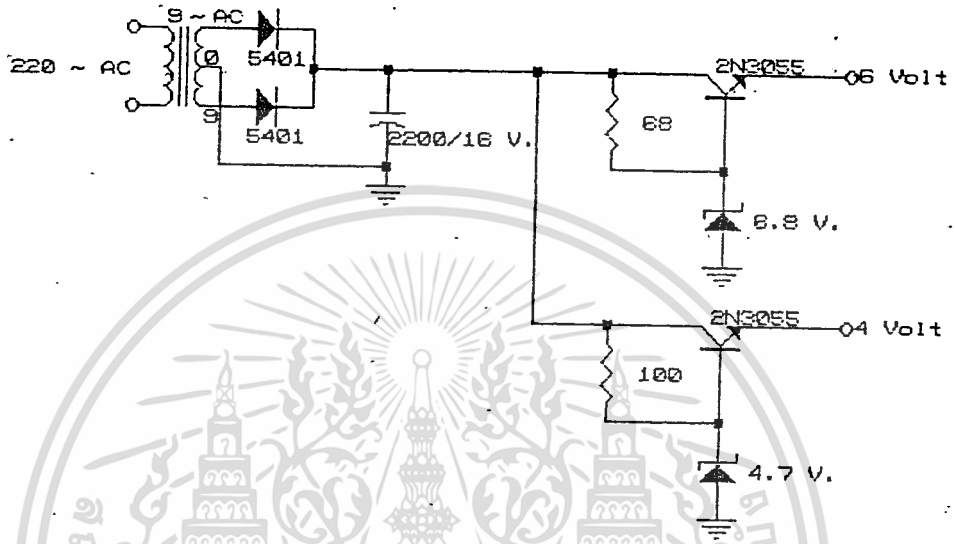
ในการใช้งานจะปัดค่าลง โดยจะใช้ค่า R1 = 68 โอห์ม

$V_{R2} = (9\ 2 - 0.7) - 4.7 = 7.3$ โวลต์

V_{R2} คือ คักดาคร่อม R2

$R2 = V_{R2} / I_{R2} = 7.3 / 55 = 132$ โอห์ม

ค่าที่ใช้งานคือ 100 โอห์ม



รูปที่ 2.11 วงจรจ่ายไฟเลี้ยงให้มอเตอร์

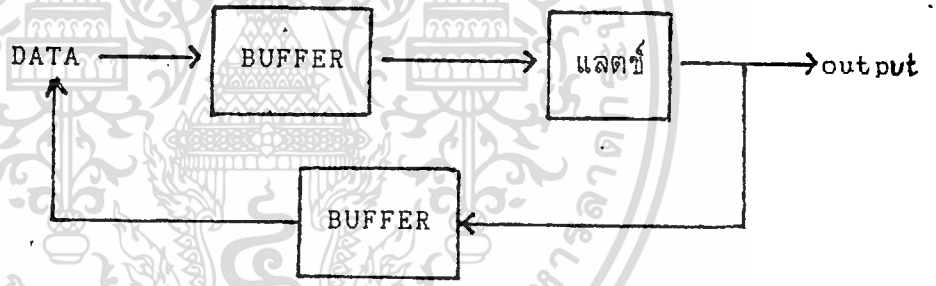
บทที่ 3

ซอฟท์แวร์ (SOFTWARE)

ในระบบคอมพิวเตอร์ สิ่งที่สำคัญอย่างหนึ่งก็คือ ซอฟท์แวร์ ซึ่งจะมีส่วนช่วยให้การทำงานเป็นไปอย่างสมบูรณ์ ง่ายต่อการใช้งาน, การเปลี่ยนแปลงแก้ไข และยังช่วยลดส่วนของฮาร์ดแวร์ได้อีกด้วย โดยจะเห็นได้จากการใช้ตัวแปร "bit" เก็บค่าที่ส่งไปยังพอร์ต 304H ซึ่งเป็นสถานะของสแกนเนอร์ แทนการใช้ การอ่านค่าผ่าน บัฟเฟอร์ ซึ่งอธิบายได้ดังรูป

รูปที่ 3.1 แสดงการย้ายข้อมูล

ก. แบบ open-loop การอ่านสถานะใช้ค่าตัวแปรที่เก็บไว้ได้เลย



ข. แบบ close-loop การอ่านสถานะจะอ่านจากสถานะจริงผ่านบัฟเฟอร์อีกตัว โดยใช้คำสั่ง Inport

ส่วนของโปรแกรมทั้งหมด จะเขียนด้วยภาษา C เนื่องจากภาษา C มีความเร็วสูงและมีลักษณะเป็นภาษาโครงสร้าง(Structural Programming Language)ทำให้สามารถใช้กับงานที่มีความซับซ้อนได้ อีกทั้งมีฟังก์ชั่นที่เกี่ยวกับการแสดงผลในแบบกราฟฟิกส์อยู่แล้ว

3.1 <u>โครงสร้างการทำงานของโปรแกรม</u> แบ่งออกเป็น 5 ไฟล์หลัก ดังนี้

- SCANNER.H เป็นไฟล์ที่เก็บค่าตัวแปร และการกำหนดค่าต่างๆของระบบไว้รวมกัน
- SCANNER.C เป็นไฟล์หลักซึ่งจะควบคุมการทำงานหลักของโปรแกรม
- MENUBAR.C เป็นไฟล์ที่เกี่ยวกับการแสดงผลของระบบเมนู ที่เป็นเมนูในลักษณะพูลดาวน์เมนู(pull down menu) ซึ่งสะดวกในการทำความเข้าใจ และการใช้งาน

- FUNCTION.C เป็นไฟล์ที่รับการทำงานต่อมาจาก MENUBAR.C การทำงานของการสแกนภาพจริงๆจะอยู่ในไฟล์นี้
- ROUTINE.C เป็นไฟล์ที่เก็บโปรแกรมย่อยต่างๆ ซึ่งจะถูกเรียกใช้จากไฟล์ FUNCTION.C

3.2 <u>หน้าที่การทำงานของโปรแกรมย่อย</u> ที่สำคัญ มีดังนี้

`init()` ใช้เตรียมสถานะต่างๆของระบบ

`barmenu()` เป็นโปรแกรมหลักในการควบคุมเมนู

`file()` เป็นโปรแกรมย่อยที่จัดการเกี่ยวกับแฟ้มข้อมูล เช่น การโหลดข้อมูล เก็บข้อมูล เป็นต้น

`scan()` เป็นโปรแกรมย่อยที่เกี่ยวกับการสแกนภาพ ซึ่งประกอบด้วย
`autoscan()` ใช้ควบคุมการสแกนตั้งแต่ต้นจนจบ
`setdelay()` ใช้ตั้งอัตราการหน่วงเวลาในการสแกนภาพ

`display()` ใช้ควบคุมส่วนของการแสดงภาพที่ได้โหลดหรือสแกนไว้ในหน่วยความจำแล้วและรวมไปถึงการพิมพ์ด้วย

`print()` ใช้ควบคุมการพิมพ์รูปในหน่วยความจำให้ออกไปยังเครื่องพิมพ์

`position()` ใช้ควบคุมตำแหน่งของหัวสแกน

`edgeset()` ใช้กำหนดขอบของการสแกนภาพ

`parameter()` ใช้กำหนดตัวแปรของระบบ

`autoscan()` มีโฟลวชาร์ตดังรูปในหน้า 55 มีตัวแปรและฟังก์ชั่นที่สำคัญคือ `msecx,msecy` คือเวลาที่ใช้ในการสแกนในแกน x และแกน y ตามลำดับ ซึ่งหาได้จาก จำนวนสเต็ปคูณกับเวลาที่ใช้ในการเคลื่อนที่ในหนึ่งสเต็ป

$$= (rgscan-lfscan) \times (1/Hz)$$

แต่เวลาที่คำนวนได้นี้ ยังไม่ได้รวมถึงเวลาอื่นๆ เช่น เวลาที่เครื่องคอมพิวเตอร์เฟ็ทช์ (fetch) โปรแกรมจากหน่วยความจำ , เวลาที่ต้องตั้งค่าต่างๆให้แก่วงจรและมอเตอร์ ดังนั้นจึงต้องมีการปรับค่านี้โดยคูณด้วยตัวประกอบหน่วงเวลา (delay factor ซึ่งในโปรแกรมจะแทนด้วยตัวแปร Scandelay) ซึ่งจะมีค่าประมาณ 0.8 ถึง 1.2 ขึ้นอยู่กับความถี่ของคอมพิวเตอร์ที่ใช้ โดยจะใช้ค่าที่น้อยกว่าหนึ่งกับเครื่องที่มีความถี่สูงกว่ามาตรฐาน และใช้ค่าที่มากกว่าหนึ่งกับเครื่องที่มีความถี่มาตรฐาน

3.3 คำต่างๆที่กำหนด(#define) ไว้ในไฟล์ SCANNER.H ที่สำคัญมีดังนี้

MAX_MENU 7	กำหนดจำนวนเมนูหลักสูงสุด ในที่นี้คือ 7 เมนู(0-7)
MAX_SUB_MENU 6	กำหนดจำนวนเมนูย่อยที่สูงสุด
MAX_CHAR 22	กำหนดจำนวนอักขระสูงสุด ในเมนูรองแต่ละแถว
MAX_Y_STEP 20	กำหนดจำนวนสเตปสูงสุด ที่มอเตอร์ Y จะเคลื่อนที่ในแต่ละครั้ง ดังนั้นจะเกี่ยวกับความละเอียดของภาพที่สแกนได้
MIN_HZ 20	ความถี่ต่ำสุดที่ส่งให้มอเตอร์ หรือคือ ความเร็วต่ำสุดในการเคลื่อนที่ของมอเตอร์
MAX_HZ 2000	ความถี่สูงสุดที่ส่งให้มอเตอร์ หรือคือ ความเร็วสูงสุดในการเคลื่อนที่ของมอเตอร์
MAX_STEP 1400	กำหนดการเคลื่อนที่ ของหัวสแกนมากที่สุด ต่อการเคลื่อนที่ 1 ครั้ง เป็นการกำหนดที่ขึ้นอยู่กับขนาดของแท่นสแกน
MAX_X 1200	
MAX_Y 1300	กำหนดขนาดของการสแกนเมื่อเริ่มต้นโปรแกรม ค่านี้สามารถเปลี่ยนค่าได้ในโปรแกรม
CLK 4772726.6667	เป็นความถี่ของเครื่อง IBM PC จะใช้ในการคำนวนค่าคงควบคุม ที่ต้องส่งให้ 8253 ของสแกนเนอร์
CNT 0x303	เป็นเบอร์พอร์ทของ 8253 ที่เป็นพอร์ทควบคุม
CNT0 0x301	
CNT1 0x302	
CNT2 0x303	เป็นเบอร์พอร์ตบอง 8253 ที่เป็นพอร์ตของเคาท์เตอร์ 0,1,2 ตามลำดับ
M_CTRL 0x304	เป็นเบอร์ของพอร์ตที่ใช้ควบคุมสถานะของมอเตอร์

3.4 ตัวแปรที่สำคัญ มีดังนี้

Hz	เก็บค่าความถี่ที่ส่งให้มอเตอร์ที่มาจาก Counter0
Step	เก็บจำนวนสเตป ที่ต้องการให้มอเตอร์เคลื่อนที่ จาก Counter1
Scanrate	เก็บค่าอัตราการสแกนที่มาจาก Counter2

bit เป็นสถานะของสแกนเนอร์ที่วงจรได้แลทช์ไว้ ซึ่งแต่ละบิทมีความหมายดังนี้

bit 0 ใช้อินาเบิ้ล/ดิสเอเบิ้ล Counter0

bit 1 ใช้กระตุ้น (trig) Counter1

bit 2 ใช้อินาเบิ้ล/ดิสเอเบิ้ล Counter2

bit 3 (MOTOR SELECT) ใช้เลือกว่าจะให้มอเตอร์ตัวใดเคลื่อนที่ โดยถ้าเป็น "0" จะเลือกมอเตอร์ X แต่ถ้าเป็น "1" จะเลือกมอเตอร์ Y

bit 4 DIRECTION ใช้กำหนดทิศทางการหมุนของมอเตอร์

lfscan,upscan,rgscan,dnscan ใช้กำหนดพื้นที่ที่ต้องการสแกน โดย

lfscan (left scan) หมายถึง ขอบซ้าย

upscan (up scan) หมายถึง ขอบบน

rgscan (right scan) หมายถึง ขอบขวา

dnscan (down scan) หมายถึง ขอบล่าง

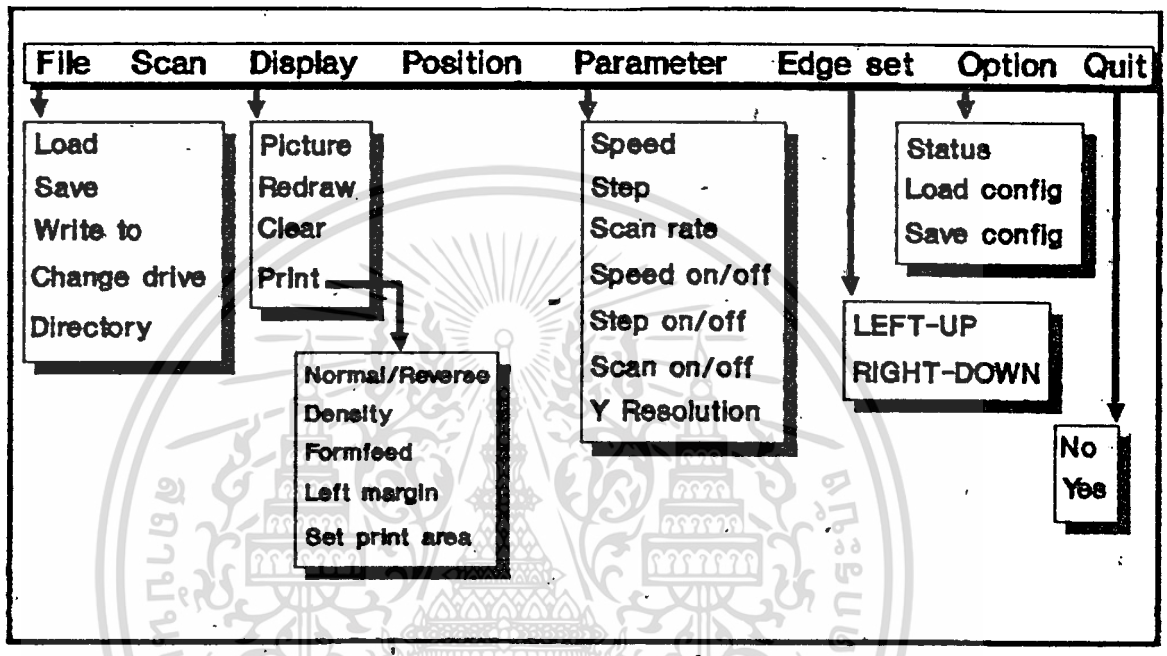
แสดงดังภาพ 3.2

รูปที่ 3.2 การกำหนดพื้นที่ของการสแกน

ystep ใช้กำหนดการเคลื่อนที่ใน 1 ครั้ง ของมอเตอร์ Y ซึ่งจะบอกถึงความละเอียดของภาพที่ได้ในแกนตั้ง

SOFTWARE

ได้ออกแบบให้ใช้ Pull-Down Menu เพื่อความง่ายในการเรียนรู้วิธีการใช้และสะดวกในการใช้งานโดยสามารถใช้เพียง Arrow Key ในการทำงานส่วนใหญ่ โดยมีโครงสร้างของเมนูดังภาพ 3.3



รูปที่ 3.3 เมนูของซอฟท์แวร์

หน้าที่ของแต่ละเมนูมีดังนี้

FILE จัดการเกี่ยวกับข้อมูลที่อยู่ใน DISKETTE

-LOAD อ่าน Filename.ext เข้ามา โดยถ้าไม่กำหนด ext (Extension) โปรแกรมจะถือว่าเป็น Filename.PIC ถ้าพบจะลบภาพเก่าทิ้ง (ถ้ามี) และแสดงภาพใหม่ขึ้นมาแทน และ ได้โปรแกรมให้สามารถอ่านไฟล์จากเครื่องสแกนรุ่นเก่าได้ด้วย

-SAVE,WRITE TO จะเก็บภาพที่อยู่ใน memory (ถ้ามี) ลงในแผ่นดิสค์โดย WRITE TO จะสามารถเปลี่ยนชื่อไฟล์ที่ต้องการเก็บได้

-CHANGE DRIVE เป็นการเปลี่ยนช่องเก็บข้อมูลไปยัง disk drive ที่ต้องการ directory ใช้ดูชื่อไฟล์ต่าง ๆ ใน current drive โดยสามารถใช้ wild card (* และ ?) ได้

SCAN เป็นการเริ่มเก็บภาพโดยจะใช้ Parameter ต่างๆ ของ Parameter Menu หรืออาจจะ Load parameter ที่เคยกำหนดไว้โดย Option Menu ก็ได้

DISPLAY เป็นเมนูที่เกี่ยวกับการจัดการภาพที่ได้ scan หรือ load เข้ามาโดย

- PICTURE จะแสดงภาพจาก memory ให้ดู
- REDRAW จะเขียนภาพที่อยู่ใน memorey ใหม่เนื่องจากอาจจะมีภาพที่ไม่ต้องการเกิดขึ้นใน Video RAM ซึ่งมีหลายสาเหตุเช่น จากโปรแกรมประเภทresident ที่ได้ pop up ขึ้นมาในระหว่างที่กำลังแสดงภาพอยู่
- CLEAR ใช้เมื่อต้องการลบภาพที่อยู่ใน memory ทิ้ง
- PRINT เป็นการพิมพ์ภาพออกเครื่องพิมพ์หรือไฟล์ก็ได้ ซึ่งสามารถพิมพ์ได้ทั้ง แบบธรรมดา(normal) และแบบกลับ (reverse) ส่วนความหนาแน่น (density) ได้กำหนดไว้ดังนี้

m	mode	dot density (dot/inch)
0	single density	60
1	low speed double density	120
2	high-speed double density	120
3	quadruple-density	240
4	CRT graphics	80
5	plotter	72
6	CRT graphics II	90

ตารางที่ 3.3 แสดงความหนาแน่นของการพิมพ์

- FORM FEED ถ้าตอบ Y หมายถึง จะเลื่อนกระดาษออกไป จนสุดแผ่นเมื่อพิมพ์เสร็จ
- LEFT MARGIN คือขอบซ้ายสุดของการพิมพ์กำหนดเป็นตัวอักษรใน Text mode

POSITION ใช้กำหนดตำแหน่งของหัวสแกนโดยใช้ ARROW KEY (หรือ Shift ARROW KEY เมื่อต้องการความเร็วในการเลื่อน) ลูกศรบนจอภาพ พร้อมทั้ง co-ordinate จะแสดงตำแหน่งของหัวสแกน เมื่ออยู่ในตำแหน่งที่ต้องการแล้ว ให้กด ENTER

PARAMETER ใช้กำหนด parameter ต่างๆ ของการสแกนดังนี้

- SPEED ใช้ตั้งความเร็วของหัวสแกนในการสแกน 1 line ได้ตั้งแต่ 18 Hz จนถึง 2000 Hz ซึ่งมีความถูกต้องสูงมาก ความเร็วนี้จะถูกจำกัดโดย Torque และ Load ของมอเตอร์

