โปรแกรมจำลองการทำงานของเครื่องควบคุมพีแอลซี

PLCs SIMULATION SOFTWARE

โดย

นาย จักรพงษ์ อธิมุตติกุล

นาย ธาดา สราวัน

นาย อัครเดช ศรียะ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมจำลองการทำงานของเครื่องควบคุมพีแอลซี

PLCs SIMULATION SOFTWARE

ผู้จัดทำ

1. นายจักรพงษ์	อธิมุตติกุล	40013400
2. นายธาดา	สราวัน	40013408
3. นายอัครเดช	ศรียะ	40013439

อาจารย์ที่ปรึกษา

.....

(อาจารย์ ทวีพล ซื่อสัตย์)

โปรแกรมจำลองการทำงานของเครื่องควบคุมพีแอลซี

โดย 1. นาย จักรพงษ์ อธิมุตติกุล 40013400
2. นาย ธาดา สราวัน 40013408
3. นาย อัครเดช ศรียะ 40013439

อาจารย์ที่ปรึกษา อาจารย์ ทวีพล ซื่อสัตย์

บทคัดย่อ

ในปัจจุบันเครื่องควบคุมแบบตรรกที่โปรแกรมได้หรือพีแอลซี ถูกใช้งานในระบบควบคุมอัตโนมัติอย่างแพร่หลาย เนื่องจากมีความเน่าเชื่อถือสูงและใช้งานได้อย่างสะดวก ปริญญานิพนธ์ที่นำเสนอโปรแกรมจำลองการทำงานของเครื่องควบคุมพีแอลซี ซึ่งสามารถใช้งานได้ในคอมพิวเตอร์ที่ทำงานภายใต้ระบบปฏิบัติการวินโดว์ 95 หรือ 98 โปรแกรมที่ใช้ไมโครซอร์ฟ วิชวลเบสิก 6 ในการพัฒนาโปรแกรม

โครงงานนี้มีความสามารถในการจำลองการทำงานพื้นฐานของ พีแอลซี เช่นการควบคุมแบบเปิด-ปิด แสดงผลในส่วนแลดเดอร์ไดอะแกรมและภาษาบูลีน ซึ่งจะเป็นแนวทางในการพัฒนาโปรแกรมให้มีความสามารถเทียบเท่ากับการทดลองบนเครื่องควบคุมจริง

PLCs SIMULATION SOFTWARE

STAFF	Mr. JAGKRAPONG ATIMOOTTIKUL	40013400
	Mr. THADA SARAWAN	40013408
	Mr. AKKARADET SRIYA	40013439

ADVISOR TAWEEPOL SUESUT

ABSTRACT

Present , The programmable Logic Controllers (PLCs) are widely used in the industrial automatic control system , because of these controllers are high reliability and convenience to use . This thesis presents the PLCs simulation software which works under the Microsoft Windows 95 or 98 operating system. This program is developed by Microsoft visual Basic 6.0

This project can be simulate the basic operation of the PLCs such as ON-OFF control , display the ladder diagram and boolean instruction , that is the guides for developing the program to be high ability same as the experiment on the real PLCs

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ในที่สุดก็สามารถลุล่วงไปได้ด้วยดี และทุกอย่างคงจะสำเร็จไปไม่ได้ถ้าปราศจากบุคคลหลายๆท่านซึ่งมีพระคุณอย่างสูง และอีกหลายคนที่คอยให้กำลังใจ คอยช่วยเหลือทุกครั้งที่มีปัญหาเกิดขึ้น ทั้งที่เกี่ยวกับโครงงาน เกี่ยวกับปริญญานิพนธ์ และเรื่องที่เกี่ยวกับผู้จัดทำเอง จึงทำให้อุปสรรคเหล่านั้นผ่านพ้นไปได้ ซึ่งผู้จัดทำขอขอบพระคุณมา ณ ที่นี้

ขอขอบพระคุณอาจารย์ ทวีพล ซื่อสัตย์ อาจารย์ที่ปรึกษาที่คอยกระตุ้นและชี้แนะแนวทางที่เป็นประโยชน์อย่างยิ่ง อีกทั้งยังคอยกำลังใจโดยไม่เคยต่อว่าให้หมดกำลังใจแม้แต่ครั้งเดียว ซึ่งผู้จัดทำไม่ทราบว่าจะนำคำใดมาขอบพระคุณอาจารย์ท่านนี้จึงจะเพียงพอ

ขอขอบพระคุณอาจารย์ทุกท่านในภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรมทุกท่านที่ประสิทธิ์ประสาทความรู้และคำปรึกษาอันมีคุณค่ายิ่งสำหรับผู้จัดทำ

ขอขอบคุณ น.ส. รุ่งอรุณ ศรีปาน ที่ช่วยเหลือในการทำปริญญานิพนธ์จนสำเร็จไปในเวลาอันรวดเร็วกว่าที่ควรจะเป็น

ขอขอบคุณพี่ๆ เพื่อนๆ น้องๆ ทุกคนที่มีส่วนร่วมทั้งกำลังกายและกำลังใจในการทำปริญญานิพนธ์จนสำเร็จลุล่วง

เหนืออื่นใดขอขอบพระคุณ คุณพ่อ คุณแม่ ที่ให้กำเนิดและคอยเป็นห่วงมาจนถึงทุกวันนี้

นาย จักรพงษ์ อธิมุตติกุล
นาย ธาดา สราวัน
นาย อัครเดช ศรียะ

สารบัญ

	หน้า
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญภาพ	VI
สารบัญตาราง	VIII
บทที่ 1 บทนำ	1
1.1 วัตถุประสงค์	1
1.2 ขอบเขตโครงงาน	1
บทที่ 2 ทฤษฎีและหลักการที่เกี่ยวข้อง	2
2.1 ทฤษฎีและหลักการพื้นฐานของ วิชวล เบสิค(VISUAL BASIC)	2
2.1.1 ที่มาและวิวัฒนาการของ วิชวล เบสิค	2
2.1.2 Event-Driven Programming กับ วิชวลเบสิค	3
2.1.3 Procedure และ Function ใน วิชวลเบสิค	3
2.1.4 การใช้ส่วนแก้ไข(Editor) ของวิชวลเบสิค	4
2.1.5 ประเภทของข้อมูล(Data Type)	6
2.1.6 กฎในการตั้งชื่อตัวแปร	8
2.1.7 การประกาศค่าตัวแปร	8
2.1.8 ปฏิบัติการ(Operate)	9
2.1.9 ส่วนควบคุมของโปรแกรม(Procss Control)	12
2.1.10 กลุ่มคำสั่งตัดสินใจ	15
2.1.11 ตัวแปรที่กำหนดขึ้นเอง	15
2.1.12 ขอบเขตการใช้งานของตัวแปร	16
2.1.13 อะเรย์(Array)	17
2.1.14 การจัดการและตรวจสอบข้อผิดพลาด	18
2.2 ทฤษฎีและหลักการพื้นฐานของเครื่องควบคุมแบบตรรกที่โปรแกรมได้	19
2.2.1 ประวัติของเครื่องควบคุมแบบตรรกที่โปรแกรมได้	19
2.2.2 ที่มาและวิวัฒนาการของเครื่องควบคุมแบบตรรกที่โปรแกรมได้	20

สารบัญ

		หน้า
	2.2.3 โครงสร้างและการทำงานของเครื่องควบคุมแบบตรรกโปรแกรมได้	22
	2.2.4 ลักษณะการโปรแกรมให้กับเครื่องควบคุมแบบตรรกโปรแกรมได้	26
	2.2.5 ตัวอย่างการประยุกต์คำสั่ง	28
	2.2.6 การทำงานของพีซี	29
บทที่ 3	การสร้างและการออกแบบ	31
	3.1 ส่วนโอเปอร์เรท(OPERATE)	31
	3.2 ส่วนแลดเดอร์ ไดอะแกรม(LADDER DIAGRAM)	32
	3.3 ส่วนภาษาบูลีน(BOOLEAN)	42
บทที่ 4	การทดลองและการใช้งาน	44
	4.1 หน้าต่างหลัก	44
	4.2 ส่วนโอเปอร์เรท(OPERATE)	44
	4.3 ฟอร์มแลดเดอร์ ไดอะแกรม(LADDER DIAGRAM)	47
	4.4 ส่วน ภาษาบูลีน(BOOLEAN)	49
	4.5 ส่วนจำลองการทำงาน(SIMULATE)	50
	4.6 ตัวอย่างการใช้งานคำสั่ง AND_LD	51
	4.7 ตัวอย่างการใช้งานคำสั่ง OR_LD	52
	4.8 ตัวอย่างการใช้งานคำสั่ง TIMER	54
	4.9 ตัวอย่างการใช้งานคำสั่ง COUNTER	55
บทที่ 5	สรุปผลการทดลองและข้อเสนอแนะ	57
	5.1 ปัญหาที่พบ	57
	5.2 วิธีการแก้ไข	57
	5.3 สรุปผลการทดลอง	57
บรรณานุกรม		58
ภาคผนวก		59
	ภาคผนวก ก	60
	คู่มือการใช้งาน พีแอลซี รุ่น C200HS	61
	ภาคผนวก ข	75
	กราฟแสดงการประมวลผลของโปรแกรม	76

สารบัญภาพ(ต่อ)

หน้า

รูปที่3.18 ตัวอย่างภาษาบูลีน 42

รูปที่4.1 หน้าต่างหลัก 44

รูปที่4.2 INPUT BOX รับค่า DATA ของ TIMER 45

รูปที่4.3 INPUT BOX รับค่าเวลาของ TIMER 45

รูปที่4.4 INPUT BOX รับค่า DATA ของ COUNTER 46

รูปที่4.5 INPUT BOX รับจำนวน PULSE ของ COUNTER 46

รูปที่4.6 หน้าต่างโอเปอร์เรท(OPERATE) 47

รูปที่4.7 แสดงฟอร์มแลดเดอร์ไดอะแกรม 48

รูปที่4.8 แสดงส่วนภาษาบูลีน 49

รูปที่4.9 แสดงส่วนซิมมูเลท(SIMULATE) 50

รูปที่4.10 แสดงส่วนโปรแกรม AND_LD 51

รูปที่4.11 แสดงส่วนแลดเดอร์ไดอะแกรม AND_LD 51

รูปที่4.12 แสดงส่วนซิมมูเลท AND_LD 52

รูปที่4.13 แสดงส่วนโปรแกรม OR_LD 52

รูปที่4.14 แสดงส่วนแลดเดอร์ไดอะแกรม OR_LD 53

รูปที่4.15 แสดงส่วนซิมมูเลท OR_LD 53

รูปที่4.16 แสดงส่วนโปรแกรม TIMER 54

รูปที่4.17 แสดงส่วนแลดเดอร์ไดอะแกรม TIMER 54

รูปที่4.18 แสดงส่วนซิมมูเลท TIMER 55

รูปที่4.19 แสดงส่วนโปรแกรม COUNTER 55

รูปที่4.20 แสดงส่วนแลดเดอร์ไดอะแกรม COUNTER 56

รูปที่4.21 แสดงส่วนซิมมูเลท COUNTER 56

สารบัญตาราง

	หน้า
ตาราง2-1 ประเภทข้อมูล	6
ตาราง2-2 กลุ่มคำสั่งทางคณิตศาสตร์	10
ตาราง2-3 กลุ่มคำสั่งเปรียบเทียบ	11
ตาราง2-4 สัญลักษณ์พื้นฐาน	26
ตาราง2-5 ตัวอย่างชุดคำสั่งบูลีน	29

บทที่ 1
บทนำ

ในปัจจุบันจะพบว่าเครื่องควบคุมพีแอลซี ถูกนำมาใช้ในระบบควบคุมอัตโนมัติอย่างแพร่หลาย เนื่องจากมีความสะดวกในการใช้งาน เมื่อเทียบกับอุปกรณ์ควบคุมชนิดอื่นเช่น รีเลย์,สวิตช์ และไม่ต้องการการบำรุงรักษามากมายนัก เนื่องจากไม่มีส่วนที่เคลื่อนไหว แต่ใช้โปรแกรมในการประมวลผลแทน ดังนั้นส่วนโปรแกรมจึงเป็นส่วนสำคัญยิ่ง ซึ่งแน่นอนว่าโปรแกรมเหล่านั้นย่อมได้รับการออกแบบมาอย่างดี

แต่มักพบปัญหาเสมอในการทดสอบกับงานจริง ดังนั้นการเขียนโปรแกรมและทดสอบบนซอฟต์แวร์เป็นทางเลือกหนึ่งที่จะทำให้โปรแกรมที่ป้อนในงานจริงมีความน่าเชื่อถือในระดับหนึ่ง

ข้อดีอีกประการคือใช้ในการศึกษาการเขียนโปรแกรมแลดเดอร์บนคอมพิวเตอร์ หรือใช้ในการศึกษาวิชา PLC โดยไม่ต้องซื้อตัว PLC เนื่องจากคอมพิวเตอร์มีขอบเขตการใช้งานกว้างกว่า

1.1วัตถุประสงค์

1. เพื่อศึกษาลักษณะการป้อนคำสั่งภาษา Boolean ของ PLC
2. เพื่อศึกษาความสัมพันธ์กันระหว่าง Ladder Diagram และ Boolean
3. เพื่อศึกษาโครงสร้างคำสั่งภาษา Boolean โดยใช้ Visaul Basic6 มาเลียนแบบลักษณะโครงสร้างของโปรแกรม
4. จำลองการทำงานของ PLC โดยใช้โปรแกรม Visaul Basic6

1.2ขอบเขตโครงงาน

1. สามารถรับค่าคำสั่ง LD , LD NOT , AND , AND NOT , OR , OR NOT และ OUT ได้
2. สามารถรับคำสั่งฟังก์ชัน TIMER และ COUNTER ได้
3. สามารถแสดงผลในส่วน Ladder Diagram ได้
4. สามารถแสดงผลในส่วนภาษา Boolean ได้
5. สามารถ Simulate ในส่วนของ สภานะ Input , สถานะ Relay input และสถานะ Relay Output
6. เป็นจุดเริ่มต้น (แนวทาง) ในการพัฒนาต่อ ๆ ไป

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1ทฤษฎีและหลักการพื้นฐาน Visual Basic

2.1.1ที่มาและวิวัฒนาการของ Visual Basic

ในปัจจุบัน ระบบปฏิบัติการ (Operating System) ในลักษณะของ Windows ได้เข้ามาแทนระบบปฏิบัติการในลักษณะเดิม ซึ่งส่วนใหญ่ที่นิยมใช้กันอยู่คือ MS-DOS เนื่องจากรูปแบบของจอภาพที่ใช้ติดต่อระหว่างคอมพิวเตอร์ และผู้ใช้ อยู่ในรูปแบบรูปภาพแทนคำสั่งต่าง ๆ ((GUI) Graphic User Interface) ซึ่งต่างจาก MS-DOS ที่รูปแบบของคำสั่งจะอยู่ในรูปแบบของตัวอักษรและเป็นแบบป้อนทีละบรรทัด ซึ่งผู้ใช้จะต้องเข้าใจคำสั่งให้ถูกต้องและแม่นยำ และด้วยเหตุนี้โปรแกรมเมอร์ ซึ่งแต่เดิมพัฒนาโปรแกรมอยู่บน MS-DOS ต้องเปลี่ยนแปลงรูปแบบและแนวความคิด และหันมาพัฒนาโปรแกรมบน Windows แทน