- STEP ใช้ในการตั้งจำนวนสเตปที่มอเตอร์ X เคลื่อนใน 1 ครั้ง อาจใช้ในการทดสอบการทำงานของมอเตอร์ก็ได้ ซึ่ง Step โปรแกรมจะคำนวนจากระยะ 1 line ของการสแกนในเมนู EDGE SET โดยอัตโนมัติ

- SCANRATE ใช้กำหนดอัตราการสแกนต่อการเคลื่อนที่ของมอเตอร์หน่วยเป็น สเตป/สแกน ซึ่งตัวเลขมากจะหมายถึงความละเอียดที่ลดลง ดังนั้นความละเอียดสูงสุดคือ 1

- SPEED,STEP,SCAN ON/OFF เป็นการ enable หรือ dissable COUNTER 0-2 ใช้ในการทดสอบเครื่อง

- Y Resolution ใช้กำหนดความละเอียดทางแนวตั้งของภาพ หมายถึงการที่มอเตอร์ Y จะเคลื่อนที่เมื่อสแกนเสร็จ 1 line ดังนั้น ความละเอียดสูงสุดคือ 1

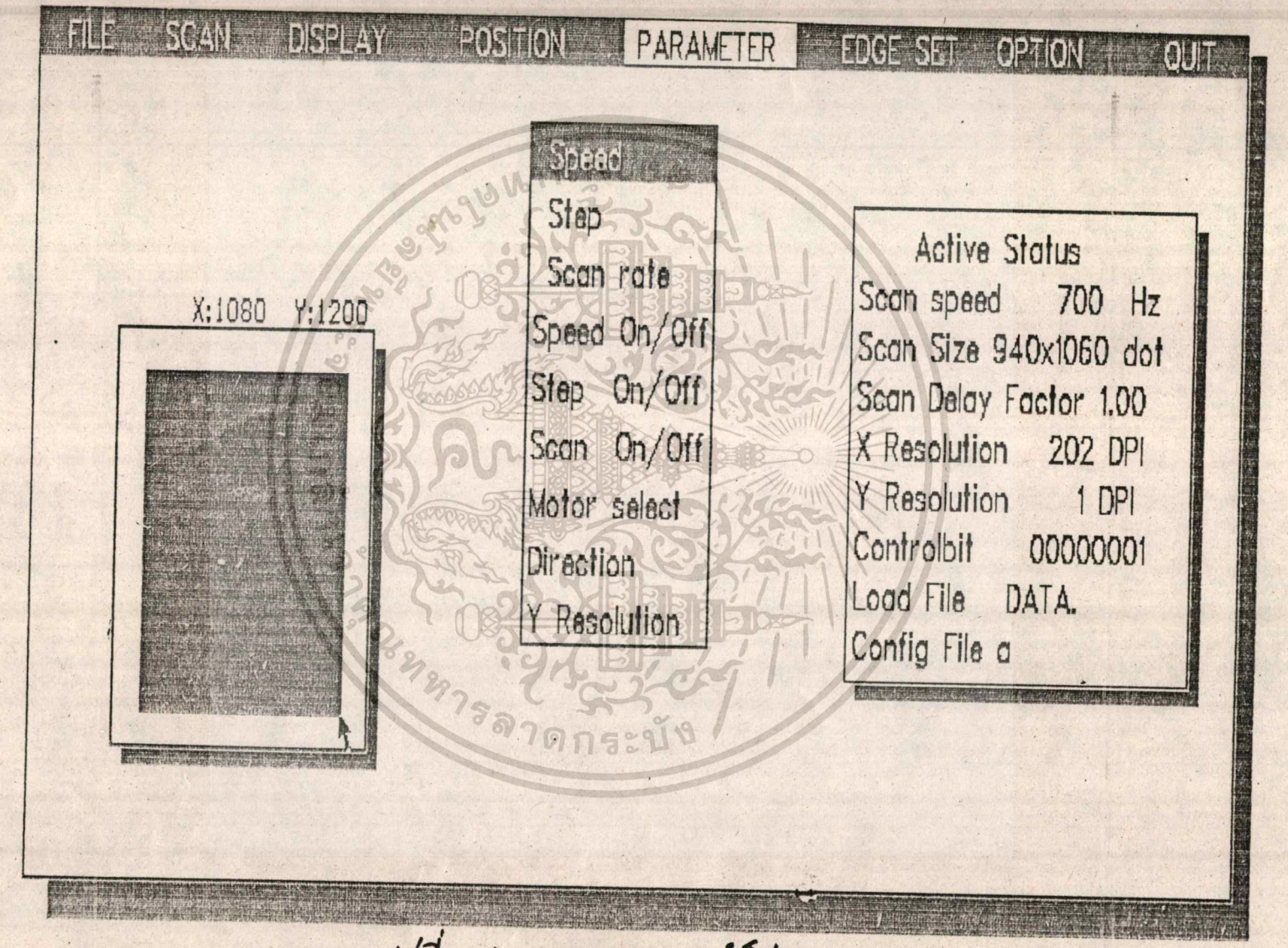
EDGE SET ใช้ตั้งขอบเขตของการสแกนโดยการตั้งมุมซ้ายบน และ/หรือ มุมขวาล่าง ถ้าขอบเขตที่สแกนเล็กเกินไป โปรแกรมจะเปลี่ยนมุมที่ต้องการตั้งนั้นเป็นตำแหน่งสูงสุดของการสแกน

OPTION เป็นเมนูที่เกี่ยวกับ parameter ของ software

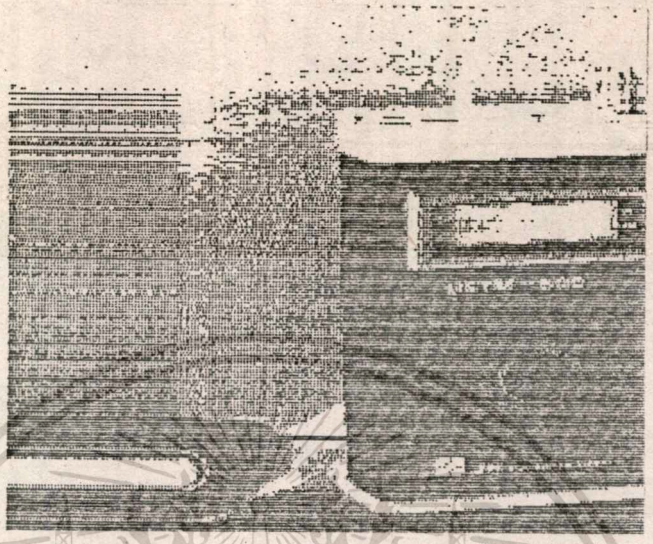
- STATUS เป็น Toggle submenu ใช้แสดงหรือไม่แสดง status window ที่บอกสถานะต่างๆ ของทั้ง software และ hardware

- LOAD, SAVE CONFIG เป็นการ load หรือ save parameter ต่างๆ ที่ได้ตั้งไว้แล้ว

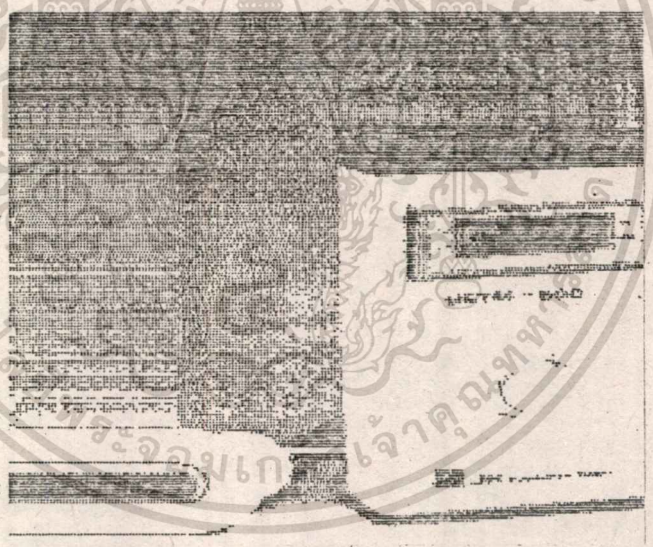
QUIT ใช้เมื่อต้องการจบการทำงาน และถ้ามีภาพที่ยังไม่ได้ save ใน memory จะมีการขอการยืนยัน และเมนูนี้จะถูกเรียกโดยอัตโนมัติ เมื่อมีการกด ESC ขณะที่อยู่ใน MENU หลัก



รูปที่ 3.4 ภาพบนจอขณะทำโปรแกรม.



ก.



ข.

รูปที่ 3.5 ตัวอย่างของภาพที่สแกนได้

ก. แบบเนกาตีฟ

ข. แบบโพสิตีฟ

บทที่ 4

การทดลองและผลการทดลอง

การทดสอบจะแบ่งเป็นส่วนๆ เพื่อสะดวกในการอธิบายดังนี้

-การทดสอบสเตปปิ้งมอเตอร์

-การทดสอบวงจรขับมอเตอร์ และ วงจรอินเตอร์เฟส

-การทดสอบทั้งระบบ

4.1 การทดสอบสเตปปิ้งมอเตอร์

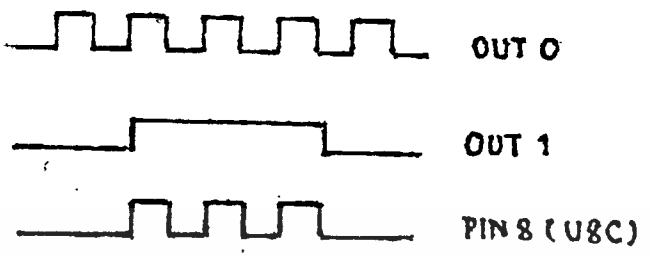
ได้ทดลองหาความเร็วสูงสุดของมอเตอร์ โดยการเพิ่มความถี่ที่ป้อนให้มอเตอร์ขณะที่ไม่มีโหลด พบว่า มอเตอร์สามารถเริ่มหมุนได้ที่ความเร็วสูงสุดได้ประมาณ 720-750 เฮิร์ต ที่ความถี่สูงกว่านี้ มอเตอร์จะไม่มีแรงพอที่จะเริ่มหมุน แต่ถ้าขณะที่มอเตอร์กำลังหมุนอยู่ สามารถเพิ่มความถี่ได้ถึง 1500 เฮิร์ต

4.2 การทดสอบวงจรขับมอเตอร์

ได้ทำการวัดกระแสที่เพาเวอร์ทรานซิสเตอร์จ่าย ขณะที่ต่อกับมอเตอร์ ปรากฏว่าวงจรสามารถจ่ายกระแสได้อย่างพอเพียง และความร้อนที่เกิดขึ้นที่ตัวทรานซิสเตอร์ขณะใช้งานอย่างต่อเนื่องมีเพียงเล็กน้อยเท่านั้น ดังนั้นจึงไม่มีความจำเป็นที่จะต้องติดแผ่นระบายความร้อน (Heat sink) ให้กับทรานซิสเตอร์

4.3 การทดสอบวงจรอินเตอร์เฟส

ได้วัดความถี่จาก counter0 โดยใช้โปรแกรมที่เขียนขึ้น ปรากฏว่ามีความถี่ถูกต้องตามที่ต้องการ โดยมีความละเอียด (resolution)ในการเปลี่ยนความถี่ถึง 1 เฮิร์ต เช่นจาก 700 Hz เป็น 701 Hz หรือ 699 Hz และลักษณะเอาท์พุทเป็นสัญญาณรูปคลื่นสี่เหลี่ยม (square wave) และเมื่อดูรูปสัญญาณที่ออกจากวงจรโมโนสเตเบิ้ล (ในที่นี้คือ counter1 ซึ่งได้ถูกโปรแกรมให้ทำงานในโหมด 1) ที่ถูก AND กับสัญญาณเอาท์พุทที่ขา OUT ของ counter0 (ในที่นี้คือสัญญาณ OUT0) จะมีลักษณะดังรูปที่ 4.1 ซึ่งจะพบว่ามีจำนวนของลูกคลื่นตรงกับที่ต้องการ



รูปที่ 4.1 แผนภูมิเวลาของวงจรอินเตอร์เฟส

4.4 <u>การทดสอบทั้งระบบ</u>

เมื่อประกอบทุกส่วนอันได้แก่ แท่นสแกน, สเตปปิ้งมอเตอร์ และส่วนของวงจร เข้าด้วยกันแล้ว พบว่าความถี่สูงสุดที่ให้แก่สเตปปิ้งมอเตอร์ในแกน x ได้จะมีค่าประมาณ 700 เฮิร์ต ถ้าป้อนความถี่ที่สูงกว่าค่าดังกล่าวนี้ให้กับมอเตอร์ หัวสแกนอาจเคลื่อนที่ไปได้บ้าง แต่จำนวนสเต็ปที่เคลื่อนไปอาจไม่ได้ครบตามที่ต้องการ ทำให้ไม่สามารถทราบตำแหน่งที่แท้จริงของหัวสแกน ขั้นต่อมาต้องทำการหาค่าตัวประกอบหน่วงเวลา (delay factor) ที่น้อยที่สุดที่จะไม่ทำให้เกิดเวลาเสียเปล่า (เวลาเสียเปล่าจะเกิดในช่วงที่มอเตอร์ทั้งสองตัวไม่มีการเคลื่อนที่ คือไม่มีการสแกนภาพ) พบว่าต้องใช้ค่าตัวประกอบหน่วงเวลาประมาณ 1.1-1.3 สำหรับเครื่องคอมพิวเตอร์ IBM ที่มีความถี่ 4.77 MHz (รายละเอียดเกี่ยวกับตัวประกอบหน่วงเวลาให้ดูจากเรื่องซอฟท์แวร์ ในบทที่ 3)

เมื่อทดลองสแกนภาพขนาด A4 ทั้งภาพ จะใช้เวลาประมาณ 30 นาที ซึ่งมีค่าใกล้เคียงกับค่าที่คำนวนไว้แล้ว

บทที่ 5

สรุปและวิจารณ์

เครื่องอิมเมจสแกนเนอร์ ที่สร้างขึ้นให้ประสิทธิภาพเป็นที่น่าพอใจ ปัญหาที่พบจะเกิดเนื่องจากเสต็ปปิ้งมอเตอร์ที่ใช้มีความเร็วและแรงบิดค่อนข้างต่ำ ทำให้อัตราการสแกนใน 1 ภาพ ต้องใช้เวลาถึง 30 นาที อย่างไรก็ดีเครื่องสแกนภาพนี้สามารถแก้ปัญหาของเครื่องสแกนภาพรุ่นที่ใช้เครื่องพิมพ์เป็นแท่นสแกนได้หลายประการดังนี้

ส่วนฮาร์ดแวร์

การทำงานของทุกส่วนได้ผลถูกต้องเป็นที่น่าพอใจ สามารถแก้ไขปัญหาของเครื่องสแกนรุ่นเดิมที่ใช้เครื่องพิมพ์เป็นตัวขับเคลื่อนหัวอ่านได้ ดังนี้

- การเคลื่อนที่ของหัวสแกนน่าจะเร็วขึ้น โดยสามารถตั้งความเร็วได้ตามความต้องการด้วยซอฟท์แวร์ ซึ่งมีผลดีคือ หากต้องมีการเปลี่ยนมอเตอร์ ที่มีความเร็วสูงสุดต่ำกว่าตัวที่เคยใช้อยู่ จะสามารถทำได้ทันที โดยเพียงแต่โปรแกรมให้ความเร็วสูงสุดสัมพันธ์กับมอเตอร์ที่จะใช้

- ความคมชัดของขอบภาพมีมากขึ้น เนื่องจากไม่ต้องมีการหน่วงเวลาในการส่งคำสั่งควบคุม ให้กับเครื่องพิมพ์เหมือนรุ่นเก่า แต่สามารถเริ่มสแกนได้ทันทีที่มอเตอเคลื่อนที่ ซึ่งจะมีผลทำให้เวลารวมในการสแกนเร็วขึ้นด้วย

- สามารถสแกนได้ทั้งสองทิศทาง คือ ทั้งจากซ้ายไปขวา และจากขวาไปซ้าย ได้ในภาพเดียวกัน โดยจะให้ผลออกมาเหมือนกัน เนื่องจากการสแกนไม่ได้ใช้การเฉลี่ยเวลาในการเคลื่อนที่ใน 1 line แต่ใช้นับจำนวนสเตปในการเคลื่อนแทน (ในรุ่นเดิมความเร็วในการเคลื่อนที่ไปและกลับไม่เท่ากัน)

- สามารถใช้กับเครื่อง IBM PC รุ่นที่มีความถี่เท่าใดก็ได้ โดยไม่ต้องมีการแก้ไขทั้ง ฮาร์ดแวร์และซอฟท์แวร์

ส่วนซอฟท์แวร์

- สามารถอ่านภาพจากข้อมูลที่เก็บไว้โดยเครื่องสแกนรุ่นเก่าได้ โดยไม่ต้องใช้โปรแกรมช่วยในการแสดงผล (Driver program)

แนวทางพัฒนา

- เมื่อต้องการความละเอียดของการสแกนมากขึ้น โดยใช้สเต็ปปิ้งมอเตอร์ตัวเดิม อาจทำได้โดยใช้การขับมอเตอร์เป็นแบบไมโครสเต็ป (micro step) ซึ่งรายละเอียด สามารถหาได้จากหนังสืออ้างอิงหมายเลข 4

- ใช้หัวอ่านแบบ CCD (Charge-Couple Device) ซึ่งจะให้ภาพที่มีความละเอียดมากกว่า 200 จุดต่อนิ้ว โดยใช้เวลาสแกนเพียง 1 นาทีต่อภาพเท่านั้น

- เมื่อปรับปรุงแก้ไขโปรแกรมอีกเล็กน้อย จะสามารถใช้ซอฟท์แวร์นี้กับเครื่องจอสีได้ทันที

ภาคผนวก ก.

8253 PROGRAMMABLE TIMER

เป็นไอซีไทเมอร์ (Timer IC) ที่สามารถเลือกการทำงานได้ 5 โหมด ส่วนประกอบที่สำคัญของไอซีเบอร์นี้ คือ วงจรนับ (programmable counter) จำนวน 3 วงจร คือ counter #0 , #1 และ #2 ซึ่งวงจรนับแต่ละตัวจะทำงานเป็นอิสระต่อกัน

สายสัญญาณที่ต่อกับวงจรนับแต่ละวงจรมี 3 สายสัญญาณ โดยเป็นสัญญาณอินพุท 2 สัญญาณ คือที่ขา CLOCK และขา GATE และมีขา OUT เป็นสัญญาณ เอาท์พุต แต่ละสายสัญญาณมีหน้าที่ดังนี้คือ

CLOCK เป็นขาที่ใช้ป้อนสัญญาณนาฬิกาให้กับ วงจรนับ (counter) โดยความถี่ของสัญญาณนาฬิกามีค่าตั้งแต่ 0 Hz (หรือไฟ DC) ถึง 2.6 MHz

GATE เป็นขารับสัญญาณอินพุทโดยสัญญาณนี้จะทำหน้าที่เป็นตัว ปิด-เปิด สัญญาณนาฬิกาที่จะผ่านเข้าไปในขา CLOCK หรือเป็นตัวกำหนดการเริ่มนับให้กับวงจรนับโดยขึ้นอยู่กับโหมดการทำงานของ 8253

OUT เป็นสัญญาณที่นำไปใช้งาน โดยลักษณะของสัญญาณที่ออกจากขา OUT จะขึ้นอยู่กับการกำหนดโหมดการทำงานของ 8253

การโปรแกรม 8253

คือการป้อนข้อมูลให้กับ ขา D0-D7 เพื่อเป็นการเลือกใช้วงจรนับเบอร์ต่างๆและสั่งให้วงจรนับผลิตสัญญาณเอาท์พุทออกมาตามที่ต้องการ โดยการให้ ขา A0 และขา A1 มีลอจิก 1 จะได้รูปแบบคำสั่งควบคุม (control word format) ดังตารางต่อไปนี้

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

ตารางที่ 1 แสดงความหมายของ D0 - D7

โดยแต่ละบิทจะมีหน้าที่ดังนี้คือ

D7,D6	ใช้เลือกวงจรนับ
D5,D4	ใช้เลือกการอ่านหรือการโหลด
D3,D2,D1	ใช้เลือกโหมด
D0	ใช้กำหนดแบบการนับถ้าเป็น"1"จะเป็นการนับแบบ BCD ถ้าเป็น"0"จะเป็นการนับแบบเลขฐานสอง

เราสามารถแจกแจงค่า D0-D7 ในการใช้งาน 8253 ดังตารางที่ 2-4

D7	D6	วงจรนับ
0	0	0
0	1	1
1	0	2
1	1	ไม่ใช้

ตารางที่ 2 แสดงการกำหนดค่า D7 และ D6 เพื่อเลือกวงจรนับ

D5	D4	เงื่อนไขในการโหลดหรืออ่านข้อมูล
0	0	ให้วงจรนับ แลตช์เพื่อให้วงจรนับส่งค่าจากตัวมันไปยังรีจิสเตอร์ เพื่อให้หน่วยประมวลผล อ่านค่า
0	1	Read หรือ Load 8 บิตล่าง(LSB)
1	0	Read หรือ Load 8 บิตบน(MSB)
1	1	Read หรือ Load 8 บิตล่างก่อนแล้วค่อยตามด้วย 8 บิตบน

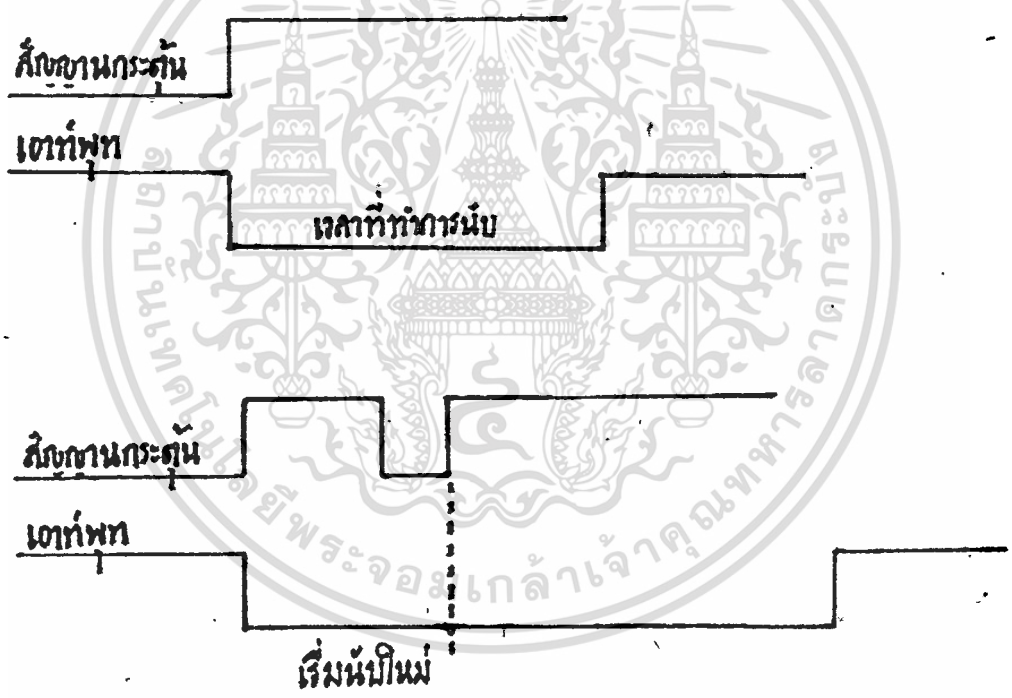
ตารางที่ 3 แสดงการกำหนดค่า D5 และ D4 เพื่อบอกว่าจะให้วงจรนับที่เลือก โดย D7 และ D6 อ่านหรือเขียนข้อมูล

D3	D2	D1	MODE VALUE
0	0	0	mode 0 : interrupt on terminal count
0	0	1	mode 1 : programmable one-shot
x	1	0	mode 2 : rate generator
x	1	1	mode 3 : square wave generator
1	0	0	mode 4 : software triggered strobe
1	0	1	mode 5 : hardware triggered strobe

ตารางที่ 4 แสดงการกำหนดค่า D3,D2 และ D1 เพื่อเลือกโหมดการทำงาน

การใช้งานโหมด 0 : interrupt on terminal count

เป็นการสร้างสัญญาณอินเทอร์รัพท์จากการนับ โดยให้วงจรนับนับสัญญาณนาฬิกา ซึ่งจะนับถอยหลังจนกระทั่งค่าที่วงจรนับนับได้เป็น 0 ขา OUT ของวงจรนับ จะมีลอจิกเป็น "1". และจะคงค่านี้จนกระทั่งมีการโหลดค่าใหม่ และในขณะที่วงจรนับทำการนับ เราอาจจะทำการยกเลิก (disable) การนับได้โดยการป้อนสัญญาณ"0"ให้ขา GATE การใช้งานในโหมดนี้ เราจะต่อขา OUT ของวงจรนับเพื่ออินเทอร์รัพท์การทำงานของ ไมโครโปรเซสเซอร์



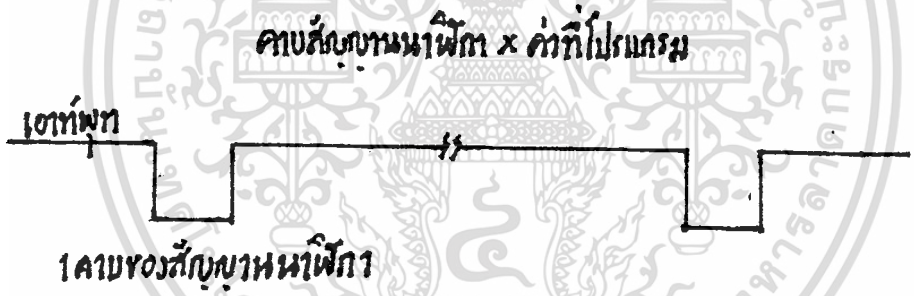
รูปที่ 1 แสดงแผนภูมิเวลา(Timing Diagram)ของ 8253 ขณะทำงานโหมด 0

การใช้งานโหมด 1 : programmable one-shot

เป็นการใช้วงจรนับเสมือนเป็นวงจรโมโนสเตเบิ้ลที่โปรแกรมได้ โดยจะให้วงจรนับสร้างสัญญาณโมโนสเตเบิ้ลที่มีความกว้างของพัลส์ เป็นจำนวนเท่าของความกว้างของคาบของสัญญาณนาฬิกา ในขั้นแรกเราจะป้อนสัญญาณกระตุ้น (trigger) ที่ขอบขาขึ้นให้กับขา GATE ซึ่งจะทำให้สัญญาณที่ขา OUT เป็น "0" โดยความกว้างของพัลส์ลบนี้ จะขึ้นกับการโปรแกรม

การใช้งานโหมด 2 : rate generator

เป็นการใช้วงจรนับให้สร้างพัลส์แคบๆต่อเนื่องกัน โดยในโหมดนี้ 8253 จะทำหน้าที่เป็นวงจรหาร n โดยสัญญาณที่ขา OUT จะมีค่าเป็น "0" ในระยะเวลา 1 คาบ ของสัญญาณนาฬิกา จากนั้นก็จะได้พัลส์บวกที่มีความกว้างของสัญญาณเท่ากับ 1 คาบของสัญญาณนาฬิกาคูณด้วยค่า n



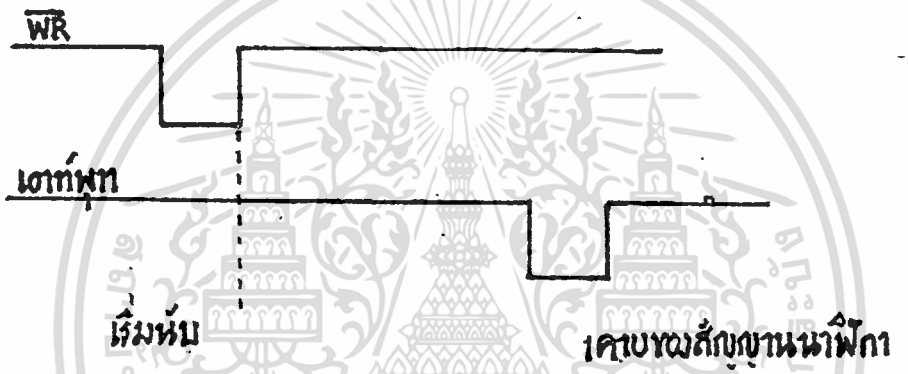
รูปที่ 2 แสดงแผนภูมิเวลาของ 8253 ขณะทำงานที่โหมด 2

การใช้งานโหมด 3 : square wave generator

เป็นการใช้วงจรนับให้สร้างสัญญาณคลื่นรูปสี่เหลี่ยม (square wave) ออกมา โดยมีการทำงานคล้ายกับโหมด 2 แต่สัญญาณที่ขา OUT ที่ได้จะเป็นคลื่นรูปสี่เหลี่ยมที่มีความถี่เป็น 1/n เท่าของความถี่ของสัญญาณนาฬิกา โดยค่า n เป็นค่าที่เราโปรแกรมให้ 8253

การใช้งานโหมด 4 : software triggered strobe

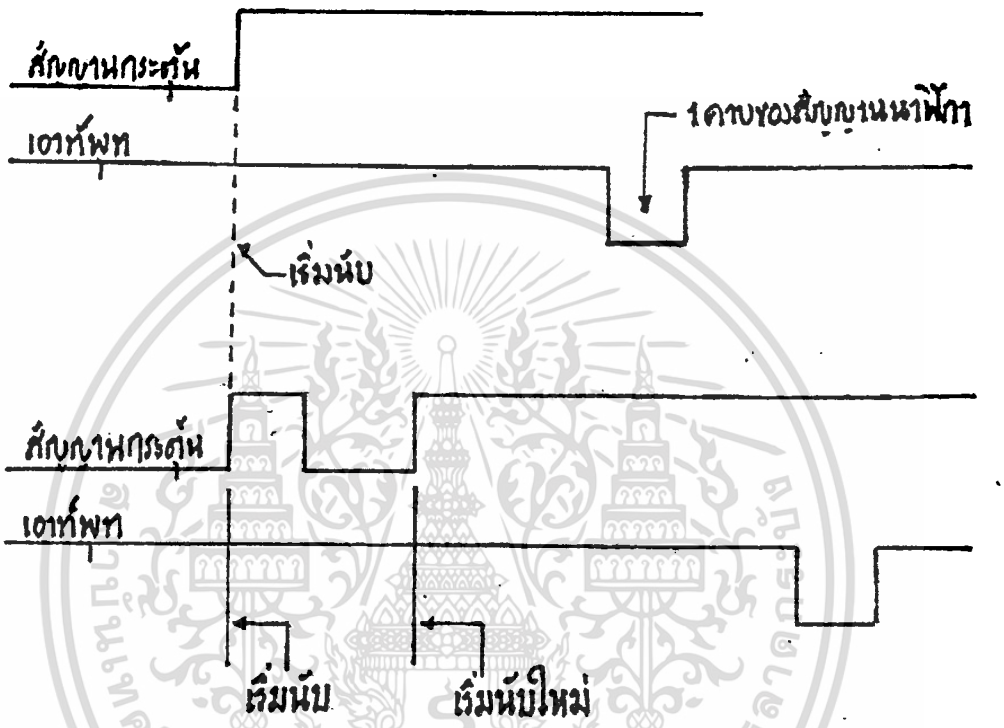
ในโหมดนี้เป็นการสั่งให้ 8253 ทำงานเป็นวงจรหน่วงเวลา โดยการทำงานจะเริ่มเมื่อรีจิสเตอร์ในวงจรนับได้รับการโหลดเรียบร้อยแล้ว (ช่วงขาขึ้นของสัญญาณWR) วงจรนับจะทำการนับลงจนได้ค่าเป็น 0 ขา OUT ของวงจรนับก็จะให้ค่าเอาท์พุทเป็นพัลส์ลบหนึ่งลูก มีความกว้างเท่ากับหนึ่งคาบของสัญญาณนาฬิกา



รูปที่ 3 แสดงแผนภูมิเวลาของ 8253 ขณะทำงานที่โหมด 4

การใช้งานโหมด 5 : hardware triggered strobe

การทำงานจะคล้ายกับโหมด 4 แต่มีข้อแตกต่างกันโดยในโหมด 4 วงจรนับจะทำการนับทันที เมื่อมีการโหลดข้อมูลเรียบร้อยแล้ว แต่ในโหมด 5 วงจรนับจะทำการนับที่ขอบขาขึ้นของสัญญาณกระตุ้นที่ขา GATE และการทำงานของวงจรนับนี้จะยอมให้มีการกระตุ้นซ้ำได้ คือ ถ้ามีสัญญาณกระตุ้นเข้ามาที่ขา GATE ในขณะที่วงจรนับยังทำการนับไม่เสร็จ วงจรนับจะเริ่มนับใหม่อีกครั้ง



รูปที่ 4 แสดงแผนภูมิเวลาของ 8253 ขณะทำงานที่โหมด 5

8253/8253-5
PROGRAMMABLE INTERVAL TIMER

- MCS-85™ Compatible 8253-5
- 3 Independent 16-Bit Counters
- DC to 2.6 MHz
- Programmable Counter Modes
- Count Binary or BCD
- Single +5V Supply
- Available in EXPRESS
  - —Standard Temperature Range
  - —Extended Temperature Range

The Intel® 8253 is a programmable counter/timer device designed for use as an Intel microcomputer peripheral. It uses nMOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

It is organized as 3 independent 16-bit counters, each with a count rate of up to 2.6 MHz. All modes of operation are software programmable.

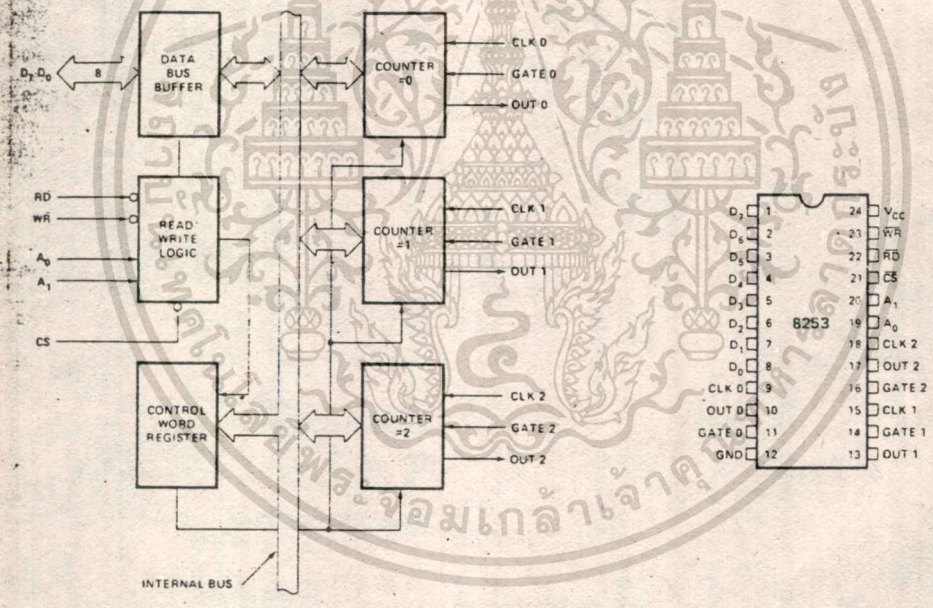


Figure 1. Block Diagram

Figure 2. Pin Configuration

AFN-00745C

FUNCTIONAL DESCRIPTION

General

The 8253 is a programmable interval timer/counter specifically designed for use with the Intel™ Microcomputer systems. Its function is that of a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8253 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in systems software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its tasks. It is easy to see that the software overhead is minimal and that multiple delays can easily be maintained by assignment of priority levels.

Other counter/timer functions that are non-delay in nature but also common to most microcomputers can be implemented with the 8253.

- Programmable Rate Generator
- Event Counter
- Binary Rate Multiplier
- Real Time Clock
- Digital One-Shot
- Complex Motor Controller

Data Bus Buffer

This 3-state, bi-directional, 8-bit buffer is used to interface the 8253 to the system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput CPU instructions. The Data Bus Buffer has three basic functions.

1. Programming the MODES of the 8253.
2. Loading the count registers.
3. Reading the count values.

Read/Write Logic

The Read/Write Logic accepts inputs from the system bus and in turn generates control signals for overall device operation. It is enabled or disabled by CS so that no operation can occur to change the function unless the device has been selected by the system logic.

$\overline{RD}$ (Read)

A "low" on this input informs the 8253 that the CPU is inputting data in the form of a counters value.

$\overline{WR}$ (Write)

A "low" on this input informs the 8253 that the CPU is outputting data in the form of mode information or loading counters.

A0, A1

These inputs are normally connected to the address bus. Their function is to select one of the three counters to be operated on and to address the control word register for mode selection.

$\overline{CS}$ (Chip Select)

A "low" on this input enables the 8253. No reading or writing will occur unless the device is selected. The $\overline{CS}$ input has no effect upon the actual operation of the counters.

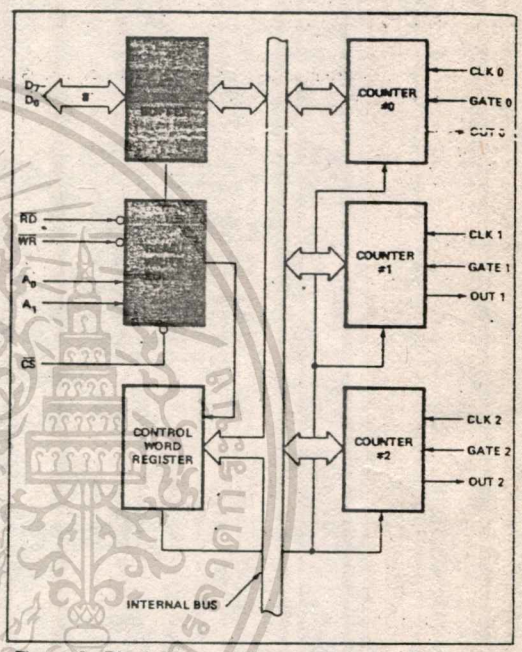


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Load Counter No. 0
0	1	0	0	1	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	0	1	1	Write Mode Word
0	0	1	0	0	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	0	1	1	0	Read Counter No. 2
0	0	1	1	1	No-Operation 3-State
1	X	X	X	X	Disable 3-State
0	1	1	X	X	No-Operation 3-State

Control Word Register

The Control Word Register is selected when A0, A1 are 11. It then accepts information from the data bus buffer and stores it in a register. The information stored in this register controls the operational MODE of each counter, selection of binary or BCD counting and the loading of each count register.

The Control Word Register can only be written into; no read operation of its contents is available.