ภาษา BASIC ถูกสร้างขึ้นในปี 2507 โดย John Keneny และ Thomas Kurtz ที่วิทยาลัย Dattmouth ในเบื้องต้น พวกเขามีจุดมุ่งหมายในการพัฒนาภาษา BASIC ขึ้น เพื่อใช้ในการสอนแนวในการเขียนโปรแกรม (Programming Concept) โดยเน้นให้รูปแบบของภาษานั้นง่ายต่อทั้งการเข้าใจและการใช้ และสามารถติดต่อกับผู้พัฒนาโปรแกรมได้ทันทีที่มีข้อผิดพลาดขณะเขียนโปรแกรม ในยุคแรก ๆ นั้นบริษัท IBM ได้นำมาใช้กับเครื่อง PC โดยถูกเรียกว่า BASICA ต่อมาในรุ่น Compatible

เมื่อไมโครซอฟท์นำ MS DOS 5.0 ออกสู่ตลาดก็ได้เปิดตัว Quick BASIC ซึ่งได้มีการพัฒนาต่อมาเรื่อย ๆ จนถึงปี 2534 ได้นำวิชวลเบสิก (Visual Basic) ออกสู่ตลาด วิชวลเบสิกสามารถออกแบบหน้าตาโปรแกรมสำหรับผู้ใช้งานแบบกราฟิก หรือที่เรียกว่ากราฟิกยูสเซอร์อินเตอร์เฟส (GUI) ได้ทันที โดยใช้เครื่องมือที่ได้เตรียมเอาไว้ให้แล้ว ทำให้มีโอกาสเห็นหน้าตาของแอพพลิเคชันตั้งแต่ ตอนพัฒนาเลยทีเดียว ซึ่งจะง่ายต่อการจัดวางรูปแบบ และแก้ไขหากความต้องการของผู้ใช้งานเปลี่ยนไป

วิชวลเบสิกพัฒนามาตั้งแต่รุ่น 1.0 ที่รันบนดอส มาจนรุ่นที่ 3.0 ที่ได้รับความนิยมอย่างมาก เพราะสร้างแอพพลิเคชันบน Windows 3.1 ได้อย่างรวดเร็ว และปัจจุบันวิชวลเบสิก 5.0 ได้พัฒนาให้สามารถใช้งานเกี่ยวกับมัลติมีเดีย (Multimedia) ฐานข้อมูล (Data Base) ได้ดีขึ้น สามารถเขียนโปรแกรมในแบบ OOP ได้เป็นอย่างดี และที่สำคัญสามารถสร้างแอพพลิเคชัน(Application) ให้ใช้งานกับอินเตอร์เนตได้ด้วยนอกจากที่กล่าวมาแล้วยังสามารถหาอุปกรณ์เสริมการทำงานและแหล่งข้อมูลอ้างอิงได้ที่ http://www.microsoft.com

2.1.2 Event – Driven Programming กับ Visual Basic

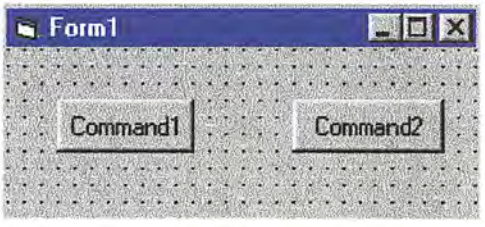
ในระบบปฏิบัติการ DOS เหตุการณ์ต่าง ๆ ที่เกิดขึ้นในโปรแกรม จะเกิดขึ้นอย่างเป็นลำดับตามขั้นตอนที่โปรแกรมกำหนดไว้เช่นโปรแกรมอาจกำหนดลำดับการทำงานไว้ให้รับข้อมูลในส่วนที่ต่าง ๆ แล้วจึงยืนยันเพื่อบันทึกข้อมูล จากนั้นก็ให้กลับไปรับข้อมูลใหม่

นับตั้งแต่ระบบปฏิบัติการ Windows ได้มีการใช้กันอย่างแพร่หลาย รูปแบบของโปรแกรมได้มีการเปลี่ยนแปลงไป เหตุการณ์ที่เกิดขึ้นไม่ได้เกิดขึ้นอย่างเป็นลำดับเหมือนเช่นเดิม แต่จะเกิดตามที่ผู้ใช้กำหนดแทน เช่น ผู้ใช้อาจกดปุ่มที่ 2 ก่อนปุ่มที่ 1 ทั้ง ๆ ที่ผู้พัฒนาโปรแกรมตั้งใจให้กดปุ่มที่ 1 ก่อน เป็นต้น

แต่อย่างไรก็ตาม ก็ยังคงต้องอาศัยแนวความคิดในการเขียนแบบเดิมอยู่บ้าง เนื่องจากการกำหนดการทำงานให้กับเหตุการณ์ (Event) ยังคงต้องกำหนดอย่างเป็นขั้นตอน โดยจะพิจารณาว่าแต่ละหน่วยย่อยที่สุดของโปรแกรม (Object) จะมีเหตุการณ์อะไรเกิดขึ้นบ้าง แล้วจึงเลือกเขียนโปรแกรมเฉพาะเหตุการณ์นั้น

2.1.3 Procedure และ Function ใน Visual Basic

Visual Basic ใช้หลักการแบ่งโปรแกรมออกเป็นส่วน ๆ ที่เรียกว่า "โมดูล" (Module) (ตามหลักการการเขียนโปรแกรมแบบ Modularity โดยจะแบ่งโปรแกรมออกเป็นกระบวนการ (Procedure) และ ฟังก์ชัน (Function) แต่เนื่องจากการกำหนดการทำงานของโปรแกรมจะกำหนดตามเหตุการณ์ที่เกิดขึ้นกับส่วนย่อยที่สุดของโปรแกรม ดังนั้นการแบ่ง กระบวนการ ใน Visual Basic จึงแบ่งตามชื่อ ส่วนย่อยที่สุดของโปรแกรม และ เหตุการณ์ เช่น การเขียนโปรแกรมให้กับส่วนย่อยที่สุดของโปรแกรม สมมุติชื่อ "Command1" ซึ่งเป็นปุ่มบนจอภาพ ภายใต้เหตุการณ์ กดปุ่มก็จะเขียนเป็นกระบวนการหนึ่ง กับอีกส่วนย่อยที่สุดของโปรแกรมหนึ่ง ซึ่งเป็นส่วนย่อยที่สุดของโปรแกรมประเภทเดียวกันแต่คนละชื่อ สมมุติชื่อ "Command2" ภายใต้เหตุการณ์ กดปุ่มเหมือนกันก็ต้องเขียนเป็นอีกกระบวนการหนึ่ง ถึงแม้จะเป็นเหตุการณ์เดียวกัน แต่เกิดกับส่วนย่อยที่สุดของโปรแกรมที่คนละตัวกัน ดังรูปที่ 2.1



รูปที่ 2.1 ฟอร์ม1(Form1)

ชื่อของกระบวนการ ใน visual basic จะอยู่ในรูปแบบดังนี้

```
[Private / Private] Sub name [(arglist)]
……..
End Sub
```

โดยที่ Private Sub จะเป็นคำเฉพาะที่ใช้บอกว่าเป็นกระบวนการ

name ได้แก่ ชื่อของกระบวนการ

arglist ได้แก่ รายชื่อข้อโต้แย้ง (Argument) ที่ใช้ภายในกระบวนการนั้น

End Sub เป็นคำเฉพาะที่ใช้สำหรับจบการทำงานของแต่ละกระบวนการ

```
    A = 1
    B = 1
    Inc bInc := 1 , aInc := 1
    Debug.Print "value of A ager function is" & A & "and B is" & B
End Sub
```

ขอบเขตการใช้งานของ กระบวนการ และ ฟังก์ชัน

ทุกกระบวนการ หรือ ฟังก์ชันที่อยู่ในฟอร์ม (Form) หนึ่ง ๆ ล้วนแต่มีขอบเขตในการใช้งาน ซึ่งสามารถแบ่งออกได้เป็น 2 ลักษณะคือ แบบทั่วไป (Public) และแบบเฉพาะตัว (Private) โดยทั่วไปกระบวนการและฟังก์ชันจะถูกกำหนดให้เป็นแบบเฉพาะตัว สามารถถูกเรียกใช้งานได้เฉพาะภายในฟอร์มที่มันถูกสร้างขึ้น โดยจะถูกเรียกใช้จากฟอร์มอื่นไม่ได้ ส่วนแบบทั่วไปจะตรงกันข้ามกับแบบเฉพาะตัวกล่าวคือ กระบวนการหรือฟังก์ชันที่จัดทำขึ้นในฟอร์มหนึ่ง สามารถถูกเรียกใช้ได้ในกระบวนการหรือฟังก์ชันที่อยู่ในอีกฟอร์มหนึ่งได้

2.1.4 การใช้ส่วนแก้ไข (Editor) ของ Visual Basic

ในการเรียกส่วนแก้ไขของ Visual Basic ขึ้นมาทางาน ทำได้โดยดับเบิ้ลคลิกที่ส่วนย่อยที่สุดของโปรแกรมที่ต้องการเขียนโปรแกรม

ส่วนแก้ไขจะประกอบไปด้วย 3 ส่วนคือ

2.1.4.1 ส่วนแสดงรายชื่อส่วนย่อยที่สุดของโปรแกรม

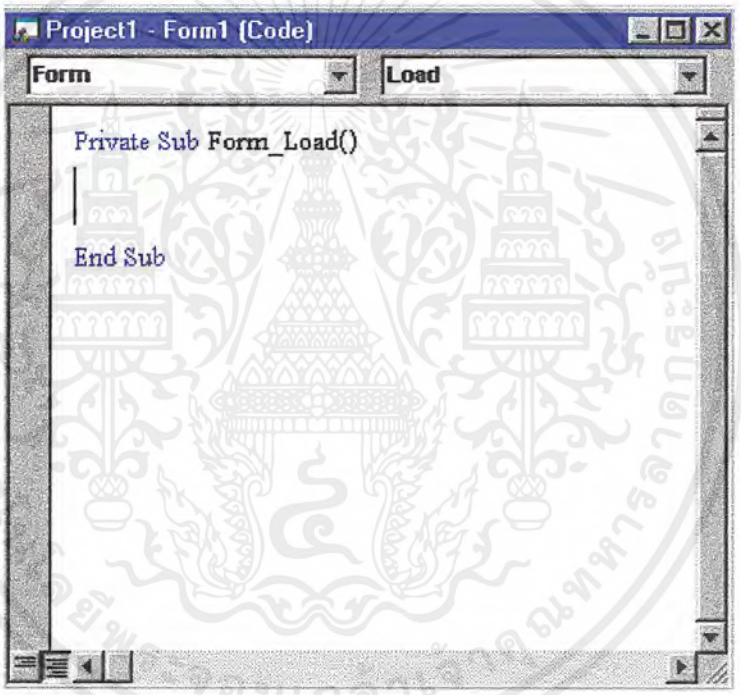
ใช้แสดงชื่อของส่วนย่อยที่สุดของโปรแกรมต่าง ๆ ที่เราได้วาดไว้บนฟอร์มรวมทั้งส่วนของทั่วไป (General) ที่ใช้ในการนิยาม (Declare)

2.1.4.2 ส่วนแสดงรายชื่อเหตุการณ์

เป็นส่วนของเหตุการณ์ภายใต้ส่วนย่อยที่สุดของโปรแกรมนั้น ซึ่งแต่ละส่วนย่อยที่สุดของโปรแกรมจะมีเหตุการณ์ที่แตกต่างกันไป

2.1.4.3 ส่วนสำหรับเขียนโปรแกรม

ใช้สำหรับเขียนโปรแกรม



รูปที่2.2 ส่วนแก้ไขโปรแกรม

ส่วนแก้ไขของ Visual Basic จะมีส่วนที่เรียกว่าส่วนตรวจเช็คข้อผิดพลาดอัตโนมัติ (Automatic Syntax Checking) ซึ่งทำหน้าที่ตรวจสอบความถูกต้องในแต่ละคำสั่ง โดยจะเริ่มตรวจสอบทันทีเมื่อมีการกดEnterซึ่งถ้ามีข้อผิดพลาดเกิดขึ้นก็จะมีข้อความมาเตือนให้ผู้ใช้ทราบในทันที นอกจากนี้ Visual Basic ยังมีความสามารถพิเศษในการแยกคำเฉพาะ (Reserved Word) เช่น คำสั่ง

End ไม่ว่าจะพิมพ์เป็น end , End หรือ END ก็ตามเมื่อพิมพ์เสร็จ Visual Basic จะเปลี่ยนเป็น End เสมอ

นอกจากนั้น Visual Basic ยังได้ใช้สีเพื่อแยกความแตกต่างของคำสั่งตัวแปรและข้อสังเกต (Comment) ออกจากกัน โดยจะเปลี่ยนสีของคำสั่งให้เป็นสีน้ำเงิน ตัวแปรเป็นสีดำ และ ข้อสังเกตเป็นสีเขียว ทั้งนี้ก็เพื่อความสะดวกในการเขียนโปรแกรม

2.1.5 ประเภทของข้อมูล (Data Type)

ใน Visual Basic จะแบ่งข้อมูลออกเป็นประเภทต่าง ๆ ดังตาราง

ประเภทตัวแปร	สัญลักษณ์	เนื้อที่ที่ใช้	ขอบเขตของค่าที่รับได้
Byte	ไม่มี	1 Byte	0 ถึง 255
Boolean	ไม่มี	2 Bytes	True หรือ False
String (กรณีไม่จำกัดความยาว)	S	10 Bytes + ความยาวของข้อความ	0 ถึง 2 พันล้านตัวอักษรโดยประมาณ
String (กรณีจำกัดความยาว)	S	ความยาวของข้อความ	1 ถึง 65,400 ตัวอักษรโดยประมาณ
Integer	%	2 Bytes	-32,768 ถึง +32,767
Long (long integer)	&	4 Bytes	-2,147,483,648 ถึง +2,147,483,647
Single (single-precision folating-point)	!	4 Bytes	-3.402823E+38 ถึง –1.401298E+45 และ +1.401298E-45 ถึง 3.402823E+38
Double (double-precision folating-point)	#	8 Bytes	-4.94065645841247E-324 (สำหรับจำนวนลบ) และ 4.906564584123E-324 ถึง 1.79769313486232E308 สำหรับจำนวนบวก
Currency (scaled integer)	@	8 Bytes	-922,337,203,685,477.5808 ถึง 922,337,203,685,477.5807

ประเภทตัวแปร	สัญลักษณ์	เนื้อที่ที่ใช้	ขอบเขตของค่าที่รับได้
Variant (กับข้อมูลที่เป็นตัวเลข)	-	16 Bytes	ขนาดเท่ากับ Double
Variant (กับข้อมูลที่เป็นข้อความ)	-	22 Bytes + ความยาวของข้อความ	ขนาดเท่ากับ String แบบไม่กำหนดความยาว
Date	ไม่มี	8 Bytes	January 1,100 ถึง December 31,9999
Object	ไม่มี	4 Bytes	Object ใด ๆ