Counter #0, Counter #1, Counter #2

These three functional blocks are identical in operation so only a single Counter will be described. Each Counter consists of a single, 16-bit, pre-settable, DOWN counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the Control Word Register.

The counters are fully independent and each can have separate Mode configuration and counting operation, binary or BCD. Also, there are special features in the control word that handle the loading of the count value so that software overhead can be minimized for these functions.

The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications and special commands and logic are included in the 8253 so that the contents of each counter can be read "on the fly" without having to inhibit the clock input.

8253 SYSTEM INTERFACE

The 8253 is a component of the Intel™ Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A0, A1 connect to the A0, A1 address bus signals of the CPU. The $\overline{CS}$ can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel® 8205 for larger systems.

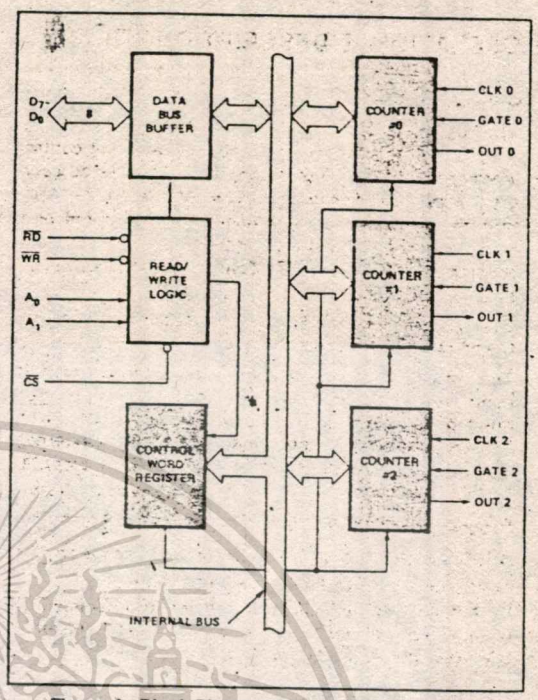


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

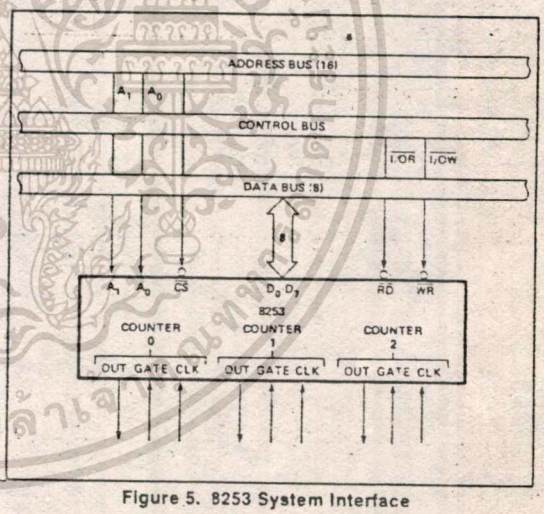


Figure 5. 8253 System Interface

OPERATIONAL DESCRIPTION

General

The complete functional definition of the 8253 is programmed by the systems software. A set of control words must be sent out by the CPU to initialize each counter of the 8253 with the desired MODE and quantity information. Prior to initialization, the MODE, count, and output of all counters is undefined. These control words program the MODE, Loading sequence and selection of binary or BCD counting.

Once programmed, the 8253 is ready to perform whatever timing tasks it is assigned to accomplish.

The actual counting operation of each counter is completely independent and additional logic is provided on-chip so that the usual problems associated with efficient monitoring and management of external, asynchronous events or rates to the microcomputer system have been eliminated.

Programming the 8253

All of the MODES for each counter are programmed by the systems software by simple I/O operations.

Each counter of the 8253 is individually programmed by writing a control word into the Control Word Register. (A0, A1 = 11)

Control Word Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

Definition of Control

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL — Read/Load:

RL1	RL0	
0	0	Counter Latching operation (see READ/WRITE Procedure Section)
1	0	Read/Load most significant byte only.
0	1	Read/Load least significant byte only.
1	1	Read/Load least significant byte first, then most significant byte.

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Counter Loading

The count register is not loaded until the count value is written (one or two bytes, depending on the mode selected by the RL bits), followed by a rising edge and a falling edge of the clock. Any read of the counter prior to that falling clock edge may yield invalid data.

MODE Definition

MODE 0: Interrupt on Terminal Count. The output will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low and the counter will count. When terminal count is reached the output will go high and remain high until the selected count register is reloaded with the mode or a new count is loaded. The counter continues to decrement after terminal count has been reached.

Rewriting a counter register during counting results in the following:

(1) Write 1st byte stops the current counting.
(2) Write 2nd byte starts the new count.

MODE 1: Programmable One-Shot. The output will go low on the count following the rising edge of the gate input.

The output will go high on the terminal count. If a new count value is loaded while the output is low it will not affect the duration of the one-shot pulse until the succeeding trigger. The current count can be read at any time without affecting the one-shot pulse.

The one-shot is retriggerable, hence the output will remain low for the full count after any rising edge of the gate input.

MODE 2: Rate Generator. Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected, but the subsequent period will reflect the new value.

The gate input, when low, will force the output high. When the gate input goes high, the counter will start from the initial count. Thus, the gate input can be used to synchronize the counter.

When this mode is set, the output will remain high until after the count register is loaded. The output then can also be synchronized by software.

MODE 3: Square Wave Rate Generator. Similar to MODE 2 except that the output will remain high until one half the count has been completed (for even numbers) and go low for the other half of the count. This is accomplished by decrementing the counter by two on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.

If the count is odd and the output is high, the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the clock by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the counter by 3. Subsequent clock pulses decrement the count by 2 until timeout. Then the whole process is repeated. In this way, if the count is odd, the output will be high for $(N+1)/2$ counts and low for $(N-1)/2$ counts.

In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following WR of a new count value.

MODE 4: Software Triggered Strobe. After the mode is set, the output will be high. When the count is loaded, the counter will begin counting. On terminal count, the output will go low for one input clock period, then will go high again.

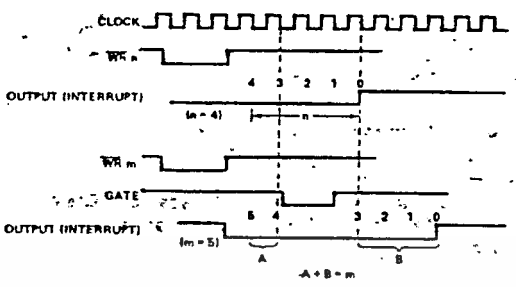
If the count register is reloaded during counting, the new count will be loaded on the next CLK pulse. The count will be inhibited while the GATE input is low.

MODE 5: Hardware Triggered Strobe. The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will not go low until the full count after the rising edge of any trigger.

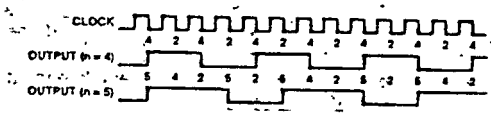
Modes \ Signal Status	Low Or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets output after next clock	—
2	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

Figure 6. Gate Pin Operations Summary

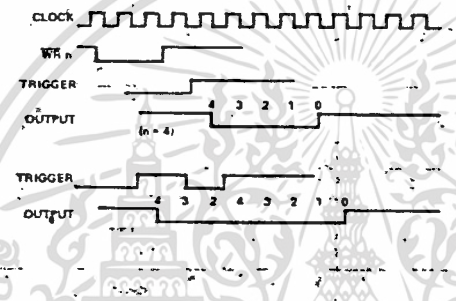
MODE 0: Interrupt on Terminal Count



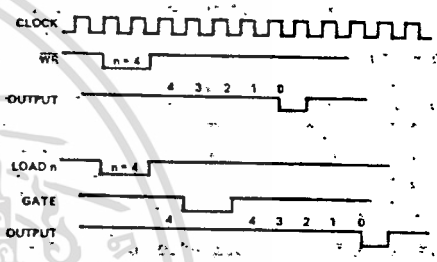
MODE 3: Square Wave Generator



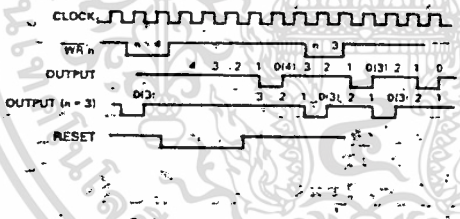
MODE 1: Programmable One-Shot



MODE 4: Software Triggered Strobe



MODE 2: Rate Generator



MODE 5: Hardware Triggered Strobe

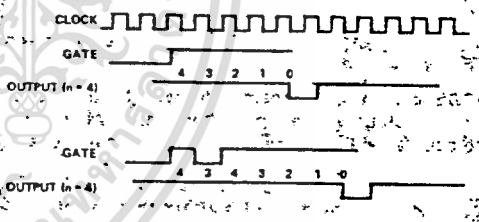


Figure 7. 8253 Timing Diagrams

8253 READ/WRITE PROCEDURE

Write Operations

The systems software must program each counter of the 8253 with the mode and quantity desired. The programmer must write out to the 8253 a MODE control word and the programmed number of count register bytes (1 or 2) prior to actually using the selected counter.

The actual order of the programming is quite flexible. Writing out of the MODE control word can be in any sequence of counter selection, e.g., counter #0 does not have to be first or counter #2 last. Each counter's MODE control word register has a separate address so that its loading is completely sequence independent. (SC0, SC1)

The loading of the Count Register with the actual count value, however, must be done in exactly the sequence programmed in the MODE control word (RL0, RL1). This loading of the counter's count register is still sequence independent like the MODE control word loading, but when a selected count register is to be loaded it must be loaded with the number of bytes programmed in the MODE control word (RL0, RL1). The one or two bytes to be loaded in the count register do not have to follow the associated MODE control word. They can be programmed at any time following the MODE control word loading as long as the correct number of bytes is loaded in order.

All counters are down counters. Thus, the value loaded into the count register will actually be decremented. Loading all zeroes into a count register will result in the maximum count (2^{16} for Binary or 10^4 for BCD). In MODE 0 the new count will not restart until the load has been completed. It will accept one of two bytes depending on how the MODE control words (RL0, RL1) are programmed. Then proceed with the restart operation.

	MODE Control Word Counter *n*
LSB	Count Register byte Counter n
MSB	Count Register byte Counter n

Note: Format shown is a simple example of loading the 8253 and does not imply that it is the only format that can be used.

Figure 8. Programming Format

		A1	A0
No. 1	MODE Control Word Counter 0	1	1
No. 2	MODE Control Word Counter 1	1	1
No. 3	MODE Control Word Counter 2	1	1
No. 4	LSB Count Register Byte Counter 1	0	1
No. 5	MSB Count Register Byte Counter 1	0	1
No. 6	LSB Count Register Byte Counter 2	1	0
No. 7	MSB Count Register Byte Counter 2	1	0
No. 8	LSB Count Register Byte Counter 0	0	0
No. 9	MSB Count Register Byte Counter 0	0	0

Note: The exclusive addresses of each counter's count register make the task of programming the 8253 a very simple matter, and maximum effective use of the device will result if this feature is fully utilized.

Figure 9. Alternate Programming Formats

Read Operations

In most counter applications it becomes necessary to read the value of the count in progress and make a computational decision based on this quantity. Event counters are probably the most common application that uses this function. The 8253 contains logic that will allow the programmer to easily read the contents of any of the three counters without disturbing the actual count in progress.

There are two methods that the programmer can use to read the value of the counters. The first method involves the use of simple I/O read operations of the selected counter. By controlling the A0, A1 inputs to the 8253 the programmer can select the counter to be read (remember that no read operation of the mode register is allowed A0, A1-11). The only requirement with this method is that in order to assure a stable count reading the actual operation of the selected counter must be inhibited either by controlling the Gate input or by external logic that inhibits the clock input. The contents of the counter selected will be available as follows:

first I/O Read contains the least significant byte (LSB).

second I/O Read contains the most significant byte (MSB).

Due to the internal logic of the 8253 it is absolutely necessary to complete the entire reading procedure. If two bytes are programmed to be read then two bytes must be read before any loading WR command can be sent to the same counter.

Read Operation Chart

A1	A0	RD	
0	0	0	Read Counter No. 0
0	1	0	Read Counter No. 1
1	0	0	Read Counter No. 2
1	1	0	Illegal

Reading While Counting

In order for the programmer to read the contents of any counter without effecting or disturbing the counting operation the 8253 has special internal logic that can be accessed using simple WR commands to the MODE register. Basically, when the programmer wishes to read the contents of a selected counter "on the fly" he loads the MODE register with a special code which latches the present count value into a storage register so that its contents contain an accurate, stable quantity. The programmer then issues a normal read command to the selected counter and the contents of the latched register is available.

MODE Register for Latching Count

A0, A1 = 11

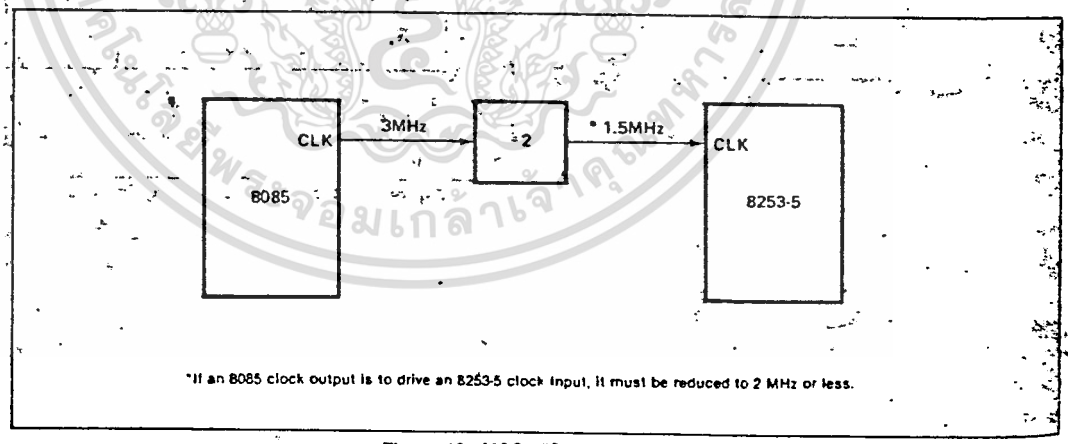
D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	X	X	X	X

SC1,SC0 — specify counter to be latched.

D5,D4 — 00 designates counter latching operation.

X — don't care.

The same limitation applies to this mode of reading the counter as the previous method. That is, it is mandatory to complete the entire read operation as programmed. This command has no effect on the counter's mode.



*If an 8085 clock output is to drive an 8253-5 clock input, it must be reduced to 2 MHz or less.

Figure 10. MCS-85™ Clock Interface*

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to +150°C
Voltage On Any Pin With Respect to Ground	−0.5 V to +7 V
Power Dissipation	1 Watt

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

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.2	V_{CC}+.5V	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 2
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to .45V
I_{CC}	V_{CC} Supply Current		140	mA	

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ± 10%, GND = 0V)

Bus Parameters (Note 3)

READ CYCLE

Symbol	Parameter	8253 Min.	8253 Max.	8253-5 Min.	8253-5 Max.	Unit
t_{AR}	Address Stable Before $\overline{READ}$	50		30		ns
t_{RA}	Address Hold Time for $\overline{READ}$	5		5		ns
t_{RR}	$\overline{READ}$ Pulse Width	400		300		ns
t_{RD}	Data Delay From $\overline{READ}$[4]		300		200	ns
t_{DF}	$\overline{READ}$ to Data Floating	25	125	25	100	ns
t_{RV}	Recovery Time Between READ and Any Other Control Signal	1		1		μs

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before $\overline{WRITE}$	50		30		ns
t_{WA}	Address Hold Time for $\overline{WRITE}$	30		30		ns
t_{WW}	$\overline{WRITE}$ Pulse Width	400		300		ns
t_{DW}	Data Set Up Time for $\overline{WRITE}$	300		250		ns
t_{WD}	Data Hold Time for $\overline{WRITE}$	40		30		ns
t_{RV}	Recovery Time Between $\overline{WRITE}$ and Any Other Control Signal	1		1		μs

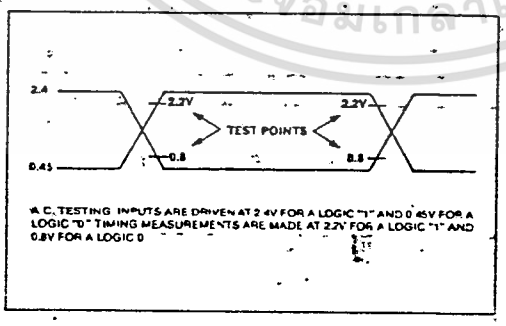
CLOCK AND GATE TIMING

Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{CLK}	Clock Period	380	dc	380	dc	ns
t_{PWH}	High Pulse Width	230		230		ns
t_{PWL}	Low Pulse Width	150		150		ns
t_{GW}	Gate Width High	150		150		ns
t_{GL}	Gate Width Low	100		100		ns
t_{GS}	Gate Set Up Time to CLK↑	100		100		ns
t_{GH}	Gate Hold Time After CLK↑	50		50		ns
t_{OD}	Output Delay From CLK↓ [4]		400		400	ns
t_{ODG}	Output Delay From Gate↓ [4]		300		300	ns

NOTES:
1. I_{OL} = 2.2 mA.
2. I_{OH} = −400 μA.
3. AC timings measured at V_{OH} 2.2, V_{OL} = 0.8.
4. C_L = 150pF.

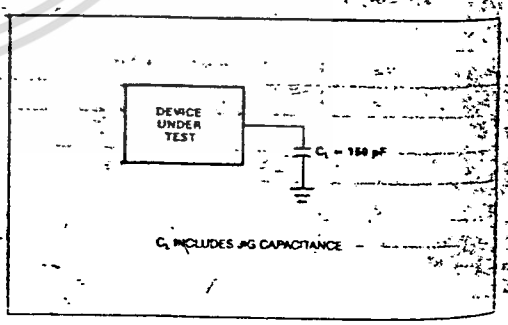
* For Extended Temperature EXPRESS, use M8253 electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0". TIMING MEASUREMENTS ARE MADE AT 2.2V FOR A LOGIC "1" AND 0.8V FOR A LOGIC 0.