ตารางที่ 2.1ประเภทของข้อมูล (Data Type)

2.5.1 Byte ใช้เก็บข้อมูลในรูป Binary

2.5.2 Boolean ใช้เก็บค่าทางตรรกะ ที่มีค่าเป็นจริง (True) หรือเท็จ (False)

2.5.3 String ใช้เก็บข้อความต่าง ๆ เช่น "Hello , world!" หรือ "John Fitzquatle" เป็นต้น หรือชุดของตัวเลขในรูปของข้อความ เช่น "1234" หรือทั้ง 2 อย่างรวมกัน เช่น "1234 Main Street"

2.5.4 Integer ใช้เก็บค่าของเลขจำนวน

2.5.5 Long จะใช้กับเลขจำนวนเต็มที่มีขนาดใหญ่

2.5.6 Single ใช้เก็บค่าของเลขจำนวนจริง

2.5.7 Double จะใช้กับเลขจำนวนจริงที่มีขนาดใหญ่

2.5.8 Currency ใช้เก็บค่าที่เป็นจำนวนเงิน

2.5.9 Variant ใช้เก็บค่าประเภทใดก็ได้ โดยจะแปรเปลี่ยนไปตามข้อมูลที่มันจัดเก็บ

2.5.10 Object ใช้อ้างถึง Object ใด ๆ (รายละเอียดจะกล่าวถึงในบทการเขียนโปรแกรมกับ Object)

2.5.11 Date ใช้เก็บข้อมูลในรูปวันที่

ตัวแปรประเภท String มีอยู่ด้วยกัน 2 แบบคือ String ในแบบที่ไม่จำกัดความยาว (Variable Length) และแบบจำกัดความยาว (Fixed Length)

ตัวอย่างการกำหนดให้ตัวแปร String เป็นแบบจำกัดความยาว และแบบไม่จำกัดความยาว

```
Dim Rstring As String 815
Lstring$ = Rstring
```

ในบรรทัดแรกจะเป็นการกำหนด String แบบจำกัดความยาวที่มีขนาด 15 ตัวอักษร ส่วนในบรรทัดที่ 2 เป็นตัวแปรไม่จำกัดความยาว โดยที่เริ่มต้นจะมีขนาด 15 ตัวอักษร แต่ขนาดสามารถเปลี่ยนแปลงได้ในภายหลัง

2.1.6 กฎในการตั้งชื่อตัวแปร

1. ชื่อของตัวแปรจะยาวได้ไม่เกิน 40 ตัวอักษร
2. ตัวอักษรตัวแรกของชื่อจะต้องเป็นตัวอักษร A-Z
3. ตัวอักษรตัวถัดไปจะเป็นตัวอักษร A-Z ตัวเลข 0-9 หรือขีด (-)
4. ตัวอักษรตัวสุดท้ายอาจเป็นเครื่องหมายที่ใช้แสดงถึงประเภทของตัวแปรได้แก่ % , & , S , # , ! , @ หรืออาจไม่มีเครื่องหมายใด ๆ ก็ได้ ในกรณีที่ ประกาศตัวแปรด้วยคำสั่ง Dim
5. ชื่อของตัวแปรจะต้องไม่ซ้ำกับคำเฉพาะ (Reserved Word)
6. ตัวอักษรในชื่อสามารถเป็นได้ทั้งตัวอักษรตัวใหญ่และตัวเล็ก

2.1.7 การประกาศ (Declare) ค่าตัวแปร

ค่าตัวแปรต่าง ๆ ที่ใช้ใน Visual Basic จะต้องมีการกำหนดประเภทของตัวแปรนั้น โดยทำได้ 2 วิธี คือ

2.1.7.1 การประกาศตัวแปรโดยชัดเจน (Explicit Declaration) กำหนดประเภทของตัวแปรโดยใช้คำสั่ง

Dim varname [As type] [,varname [As type]] ...

โดยที่ varname หมายถึง ชื่อตัวแปร

type หมายถึง ประเภทข้อมูล

การกำหนดในลักษณะนี้ สามารถกำหนดได้ทั้งในส่วนทั่วไป และต้นโปรแกรมของแต่ละกระบวนการ มีข้อแตกต่างกันคือในส่วนทั่วไปจะเป็นตัวแปรส่วนกลางที่ กระบวน หรือ ฟังก์ชันต่าง ๆ ภายในฟอร์มเดียวกันสามารถเรียกใช้งานได้ แต่กรณีที่ประกาศไว้ในกระบวนการหรือฟังก์ชัน ตัวแปรที่ประกาศไว้จะใช้ได้ภายในกระบวนการหรือฟังก์ชันที่มีการประกาศ ตัวแปรนั้นไว้เท่านั้น

2.1.7.2 การประกาศตัวแปรไม่ชัดเจน (Implicit Declaration) การกำหนดประเภทของตัวแปรจะทำอยู่ในรูป

varname <Type Identifier>

เช่น Mradius! , Vamount@ เป็นต้น ซึ่งการกำหนดประเภทตัวแปรลักษณะนี้ ตัวแปรนั้นจะต้องเขียนในลักษณะนี้ตลอดการใช้งาน กระบวนการนั้น

2.1.8 ปฏิบัติการ (Operator)

ปฏิบัติการที่ใช้ใน Visual Basic จะแบ่งออกได้ 4 ประเภท ดังนี้

2.1.8.1 ปฏิบัติการกำหนดค่า (Assignment Operator)

ได้แก่เครื่องหมาย "=" ซึ่งใช้กำหนดค่าให้กับตัวแปร (Property) ของส่วนย่อยที่สุดของโปรแกรม

```
Height% = 100
Area! PI * Radius ^2
MyBox.Text = MyPrompt$
```

ปฏิบัติการนี้ จะกำหนดค่าให้กับตัวแปรของส่วนย่อยที่สุดของโปรแกรม ที่อยู่ทางซ้ายของเครื่อง "=" เช่น การกำหนดให้ตัวแปร "Height" เป็นตัวแปรประเภท Integer (มีเครื่องหมาย % อยู่หลังตัวแปร) มีค่า 100 ตัวแปร Area ซึ่งกำหนดให้เป็นตัวแปรประเภท Single (มีเครื่องหมาย ! อยู่หลังตัวแปร)

โดยค่าที่กำหนดให้ ได้มาจากตัวแปร MyPrompt ซึ่งถูกกำหนดให้เป็นตัวแปรประเภท String (มีเครื่องหมาย $ อยู่หลังตัวแปร)

2.1.8.2 ปฏิบัติการทางคณิตศาสตร์ (Arithmetic Operators)
เป็นปฏิบัติการที่ใช้ในการคำนวณ ประกอบด้วย

เครื่องหมาย	ความหมาย	ตัวอย่าง
^	ยกกำลัง	A! = 3^4 (3 ยกกำลัง 4)
-	เครื่องหมายลบ	A! = -1
*,/	คูณ หรือหาร	A! = B! * C! หรือ A! = B! / C!
\	การหารแบบ Integer	A% = B% \ C%
Mod	Modulo	A% = B% Mod C%
+ , -	บวก หรือ ลบ	A% = B% + C% หรือ A% = B% - C%
&	การต่อ String	A$ = B$ & C$

ตาราง2.2 กลุ่มคำสั่งทางคณิตศาสตร์

ในการต่อ String มีสิ่งที่น่าสนใจ คือ เราสามารถใช้ปฏิบัติการ "+" แทนปฏิบัติการ "&W ได้ เช่น

```
A$ = "Hello"
B$ = "World"
C$ = A$ & "," & B$
D$ = A$ + ","+B$
```

ค่าของ C$ และ D$ จะมีค่าเท่ากันคือ "Hello , World" แต่ถ้าเราใช้ปฏิบัติการ "+" ในการต่อ String ซึ่งเก็บอยู่ในกับตัวแปรประเภท Variant กลับให้ผลที่ต่างกับการใช้ปฏิบัติ "&" ในการต่อ เช่น

```
A = "5"
B = "6"
C = A + B
D = A & B
```

ตัวแปร A และ B ซึ่งไม่ได้ประกาศจึงถูกกำหนดให้เป็นตัวแปรประเภท Variant และใช้เก็บ String "5" และ "6" แต่ตัวแปร C จะมีค่าเป็น "11" เพราะค่าของ A และ B จะถูกแปลงไปเป็นตัวเลข ในขณะที่ D จะมีค่าเป็น "56"

2.1.8.3 ปฏิบัติการเปรียบเทียบ (Relational Operators)

ใช้สำหรับเปรียบเทียบประโยคคำสั่งตั้งแต่ 2 ประโยคขึ้นไป ซึ่งผลที่ได้จะเป็นค่าจริง (True) หรือเท็จ (False) ดังนั้นจึงมักใช้ในประโยคคำสั่ง If

ปฏิบัติการ	ความหมาย	ตัวอย่าง
=	เท่ากับ	A% = B% จะเป็นจริง ถ้าค่าของ A% เท่ากับค่าของ B%
<>	ไม่เท่ากับ	A% <> B% จะเป็นจริงถ้าค่าของ A% ไม่เท่ากับค่าของ B%
>	มากกว่า	A% > B% จะเป็นจริงถ้าค่าของ A% มากกว่าค่าของ B%
<	น้อยกว่า	A% < B% จะเป็นจริงถ้าค่าของ A% น้อยกว่าค่าของ B%
>=	มากกว่าหรือเท่ากับ	A% >= B% จะเป็นจริงถ้าค่าของ A% มากกว่าหรือเท่ากับค่าของ B%
<=	น้อยกว่าหรือเท่ากับ	A% <= B% จะเป็นจริงถ้าค่าของ A% น้อยกว่าหรือเท่ากับค่าของ B%
Like	ใช้เปรียบเทียบ String	AS like BS จะเป็นจริง ถ้าค่าของ String A ตรงกับค่าของ String B
Is	ใช้เปรียบเทียบ 2 Object	A Is B จะเป็นจริงถ้าค่าของ A และ B อ้างถึงส่วนย่อยที่สุดของโปรแกรมเดียวกัน

ตารางที่2.3ปฏิบัติการเปรียบเทียบ (Relational Operators)

เช่น ประโยคคำสั่ง If A% = B% Then Label1.Caption = "These are equal" ซึ่งหมายถึง ถ้าค่าของตัวแปร A เท่ากับค่าของตัวแปร B แล้วให้ใส่ข้อความ "These are equal" ในตัวแปร "Caption" ของส่วนย่อยที่สุดของโปรแกรม "Label1"

2.1.8.4 ปฏิบัติการลอจิก (Logical (Bitwise) Operators)

ปฏิบัติการนี้ใช้เชื่อมประโยคคำสั่ง (Expression) หลาย ๆ ประโยคเข้าด้วยกัน โดยผลที่ได้จะมีค่าเป็น จริง หรือ เท็จ แต่จะเกิดจากการนำเอาค่า จริงหรือเท็จ ของแต่ละประโยคคำสั่งมาประมวล ซึ่งจะกล่าวอีกครั้งในเรื่องของ PLC

2.1.9 ส่วนควบคุมการทำงานของโปรแกรม (Process Control)

ประกอบไปด้วยกลุ่มคำสั่ง 3 กลุ่มด้วยกันคือ

2.1.9.1 กลุ่มคำสั่ง Branching

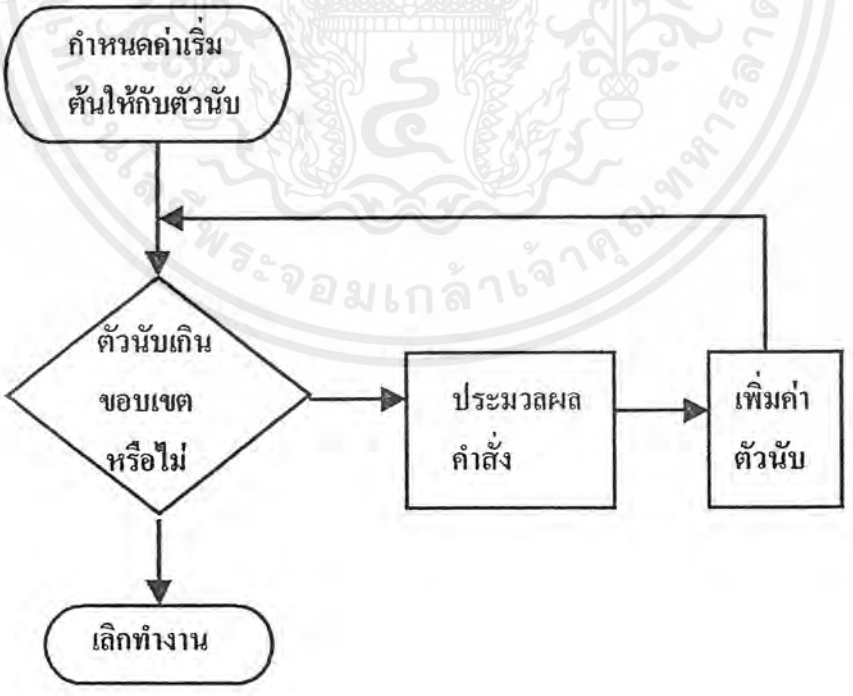
เป็นกลุ่มคำสั่งที่ใช้ในการกระโดดข้ามการทำงานจากคำสั่งหนึ่งไปอีกคำสั่งหนึ่ง แบ่งเป็น 2 กลุ่มย่อย คือ กลุ่มแรกเป็นกลุ่มคำสั่งที่เมื่อกระโดดข้ามไปยังอีกคสั่งหนึ่งแล้วจะไม่กลับมาทำงานยังคำสั่งที่อยู่ถัดจากคำสั่งที่กระโดดไป และกลุ่มที่สองเมื่อกระโดดไปทำงานยังคำสั่งอีกส่วนหนึ่งเรียบร้อยแล้ว ก็จะกลับมาทำงานยังคำสั่งถัดจากคำสั่งที่กระโดดไปต่อไป

2.1.9.2 กลุ่มคำสั่ง Iteration

เป็นกลุ่มคำสั่งที่มีลักษณะการทำซ้ำ บ่อยครั้งที่โปรแกรมจะมีลักษณะการทำงานที่ซ้ำคำสั่งเดิมเป็นจำนวนครั้ง ภายใต้เงื่อนไขที่กำหนด

2.1.9.2.1 กลุ่ม For … Next

มีลักษณะการทำซ้ำในแบบ Block โดยสามารถกำหนดค่าเริ่มต้นและค่าสิ้นสุดที่ใช้ควบคุมจำนวนครั้งของการทำซ้ำ ซึ่งถูกควบคุมโดยตัวนับ (Counter) ซึ่งจะเปลี่ยนแปลงค่าทุกครั้งในแต่ละรอบ การทำซ้ำจนกระทั่งค่าตัวนับมีค่าเกินค่าสิ้นสุดที่กำหนดไว้ ก็ออกจาก Block การทำซ้ำ ดังรูป