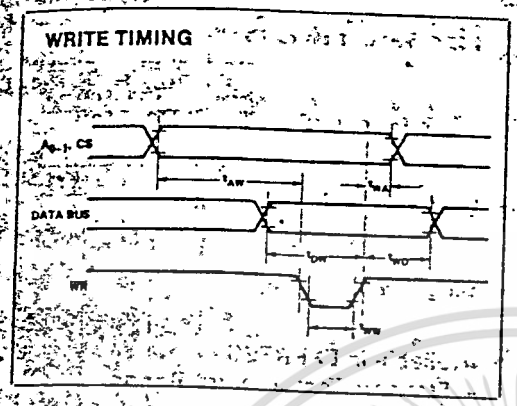
A.C. TESTING LOAD CIRCUIT



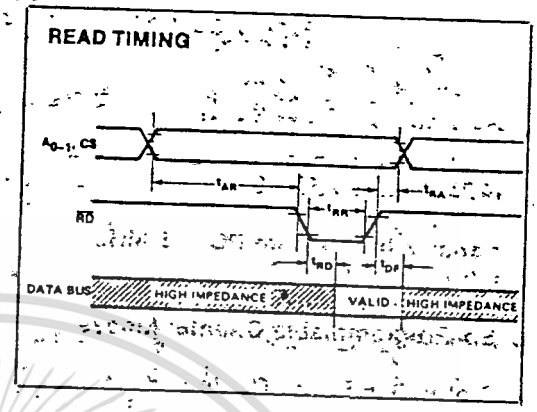
AFN-00745

WAVEFORMS

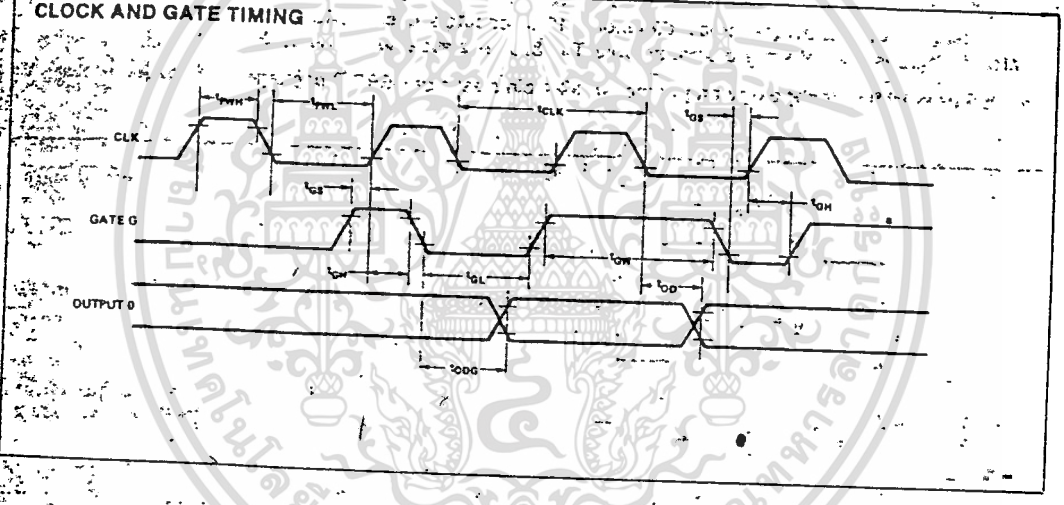
WRITE TIMING



READ TIMING



CLOCK AND GATE TIMING



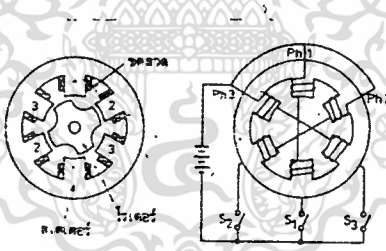
AFN-00745C

ภาคผนวก ข.

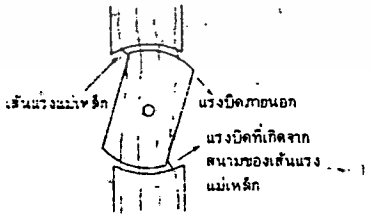
สเตปปิงมอเตอร์ (STEPPING MOTOR)

เป็นมอเตอร์ชนิดหนึ่งซึ่งเราสามารถควบคุมการหมุน โดยการป้อนพัลส์ให้กับขดลวดที่สเตเตอร์แต่ละเฟสเรียงกันไปเรื่อยๆ ส่วนของโรเตอร์ก็จะหมุนเป็นสเตปต่อเนื่องกันไปตามพัลส์ที่ป้อน ซึ่งถ้าเราป้อนพัลส์ให้มีความถี่สูงๆก็จะทำให้มอเตอร์หมุนได้ ข้อดีของมอเตอร์ชนิดนี้ คือ เราสามารถควบคุมการหมุนของมอเตอร์ รวมทั้งสามารถรู้ถึงตำแหน่งที่แน่นอนของมอเตอร์ได้ เนื่องจากมอเตอร์จะเคลื่อนที่เป็นสเตป

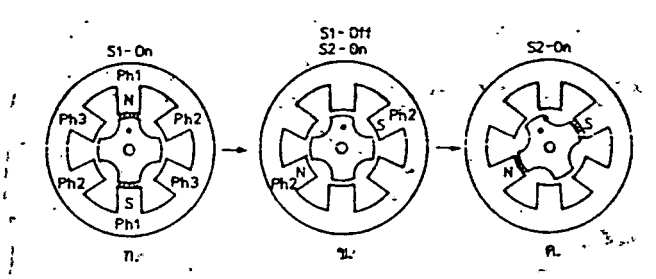
ส่วนประกอบของสเตปปิงมอเตอร์มีโรเตอร์, สเตเตอร์ และขดลวดที่พันเป็นขั้วแม่เหล็กที่สเตเตอร์ ดังรูปที่ 1 โดยการพันขดลวดนั้นจะพันเป็นเฟสต่างกัน ดังนั้นถ้าเราป้อนพัลส์หนึ่งลูกให้กับขดลวดหนึ่งเฟสของมอเตอร์แล้ว ขดลวดจะเหนี่ยวนำให้เกิดเส้นแรงแม่เหล็กตัดผ่านตัวโรเตอร์ ทำให้ตัวโรเตอร์หมุนไปหยุดในตำแหน่งที่สนามแม่เหล็กตัดผ่านตัวมันมากที่สุด ที่กล่าวมานี้เป็นหลักการเบื้องต้นในการทำให้สเตปปิงมอเตอร์หมุนแต่ต้องให้สนามแม่เหล็กที่เกิดขึ้นรับช่วงต่อกันไปเรื่อยๆตามเฟสของขดลวด ดังรูปที่ 6 และ 7



รูปที่ 5 ภาพหน้าตัดของสเตปปิงมอเตอร์แบบ 3 เฟส



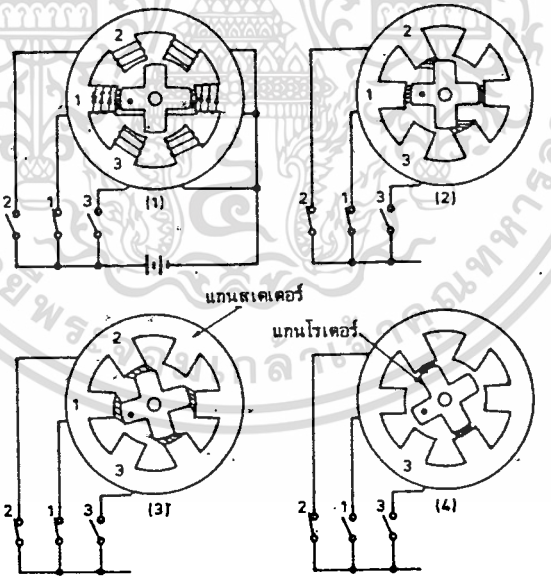
รูปที่ 6 เส้นแรงแม่เหล็กที่ทำให้เกิดแรงบิด



รูปที่ 7 แสดงการเคลื่อนที่ทีละสเตป เมื่อกระตุ้นเฟสหนึ่ง เฟสสอง

เพราะฉะนั้นถ้าเราต้องการให้สเตปปิงมอเตอร์หมุน เราต้องกระตุ้นแต่ละเฟส ของขดลวดโดยเรียงกันไปทีละขด ดังรูปที่ 8 และเมื่อต้องการให้มอเตอร์หยุดหมุน เราก็หยุดการป้อนพัลส์ให้ขดลวด มอเตอร์จะหยุดที่ตำแหน่งสุดท้ายที่มีการขับที่สเตเตอร์ เราสามารถที่จะรู้ตำแหน่งสุดท้ายของมอเตอร์ได้จากสูตร

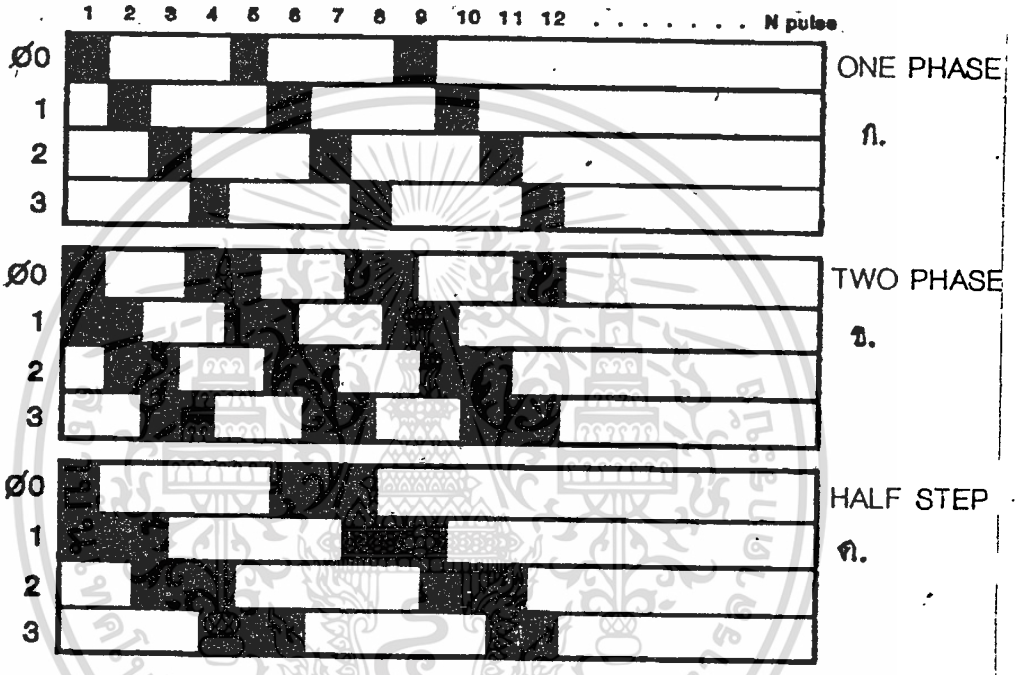
มุมที่เปลี่ยนไป = ค่ามุมต่อสเตป x จำนวนพัลส์ที่ป้อนให้



รูปที่ 8 แสดงการทำงานของสเตปปิงมอเตอร์

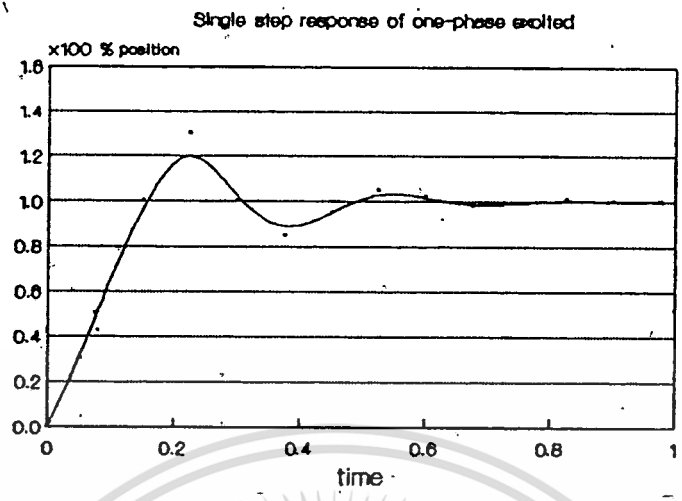
การกระตุ้น(exitation)สเตปปิงมอเตอร์ แบบที่นิยมมีอยู่ 3 แบบ คือ

1. กระตุ้นเฟสเดียว (one phase excitation)
2. กระตุ้น 2 เฟส (two phase excitation)
3. กระตุ้นครึ่งสเตป (half step excitation)



รูปที่ 9 ก.ตารางการกระตุ้นเฟสเดียว
ข.ตารางการกระตุ้นสองเฟส
ค.ตารางการกระตุ้นครึ่งสเตป

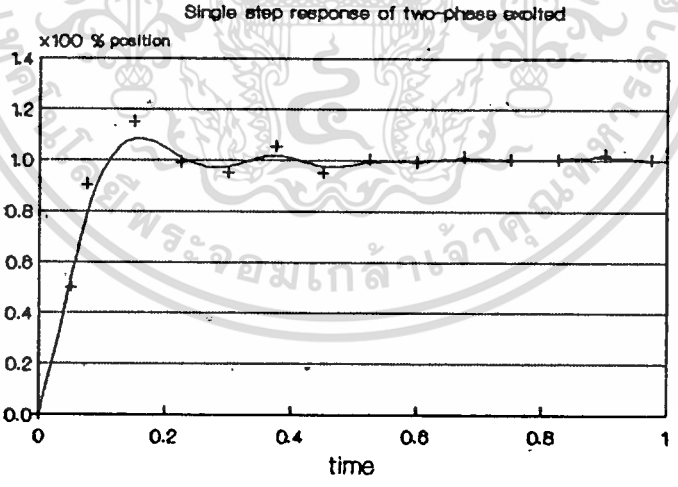
- การกระตุ้นแบบเฟสเดียว มีตารางการกระตุ้นแต่ละเฟสดังรูปที่ 5 ก.
แบบนี้จะมีแรงบิดน้อยที่สุดทั้งในขณะเริ่มเคลื่อนที่และเคลื่อนที่ไปแล้ว มีโอ-เวอร์ชูทสูงเข้าสู่ตำแหน่งแต่ละสเตปช้า ข้อดีคือ เมื่อโรเตอร์เข้าตำแหน่งในแต่ละสเตป ในสภาวะคงตัวจะไม่มีการออสซิเลท ดังรูปที่ 10



รูปที่ 10 กราฟแสดงผลตอบสนองของสเตปปิงมอเตอร์ต่อการกระตุ้นเฟสเดียว

- การกระตุ้นแบบ 2 เฟส มีตารางการกระตุ้นแต่ละเฟสดังรูปที่ 9 ข.

การกระตุ้นแบบนี้จะมีแรงบิดขณะเริ่มต้นสูง เนื่องจากเส้นแรงแม่เหล็กไม่ได้ผ่านตัวโรเตอร์ไปยังอีกด้าน แต่จะวกกลับมายังเฟสข้างๆแทนมีโอเวอร์ชูทต่ำ แต่เมื่อเข้าตำแหน่งแต่ละเฟสจะมีออสซิเลทเล็กน้อยในสภาวะคงตัว ดังรูปที่ 11



รูปที่ 11 กราฟแสดงผลตอบสนองของสเตปปิงมอเตอร์ต่อการกระตุ้นสองเฟส

- การกระตุ้นแบบครึ่งขั้น มีตารางการกระตุ้นแต่ละเฟสดังรูปที่ 9 ค.

การกระตุ้นแบบนี้จะรวมข้อดีของทั้งสองแบบข้างต้นไว้ โดยขณะเริ่มต้นจะกระตุ้นแบบ 2 เฟส ทำให้ได้แรงบิดสูงที่สุดและเมื่อเข้าตำแหน่งกระตุ้นเพียงเฟสเดียวแล้วจะไม่เกิดออสซิเลท

ภาคผนวก ค.

ความเร็วของการสแกนภาพ

ในการให้หัวอ่านเคลื่อนไปบนภาพที่ต้องการสแกนนั้น ต้องใช้สเตปปิงมอเตอร์ 2 ตัว โดยตัวหนึ่งจะเป็นตัวขับเคลื่อนให้หัวอ่านวิ่งไปตามขวางของภาพ (หรือในแนวแกน X) ส่วนอีกตัวจะขับเคลื่อนทั้งหัวอ่านและมอเตอร์ตัวแรก (มอเตอร์ X) ให้ไปตามแนวยาว (หรือในแนวแกน Y) ดังนั้นมอเตอร์ที่ขับในแกน X ควรจะเป็นตัวที่มีน้ำหนักน้อย เพราะต้องเป็นโหลดของมอเตอร์อีกตัว อีกทั้งต้องมีความเร็วรอบและความละเอียดสูง เพื่อให้การสแกนที่ความละเอียดมาก ๆเป็นไปด้วยความรวดเร็ว ซึ่งเวลาน้อยที่สุดที่ใช้ในการสแกนใน 1 line โดยใช้การขับมอเตอร์แบบ Two-phase excited สามารถหาได้จากสมการ

$$L/(r\theta f_{max})$$

โดย	f_{max}	/	ความถี่สูงสุดของมอเตอร์	(Hz/4phase)
	r		รัศมีของแกน Pulley	(inch)
	θ		มุมของการเคลื่อนที่ใน 1 สเตป	(Radian)
	L		ความยาวของการสแกนใน 1 line	(inch)
หรือ	$1/f_{max}$	คือ	เวลาที่ใช้ใน 1 สเตป	(sec)
	$r\theta$	คือ	ระยะทางที่เคลื่อนได้ใน 1 สเตป	(inch)

ดังนั้นหากภาพที่สแกนเป็นกระดาษขนาด A4 ที่มีความกว้าง L=8.5 นิ้ว รัศมีของ Pulley r=0.2" มอเตอร์ที่ใช้มี θ=1.8 องศา f_{max}=700 Hz จะใช้เวลา 1.9 วินาที

จะเห็นได้ว่า เป็นเวลาที่ไม่มากนัก แต่ถ้าต้องการความละเอียดในแกน Y ให้สูง เช่น 100 DPI ในขนาดกระดาษ A4 ที่มีความยาว = 11.5 นิ้ว ต้องสแกนถึง

100 x 11.5 = 1150 line

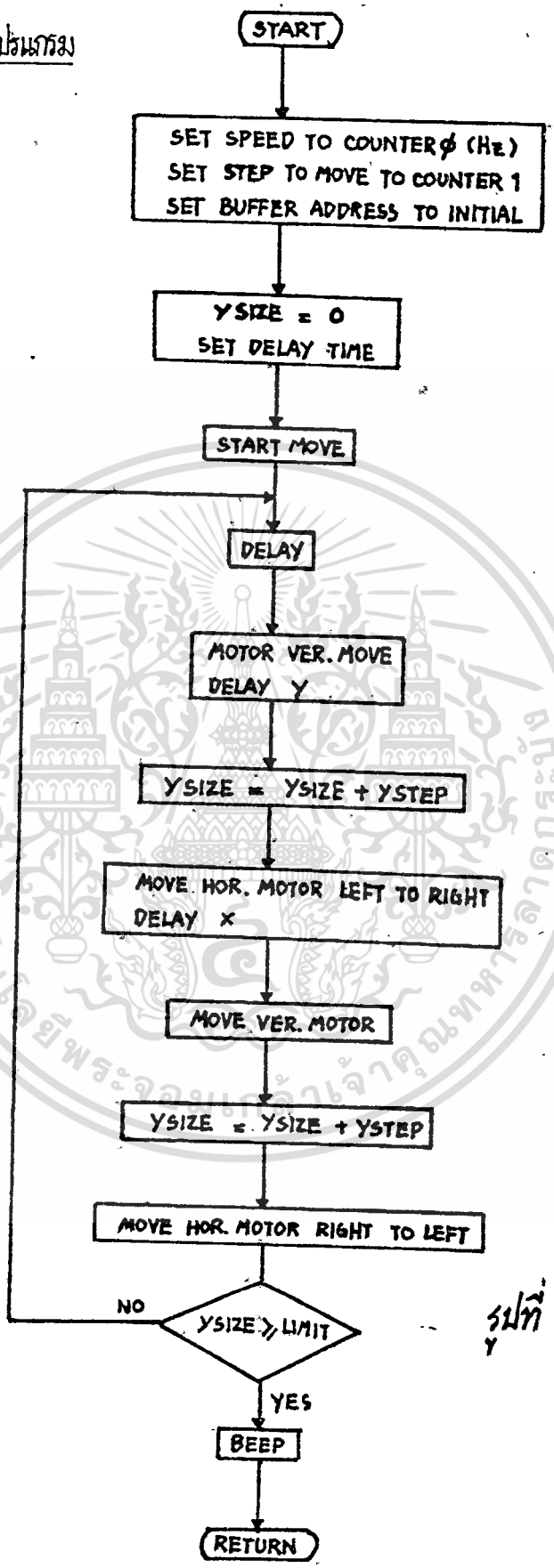
เมื่อคิดเวลารวมในการสแกน 1 แผ่นจะใช้เวลา

(เวลาในการสแกน 1 line)x(จำนวน line) = 1.9x1150 = 2222 วินาที หรือประมาณ 37 นาที ซึ่งเป็นเวลาที่นานมากสำหรับ 1 ภาพ และยังต้องรวมเวลาที่ใช้เคลื่อนในแนวแกน Y เวลาในการส่งระหัสควบคุม (control code)และเวลาในการย้ายข้อมูลจาก Buffer มาเก็บยังคอมพิวเตอร์อีกด้วย แต่เวลจที่เพิ่มมาดังกล่าวจะมีผลเพียงเล็กน้อยเมื่อเทียบกับเวลาในการเคลื่อนที่ในแกน X กับความละเอียดสูงๆ ที่ต้องการ

การลดเวลาในการสแกน สามารถทำได้โดยการปรับช่วงเวลาการสแกนให้สแกนเฉพาะส่วนของภาพที่ต้องการ ซึ่งการลดความยาวของการสแกนจากขอบกระดาษทั้งสี่เพียง 1 นิ้ว (ซึ่งขนาดของภาพโดยปกติจะไม่เต็มขนาด A4 อยู่แล้ว) จะสามารถลดเวลาในการสแกนได้ถึง 13 นาที

ที่กล่าวมาข้างต้นเป็นความละเอียดในการสแกนโดย parameter ของแท่นสแกน ส่วนความละเอียดที่สามารถใช้สแกนภาพจริงนั้นจะขึ้นอยู่กับประสิทธิภาพของหัวอ่าน ซึ่งหัวอ่านที่เคยใช้ยังมีความละเอียดน้อยกว่าแท่นสแกน จะทำให้ความละเอียดสูงสุดขึ้นอยู่กับหัวอ่านแทน