รูปที่2.3กลุ่ม For … Next

รูปแบบคำสั่ง

```
For Counter = Start to end (Step Step)
---
[Exit For]
---
Next
```

โดยที่ Counter หมายถึงตัวแปรที่ใช้นับจำนวนลูป (Loop)

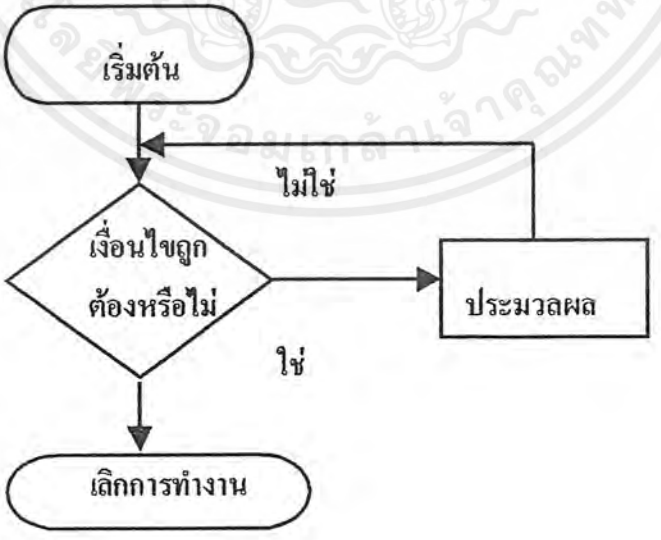
Start หมายถึงค่าเริ่มต้นในการนับ

End หมายถึงค่าสิ้นสุดการนับ

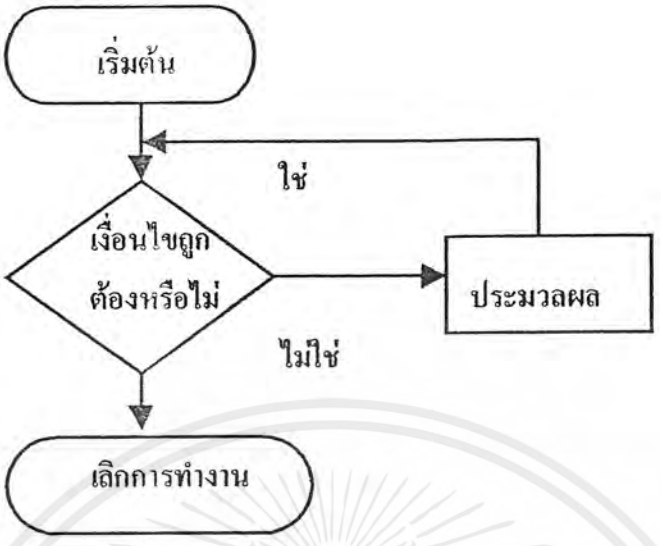
Step หมายถึงลำดับในการนับ

2.1.9.2.2 Do … loop

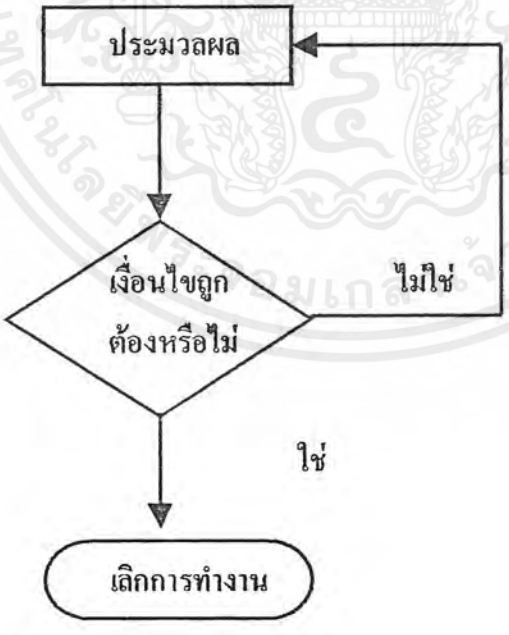
มีด้วยกัน 4 แบบคือ Do While , Do Until , Loop While และ Loop Until ขั้นตอนการทำงานแสดงด้วย Flow Chart



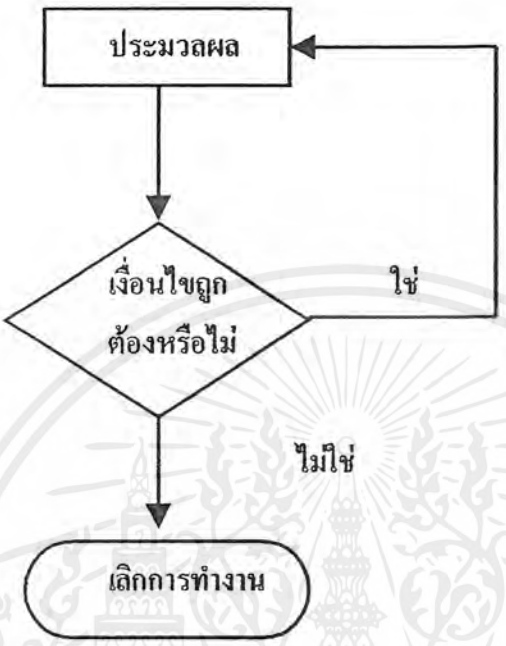
รูปที่ 2.4 Do Until … loop



รูปที่2.5 Do While ... loop



รูปที่2.6 Do ... loop Until



รูปที่2.7 Do ... loop While

2.1.10 กลุ่มคำสั่งตัดสินใจ

เป็นกลุ่มคำสั่งที่ใช้ในการตัดสินใจในการทำงานตามเงื่อนไขต่าง ๆ ภายในโปรแกรม เช่น Go to , Go sub

2.1.11 ตัวแปรที่กำหนดขึ้นใช้เอง

นอกเหนือจากตัวแปรประเภทต่าง ๆ ที่เป็นมาตรฐานใน Visual Basic แล้ว เรายังสามารถกำหนดรูปแบบของตัวแปรขึ้นใช้เองได้โดยใช้คำสั่ง Type ... End Type

รูปแบบ

```
[Private | Public] Type varname
elementName [([subcripts])] As type
[elementName [([subscripts])] As type]
...
End Type
```

โดยที่	varname	หมายถึงชื่อของตัวแปรที่ต้องการสร้างขึ้น
	elementName	หมายถึงชื่อของตัวแปรที่อยู่ภายใต้ตัวแปรที่สร้างขึ้น
	subscripts	หมายถึงขนาดของ Array ใช้ในกรณีที่ elementName ที่กำหนดขึ้นเป็น Array
	type	หมายถึงประเภทของข้อมูล

คำสั่ง **Public** (ค่า Default) จะใช้ในกรณีที่ต้องการให้ตัวแปรที่กำหนดขึ้น สามารถใช้ในทุกกระบวนการของโปรเจค ส่วนคำสั่ง **Private** (กำหนดหรือไม่ก็ได้) ใช้ในกรณีที่ต้องการให้ตัวแปรที่กำหนดขึ้น สามารถใช้ได้เฉพาะในกระบวนการของโปรเจคที่ประกาศตัวแปรนั้น

2.1.12 ขอบเขตการใช้งานของตัวแปร

ตัวแปรที่ถูกกำหนดขึ้นด้วยคำสั่งการประกาศจะมีขอบเขตในการใช้งานภายใต้กระบวนการหรือฟังก์ชันต่าง ๆ ขึ้นอยู่กับว่าการประกาศตัวแปรนั้นกระทำ ณ ที่ใดของโปรแกรม ดังนั้นจึงสามารถแบ่งประเภทของตัวแปรตามขอบเขตการใช้งานได้ออกเป็น 2 แบบดังนี้

2.1.12.1 ตัวแปรแบบเฉพาะ

ตัวแปรชนิดนี้จะมีขอบเขตการใช้งานอยู่ในเฉพาะกระบวนการหรือฟังก์ชันที่ประกาศไว้ ดังนั้นตัวแปรชื่อเดียวกันจึงสามารถประกาศ ในต่างกระบวนการหรือฟังก์ชันกันได้ โดยที่ Visual Basic จะมองตัวแปรเหล่านั้นเป็นตัวแปรคนละตัวกัน เมื่อเปลี่ยนแปลงค่าของตัวแปรประเภทนี้ในกระบวนการหรือฟังก์ชันหนึ่ง จะไม่ส่งผลต่อตัวแปรชื่อเดียวกันในกระบวนการหรือฟังก์ชันอื่น ๆ แต่อย่างใด ในการประกาศตัวแปรประเภทนี้ ทำได้ทั้งแบบไม่ชัดเจนและแบบชัดเจน แต่ต้องกำหนดในกระบวนการหรือฟังก์ชันที่ต้องการใช้งาน

2.1.12.2 ตัวแปรแบบทั่วไป

ตัวแปรประเภทนี้จะต่างจากแบบแรก กล่าวคือ จะมีขอบเขตการใช้งานอยู่ในทุก ๆ กระบวนการหรือฟังก์ชัน ภายในฟอร์มเดียวกัน เมื่อเปลี่ยนแปลงค่าของตัวแปรประเภทนี้ ไม่ว่าที่กระบวนการหรือฟังก์ชัน ใดภายใต้ฟอร์มเดียวกัน จะส่งผลต่อตัวแปรชื่อเดียวกันในกระบวนการหรือฟังก์ชันอื่นด้วย ในการประกาศจึงกระทำในส่วนของทั่วไป

2.1.13 อะเรย์ (Array)

อะเรย์คือชุดของตัวแปรที่แสดงอยู่ในรูปของลำดับที่เพื่อใช้สำหรับเก็บค่าของข้อมูลที่อยู่ในกลุ่มเดียวกัน อะเรย์จะแตกต่างจากตัวแปรโดยทั่วไป กล่าวคือ ตัวแปรทั่วไปจะถูกเก็บอยู่ในหน่วยความจำในแต่ละตำแหน่งที่ไม่ต่อเนื่องกัน ส่วนตัวแปรอะเรย์จะถูกเก็บอยู่ในหน่วยความจำในตำแหน่งที่ต่อเนื่องกันเป็นผืน

อะเรย์แบ่งออกได้เป็น 2 ประเภทคือ

2.1.13.1 Static Array

เป็นอะเรย์ที่มีขนาดของมิติตายตัว เปลี่ยนแปลงไม่ได้ การสร้างตัวแปรอะเรย์ประเภทนี้ต้องระบุขนาดของอะเรย์ ซึ่งเป็นตัวเลขในวงเล็บหลังตัวแปรที่เป็นอะเรย์ เช่น

```
Dim A(100) As Integer
```

จากตัวอย่าง ตัวแปร A จะเป็นอะเรย์ขนาดมิติเดียวจำนวน 100 ชุด เริ่มจาก A(1) , A(2) , A(3) , … , A(100)

2.1.13.2 Dynamic Array

เป็นอะเรย์ที่เปลี่ยนแปลงขนาดของมิติได้ ดังนั้นในการประกาศจึงไม่ต้องระบุขนาดในวงเล็บ ตัวอย่างเช่น

```
Dim A ( ) As Integer
```

2.1.14 การจัดการและตรวจสอบข้อผิดพลาด

บ่อยครั้งในการเขียนโปรแกรม มักจะพบกับข้อผิดพลาดของโปรแกรม ถ้าโปรแกรมมีขนาดเล็ก ผู้พัฒนาโปรแกรมก็สามารถตรวจสอบโปรแกรมทีละบรรทัดได้ แต่ถ้าโปรแกรมมีขนาดใหญ่ การแก้ไขย่อมทำได้ยาก Visual Basic จึงสร้างเครื่องมือที่ช่วยในการตรวจหาข้อผิดพลาดของโปรแกรมเรียกว่า "Debugger" กัน

2.1.14.1 ประเภทของข้อผิดพลาด

ข้อผิดพลาดแบ่งออกเป็น 3 ประเภท

2.1.14.1.1 Syntax Errors เกิดเนื่องจากเขียนโปรแกรม ไม่ตรงตามรูปแบบของภาษาที่กำหนดไว้

```
If I = 0          I = I+I
```

ตัวอย่างจะเห็นได้ว่าขาดคำว่า Then ไป ข้อผิดพลาดประเภทนี้ใน Visual Basic ไม่ใช่เรื่องใหญ่แต่ประการใดเนื่องจากตัว Visual Basic จะมีส่วนที่คอยตรวจสอบข้อผิดพลาดประเภทนี้ที่เรียกว่า "Automatic Syntax Checking" ไว้ให้แล้ว

2.1.14.1.2 Runtime Errors เป็นข้อผิดพลาดที่เกิดขึ้นเมื่อทำการรัน (Run) และจะเกิดขึ้นในบางกรณีเท่านั้น เช่น

```
J = 6
For I = 6 To 0 Step –1
        A = J / I
Next
```

ตัวอย่างจะเห็นว่าโปรแกรมเกิดข้อผิดพลาด เมื่อคำนวณไปจนถึง Loop ที่ I มีค่าเป็น 0 เนื่องจากมีการหารด้วยเลข 0 ซึ่งข้อผิดพลาดประเภทนี้จะไม่สามารถตรวจสอบได้ จนกว่าจะทำการรันและเข้าในกรณีนั้น ๆ เท่านั้น

2.1.14.1.3 Logical Errors เป็นข้อผิดพลาดที่ค่อนข้างค้นหาได้ยากที่สุดเนื่องจากเป็นข้อผิดพลาดที่เกิดขึ้นจากความผิดพลาดด้านแนวคิดในการเขียนโปรแกรมของผู้พัฒนาโปรแกรมเอง ซึ่งการแก้ไขต้องอาศัยประสบการณ์ของผู้พัฒนาโปรแกรมเข้ามาช่วยในการแก้ไขข้อผิดพลาด

2.1.14.2 เครื่องมือตรวจสอบและแก้ไขข้อผิดพลาด

2.1.14.2.1 กรอบแสดงคำเตือน (Warning Message)

ในขณะที่กำลังเขียนโปรแกรม เกิดมีข้อผิดพลาด จากการใช้รูปประโยคผิดพลาด การใช้คำสั่งผิด ใช้ตัวแปรไม่ถูกต้อง หรือจากสาเหตุอื่น ๆ ก็จะปรากฏการณ์กรอบคำเตือนโดย

อัตโนมัติ เพราะวิชวลเบสิกใช้การคอมพล์ แบบ Interpreter คือ เมื่อเกิดข้อผิดพลาดจุดใดก็จะแสดงข่าวสารออกมาทันทีทำให้รูปได้ว่า เกิดข้อผิดพลาดตรงไหน และไม่ได้แก้ไขให้ถูกต้องโปรแกรมจะทำงานต่อไปไม่ได้

2.1.14.2.2 หน้าต่างอิมมิเดียต (Immediate Window)

การใช้หน้าต่างอิมมิเดียตแสดงค่าต่าง ๆ จากโปรแกรมเช่น ค่าตัวแปร และผลของการทำงานตรวจสอบข้อผิดพลาด เป็นต้น ช่วยให้ติดตามผลการทำงานได้ทุกขั้นตอนอย่างชัดเจน โดยใช้คำสั่ง Debug.Print ในการเรียกใช้หน้าต่างอิมมิเดียต

2.1.14.2.3 Error Handlers

บางครั้งผู้พัฒนาโปรแกรมก็ต้องการที่จะจัดการกับข้อผิดพลาดเหล่านั้นเอง ซึ่งสามารถทำได้โดยอาศัยฟังก์ชัน Error และ ส่วนย่อยที่สุดของโปรแกรม "Err" รูปแบบคำสั่งของข้อผิดพลาดมีดังนี้

Error[(errornumber)]

ส่วนย่อยที่สุดของโปรแกรม"Err"มีหน้าที่ในการคืนค่าหมายเลขของข้อผิดพลาดที่เกิดขึ้นล่าสุด ประโยคคำสั่ง **On Error** ซึ่งมีรูปแบบดังนี้

On Error command

โดยที่ command ได้แก่ คำสั่งที่ต้องการให้ทำ เมื่อเกิดข้อผิดพลาดของโปรแกรมขึ้น ซึ่งมีอยู่ 2 คำสั่งดังนี้

1. คำสั่ง **GoTo** line เป็นคำสั่งที่ให้ย้ายการทำงานของโปรแกรมไปยังบรรทัดคำสั่งที่กำหนดใน line เช่น

On Error GoTo error1 ย้ายไปยังบรรทัด "error1" เมื่อเกิดข้อผิดพลาดขึ้น

2. คำสั่ง **Resume Next** เป็นคำสั่งที่ให้ย้ายการทำงานของโปรแกรมไปยังบรรทัดถัดไป

2.2ทฤษฎีและหลักการพื้นฐานของเครื่องควบคุมแบบตรรกที่โปรแกรมได้

2.2.1 ประวัติของเครื่องควบคุมแบบตรรกที่โปรแกรมได้

บริษัท GM (Gerneral Motors) แห่งประเทศอเมริกาได้เสนอให้มีการพัฒนา PC ขึ้นมาใช้งานในอุตสาหกรรมและในปี พ.ศ. 2511 ก็มีผู้ประดิษฐ์ PC ตัวแรกได้สำเร็จ ต่อมาในปี พ.ศ. 2512 PC ก็ถูกผลิตขายในประเทศอเมริกาเป็นแห่งแรก

ในประเทศญี่ปุ่นเริ่มมีการผลิต PC ขายในปี พ.ศ. 2513 และPC ที่ใช้ได้กับงานทั่วไปตั้งแต่ปี พ.ศ. 2513 เป็นต้นมาบริษัมิตซูมิชิได้ผลิต PC ในรูปบอร์ดเดี่ยวตั้งแต่ปี พ.ศ. 2514 ต่อมาได้เริ่มผลิตรุ่น PC ตั้งแต่ปี พ.ศ. 2518 และในปี พ.ศ. 2519 ได้ผลิตรุ่น F ซึ่งเป็น PC ขนาดเล็ก ออกขายในท้องตลาดและเป็นที่นิยมใช้กันจนถึงปัจจุบัน

2.2.2 ที่มาและวิวัฒนาการของเครื่องควบคุมแบบตรรกที่โปรแกรมได้

การควบคุมแบบลำดับขั้น (Sequence Control) เป็นการควบคุมแบบหนึ่งที่ต้องการให้เครื่องจักรหรือระบบการทำงานตามช่วงเวลา ตามลำดับขั้นตอน ตลอดจนตามเงื่อนไขที่ได้กำหนดไว้โดยมีลักษณะของการควบคุมเป็นแบบ "ON" หรือ "OFF" การควบคุมแบบลำดับขั้นหรือแบบซีเควนซ์นี้จะพบ เห็นอยู่เสมอในงานอุตสาหกรรมแทบทุกชนิด ตัวอย่างของการควบคุมแบบนี้ได้แก่ ระบบควบคุมเครื่องจักร ระบบล้างรถอัตโนมัติ ระบบป้อนวัสดุ ระบบการผสมวัสดุ ระบบควบคุมลิฟท์ เป็นต้น

ในอดีตที่ผ่านมา ระบบควบคุมแบบซีเควนซ์จะใช้อุปกรณ์ไฟฟ้า เชิงกล (Electromechanical Device) เช่น รีเลย์แม่เหล็กไฟฟ้า (Relay) ตัวตั้งเวลา (Timer) ตัวนับ (Counter) มาประกอบกันเป็นวงจรควบคุมเพื่อให้เครื่องจักรหรือระบบกระบวนการทำงานตามช่วงเวลา ตามลำดับขั้นตอน และตามเงื่อนไข ที่วิศวกรหรือผู้ออกแบบระบบกำหนดได้ อย่างไรก็ตามระบบควบคุมแบบซีเควนซ์ที่ใช้อุปกรณ์ไฟฟ้าชิงกลที่กล่าวมานี้ มีข้อเสียมาก คือขนาดของวงจรควบคุมจะมีขนาดใหญ่ สิ้นเปลืองทั้งเนื้อที่และพลังงานสูง ราคาแพง ไม่สามารถจะใช้กับระบบควบคุมที่มีขั้นตอนการทำงานยุ่งยากซับซ้อน เมื่อมีปัญหาเกิดขึ้นในวงจรควบคุมก็ตรวจสอบแก้ไขยาก การขยายระบบทำได้ยาก และที่สำคัญ คือถ้าต้องการเปลี่ยนแปลงลำดับขั้นตอนหรือเงื่อนไขของการทำงาน วงจรควบคุมแบบรีเลย์ที่มีอยู่เดิม จะเปลี่ยนแปลงเพื่อใช้กับงานใหม่ได้ยาก ต่อมาเมื่ออุปกรณ์สารกึ่งตัวนำ (Solid State Device) ได้แพร่หลาย และก้าวหน้าขึ้นจึงได้มีการนำเอาเกท (Gate) ต่าง ๆ มาประกอบกันเป็นวงจรควบคุมแทนรีเลย์และสามารถลดข้อเสียของระบบเดิมลงไปได้มาก เช่น วงจรควบคุมแทนรีเลย์ และสามารถลดข้อเสียการขยายระบบทำได้ง่ายขึ้น และยังสามารถต่อเข้ากับคอมพิวเตอร์ เพื่อการเก็บข้อมูลและอื่น ๆ ได้เป็นต้น แต่ก็ยังไม่สามารถจะลดข้อเสียบางอย่างที่สำคัญลงได้ นั่นคือการตรวจสอบแก้ไข เมื่อมีปัญหาก็ยังทำได้ยาก และไม่สามารถจะเปลี่ยนแปลงลำดับขั้นตอน หรือเงื่อนไขของการทำงาน จึงได้มีการสร้างเครื่องควบคุมที่

สามารถกำหนดโปรแกรมการทำงานของ (Programmable Controller) ขึ้นมาเพื่อให้นำมาประยุกต์ใช้กับระบบควบคุมแบบซีเควนซ์ได้ทุกระบบ โดยไม่ต้องเปลี่ยนแปลงแก้ไขวงจรควบคุม

2.2.2.1 วิวัฒนาการของเครื่องควบคุมแบบตรรกที่โปรแกรมได้

PLC หรือ Programmable Logic Controllers และ PC หรือ Programmable Controller เป็นเครื่องควบคุมที่สามารถกำหนดโปรแกรมการทำงานได้ PLC/PCที่ใช้ไมโครโปรเซสเซอร์นอกจากจะใช้ในการควบคุมเครื่องจักรโดยทั่วไปแล้ว ยังมีการพัฒนาให้มีความสามารถและขอบเขตของงานได้กว้างขึ้น เช่น มีฟังก์ชันทางคณิตศาสตร์เพิ่มขึ้น ทำให้การควบคุมเป็นได้ทั้งแบบ ON-OFF หรือแบบอนาล็อก (Analog) เช่น PID (Progortional +Integral +Derivative) สามารถต่อเข้ากับอุปกรณ์วัดและควบคุม (Instrumentation) อื่น ๆ จำนวนของอินพุท / เอาท์พุท (Input/Output) ขยายให้มากขึ้นได้ หรือการใช้อินพุท/เอาท์พุทแบบรีโมท (Remote) เพื่อลดการเดินสาย การติดต่อสื่อสารข้อมูลกับคอมพิวเตอร์ ตลอดจนอุปกรณ์พิเศษอื่น ๆ เช่น เครื่องอ่านรหัสแถบเพื่อจำแนกหรือ จัดทำฐานข้อมูล การเชื่อมโยงข้อมูลลักษณะโครงข่ายท้องถิ่น (LAN) เป็นต้น หน่วยความจำก็ขยายได้มาก ทำให้ใช้กับระบบกระบวนการใหญ่ ๆ ได้ และมีการจัดการเกี่ยวกับข้อมูลได้ตามต้องการ

2.2.2.2 สรุปข้อดีของ PLC/PC

ข้อดีของ PLC/PC สามารถจำแนกออกได้เป็นข้อ ๆ ดังนี้

1. สิ้นเปลืองเนื้อที่น้อยเพราะมีขนาดเล็ก
2. สามารถจะใช้ควบคุมเครื่องจักรหรือระบบกระบวนการใด ๆ ก็ได้ ถ้าเลือกขนาดของ PLC/PCV ที่เหมาะสม
3. การเปลี่ยนลำดับขั้นตอนหรือเงื่อนไขของการทำงานก็ทำได้ตามต้องการ เพราะใช้หลักของการโปรแกรม
4. ตัวตั้งเวลาและตัวนับจะเป็นซอฟท์แวร์ ทำให้การกำหนดค่าต่าง ๆ ง่าย เปลี่ยนแปลงค่าได้ตลอดเวลาไม่ต้องมีฮาร์ดแวร์ร่วม และทำให้ราคาถูกลง
5. รีเลย์ภายใน (Interanl relay) ก็เป็นซอฟท์แวร์เช่นเดียวกัน ลดค่าใช้จ่ายในการเดินสาย ลดฮาร์ดแวร์ และทำให้ขนาดเล็กลงด้วย
6. การติดตั้งทำได้ง่ายและสะดวก
7. การขยายระบบให้ใหญ่ขึ้นทำได้โดยง่าย
8. ราคาถูกกว่าระบบรีเลย์
9. ความน่าเชื่อถือ (Reliability) ดีเพราะเป็นอุปกรณ์สารกึ่งตัวนำ ไม่มีการเดินสายมาก ไม่มีปัญหาเกี่ยวกับหน้าสัมผัส (Contact) แบบรีเลย์

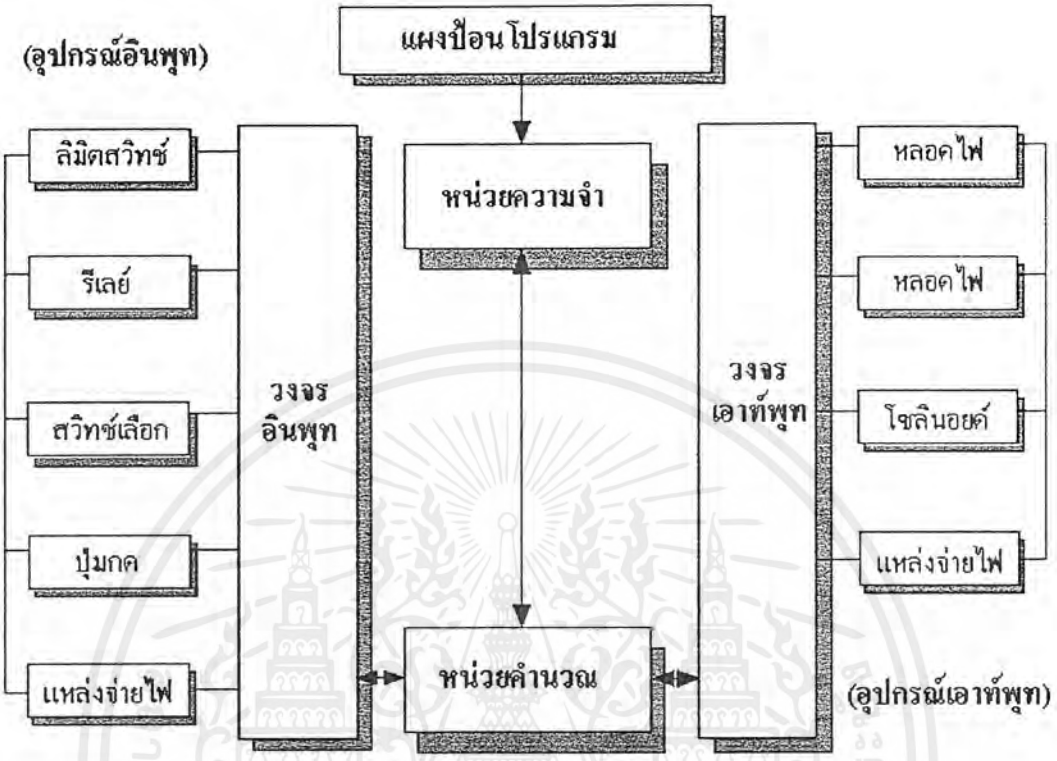
10. มีระบบตรวจสอบหาที่ผิดพลาดด้วยตัวเอง การตรวจสอบแก้ไขเมื่อมีปัญหาจึงทำได้รวดเร็ว
11. ลดการเดินสายยาว ๆ และลดค่าใช้จ่ายในการเดินสายได้ เพราะมีอินพุท เอาท์พุทแบบรีโมท
12. การบำรุงรักษาทำได้ง่าย
13. เวลาในการทำงานเร็วกว่าระบบที่ใช้รีเลย์
14. มีฟังก์ชันทางคณิตศาสตร์ได้แก่ บวก ลบ คูณ หาร และอื่น ๆ ทำให้สามารถใช้สำหรับการควบคุมแบบ ON-OFF หรือแบบอนาล็อก เช่น PID ได้
15. สามารถเชื่อมต่อ กับอุปกรณ์การวัด เช่น เทอร์โมคัปเปิล (Thermocouple) และอื่น ๆ ได้ นอกเหนือจากอุปกรณ์ตรวจวัดที่เป็นสวิทช์
16. ต่อเข้ากับคอมพิวเตอร์ เพื่อวัตถุประสงค์ใด ๆ เช่น การเก็บข้อมูลการกระจายการควบคุม (Distributed Control) เป็นต้น
17. การโปรแกรมทำได้หลายแบบ เช่น คำสั่งในรูปของแลดเดอร์ไดอะแกรม คำสั่งบูลีน (Boolean Instruction) คำสั่งในรูปบล็อค (Block Instruction) หรือ คำสั่งภาษาเบสิค
18. ใช้ได้ในทุกสภาพแวดล้อมของงานอุตสาหกรรม
19. การโปรแกรมทำได้โดยใช้เครื่องป้อนโปรแกรม (Program Loader) หรือโปรแกรมลงบน CRT

จากที่กล่าวมานี้จะเห็นว่า PLC/PC เป็นเครื่องควบคุมที่มีประสิทธิภาพและมีประโยชน์อย่างยิ่งต่องานอุตสาหกรรมในปัจจุบันและในอนาคต