ภาคผนวก ง. ตัวโปรแกรม



รูปที่ 12 โฟลว์ชาร์ต (flow chart) ของโปรแกรมควบคุมการหมุนของมอเตอร์

```
 1  #define MAX_MENU 7
 2  #define MAX_SUB_MENU 8
 3  #define MAX_CHAR 22
 4  #define MAX_Y_STEP 20
 5  #define MOV_SPEED 20
 6  #define OFX   4
 7  #define OFY   4
 8  #define MIN_HZ    20
 9  #define MAX_HZ    2000
10  #define MAX_STEP  1400
11  #define MAX_X     1200
12  #define MAX_Y     1300
13  #define CLK       4772726.6667
14  #define CNT       0x303
15  #define CNT0      0x300
16  #define CNT1      0x301
17  #define CNT2      0x302
18  #define M_CTRL    0x304
19  #define MINSCAN   20
20  #define PLATE_LF  50
21  #define PLATE_UP  120
22  #define PLATE_RG  200
23  #define PLATE_DN  280
24  #define KX        (float)(PLATE_RG-PLATE_LF)/MAX_X
25  #define KY        (float)(PLATE_DN-PLATE_UP)/MAX_Y
26
27
28  #if !defined(MAIN)
29
30  extern void far *arrow;
31  extern char *picbuffer,loadfile[30],configfile[30],interactstat;
32  extern float scandelay;
33  extern unsigned  Hz,Step,Scanrate,bit;
34  extern int tw,th,posx,posy,lfscan,upscan,rgscan,dnscan,arrowpoint[],ystep;
35  extern char buff[15],havedata;
36  extern struct menus { char str[12];
37                        int  x;
38                        void far *ptr;
39                      } menu[MAX_MENU+1];
40  #endif
```

```
#define MAX_MENU 7
#define MAX_SUB_MENU 8
#define MAX_CHAR 22
#define MAX_Y_STEP 20
#define MOV_SPEED 20
#define OFX   4
#define OFY   4
#define MIN_HZ    20
#define MAX_HZ    2000
#define MAX_STEP  1400
#define MAX_X     1200
#define MAX_Y     1300
#define CLK       4772726.6667
#define CNT       0x303
#define CNT0      0x300
#define CNT1      0x301
#define CNT2      0x302
#define M_CTRL    0x304
#define MINSCAN   20
#define PLATE_LF  50
#define PLATE_UP  120
#define PLATE_RG  200
#define PLATE_DN  280
#define KX        (float)(PLATE_RG-PLATE_LF)/MAX_X
#define KY        (float)(PLATE_DN-PLATE_UP)/MAX_Y


#if !defined(MAIN)

extern void far *arrow;
extern char *picbuffer,loadfile[30],configfile[30],interactstat;
extern float scandelay;
extern unsigned  Hz,Step,Scanrate,bit;
extern int tw,th,posx,posy,lfscan,upscan,rgscan,dnscan,arrowpoint[],ystep;
extern char buff[15],havedata;
extern struct menus { char str[12];
                      int  x;
                      void far *ptr;
                    } menu[MAX_MENU+1];
#endif
```

```
1  #define MAIN
2
3  #include "scanner.h"
4
5  void far *arrow;
6  char *picbuffer,loadfile[30],configfile[30],interactstat=1;
7  int arrowpoint[]={10,10, 10,18, 13,16, 15,23, 16,23, 13,16, 16,16, 10,10};
8  float scandelay=1.0;
9  unsigned Hz=700,Step=1000,Scanrate=1,bit;
10 int tw,th,posx,posy;
11 int lfscan=0,upscan=0,rgscan=MAX_X,dnscan=MAX_Y,ystep=MAX_Y_STEP;
12 char buff[15],havedata;
13 struct menus { char str[12];
14                 int  x;
15                 void far *ptr;
16               }menu[MAX_MENU+1];
17 main()
18 { int fn;
19   init();
20   boarder();
21   status();
22   fn=barmenu();
23 }
```

```
#include <graphics.h>
#include <stdarg.h>
#include <string.h>
#include <bios.h>
#include <alloc.h>
#include <dos.h>
#include "zspeckey.h"
#include "scanner.h"


static char submenu[MAX_MENU+1][MAX_SUB_MENU+1][MAX_CHAR];
static int sm[MAX_MENU+1];

void far *sbar;
void far *tem0;
void far *tem1;
static int tw1,sublf=10,subup=20;

void init()      /* xmax=719  ymax=347 */
{ int g_error,gdrv=DETECT,gmode,i;
   if(registerbgidriver(Herc_driver)<0) exit (1);
   if(registerbgifont(sansserif_font)<0) exit (1);
   if(registerbgifont(small_font)<0) exit (1);
   initgraph(&gdrv,&gmode,"");
   g_error=graphresult();
   if (g_error!=grOk)
   {  printf("Initgraph error : %s.\n",grapherrormsg(g_error));
      exit(1);
   }
   outtextxy(getmaxx()/2-110,getmaxy()/2,"P l e a s e  w a i t ...");
   settextjustify(LEFT_TEXT,TOP_TEXT);
   settextstyle(3,HORIZ_DIR,USER_CHAR_SIZE);
   setusercharsize(1,2,1,2);
   tw=textwidth("H"); th=textheight("H");
   tw1=textwidth(" Config file  ");

   strcpy(menu[0].str," FILE ");     menu[0].x=10;
   strcpy(menu[1].str," SCAN ");     menu[1].x=67;
   strcpy(menu[2].str," DISPLAY ");  menu[2].x=140;
   strcpy(menu[3].str," POSITION "); menu[3].x=240;
   strcpy(menu[4].str," PARAMETER ");menu[4].x=344;
   strcpy(menu[5].str," EDGE SET "); menu[5].x=464;
   strcpy(menu[6].str," OPTION ");   menu[6].x=555;
   strcpy(menu[7].str," QUIT ");     menu[7].x=652;


   setactivepage(1); setvisualpage(0);

   arrow=(void far *)farmalloc(imagesize(10,10,16,23));
   if (arrow==0) {closegraph(); exit(2); }
   fillpoly(8,arrowpoint);
   getimage(10,10,16,23,arrow);

   setcolor(0); setbkcolor(1);
   floodfill(1,1,1);
```

```
56
57
58      sbar=(void far *)farmalloc(imagesize(sublf+1,subup+1,\
59           sublf-1+2*OFX+tw1,subup+2*OFY+th-1) );
60      getimage(sublf+1,subup+1,sublf-1+2*OFX+\
61           tw1,subup+2*OFY+th-1,sbar);
62      for (i=0;i<=MAX_MENU;i++)
63      { menu[i].ptr=(void far *)farmalloc(\
64           imagesize(0,0,textwidth(menu[i].str),\
65                th+2) );
66        getimage(0,0,textwidth(menu[i].str),\
67                th+2,menu[i].ptr);
68      }
69
70
71      strcpy(submenu[0][0]," Load ");
72      strcpy(submenu[0][1]," Save ");
73      strcpy(submenu[0][2]," Write to.. ");
74      strcpy(submenu[0][3]," Change drive");sm[0]=4;
75      strcpy(submenu[1][0]," Start Scan ");
76      strcpy(submenu[1][1]," Set Delay ");  sm[1]=2;
77      strcpy(submenu[2][0]," Picture");
78      strcpy(submenu[2][1]," Redraw");
79      strcpy(submenu[2][2]," Clear ");
80      strcpy(submenu[2][3]," Print ");      sm[2]=4;
81      strcpy(submenu[4][0]," Speed ");
82      strcpy(submenu[4][1]," Step ");
83      strcpy(submenu[4][2]," Scan rate ");
84      strcpy(submenu[4][3],"Speed On/Off");
85      strcpy(submenu[4][4],"Step  On/Off");
86      strcpy(submenu[4][5],"Scan  On/Off");
87      strcpy(submenu[4][6],"Motor select");
88      strcpy(submenu[4][7],"Direction");
89      strcpy(submenu[4][8],"Y Resolution"); sm[4]=9;
90      strcpy(submenu[5][0]," LEFT-UP ");
91      strcpy(submenu[5][1]," RIGHT-DOWN "); sm[5]=2;
92      strcpy(submenu[6][0]," Status D/A");
93      strcpy(submenu[6][1]," Load Config ");
94      strcpy(submenu[6][2]," Save Config ");sm[6]=3;
95      strcpy(submenu[7][0]," NO ");
96      strcpy(submenu[7][1]," YES ");        sm[7]=2;
97
98      clearviewport();
99      setactivepage(0); setcolor(1); setbkcolor(0);
100     clearviewport();
101   }
102   boarder()
103   { int i;
104     rectangle(0,0,getmaxx()-11,getmaxy()-16);
105     line(0,th+4,getmaxx()-11,th+4);
106     floodfill(getmaxx()/2,(th+4)/2,1);
107     moveto(15,getmaxy()-14);
108     lineto(15,getmaxy());
109     lineto(getmaxx(),getmaxy());
110     lineto(getmaxx(),10);
```

```
111     lineto(getmaxx()-9,10);
112     lineto(getmaxx()-9,getmaxy()-14);
113     lineto(15,getmaxy()-14);
114
115     floodfill(getmaxx()-1,11,1);
116
117  /* for (i=0;i<=(th+4)/2;i++)
118       rectangle( (th+4)/2-i, (th+2)/2-i,getmaxx()-(th+4)/2+i,(th+4)/2+i);
119  */
120
121     setcolor(0); setbkcolor(1);
122     for (i=0;i<=MAX_MENU;i++)
123     { outtextxy(menu[i].x,1,menu[i].str);
124     }
125     setcolor(1); setbkcolor(0);
126     rectangle(PLATE_LF,PLATE_UP,PLATE_RG,PLATE_DN);
127     moveto(PLATE_LF+7,PLATE_DN+2);
128     lineto(PLATE_LF+7,PLATE_DN+7);
129     lineto(PLATE_RG+7,PLATE_DN+7);
130     lineto(PLATE_RG+7,PLATE_UP+7);
131     lineto(PLATE_RG+2,PLATE_UP+7);
132     lineto(PLATE_RG+2,PLATE_DN+2);
133     lineto(PLATE_LF+7,PLATE_DN+2);
134     floodfill(PLATE_LF+8,PLATE_DN+3,1);
135     putimage(PLATE_LF,PLATE_UP,arrow,XOR_PUT);
136  }
137  int subbarmenu(int sel)
138  {int sub=0,i,submen=0;
139
140     setcolor(1); setbkcolor(0);
141     tem0=(void far *)farmalloc( imagesize(sublf,subup,\
142                                 sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel]));
143     getimage(sublf,subup,sublf+2*OFX+twl,\
144           subup+(2*OFY+th)*sm[sel],tem0);
145     putimage(sublf,subup,tem0,XOR_PUT);
146     tem1=(void far *)farmalloc( imagesize(sublf,subup,\
147                                 sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel]) );
148
149     rectangle(sublf,subup,sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel]);
150     for (i=0;i<sm[sel];i++)
151     { outtextxy(sublf+OFX,subup+1+OFY/2+(2*OFY+th)*i,submenu[sel][i]);
152     }
153
154     putimage(sublf+1,subup+1,sbar,XOR_PUT);
155
156     while ( (sub!=ESC)&&(sub!=CR) )
157     { sub=getkey();
158       switch(sub)
159       { case '8'     :
160         case UPKEY   : if (bioskey(2) & 0x10)
161                          if (subup-MOV_SPEED>20)
162                          {  getimage(sublf,subup,\
163                               sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel],tem1);
164                             putimage(sublf,subup,tem0,COPY_PUT);
165                             subup-=MOV_SPEED;
```

```
166                         getimage(sublf,subup,\
167                           sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel],tem0);
168                         putimage(sublf,subup,tem1,COPY_PUT);
169                       }
170                       else beep();
171                     else
172                     { putimage(sublf+1,subup+1+(2*OFY+th)*submen,sbar,XOR_PUT);
173                       submen=(--submen<0) ? sm[sel]-1 : submen;
174                       putimage(sublf+1,subup+1+(2*OFY+th)*submen,sbar,XOR_PUT);
175                     }
176                       break;
177       case '2'    :
178       case DOWNKEY : if (bioskey(2) & 0x10)
179                       if (subup+(2*OFY+th+1)*MAX_SUB_MENU+MOV_SPEED+8<getmaxy())
180                       {  getimage(sublf,subup,\
181                            sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel],tem1);
182                          putimage(sublf,subup,tem0,COPY_PUT);
183                          subup+=MOV_SPEED;
184                          getimage(sublf,subup,\
185                            sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel],tem0);
186                          putimage(sublf,subup,tem1,COPY_PUT);
187                       }
188                       else beep();
189                     else
190                     { putimage(sublf+1,subup+1+(2*OFY+th)*submen,sbar,XOR_PUT);
191                       submen=(++submen>=sm[sel]) ? 0 : submen;
192                       putimage(sublf+1,subup+1+(2*OFY+th)*submen,sbar,XOR_PUT);
193                     }
194                     break;
195       case LEFTKEY : if (bioskey(2) & 0x10)
196                       if (sublf-MOV_SPEED>0)
197                       {  getimage(sublf,subup,\
198                            sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel],tem1);
199                          putimage(sublf,subup,tem0,COPY_PUT);
200                          sublf-=MOV_SPEED;
201                          getimage(sublf,subup,\
202                            sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel],tem0);
203                          putimage(sublf,subup,tem1,COPY_PUT);
204                       }
205                       else beep();
206                     break;
207       case RIGHTKEY :if (bioskey(2) & 0x10)
208                       if (sublf+2*OFX+twl+MOV_SPEED<getmaxx())
209                       {  getimage(sublf,subup,\
210                            sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel],tem1);
211                          putimage(sublf,subup,tem0,COPY_PUT);
212                          sublf+=MOV_SPEED;
213                          getimage(sublf,subup,\
214                            sublf+2*OFX+twl,subup+(2*OFY+th)*sm[sel],tem0);
215                          putimage(sublf,subup,tem1,COPY_PUT);
216                       }
217                       else beep();
218                     break;
219       case CR     :
220       case ESC    :   break;
```

```
221      }
222    }
223    putimage(sublf,subup,tem0,COPY_PUT);
224    if (sub==CR) return(submen);
225    else return(-1);
226  }
227  quit()
228  { int i;
229     farfree(sbar); farfree(tem0); farfree(tem1);
230     for (i=0;i<=MAX_MENU;i++); farfree(menu[i].ptr);
231     if (havedata) free(picbuffer);
232     closegraph();
233  }
234
235  barmenu()   /* xmax=719  ymax=347 */
236  { int m,men=0;
237
238     putimage(menu[0].x,1,menu[0].ptr,XOR_PUT);
239     do
240     { m=getkey();
241       switch (m)
242       { case '4'       :
243         case LEFTKEY   : putimage(menu[men].x,1,menu[men].ptr,XOR_PUT);
244                          men=(--men<0) ? MAX_MENU : men;
245                          putimage(menu[men].x,1,menu[men].ptr,XOR_PUT);
246                          break;
247         case '6'       :
248         case RIGHTKEY  : putimage(menu[men].x,1,menu[men].ptr,XOR_PUT);
249                          men=(++men>MAX_MENU) ? 0 : men;
250                          putimage(menu[men].x,1,menu[men].ptr,XOR_PUT);
251                          break;
252         case DOWNKEY   :
253         case CR        : switch(men)
254                          { case 0 : file(subbar=menu(men));      break;
255                            case 1 : scan(subbar=menu(men));      break;
256                            case 2 : display(subbar=menu(men));   break;
257                            case 3 : position(0);                 break;
258                            case 4 : parameter(subbar=menu(men)); break;
259                            case 5 : edgeset(subbar=menu(men));   break;
260                            case 6 : option(subbar=menu(men));    break;
261                            case 7 : if (subbar=menu(men)==1) m=0;break;
262                          }
263                          break;
264         case ESC       : putimage(menu[men].x,1,menu[men].ptr,XOR_PUT);
265                          men=7;
266                          putimage(menu[men].x,1,menu[men].ptr,XOR_PUT);
267                          if (subbar=menu(men)==1) m=0; break;
268       }
269     }while (m!=0);
270
271     quit();
272  }
273
274
```

```
#include <graphics.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <dir.h>
#include <dos.h>
#include <stdlib.h>
#include "zspeckey.h"
#include "scanner.h"
#define MIN_DELAY 0.01
#define MAX_DELAY 100.0

file(int f)
{ void far *fi;
  char *str,ch;
  FILE *binfile;
   if (f==-1) return;
   fi=(void far *)farmalloc(imagesize(menu[0].x,20,menu[0].x+200,th+24));
   getimage(menu[0].x,20,menu[0].x+200,th+24,fi);
   putimage(menu[0].x,20,fi,XOR_PUT);
   rectangle(menu[0].x,20,menu[0].x+200,th+24);

   switch(f)
   { case 0 : str=(char *)readkey(menu[0].x+5,20,12);
              if (str==NULL) break;
              if ( strchr(str,'.')==NULL ) { strcat(str,".pic");}
              if (havedata)
              { free(picbuffer);
                havedata=0;
                loadfile[0]='\0';
              }
              setactivepage(1);
              clearviewport();
              setactivepage(0);
              if ( strpbrk(str,"?*")!=NULL )
              { if (dirmenu(str)==CR) strncpy(str,buff,12);
                else break;
              }
              binfile=fopen(str,"rb");
              if (binfile==NULL)
              { beep(); beep();
                setcolor(0); outtextxy(menu[0].x+5,20,str);
                setcolor(1); outtextxy(menu[0].x+5,20,"NOT EXIST");
                delay(1000);
                setcolor(0); outtextxy(menu[0].x+5,20,"NOT EXIST");
                break;
              }
              else strcpy(loadfile,str);

              if (havedata) free(picbuffer);
              picbuffer=(char *)malloc(sizeof(char)*0x4000);
              if (picbuffer==NULL)
              { beep(); beep();
                setcolor(0); outtextxy(menu[0].x+5,20,str);
```

```
56                setcolor(1); outtextxy(menu[0].x+5,20,"NOT ENOUGH MEMORY");
57                delay(1000);
58                setcolor(0); outtextxy(menu[0].x+5,20,"NOT ENOUGH MEMORY");
59                setcolor(1);
60              }
61              else
62              { fread((char *)picbuffer,sizeof(char),0x4000,binfile);
63                fclose(binfile);
64                havedata=1;
65                setactivepage(1);
66                clearviewport();
67                buffer2page1();
68                if (interactstat) status();
69              }
70              break;
71      case 1 :
72      case 2 : if (!havedata)
73              { beep(); beep();
74                setcolor(1); outtextxy(menu[0].x+5,20,"NO PICTURE TO SAVE");
75                delay(1000);
76                setcolor(0); outtextxy(menu[0].x+5,20,"NO PICTURE TO SAVE");
77                setcolor(1);
78                break;
79              }
80              str=(char *)readkey(menu[0].x+5,20,12);
81              if (str==NULL) break;
82              if ( strchr(str,'.')==NULL ) { strcat(str,".pic");}
83
84              binfile=fopen(str,"rb");
85              if (binfile!=NULL)
86              { beep();
87                fclose(binfile);
88                setcolor(0); outtextxy(menu[0].x+5,20,str);
89                setcolor(1); outtextxy(menu[0].x+5,20,"ALREADY EXIST");
90                delay(1000);
91                setcolor(0); outtextxy(menu[0].x+5,20,"ALREADY EXIST");
92                setcolor(1); outtextxy(menu[0].x+5,20,"OVERWRITE Y/N");
93                while ( strchr("YyNn",ch=getch())==NULL );
94                setcolor(0); outtextxy(menu[0].x+5,20,"OVERWRITE Y/N");
95                setcolor(1);
96                if ( strchr("Nn",ch)!=NULL )
97                {  break;
98                }
99              }
100             binfile=fopen(str,"w+b");
101             if (fwrite((char *)picbuffer,sizeof(char),0x4000,binfile)!=0x4000)
102             { beep(); beep();
103               setcolor(0); outtextxy(menu[0].x+5,20,str); setcolor(1);
104               outtextxy(menu[0].x+5,20,"DISK FULL"); delay(1000);
105               setcolor(0); outtextxy(menu[0].x+5,20,"DISK FULL");
106               setcolor(1);
107               unlink(str);
108             }
109             break;
110     case 3 : str=(char *)readkey(menu[0].x+5,20,20);
```

```
111                 if (str==NULL) break;
112                 setcolor(0); outtextxy(menu[0].x+5,20,str); setcolor(1);
113                 if (chdir(str)!=0) /* beware path sign must be double, "\\" */
114                 { outtextxy(menu[0].x+5,20,"INVALID PATH"); delay(1000);
115                   setcolor(0); outtextxy(menu[0].x+5,20,"INVALID PATH");
116                   setcolor(1);
117                 }
118                 break;
119     }
120     putimage(menu[0].x,20,fi,COPY_PUT);
121     farfree(fi);
122 }
123 scan(int s)
124 { void far *sca;
125   char *str;
126   double tem;
127    switch(s)
128    { case 0 : autoscan();
129               break;
130      case 1 : sca=(void far *)farmalloc(imagesize(menu[1].x,20,menu[1].x+120,th+24));
131               getimage(menu[1].x,20,menu[1].x+120,th+24,sca);
132               putimage(menu[1].x,20,sca,XOR_PUT);
133               rectangle(menu[1].x,20,menu[1].x+120,th+24);
134               outtextxy(menu[1].x+70,21,"Delay");
135               do
136               { str=(char *)readkey(menu[1].x+5,20,5);
137                 if (str==NULL) {putimage(menu[1].x,20,sca,COPY_PUT);break;}
138                 tem=atof(str);
139                 if ( (tem!=0)&&(tem>MIN_DELAY)&&(tem<MAX_DELAY) )
140                 {  scandelay=(float)tem;
141                   putimage(menu[1].x,20,sca,COPY_PUT);
142                    farfree(sca);
143                   if (interactstat) status();
144                   break;
145                 }
146                 else
147                 { setcolor(0); outtextxy(menu[1].x+5,20,str); setcolor(1);
148                   continue;
149                 }
150               } while (1);
151     }
152 }
153 display(int d)
154 { void far *di;
155   char pdensity,pmode,ch,ff,*str;
156   static char leftmargin=8;
157   static int lf=0,up=0,rg=719,dn=347;
158   int parea;
159    if (d==-1) return;
160    switch(d)
161    { case 0 : setvisualpage(1);
162               getch();
163               setvisualpage(0);
164               break;
165      case 1 : if (havedata) { setactivepage(1); clearviewport();
```

```
166                         buffer2page1();
167                       }
168             break;
169     case 2 : if (!havedata) break;
170             setactivepage(1);
171             clearviewport();
172             free(picbuffer);
173             havedata=0;
174             setactivepage(0);
175             loadfile[0]='\0';
176             if (interactstat) status();
177             break;
178     case 3 : if (havedata)
179             { di=(void far *)farmalloc(imagesize(menu[2].x,20,\
180                                        menu[2].x+200,th+24));
181               getimage(menu[2].x,20,menu[2].x+200,th+24,di);
182               putimage(menu[2].x,20,di,XOR_PUT);
183               rectangle(menu[2].x,20,menu[2].x+200,th+24);
184
185               setcolor(1);outtextxy(menu[2].x+5,20,"Normal/Reverse (N/R). N");
186               while ( strchr("NnRr ",ch=getch())==NULL );
187               setcolor(0);outtextxy(menu[2].x+5,20,"Normal/Reverse (N/R). N");
188               setcolor(1);
189               if ( (strchr("Nn",ch)!=NULL)||(ch==' ') ) pmode=0x00;
190               else pmode=0xFF;
191
192               setcolor(1);outtextxy(menu[2].x+5,20,"Density (0-3).  1");
193               while ( strchr("0123456 ",ch=getch())==NULL );
194               setcolor(0);outtextxy(menu[2].x+5,20,"Density (0-3).  1");
195               if (ch==' ') ch='1';
196               pdensity=ch;
197
198               setcolor(1);outtextxy(menu[2].x+5,20,"Fromfeed (Y/N).  N");
199               while ( strchr("YyNn ",ch=getch())==NULL );
200               setcolor(0);outtextxy(menu[2].x+5,20,"Fromfeed (Y/N).  N");
201               if (strchr("Nn ",ch)!=NULL) ff=0;
202               else ff=1;
203
204               setcolor(1);
205               gprintf(menu[2].x+5,20,"Left margin %d",leftmargin);
206               if ((ch=getch())!=13)&&(ch!=' '))
207               { setcolor(0);
208                 gprintf(menu[2].x+5,20,"Left margin %d",leftmargin);
209                 setcolor(1);
210                 ungetch(ch);
211                 do
212                 { str=(char *)readkey(menu[2].x+150,20,2);
213                   outtextxy(menu[2].x+150,20,str);
214                 } while ((atoi(str)>20)||(atoi(str)<0));
215                 leftmargin=atoi(str);
216               }
217               setcolor(0);
218               gprintf(menu[2].x+5,20,"Left margin %d",leftmargin);
219
220               setcolor(1);
```

```
221             outtextxy(menu[2].x+5,20,"Set print area (Y/N).  N");
222             while ( strchr("YyNn ",ch=getch())==NULL );
223             putimage(menu[2].x,20,di,COPY_PUT);
224             if ( strchr("Nn ",ch)==NULL)
225             { setactivepage(1); setvisualpage(1);
226               outtextxy(getmaxx()/3,getmaxy()-30,"  LEFT - UP");
227               setwritemode(1);
228               rectangle(lf,up,rg,dn);
229               while ( (parea=getkey()) != CR )
230               { switch(parea)
231                 { case '8'      :
232                   case UPKEY    : rectangle(lf,up,rg,dn);
233                                   if (up>0) up--;
234                                   rectangle(lf,up,rg,dn);
235                                   break;
236                   case '2'      :
237                   case DOWNKEY  : rectangle(lf,up,rg,dn);
238                                   if (up<dn) up++;
239                                   rectangle(lf,up,rg,dn);
240                                   break;
241                   case '4'      :
242                   case LEFTKEY  : rectangle(lf,up,rg,dn);
243                                   if (lf>0) lf--;
244                                   rectangle(lf,up,rg,dn);
245                                   break;
246                   case '6'      :
247                   case RIGHTKEY : rectangle(lf,up,rg,dn);
248                                   if (lf<rg) lf++;
249                                   rectangle(lf,up,rg,dn);
250                                   break;
251                 }
252               }
253             setwritemode(0);
254             setcolor(0);
255             outtextxy(getmaxx()/3,getmaxy()-30,"  LEFT - UP");
256             setcolor(1);
257             outtextxy(getmaxx()/3,getmaxy()-30,"  RIGHT - DOWN");
258             setwritemode(1);
259             while ( (parea=getkey())!=CR )
260             { switch(parea)
261               { case '8'      :
262                 case UPKEY    : rectangle(lf,up,rg,dn);
263                                 if (dn>up) dn--;
264                                 rectangle(lf,up,rg,dn);
265                                 break;
266                 case '2'      :
267                 case DOWNKEY  : rectangle(lf,up,rg,dn);
268                                 if (dn<getmaxy()) dn++;
269                                 rectangle(lf,up,rg,dn);
270                                 break;
271                 case '4'      :
272                 case LEFTKEY  : rectangle(lf,up,rg,dn);
273                                 if (rg>lf) rg--;
274                                 rectangle(lf,up,rg,dn);
275                                 break;
```

```
276                     case '6'      :
277                     case RIGHTKEY : rectangle(lf,up,rg,dn);
278                                     if (rg<getmaxx()) rg++;
279                                     rectangle(lf,up,rg,dn);
280                                     break;
281                   }
282                 }
283                 rectangle(lf,up,rg,dn);
284                 setwritemode(0);
285                 setcolor(0);
286                 outtextxy(getmaxx()/3,getmaxy()-30,"  RIGHT - DOWN");
287                 setcolor(1);
288               }
289               farfree(di);
290               print(pdensity,pmode,leftmargin,ff,lf,up,rg,dn);
291             }
292             break;
293     }
294  }
295  int position(int edg)
296  { int direct;
297    settextstyle(SMALL_FONT,HORIZ_DIR,5);
298    settextjustify(RIGHT_TEXT,BOTTOM_TEXT);
299    gprintf(PLATE_RG,PLATE_UP-3,"X:%d  Y:%d",posx,posy);
300    while ( (direct=getkey())!= ESC )
301    { setcolor(0); setbkcolor(1);
302      gprintf(PLATE_RG,PLATE_UP-3,"X:%d  Y:%d",posx,posy);
303      switch(direct)
304      { case UPKEY     : up(1);    break;
305        case '8'       : up(20);   break;
306        case DOWNKEY   : down(1);  break;
307        case '2'       : down(20); break;
308        case LEFTKEY   : left(1);  break;
309        case '4'       : left(20); break;
310        case RIGHTKEY  : right(1); break;
311        case '6'       : right(20);break;
312        case CR        : break;
313        default        : continue;
314      }
315      setcolor(1); setbkcolor(0);
316      gprintf(PLATE_RG,PLATE_UP-3,"X:%d  Y:%d",posx,posy);
317      if (direct==CR) break;
318    }
319    if (edg==0)
320    {  settextstyle(SANS_SERIF_FONT,HORIZ_DIR,USER_CHAR_SIZE);
321       setusercharsize(1,2,1,2);
322       settextjustify(LEFT_TEXT,TOP_TEXT);
323    }
324    beep();
325    return(direct);
326  }
327  void edgeset(int e)
328  { void far *edge;
329    int temp;
330    if (e==-1) return;
```

```
331     if ( position(1)==CR )
332     {  switch(e)
333        { case 0 : lfscan=posx; upscan=posy; break;
334          case 1 : rgscan=posx; dnscan=posy; break;
335        }
336        if ( (lfscan>rgscan)||(upscan>dnscan) )
337        {  outtextxy(PLATE_RG,PLATE_DN+25,"CORNER SWAPPING");
338           if (lfscan>rgscan) {temp=lfscan; lfscan=rgscan; rgscan=temp; }
339           if (upscan>dnscan) {temp=upscan; upscan=dnscan; dnscan=temp; }
340           delay(500); setcolor(0); setbkcolor(1);
341           outtextxy(PLATE_RG,PLATE_DN+25,"CORNER SWAPPING");
342           setcolor(1); setbkcolor(0);
343        }
344        if ( (rgscan-lfscan<MINSCAN)||(dnscan-upscan<MINSCAN) )
345        { outtextxy(PLATE_RG,PLATE_DN+25,"TOO SMALL SCAN SIZE");
346          delay(1000); setcolor(0); setbkcolor(1);
347          outtextxy(PLATE_RG,PLATE_DN+25,"TOO SMALL SCAN SIZE");
348          setcolor(1); setbkcolor(0);
349          outtextxy(PLATE_RG,PLATE_DN+25,"SCAN SIZE SET TO MAXIMUM");
350          delay(1000); setcolor(0); setbkcolor(1);
351          outtextxy(PLATE_RG,PLATE_DN+25,"SCAN SIZE SET TO MAXIMUM");
352          setcolor(1); setbkcolor(0);
353          lfscan=upscan=0; rgscan=MAX_X; dnscan=MAX_Y;
354        }
355        putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
356        setviewport(PLATE_LF+1,PLATE_UP+1,PLATE_RG-1,PLATE_DN-1,0);
357        clearviewport();
358        setviewport(0,0,getmaxx(),getmaxy(),1);
359        rectangle(PLATE_LF+lfscan*KX,PLATE_UP+upscan*KY,\
360                  PLATE_LF+rgscan*KX,PLATE_UP+dnscan*KY);
361        floodfill(PLATE_LF+lfscan*KX+1,PLATE_UP+upscan*KY+1,1);
362        putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
363     }
364     settextstyle(SANS_SERIF_FONT,HORIZ_DIR,USER_CHAR_SIZE);
365     setusercharsize(1,2,1,2);
366     settextjustify(LEFT_TEXT,TOP_TEXT);
367     if (interactstat) status();
368  }
369  option(int o)
370  { char *str;
371    FILE *configfi;
372    void far *opt;
373      switch(o)
374      { case 0 : interactstat ^= 1;
375                 break;
376        case 1 : opt=(void far *)farmalloc(imagesize(menu[6].x,20,menu[6].x+140,th+24));
377                 if (opt==NULL) break;
378                 getimage(menu[6].x,20,menu[6].x+140,th+24,opt);
379                 putimage(menu[6].x,20,opt,XOR_PUT);
380                 rectangle(menu[6].x,20,menu[6].x+140,th+24);
381                 outtextxy(menu[6].x+2,22,"Load");
382                 str=(char *)readkey(menu[6].x+5+textwidth("Load"),22,12);
383                 if (str==NULL) { putimage(menu[6].x,20,opt,COPY_PUT); break;}
384                 if ((configfi=fopen(str,"rt"))==NULL) beep();
385                 else
```

```
386                 { strcpy(configfile,str);
387                   fscanf(configfi,"%6u%6u%6u%6d%6d%6d%6d%6u%6d%4.2f",\
388                        &Hz,&Step,&Scanrate,&lfscan,&upscan,&rgscan,&dnscan,\
389                        &bit,&ystep,&scandelay);
390                   fclose(configfi);
391
392                   putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
393                   setviewport(PLATE_LF+1,PLATE_UP+1,PLATE_RG-1,PLATE_DN-1,0);
394                   clearviewport();
395                   setviewport(0,0,getmaxx(),getmaxy(),1);
396                   rectangle(PLATE_LF+lfscan*KX,PLATE_UP+upscan*KY,\
397                            PLATE_LF+rgscan*KX,PLATE_UP+dnscan*KY);
398                   floodfill(PLATE_LF+lfscan*KX+1,PLATE_UP+upscan*KY+1,1);
399                   putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
400
401                   interactstat=1;
402                 }
403                 putimage(menu[6].x,20,opt,COPY_PUT);
404                 farfree(opt);
405                 break;
406         case 2 : opt=(void far *)farmalloc(imagesize(menu[6].x,20,menu[6].x+140,th+24));
407                 if (opt==NULL) break;
408                 getimage(menu[6].x,20,menu[6].x+140,th+24,opt);
409                 putimage(menu[6].x,20,opt,XOR_PUT);
410                 rectangle(menu[6].x,20,menu[6].x+140,th+24);
411                 outtextxy(menu[6].x+2,22,"Save");
412                 str=(char *)readkey(menu[6].x+5+textwidth("Save"),22,12);
413                 if (str==NULL) { putimage(menu[6].x,20,opt,COPY_PUT); break;}
414                 if ((configfi=fopen(str,"wt"))==NULL) beep();
415                 else
416                 { strcpy(configfile,str);
417                   fprintf(configfi,"%6u%6u%6u%6d%6d%6d%6d%6u%6d%4.2f",\
418                        Hz,Step,Scanrate,lfscan,upscan,rgscan,dnscan,\
419                        bit,ystep,scandelay);
420                   fclose(configfi);
421                   interactstat=1;
422                 }
423                 putimage(menu[6].x,20,opt,COPY_PUT);
424                 farfree(opt);
425                 break;
426         default : return(-1);
427
428       }
429       status();
430   }
431   void parameter(int p)
432   { void far *para;
433     unsigned int tem;
434     char *str;
435       if (p==-1) return;
436       para=(void far *)farmalloc(imagesize(menu[4].x,20,menu[4].x+120,th+24));
437       if ((p<=2)||(p==8))
438       { getimage(menu[4].x,20,menu[4].x+120,th+24,para);
439         putimage(menu[4].x,20,para,XOR_PUT);
440         rectangle(menu[4].x,20,menu[4].x+120,th+24);
```

```
441     }
442     switch(p)
443     { case 0 : outtextxy(menu[4].x+70,22,"Hertz");
444                do
445                { str=(char *)readkey(menu[4].x+5,20,5);
446                  if (str==NULL) break;
447                  tem=atoi(str);
448                  if ( (tem!=0)&&(tem>MIN_HZ)&&(tem<MAX_HZ) )
449                  {  Hz=tem;
450                     tem=CLK/(4.0*(float)Hz);
451                     outportb(CNT,0x36);
452                     outportb(CNT0,0x00ff & tem);
453                     outportb(CNT0,tem>>8);
454                     if (interactstat) status();
455                     break;
456                  }
457                  else
458                  { setcolor(0); outtextxy(menu[4].x+5,20,str); setcolor(1);
459                    continue;
460                  }
461                } while (1);
462                break;
463      case 1 : outtextxy(menu[4].x+70,20," Step");
464               do
465               { str=(char *)readkey(menu[4].x+5,20,5);
466                 if (str==NULL) break;
467                 tem=atoi(str);
468                 if ( (tem>0)&&(tem<MAX_STEP) )
469                 { Step=tem;
470                   outportb(CNT,0x72);
471                   outportb(CNT1,0x00ff & tem);
472                   outportb(CNT1,tem>>8);
473                   break;
474                 }
475                 else
476                 { setcolor(0); outtextxy(menu[4].x+5,20,str); setcolor(1);
477                   continue;
478                 }
479               } while (1);
480               break;
481     case 2 : outtextxy(menu[4].x+70,20," st/sc");
482              do
483              { str=(char *)readkey(menu[4].x+5,20,5);
484                if (str==NULL) break;
485                tem=atoi(str);
486                if (tem>=1)
487                { Scanrate=tem;
488                  outportb(CNT,0xb6);
489                  outportb(CNT2,0x00ff & tem);
490                  outportb(CNT2,tem>>8);
491                  if (interactstat) status();
492                  break;
493                }
494              else
495              { setcolor(0); outtextxy(menu[4].x+5,20,str); setcolor(1);
```

```
496                   continue;
497                  }
498                } while (1);
499                break;
500       case 3 : bit ^= 0x0001;
501                outportb(M_CTRL,bit);
502                if (interactstat) status();
503                break;
504       case 4 : bit &= 0x00fd;
505                outportb(M_CTRL,bit);
506                bit |= 0x0002;
507                outportb(M_CTRL,bit);
508                break;
509       case 5 : bit ^= 0x0004;
510                outportb(M_CTRL,bit);
511                if (interactstat) status();
512                break;
513       case 6 : bit ^= 0x0008;
514                outportb(M_CTRL,bit);
515                if (interactstat) status();
516                break;
517       case 7 : bit ^= 0x0010;
518                outportb(M_CTRL,bit);
519                if (interactstat) status();
520                break;
521       case 8 : outtextxy(menu[4].x+70,20,"YStep");
522                do
523                { str=(char *)readkey(menu[4].x+5,20,5);
524                  if (str==NULL) break;
525                  tem=atoi(str);
526                  if ( (tem>0)&&(tem<MAX_Y_STEP) )
527                  { ystep=tem;
528                    if (interactstat) status();
529                    break;
530                  }
531                  else
532                  { setcolor(0); outtextxy(menu[4].x+5,20,str); setcolor(1);
533                    continue;
534                  }
535                } while (1);
536                break;
537     }
538     putimage(menu[4].x,20,para,COPY_PUT);
539     farfree(para);
540   }
```

```
#include <graphics.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>
#include <bios.h>
#include <alloc.h>
#include <dos.h>
#include <dir.h>
#include <ctype.h>
#include "zspeckey.h"
#include "scanner.h"

  /*Motor definitions*/
#define LF     10
#define STOP   0
#define START  1
#define CW     0
#define CCW    1
#define UP     CCW
#define DOWN   CW
#define LEFT   CCW
#define RIGHT  CW
#define HOR    0
#define VER    1
#define RPULLYX 0.15748
#define RPULLYY 0.15748

  /*Status definitions*/
#define S_LF   480
#define S_LFT  485
#define S_UP   70
#define S_RG   680
#define S_DN   250

  /*directory definitions*/
#define D_LF 10
#define D_UP 150-dirnums        /*approx: maxx/2-((dirnums*13)/6+60)/2*/
#define D_RG 695
#define D_DN (dirnums*13)/6+23+D_UP

  static char dirarray[112][9],dirext[112][4],wildcard[12];

char *readkey(int x,int y,int digit)
{ char ch,gr[2];
  int i=0;
  gr[1]='\0';
  while( ((ch=getch())!=13)&&(i<digit) )
  { if (ch==27) return(0);
    if (ch!=8)
    {  buff[i++]=ch;
       sprintf(gr,"%c",ch);
       outtextxy(x,y,gr); x += textwidth(gr);
    }
    else
    {  if (i>0)
```

```
56-         { --i;
57-           gr[0]=buff[i];
58            x = x-textwidth(gr);
59            setcolor(0);
60            outtextxy(x,y,gr);
61            setcolor(1);
62          }
63       }
64       buff[i]='\0';
65     }
66     return(buff);
67   }
68   int getkey(void)
69   {int key, lo, hi;
70    key = bioskey(0);
71    lo = key & 0X00FF;
72    hi = (key & 0XFF00) >> 8;
73    return((lo == 0) ? hi + 256 : lo);
74   }
75
76   int gprintf(int xloc,int yloc,char *fmt,...)
77   { va_list  argptr;
78     char str[140];
79       va_start(argptr,format );
80         vsprintf(str,fmt,argptr);
81         outtextxy(xloc,yloc,str);
82       va_end(argptr);
83   }
84   beep()
85   { sound(1500);delay(40);sound(1000);delay(20);nosound();
86   }
87
88   void motor(int m)
89   { if (m==HOR) { bit &= 0xf7; outportb(M_CTRL,bit); }
90     else        { bit |= 0x08; outportb(M_CTRL,bit); }
91   }
92   void direction(int d)
93   { if (d==CW)  { bit &= 0xef; outportb(M_CTRL,bit); }
94     else        { bit |= 0x10; outportb(M_CTRL,bit); }
95   }
96   void steptomove(unsigned s)
97   { if ( (s>0)&&(s<MAX_STEP) )
98     { outportb(CNT,0x72);
99       outportb(CNT1,0x00ff & s);
100      outportb(CNT1,s>>8);
101    }
102  }
103  void move(unsigned s)  /*send rising pulse to counter 1*/
104  { if (s==START)
105    { bit |= 0x0001;
106      outportb(M_CTRL,bit & 0xfd);
107      outportb(M_CTRL,bit | 0x02);
108    }
109    else { bit &= 0xfe; outportb(M_CTRL,bit); }
110  }
```

```
111  void up(int decr)
112  { if (posy-decr<0) beep();
113    else
114    { motor(VER); direction(UP);
115      steptomove(decr);
116      move(START);
117      putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
118      posy -= decr;
119      putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
120    }
121  }
122  void down(int incr)
123  { if (posy+incr>MAX_Y) beep();
124    else
125    { motor(VER); direction(DOWN);
126      steptomove(incr);
127      move(START);
128      putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
129      posy += incr;
130      putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
131    }
132  }
133  void left(int decr)
134  { if (posx-decr<0) beep();
135    else
136    { motor(HOR); direction(LEFT);
137      steptomove(decr);
138      move(START);
139      putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
140      posx -= decr;
141      putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
142    }
143  }
144  void right(int incr)
145  { if (posx+incr>MAX_X) beep();
146    else
147    { motor(HOR); direction(RIGHT);
148      steptomove(incr);
149      move(START);
150      putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
151      posx += incr;
152      putimage(PLATE_LF+posx*KX,PLATE_UP+posy*KY,arrow,XOR_PUT);
153    }
154  }
155  autoscan()
156  { unsigned int tem,ysize;
157    unsigned msecx,msecy;
158
159        outportb(M_CTRL,0);
160
161     /*set 3 counter*/
162        tem=CLK/(4.0*(float)Hz);
163        outportb(CNT,0x36);
164        outportb(CNT0,0x00ff & tem);
165        outportb(CNT0,tem>>8);
```

```
166
167         steptomove(rgscan-lfscan);
168
169         tem=Scanrate;
170         outportb(CNT,0xb6);
171         outportb(CNT2,0x00ff & tem);
172         outportb(CNT2,tem>>8);
173
174         /*set RAM address counter to 0*/
175
176         ysize=0;
177         msecx=((float)(rgscan-lfscan)*1000.0*scandelay)/Hz;
178         msecy=(ystep*1000.0*scandelay)/Hz;
179         move(START);
180         while ((ysize<dnscan-upscan)&&(!kbhit()))
181         {  delay(msecx);
182           motor(VER); direction(DOWN); steptomove(ystep); move(START);
183            delay(msecy);
184           ysize += ystep;
185           motor(HOR); direction(LEFT);  steptomove(rgscan-lfscan); move(START);
186            delay(msecx);
187           motor(VER); direction(DOWN);  steptomove(ystep);        move(START);
188            delay(msecy);
189           ysize += ystep;
190           motor(HOR); direction(RIGHT); steptomove(rgscan-lfscan); move(START);
191         }
192         beep();
193 }
194 directory(char *wild)
195 { char *dirbuff,*ptr;
196   int i,j,k;
197     i=j=k=0;
198     while (i<8)
199     { if ( isalnum(*(wild+j)) || (*(wild+j)=='?') )
200       { wildcard[i++]=toupper(*(wild+j++)); }
201       else if ( *(wild+j)=='*' ) { while (i<8) wildcard[i++]='?'; ++j; }
202            else while (i<8) wildcard[i++]=' ';
203     }
204     ++j;
205     while (i<11)
206     { if ( isalnum(*(wild+j)) || (*(wild+j)=='?') )
207          wildcard[i++]=toupper(*(wild+j++));
208       else if ( *(wild+j)=='*' ) { while (i<11) wildcard[i++]='?'; ++j; }
209            else while (i<11) wildcard[i++]=' ';
210     }
211     dirbuff=(char *)malloc(sizeof(char)*0x200*7);
212     if (dirbuff==NULL) { printf("\7"); return(-1); }
213     if (absread(getdisk(),7,5,dirbuff)!=0) { printf("\7"); return(-2); }
214     ptr=dirbuff;
215
216     for(i=0;i<112;i++)
217     { if ( isprint(*ptr)&&(*(ptr+11)==0x20) ) /*0x20 is archive attribute*/
218       {  for (j=0;j<11;j++)
219          { if ( (wildcard[j]=='?') || (wildcard[j]==*(ptr+j)) ) continue;
220            else break;
```

```
221         }
222         if (j==11)
223         { strncpy(dirarray[k],ptr,8);
224           strncpy(dirext[k++],(ptr+8),3);
225         }
226       }
227       ptr += 32;
228     }
229     free(dirbuff);
230     return(k);
231 }
232 dirmenu(char *wild)
233 {
234   struct textsettingstype oldtext;
235   int dirnums,i,j,k,l,dir;
236   void far *dir0,far *dir1,far *bar;
237
238    dirnums=directory(wild);
239    gettextsettings(&oldtext);
240    settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
241    dir0=(void far *)farmalloc(imagesize(D_LF,D_UP,D_RG,D_DN));
242    dir1=(void far *)farmalloc(imagesize(D_LF,D_UP,D_RG,D_DN));
243    bar =(void far *)farmalloc(imagesize(0,0,108,13));
244    if((dir0==NULL)||(dir1==NULL)||(bar==NULL)) exit(3);
245
246    setactivepage(1);
247
248    rectangle(0,0,108,13); floodfill(1,1,1); getimage(0,0,108,13,bar);
249
250    for (i=0;i<=dirnums/6;i++)
251    { for (j=0;j<6;j++)
252      { if (j+i*6<dirnums)
253        { moveto(D_LF+10+113*j,D_UP+10+13*i);
254          outtext(dirarray[j+i*6]); outtext("."); outtext(dirext[j+i*6]);
255        }
256      }
257    }
258    rectangle(D_LF+4,D_UP+4,D_RG-4,D_DN-4);
259    getimage(D_LF,D_UP,D_RG,D_DN,dir1);
260    putimage(D_LF,D_UP,dir1,NOT_PUT);
261    getimage(D_LF,D_UP,D_RG,D_DN,dir1);
262    setactivepage(0);
263    getimage(D_LF,D_UP,D_RG,D_DN,dir0);
264    putimage(D_LF,D_UP,dir1,COPY_PUT);
265    i=j=dir=0;
266    putimage(D_LF+6+113*j,D_UP+6+13*i,bar,XOR_PUT);
267    while ( (dir!=ESC)&&(dir!=CR) )
268    { dir=getkey();
269      switch(dir)
270      { case '6'      :
271        case RIGHTKEY : putimage(D_LF+6+113*j,D_UP+6+13*i,bar,XOR_PUT);
272                        if ((++j)>5)||(j+6*i)>=dirnums))
273                        { j=0;  if (++i)>dirnums/6) i=0; }
274                        break;
275        case '4'      :
```

```
276         case LEFTKEY  : putimage(D_LF+6+113*j,D_UP+6+13*i,bar,XOR_PUT);
277                         if (--j<0) { j=5;  if (--i<0)
278                                              { i=dirnums/6; j=(dirnums%6)-1;}
279                                    }
280                         break;
281         case '8'      :
282         case UPKEY    : putimage(D_LF+6+113*j,D_UP+6+13*i,bar,XOR_PUT);
283                         if (--i<0)
284                           i=(j+6*(dirnums/6)>=dirnums) ? dirnums/6-1:dirnums/6;
285                         break;
286         case '2'      :
287         case DOWNKEY  : putimage(D_LF+6+113*j,D_UP+6+13*i,bar,XOR_PUT);
288                         if ((++i>dirnums/6)||(j+6*i)>=dirnums)) i=0;
289                         break;
290         case CR  :
291         case ESC : break;
292         default  : printf("\7"); continue;
293       }
294       putimage(D_LF+6+113*j,D_UP+6+13*i,bar,XOR_PUT);
295
296     }
297     settextstyle(oldtext.font,oldtext.direction,oldtext.charsize);
298     if (dir==CR)
299     { k=0; while ((dirarray[j+6*i][k]!=' ')&&k<8)
300           { buff[k]=dirarray[j+6*i][k++];
301           }
302           buff[k++]='.';
303       l=0; while ((dirext[j+6*i][l]!=' ')&&l<3)
304           { buff[k++]=dirext[j+6*i][l++];
305           }
306       buff[k]='\0';
307     }
308     putimage(D_LF,D_UP,dir0,COPY_PUT);
309     farfree(dir0); farfree(dir1); farfree(bar);
310     return(dir);
311 }
312 buffer2page1()
313 { unsigned CGArow,CGAcol,HERCscr,CGAscr;
314   char far *scr;
315    scr=(char far *)MK_FP(0xB800,0);
316    setactivepage(1); setvisualpage(1);
317    for (CGArow=0;CGArow<100;CGArow++)
318    { for (CGAcol=0;CGAcol<80;CGAcol++)
319      {  HERCscr = (CGArow%2)*0x4000 + CGAcol + (CGArow/2)*90;
320         CGAscr  = CGArow*80 + CGAcol;
321         *(scr+HERCscr) = *(picbuffer+CGAscr);
322         *(scr+HERCscr+0x2000) = *(picbuffer+CGAscr+0x2000);
323      }
324    }
325    setvisualpage(1);
326    beep();
327    getch();
328    setactivepage(0); setvisualpage(0);
329 }
330 void printonebyte(unsigned char byte)
```

```
331 {   _AH = 0;
332     _AL = byte;
333     _DX = 0;
334     geninterrupt(0x17);
335 }
336 print(pdensity,pmode,leftmargin,ff,lf,up,rg,dn)
337 char pdensity,pmode,leftmargin,ff;
338 int lf,up,rg,dn;
339 { int scanline,xypixel,xy;
340   unsigned char printbyte,bits[]={128,64,32,16,8,4,2,1},bit,n1,n2;
341   char ch;
342
343   setactivepage(1); setvisualpage(1);
344   fprintf(stdprn,"%c%c%c%c%c%c",ESC,'3',24,ESC,'l',leftmargin);
345   n1 = (rg-lf)%256 ;
346   n2 = (rg-lf)/256 ;
347   for (scanline=up;scanline<=(dn/8);scanline++)
348   { fprintf(stdprn,"%c%c%c%c%c",ESC,'*',pdensity,n1,n2);
349     for (xypixel=lf;xypixel<=rg;xypixel++)
350     { xy = scanline * 8;
351       printbyte = 0;
352       for (bit=0;bit<=7;bit++)
353       { if (getpixel(xypixel,xy+bit)!=0) printbyte += bits[bit];
354       }
355       printonebyte(printbyte^pmode);
356     }
357     printonebyte(10); /*LF*/
358     if (kbhit())
359     {setcolor(1);outtextxy(getmaxx()/3,getmaxy()-30,"CANCEL PRINTING ? (Y/N).");
360      while ( strchr("YyNn",ch=getch())==NULL );
361      setcolor(0);outtextxy(getmaxx()/3,getmaxy()-30,"CANCEL PRINTING ? (Y/N).");
362      setcolor(1);
363      if ( strchr("Yy",ch)!=NULL )
364      {  break;
365      }
366     }
367   }
368   fprintf(stdprn,"%c%c%c%c%c%c%c",ESC,2,ESC,'l',0,ESC,'<');
369   if (ff) printonebyte(0x0C);
370
371   setactivepage(0); setvisualpage(0);
372 }
373 static char *hex2bin(unsigned bit)
374 { static unsigned char test,bin[9];
375   char i;
376    test=1;
377    for (i=7;i>=0;i--)
378    { bin[i]=(((unsigned char)bit & test)>0)+48;
379      test <<=1;
380    }
381    return(bin);
382 }
383 status()
384 { void far *s;
385    s=(void far *)farmalloc(imagesize(S_LF,S_UP, S_RG+7,S_DN+7));
```

```
386     getimage(S_LF,S_UP, S_RG+7,S_DN+7,s);
387     putimage(S_LF,S_UP,s,XOR_PUT);
388     rectangle(S_LF,S_UP, S_RG,S_DN);
389     moveto(S_LF+7,S_DN+2);
390     lineto(S_LF+7,S_DN+7);
391     lineto(S_RG+7,S_DN+7);
392     lineto(S_RG+7,S_UP+7);
393     lineto(S_RG+2,S_UP+7);
394     lineto(S_RG+2,S_DN+2);
395     lineto(S_LF+7,S_DN+2);
396     floodfill(S_LF+8,S_DN+3,1);
397
398     if (interactstat) gprintf(S_LFT,S_UP+5,"    Active Status");
399     else gprintf(S_LFT,S_UP+5,"    De-Active Status");
400     gprintf(S_LFT,S_UP+5+th+5,"Scan speed   %4d  Hz",Hz);
401     gprintf(S_LFT,S_UP+5+2*(th+5),\
402       "Scan Size %dx%d dot",rgscan-lfscan,dnscan-upscan);
403     gprintf(S_LFT,S_UP+5+3*(th+5),\
404       "Scan Delay Factor %4.2f",scandelay);
405     gprintf(S_LFT,S_UP+5+4*(th+5),\
406       "X Resolution   %3d DPI",(int)(1.0/(0.031416*RPULLYX*(float)Scanrate)));
407     gprintf(S_LFT,S_UP+5+5*(th+5),\
408       "Y Resolution   %3d DPI",(int)(1.0/(0.031416*RPULLYY*(float)ystep)));
409     gprintf(S_LFT,S_UP+5+6*(th+5),"Controlbit    %s",hex2bin(bit));
410     gprintf(S_LFT,S_UP+5+7*(th+5),"Load File   %s",loadfile);
411     gprintf(S_LFT,S_UP+5+8*(th+5),"Config File %s",configfile);
412     farfree(s);
413 }
414
415
```

กิตติกรรมประกาศ

ขอขอบคุณอาจารย์ รัตติกร วรากูลศิริพันธุ์ และ นักศึกษาปริญญาโท เป็นอย่างสูงที่ช่วยให้คำปรึกษา จนโครงงานนี้เสร็จลุล่วงไปด้วยดี

บรรณานุกรม

1. Intel Corporation, Microsystem Component Handbook Volume II

2. International Business Machines, IBM TECHNICAL REFERENCE

3. James W. Coffron , Z80 Applications , Sybex Inc. , 1983

4. Kenjo, Takash, Stepping motors and their microprocessor controls, clarendon press , oxford 1984