2.2.3 โครงสร้างและหลักการทำงานของเครื่องควบคุมแบบตรรกที่โปรแกรมได้

เครื่องควบคุมแบบตรรกที่โปรแกรมได้ หรือ PLC/PC เป็นอุปกรณ์สารกึ่งตัวนำ ที่ใช้สำหรับควบคุมเครื่องจักรหรือกระบวนการให้ทำงานตามโปรแกรมคำสั่งของผู้ใช้ (User Program) และข้อมูลต่าง ๆ ที่ได้รับจากอินพุท/เอาท์พุทของ PLC/PC จะเป็นได้ทั้งการทำงานตามช่วงเวลาและ ตามลำดับขั้นตอนฟังก์ชันทางคณิตศาสตร์ และอื่น ๆ PCL/PC มีส่วนประกอบสำคัญ 4 ส่วน คือ

1. หน่วยประมวลผลกลางคือ CPU (Central Processing Unit)
2. หน่วยจ่ายกำลัง (Power Supply)
3. หน่วยอินพุท/เอาท์พุท (Input/Output Unit)
4. หน่วยป้อนโปรแกรม (Programming Unit)



รูปที่2.8 โครงสร้างของPLC

2.2.3.1 หน่วยประมวลผลกลาง

หน่วยประมวลผลกลางหรือ CPU ประกอบด้วย 3 ส่วนคือ

3.3.1.1 หน่วยประมวลผลกลาง (Processor)

3.3.1.2 หน่วยความจำ (Memory)

3.3.1.3 หน่วยจ่ายกำลัง (Power Supply)

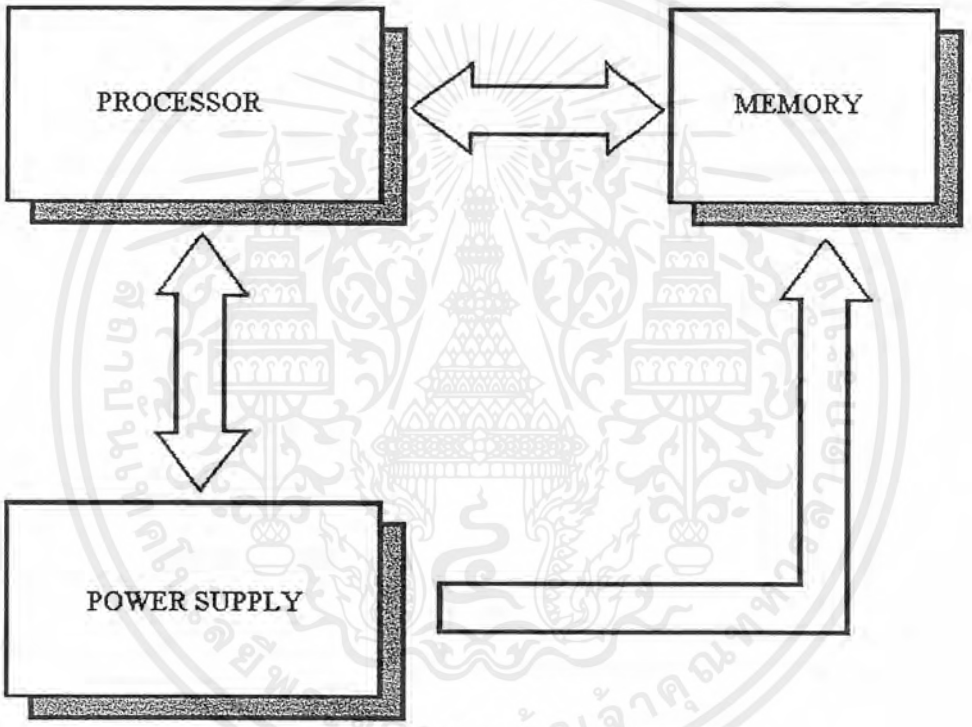
รูปที่ 3-2 แสดงส่วนประกอบของ CPU โดยทั่วไป CPU จะใช้ไมโครโปรเซสเซอร์ เป็นวงจรรวม (I.C. : Integrated Circuit) ที่มีความสามารถทั้งในการคำนวณทางคณิตศาสตร์ การจัดการข้อมูล และการตรวจสอบตัวเอง ในขณะที่วงจรควบคุมที่ใช้รีเลย์หรือไอซีพวกเกทต่าง ๆ ไม่สามารถที่จะทำได้ PLC/PC จึงมีข้อดีและประโยชน์มากกว่าระบบแบบเก่าอย่างเห็นได้ชัด

หน่วยความจำของผู้ใช้จะแตกต่างกันตามขนาดของ PLC/PC ใน PLC หรือ PC ขนาดใหญ่ ขนาดของหน่วยความจำดังกล่าวจะสามารถเปลี่ยนแปลงขนาดได้ตามต้องการ การเลือกใช้ PLC/PCจึงต้องคำนึงถึงขีดความสามารถและขีดจำกัดต่างๆประกอบด้วยเช่นขนาดโปรแกรมคำสั่ง ของผู้ใช้จำนวนอินพุท/เอาท์พุท ที่จะขยายได้สูงสุด ขนาดของหน่วยความจำภายในที่

ใช้เก็บข้อมูล และฟังก์ชันพิเศษต่าง ๆ เช่น รีเลย์ภายใน ตัวตั้งเวลา และตัวนับ

2.2.3.2 หน่วยจ่ายกำลัง

หน่วยจ่ายกำลังจะทำหน้าที่จ่ายและรักษาระดับแรงดันไฟฟ้ากระแสตรง (D.C. Voltage) ให้กับหน่วยประมวลผล หน่วยความจำ และหน่วยอินพุท/เอาท์พุทตามความต้องการและทำหน้าที่เตือนให้หน่วยประมวลผลทราบเมื่อเกิดปัญหา



รูปที่ 2.9 บล็อคไดอะแกรม

2.2.3.3 หน่วยอินพุท/เอาท์พุท

หน่วยอินพุททำหน้าที่เชื่อมต่อระหว่าง CPU กับอุปกรณ์ภายนอกโดยรับค่าสภาวะ หรือปริมาณทางกายภาพต่าง ๆ จากอุปกรณ์ตรวจวัด (Sensor) ของเครื่องจักรหรือกระบวนการ อาทิเช่น ลิมิตสวิทช์ (Limit Switch) พร็อกซิมิตี้สวิทช์ (Proximity Switch) ตำแหน่ง อุณหภูมิ ระดับ แรงดัน กระแสไฟ และอื่น ๆ ส่งไปยัง CPU เพื่อประมวลผลตามโปรแกรมคำสั่งของผู้ใช้

หน่วยเอาท์พุททำหน้าที่รับค่าสภาวะหรือคำสั่งควบคุมที่ได้จาก CPU เพื่อส่งไปควบคุมอุปกรณ์ภายในเครื่องจักรหรือกระบวนการเช่น วาล์ว (Valve) มอเตอร์ (Motor) ปั๊ม (Pump) และอื่น ๆ ให้ทำงานตามค่าสภาวะหรือคำสั่งที่ CPU ต้องการ

ดังที่ได้กล่าวมาแล้วว่า ในช่วงแรก PCL/PC ถูกใช้แทนรีเลย์สำหรับการควบคุมแบบ ON-OFF หน่วยอินพุท/เอาท์พุท ในขณะนั้นจึงเป็นแบบที่ใช้สำหรับค่าสภาวะลอจิก (Discrete Input / Output) เท่านั้น แต่ต่อมาเมื่อ PCL/PC ได้ถูกพัฒนาจนมีการควบคุมแบบอนาล็อกรวมอยู่ด้วย โดยใช้หน่วยอินพุท/เอาท์พุท ที่ถูกพัฒนาขึ้นเพื่อรับส่งสัญญาณแบบอนาล็อคโดยเฉพาะ

2.2.3.4 หน่วยป้อนโปรแกรม

หน่วยป้อนโปรแกรมของ PLC/PC ทำหน้าที่ติดต่อระหว่างผู้ใช้ และ PLC/PC ทั้งระบบ คือ การป้อนโปรแกรมคำสั่งของผู้ใช้เข้าสู่หน่วยความจำของ CPU ตรวจสอบสภาวะการทำงาน รับ-ส่งข้อมูลระหว่าง CPU กับหน่วยป้อนโปรแกรม รับ-ส่งข้อมูลระหว่าง CPU กับเทป หรืออื่น ๆ ตามฟังก์ชันการใช้งานที่มีอยู่ หน่วยป้อนโปรแกรมของ PLC/PC แบ่งออกได้หลายชนิดคือ

2.2.3.4.1 เครื่องป้อนโปรแกรมแบบ CRT

หน่วยป้อนโปรแกรม CRT ประกอบด้วยจอภาพและแป้นพิมพ์ แบ่งออกเป็น 2 ชนิด คือ แบบดัมบ์ CRT (Dumb CRT) ซึ่งไม่มีหน่วยประมวลอยู่ภายในตัว การทำงานทุกอย่างจะถูกควบคุมจาก CPU ของ PLC/PC และแบบอินเทลลิเจนต์ CRT (Intelligent CRT) ซึ่งจะมีส่วนประมวลอยู่ภายในตัวการทำงานเป็นอิสระจาก CPU จึงมีประสิทธิภาพดีกว่าแบบดัมบ์

2.2.3.4.2 เครื่องป้อนโปรแกรมขนาดเล็ก (Mini Programmer)

เป็นเครื่องป้อนโปรแกรมขนาดเล็กที่สามารถพกติดตัวและเคลื่อนย้ายไปมาได้สะดวก แต่ประสิทธิภาพจะด้อยกว่าเครื่องป้อนโปรแกรมแบบ CRT ปกติแล้วจะใช้จอ LCD ในการแสดงผล เครื่องป้อนโปรแกรมแบบนี้จะแบ่งออกเป็น 2 ชนิดคือ แบบอินเทลลิเจนต์ และแบบดัมบ์

2.2.3.4.3 เครื่องป้อนโปรแกรมลงในหน่วยความจำของ PLC/PC หรือเทป

เครื่องป้อนโปรแกรมแบบนี้สามารถใช้ป้อนหรือแก้ไขโปรแกรมคำสั่งของผู้ใช้เข้าไปในหน่วยความจำของ PLC/PC ได้โดยตรง หรือใช้ในการรับ-ส่ง โปรแกรมและส่งข้อมูลต่าง ๆ ระหว่าง PLC/PC และเทป

2.2.3.4.4 เครื่องป้อนโปรแกรมลงบนหน่วยความจำ (Memory Burner)

เครื่องป้อนโปรแกรมแบบนี้ทำหน้าที่สำหรับถ่ายโปรแกรมที่มีอยู่ลงเก็บไว้ในหน่วยความจำแบบ ROM เพื่อให้โปรแกรมต่าง ๆ คงอยู่ตลอดเวลา เมื่อโปรแกรมต่าง ๆ ได้ถูกตรวจสอบและแก้ไขจนเรียบร้อยสมบูรณ์แล้ว

2.2.3.4.5 คอมพิวเตอร์

เครื่องควบคุม PLC/PC บางรุ่นได้ออกแบบให้ สามารถติดต่อกับคอมพิวเตอร์ได้ เพื่อให้การโปรแกรมทุกอย่างสามารถทำบนคอมพิวเตอร์ได้

2.2.4 ลักษณะการโปรแกรมให้กับเครื่องควบคุมแบบตรรกที่โปรแกรมได้

ภาษาที่ใช้ในการเขียนโปรแกรมคำสั่งสำหรับ PLC/PC นั้นสามารถจะจำแนกออกได้เป็น 4 ภาษา คือ

1. ภาษาแลดเดอร์ไดอะแกรม
2. ภาษาคำสั่งบูลีน
3. ภาษาคำสั่งในรูปบล็อก
4. ภาษาระดับสูง

ภาษาแลดเดอร์ไดอะแกรมหรือคำสั่งบูลีนเป็นภาษาที่นิยมใช้และเข้าใจ ได้ง่ายสำหรับการเขียนโปรแกรมคำสั่งของPLC/PC ในการควบคุมแบบซีเควนซ์ที่เป็น ON-OFF การเขียนโปรแกรมคำสั่งในภาษาแลดเดอร์ไดอะแกรมหรือคำสั่งบูลีนจึงใช้กันมากใน PLC หรือ PC ทุกขนาด ส่วนภาษาคำสั่งในรูปแบบ บล็อกหรือ ภาษาระดับสูง เช่น ภาษาซี ภาษาปาสคาล ภาษาฟอร์-แทรน นั้นจะเหมาะสำหรับ PLC/PC ในการควบคุมแบบอนาล็อกการจัดการเกี่ยวกับข้อมูลการทำรายงานและอื่น ๆ ซึ่งมักจะเป็น PLC ขนาดใหญ่ PLC แต่ละแบบอาจใช้คำสั่งเพียงภาษาเดียว หรือหลายภาษารวมกัน ในการเขียนโปรแกรมคำสั่งก็ได้ เช่น ภาษาแลดเดอร์ ไดอะแกรม ร่วมกับคำสั่งบูลีน เพียงภาษาเดียวหรือใช้ภาษาแลดเดอร์ไดอะแกรมหรือคำสั่งในรูปบล็อก ภาษาคำสั่งบูลีนร่วมกับคำสั่งในรูปบล็อกหรือภาษาคำสั่งบูลีนร่วมกับภาษาระดับสูง เป็นต้น อย่างไรก็ตามในหัวข้อนี้จะกล่าวถึงการเขียนโปรแกรมคำสั่งของ PLC/PC โดยใช้ภาษาแลดเดอร์ไดอะแกรมและคำสั่งบูลีนเท่านั้น

2.2.4.1 โปรแกรมภาษาแลดเดอร์ไดอะแกรม

สัญลักษณ์พื้นฐานของภาษาแลดเดอร์ไดอะแกรม ดังที่ทราบกันแล้วว่า PLC/.PC เป็นเครื่องควบคุมที่ใช้วิธีการโปรแกรม และใช้ซอฟท์แวร์แทนอุปกรณ์ต่าง ๆ เช่น รีเลย์ ตัวตั้งเวลา ตัวนับ ดังนั้น การเขียนโปรแกรมคำสั่งจึงใช้สัญลักษณ์พื้นฐานต่อไปนี้แทนอุปกรณ์ต่าง ๆ

สัญลักษณ์	ความหมาย
+--] [--+	จะใช้แทนอุปกรณ์ทางด้านอินพุทต่าง ๆ ได้แก่ สวิทช์ ไฟฟ้าหรือหน้าสัมผัสของรีเลย์ ตัวตั้งเวลา ตัวนับ หรือรีเลย์ภายใน เป็นต้น โดยที่อุปกรณ์นั้นปกติจะมีค่า สภาวะเป็นลอจิก "0" หน้าสัมผัสปกติเปิดหรือ N.O.
+--] / [--+	จะใช้แทนอุปกรณ์ทางด้านอินพุทต่าง ๆ ได้แก่ สวิทช์ไฟฟ้าหรือหน้าสัมผัส

	ของรีเลย์ ตัวตั้งเวลา ตัวนับ หรือรีเลย์ภายใน เป็นต้น โดยที่อุปกรณ์นั้นปกติจะมีค่าสภาวะเป็นลอจิก "1"
+--( )--+	เอาท์พุทปกติไม่ทำงานจะใช้แทนอุปกรณ์ทางด้านเอาท์พุทต่าง ๆ ได้แก่ หลอดไฟฟ้า มอเตอร์ไฟฟ้า ขดลวดของรีเลย์ ขดลวดโซลีนอยด์ (Solenoid) ตัวตั้งเวลา ตัวนับ รีเลย์ภายใน เป็นต้น โดยที่ปกติแล้วอุปกรณ์ทางด้านเอาท์พุทจะมีค่าสภาวะเป็นลอจิก "0"
+--(/)--+	เอาท์พุทปกติทำงาน จะใช้แทนอุปกรณ์ทางด้านเอาท์พุทต่าง ๆ ได้แก่ หลอดไฟฟ้า มอเตอร์ไฟฟ้า ขดลวดของรีเลย์ ขดลวดโซลีนอยด์ (Solenoid) ตัวตั้งเวลา ตัวนับ รีเลย์ภายใน เป็นต้น โดยที่ปกติแล้วอุปกรณ์ทางด้านเอาท์พุทจะมีค่าสภาวะเป็นลอจิก "1"

ตารางที่ 2.4 แสดงถึงตัวอย่างของการใช้ฟังก์ชันลอจิก และสัญลักษณ์พื้นฐานในการเขียนโปรแกรมแลดเดอร์ของ PLC/PC

การโปรแกรมด้วยแลดเดอร์ไดอะแกรมจะสะดวกและมีประสิทธิภาพมากแต่ข้อเสียของการโปรแกรมแบบนี้คืออุปกรณ์ที่ใช้ในการเขียนโปรแกรมจะต้องมีหน่วยแสดงผลที่สามารถแสดงอักษรหรืออุปกรณ์ที่ใช้ในการโปรแกรมแบบนี้มักจะใช้จอซีอาร์ที(CRT)แต่ปัจจุบันจะมีซอฟท์แวร์สนับสนุนการเขียนแลดเดอร์บนคอมพิวเตอร์และเชื่อมโยงข้อมูลกับ PLC/PC ทางพอร์ทอนุกรม โดยคอมพิวเตอร์ทำหน้าที่ในการสร้างและแก้ไขแลดเดอร์ไดอะแกรมแล้วทำการแปลเป็นชุดคำสั่งของเครื่องควบคุม นอกจากนั้นแล้วขณะที่เครื่องควบคุมทำงานอยู่ก็สามารถตรวจสอบการทำงานได้โดยการแสดงผลซึ่งจะแสดงเป็นแลดเดอร์ไดอะแกรมและเมื่ออินพุทหรือเอาท์พุทตัวใดมีการเปลี่ยนแปลงสภาวะก็จะมีการเปลี่ยนแปลงตามสภาวะของอินพุทหรือเอาท์พุทตัวนั้น ๆ ด้วย

2.2.4.2 โปรแกรมภาษาคำสั่งแบบบูลีน

การโปรแกรมชนิดนี้มีลักษณะคล้ายกับสัญลักษณ์ของพีชคณิตบูลีน นอกจากจะเป็นสัญลักษณ์ของพีชคณิตบูลีนแล้วก็ยังมีการใช้ฟังก์ชันพิเศษต่าง ๆ อีกมากเช่น ตัวนับ ตัวตั้งเวลา เป็นต้น ภาษาคำสั่งบูลีน จะมีความสัมพันธ์กันกับ ภาษาแลดเดอร์ไดอะแกรม สามารถที่จะแปลความหมายถึงกันได้ดังตัวอย่างในตารางที่ 2.5ซึ่งเป็นชุดคำสั่งบูลีนที่เขียนจากแลดเดอร์ไดอะแกรมในรูปที่ 2.12 การเขียนโปรแกรมเป็นแลดเดอร์ไดอะแกรมสามารถแสดงกระบวนการของโปรแกรมได้ง่ายต่อการเข้าใจ แต่ไม่สะดวกในการป้อนลงบนเครื่องควบคุม จึงมีการแปลงจากแลดเดอร์ไดอะแกรม เป็นภาษาคำสั่งบูลีน เพื่อป้อนลงชุดรับข้อมูลแบบพกพาของเครื่องควบคุม PCL/PC ซึ่งจะกระทำได้สะดวกมากขึ้น

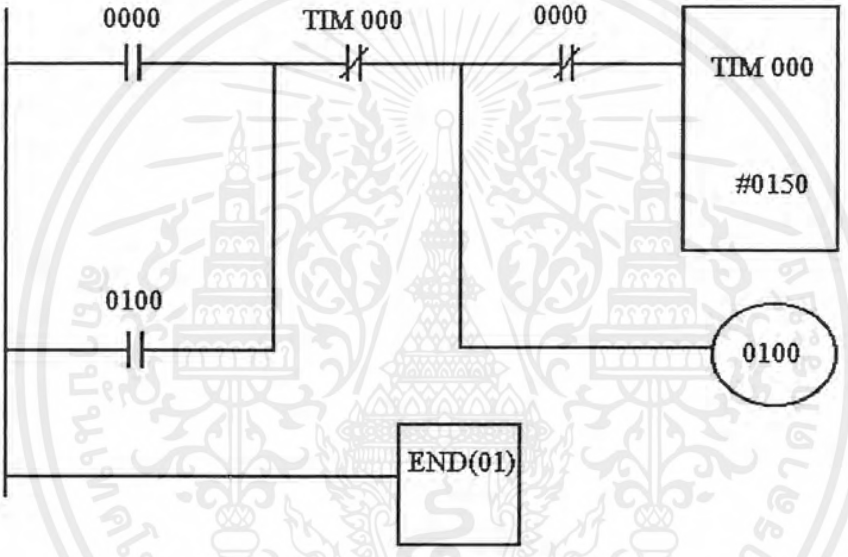
แลดเดอร์ไดอะแกรม เป็นภาษาคำสั่งบูลีน เพื่อป้อนลงชุดรับข้อมูลแบบพกพาของเครื่องควบคุม PCL/PC ซึ่งจะกระทำได้สะดวกมากขึ้น

2.2.5 ตัวอย่างการประยุกต์คำสั่ง

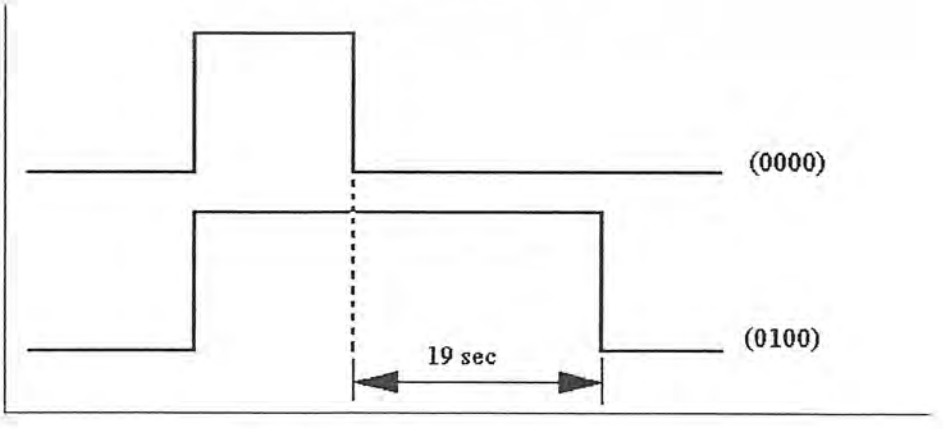
- วงจรตั้งเวลา OFF Delay Timer

2.2.5.1 แลดเดอร์ไดอะแกรม

ตัวอย่างคำสั่งวงจรตั้งเวลา(Off delay time)



รูปที่2.10 แลดเดอร์ไดอะแกรม



รูปที่2.11 ไทม์มิ่งไดอะแกรม

2.2.5.3 คำสั่งภาษาบูลีน

ADDRESS	INSTRUCTION	DATA
0000	LD	0000
0001	OR	0100
0003	AND NOT	TIM 000
0004	AND NOT	0000
0005	TIM	0000
		# 0015 Sec
0006	OUT	0100
0007	END (01)	

ตารางที่ 2.5 ตัวอย่างชุดคำสั่งบูลีน

2.2.5.4 คำอธิบาย

0000 เป็นอินพุทของการเริ่มตั้งเวลา ถ้า 0000 เปลี่ยนจากปิดเป็นเปิด จากนั้น 15 วินาที TIM 000 จะหยุดทำงาน

วงจรตั้งเวลานับจากการ OFF ของหน้าสัมผัส เรียกว่า OFF Delay Timer

2.2.6 การทำงานของ PC

PC จะรับสัญญาณที่เป็นคำสั่งจากสวิตซ์ปุ่มกด สวิตซ์เลือก และสวิตซ์ตัวเลข ซึ่งทั้งหมดอยู่ที่แผงควบคุมเครื่องจักร นอกจากนั้น ยังรับสัญญาณที่มาจากอุปกรณ์ตรวจวัดสภาพการทำงานของเครื่องจักร เช่น ลิมิตสวิตซ์ (limit switch) พล็อกซิมิตีสวิตซ์ (Proximity switch) สวิตซ์แสง และสวิตซ์ตรวจจับชนิดต่าง ๆ จากนั้น PC จะส่งสัญญาณออกไปที่ขั้วออกเพื่อขับเคลื่อนอุปกรณ์ต่าง ๆ เช่น มอเตอร์โซลินอยด์ และคลัตซ์แม่เหล็กไฟฟ้า หรือขับพวกหลอดแสดง หลอดตัวเลข

PC จะรับสัญญาณเข้ามาทางขั้วเข้า และให้สัญญาณออกทางขั้วออก การให้สัญญาณออกนี้จะเป็นไปตามโปรแกรมที่เก็บไว้ในเครื่อง PC การขับอุปกรณ์ ซึ่งเป็นโหลดขนาดเล็ก เช่น โซลินอยด์ตัวเล็ก หรือหลอดแสดงนั้น PC สามารถขับโดยตรงจากขั้วออกได้ แต่ถ้าเป็นอุปกรณ์ที่เป็นโหลดขนาดใหญ่ ใช้ไฟมาก เช่น มอเตอร์สามเฟส หรือโซลินอยด์ตัวใหญ่ จำเป็นต้องต่อผ่าน

แทคเตอร์ และรีเลย์เพื่อช่วยขยายกำลังขับ พวกคอนแทคเตอร์ รีเลย์ขับเบรกเกอร์ เหล่านี้ จะถูกติดตั้งอยู่ภายในตู้ควบคุมเดียวกันกับตัว PC

บทที่ 3

การสร้างและการออกแบบ

ในบทนี้จะกล่าวถึงวิธีการออกแบบเพื่อการจะได้มาของการ Simulate ซึ่งมีขั้นตอนหลัก ๆ 4 ขั้นตอน ดังนี้

3.1 ส่วน Operator

3.2 ส่วน Ladder Diagram

3.3 ส่วนภาษา Boolean

3.4 ส่วน Simulate

3.1 ส่วน Operator

คือ ส่วนหลักในการป้อนข้อมูลให้เกิดการแสดงผล ในส่วน Ladder Diagram ภาษา Boolean และส่วน Simulate เราอาจจะเปรียบส่วน Operaotr นี้เหมือนกับส่วนป้อนโปรแกรมของ PLC

โดยเมื่อ

คลิก	LD	จะเป็นการรับคำ	“LOAD”	เข้าไปในโปรแกรม
คลิก	AND	จะเป็นการรับคำ	“LOAD”	เข้าไปในโปรแกรม
คลิก	OR	จะเป็นการรับคำ	“LOAD”	เข้าไปในโปรแกรม
คลิก	LD_NOT	จะเป็นการรับคำ	“LOAD NOT”	เข้าไปในโปรแกรม
คลิก	AND_NOT	จะเป็นการรับคำ	“AND NOT”	เข้าไปในโปรแกรม
คลิก	OR_NOT	จะเป็นการรับคำ	“OR NOT”	เข้าไปในโปรแกรม
คลิก	AND_LD	จะเป็นการรับคำ	“AND LOAD”	เข้าไปในโปรแกรม
คลิก	OR_LD	จะเป็นการรับคำ	“OR LOAD”	เข้าไปในโปรแกรม
คลิก	TIMER	จะเป็นการรับคำ	“TIMER”	เข้าไปในโปรแกรม
คลิก	COUNTER	จะเป็นการรับคำ	“COUNTER”	เข้าไปในโปรแกรม
คลิก	OUT	จะเป็นการรับคำ	“OUT PUT”	เข้าไปในโปรแกรม

โดยในส่วนนี้จะเป็นการรับค่าเข้าไปในส่วนของ Instruction

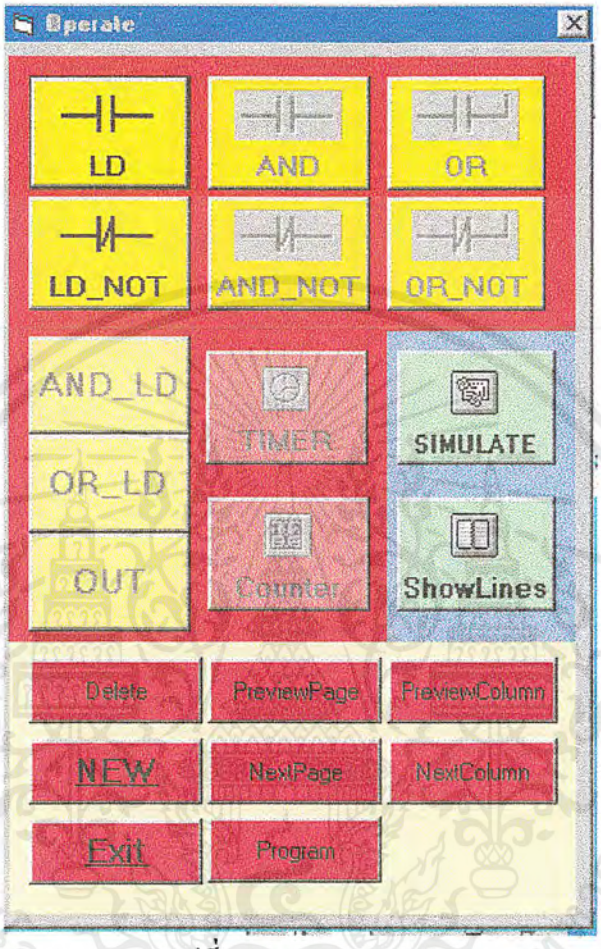
ส่วน DATA จะรับจาก Input Box ซึ่งจะปรากฏทุกครั้งเมื่อมีการคลิกใน Button ต่าง ๆ

เมื่อคลิก Sim จะเป็นการเข้าไปสู่ฟอร์มซิมมูเลท

เมื่อคลิก Deleate จะเป็นการลบข้อมูลที่ป้อนเข้าไปเมื่อท้ายสุด

เมื่อคลิก New เป็นการเริ่มต้นใหม่

เมื่อคลิก Exit เป็นการออกจากโปรแกรม



รูปที่ 3.1 form Operator

3.2 ส่วน Ladder Diagram

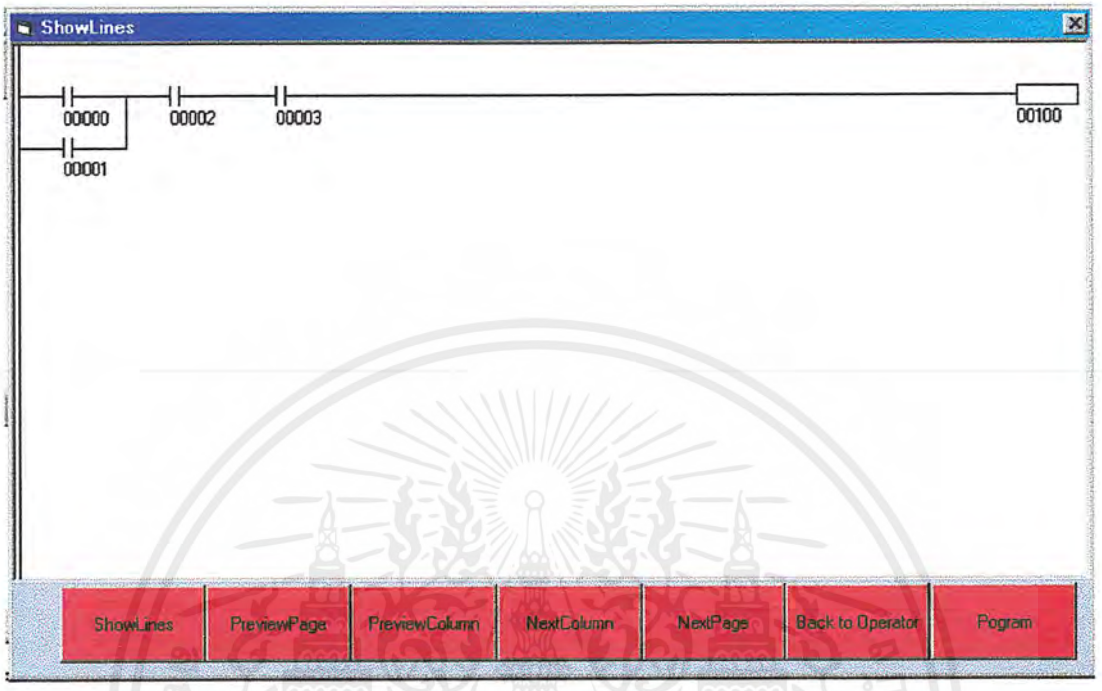
เนื่องจากเป็นส่วนที่ต้องมีรูปภาพประกอบ จากการทดลองในหลาย ๆ วิธีของ Visual-Basic พบว่าการใช้คำสั่ง "Line" ได้ผลดีที่สุด ซึ่งมีรูปคำสั่งดังเช่น

```
Line (x+00 , y+100) – (x+100 , y+100)
```

จาก Block คำสั่งด้านบนอธิบายได้ว่า

จุดเริ่มต้น แกน X ที่ตำแหน่ง 0 แกน Y ที่ตำแหน่งที่ 100

จุดสิ้นสุด แกน X ที่ตำแหน่ง 100 แกน Y ที่ตำแหน่งที่ 100



รูปที่ 3.2 LADDER DIAGRAM แสดงผลคำสั่ง Line

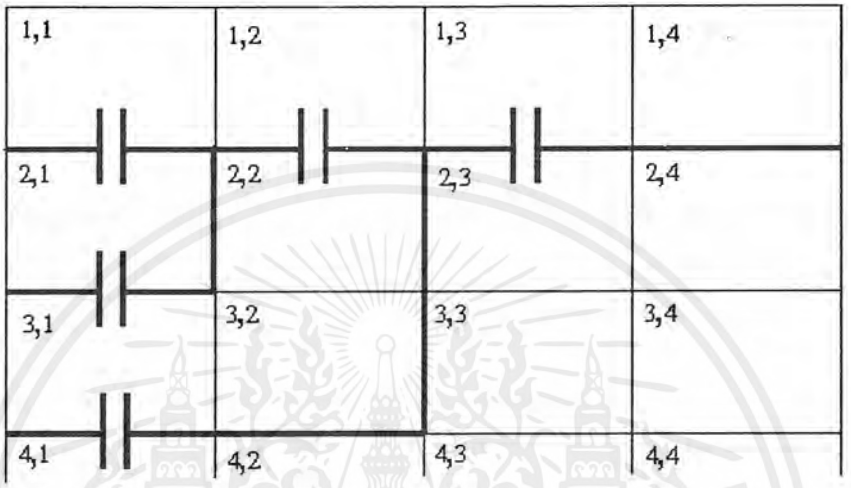
3.2.1 การเขียนรูปลงในฟอร์ม

หลักการ จะใช้หลักการลากเส้นลงบน Property Picture โดยสมมุติ Property Picture ให้เป็นส่วนย่อย ๆ ในแนวนอนให้เป็นแกน x เรียกว่า บรรทัด หรือ คอลัมน์ (Column) ให้มีความกว้าง 500 Twip และในแนวตั้งให้เป็นแกน y หรือเรียกว่า หลัก หรือ Row

จำนวนของหลักมี 10 หลัก จำนวนบรรทัดที่ได้ 1,000 บรรทัด แต่ละส่วนจะสมมุติให้มีเลขกำกับไว้ เช่น บรรทัดที่ 1 แถวที่ 1 ดังรูป

รูปที่3.3 กำหนดภาพบนฟอร์ม

จากหลักการลากเส้นบน Property Picture ก็เพื่อจุดประสงค์หลักคือเป็นตัวกำหนดการวาดภาพ ในช่องว่างเหล่านั้นและสามารถสื่อสารกับโปรแกรมได้ เพื่อจะประกอบกันเป็น Ladder Diagram



รูปที่3.4 แสดงการวางภาพเป็น LADDER DIAGRAM

จากรูปที่ 3.4 จะได้ว่า

บรรทัด ที่ 1 แถว ที่ 1 ใส่รูป
บรรทัด ที่1 แถว ที่ 2 ใส่รูป
บรรทัด ที่ 1 แถว ที่ 3 ใส่รูป
บรรทัด ที่ 1 แถว ที่ 4 ใส่รูป
บรรทัด ที่ 2 แถว ที่ 1 ใส่รูป
บรรทัด ที่ 2 แถว ที่ 2 ใส่รูป
บรรทัด ที่ 2 แถว ที่ 3 ใส่รูป
บรรทัด ที่ 2 แถว ที่ 4 ใส่รูป (ว่าง)
บรรทัด ที่ 3 แถว ที่ 1 ใส่รูป
บรรทัด ที่ 3 แถว ที่ 2 ใส่รูป
บรรทัด ที่ 3 แถว ที่ 3 ใส่รูป
บรรทัด ที่ 3 แถว ที่ 4 ใส่รูป (ว่าง)

3.2.2 องค์ประกอบหลัก ๆ ในการสร้าง Ladder Diagram

1. ทุกครั้งในการเริ่มโปรแกรม จะต้องเริ่มจากสภาวะ "LOAD" หรือ "LOAD NOT" เสมอ ซึ่งจะได้จาก การคลิก "LOAD" ที่ฟอร์ม Operate และจะเป็นผลให้ฟอร์ม LADDER นั้นแสดงภาพ รีเลย์ LOAD โดยใช้คำสั่ง Line กำหนดค่าเริ่มต้น ดังนี้

เส้นที่ 1 ———

(x ที่ตำแหน่ง 0 , y ที่ตำแหน่ง 250) – (x ที่ตำแหน่ง 475 , y ที่ตำแหน่ง 250)

เส้นที่ 2 |

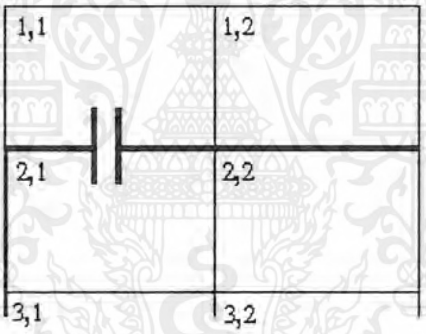
(x ที่ตำแหน่ง 475 , y ที่ตำแหน่ง 150) – (x ที่ตำแหน่ง 475 , y ที่ตำแหน่ง 350)

เส้นที่ 3 |

(x ที่ตำแหน่ง 525 , y ที่ตำแหน่ง 150) – (x ที่ตำแหน่ง 525 , y ที่ตำแหน่ง 350)

เส้นที่ 4 ———

(x ที่ตำแหน่ง 525 , y ที่ตำแหน่ง 250) – (x ที่ตำแหน่ง 1000 , y ที่ตำแหน่ง 250)

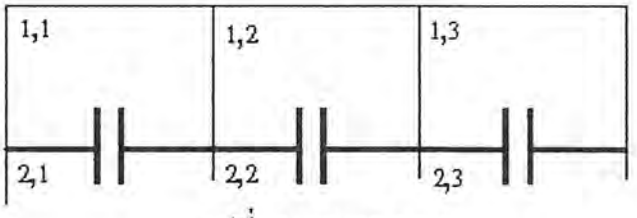


รูปที่3.5 สภาวะ Load

2. สภาวะ And

หลักการ รับค่า สภาวะ AND โดยการคลิก "AND" จากฟอร์ม Operate มากำหนดค่าให้แสดงภาพ รีเลย์ "AND" โดย

เพิ่มค่าในแกน x ทีละ 1000 Twip ทุกครั้งมีการ คลิก "AND" ส่วนค่าแกน y คงค่าไว้ที่ค่าเดิม

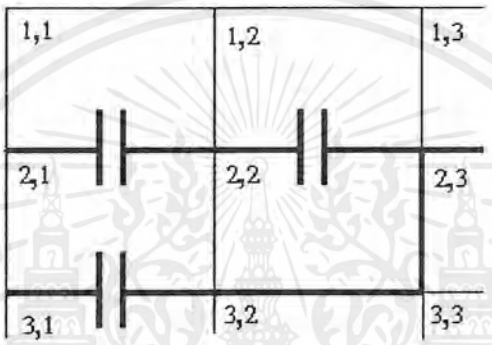


รูปที่3.6 สภาวะ AND

3. สภาวะ OR

หลักการ รับค่า สภาวะ OR โดยการคลิก "OR" จากฟอร์ม Operate มากำหนดให้แสดงภาพ รีเลย์ "OR" โดยสร้างอะเรย์ขึ้นมาจดจำว่ารีเลย์ "OR" ตัวนี้ขึ้นกับ รีเลย์ "LOAD" ตัวใด และจะจดจำค่าแกน x ของ รีเลย์ "LOAD" ตัวนั้นไว้ แล้วเพิ่มค่าในแกน y ที่ละ 500 Twip

จากรูปที่ 3.7 รีเลย์ "OR" ที่ตำแหน่ง (2,1) ขึ้นกับ รีเลย์ "LOAD" ที่ตำแหน่ง (1,1) จึงเพิ่มค่าในแกน y 500 Twip ก็จะได้ภาพ รีเลย์ "OR"



รูปที่3.7 สภาวะ Or

4. สภาวะ LD NOT

หลักการ รับค่าสภาวะ "LOAD NOT" โดยคลิก "LD_NOT" จากฟอร์ม Operate มากำหนดให้แสดงภาพ รีเลย์ "LOAD NOT" โดยใช้คำสั่ง LINE กำหนดค่าเริ่มต้น ดังนี้

เส้นที่ 1 ———

(x ที่ตำแหน่ง 0 , y ที่ตำแหน่ง 250) – (x ที่ตำแหน่ง 475 , y ที่ตำแหน่ง 250)

เส้นที่ 2 |

(x ที่ตำแหน่ง 475 , y ที่ตำแหน่ง 150) – (x ที่ตำแหน่ง 475 , y ที่ตำแหน่ง 350)

เส้นที่ 3 /

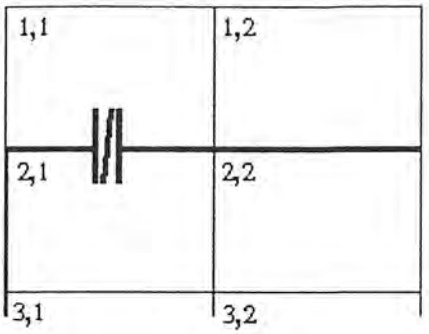
(x ที่ตำแหน่ง 480 , y ที่ตำแหน่ง 350) – (x ที่ตำแหน่ง 520 , y ที่ตำแหน่ง 150)

เส้นที่ 4 |

(x ที่ตำแหน่ง 525 , y ที่ตำแหน่ง 150) – (x ที่ตำแหน่ง 525 , y ที่ตำแหน่ง 350)

เส้นที่ 5 ———

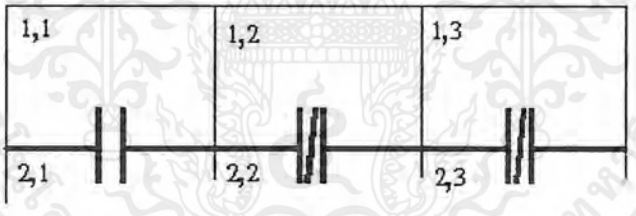
(x ที่ตำแหน่ง 525 , y ที่ตำแหน่ง 250) – (x ที่ตำแหน่ง 1000 , y ที่ตำแหน่ง 250)



รูปที่3.8 สภาวะ Load Not

5. สภาวะ AND NOT

หลักการ รับค่า สภาวะ "AND NOT" โดยการคลิก "AND_NOT" จากฟอร์ม Operate มากำหนดค่าให้แสดงภาพ รีเลย์ "AND NOT" โดยเพิ่มค่าในแกน x ทีละ 1000 Twip ทุกครั้งมีการคลิก "AND NOT" ส่วนค่าแกน y คงค่าไว้ที่ค่าเดิม

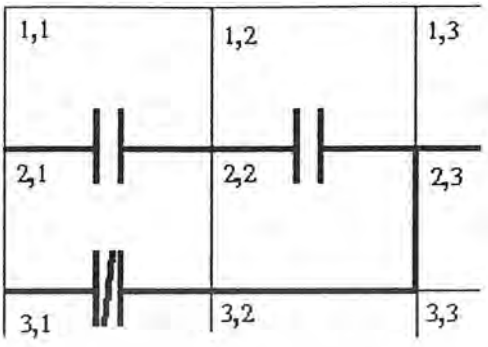


รูปที่ 3.9 สภาวะ And Not

6. สภาวะ OR NOT

หลักการ รับค่าสภาวะ "OR NOT" โดยคลิก "OR" จากฟอร์ม Operate มากำหนดให้แสดงภาพรีเลย์ "OR NOT" โดยสร้างอะเรย์ขึ้นมาจดจำรีเลย์ "OR NOT" ตัวนี้ ขึ้นกับรีเลย์ "LOAD" หรือ รีเลย์ "LOAD NOT" ตัวไหน และจดจำค่าแกน x ไว้ แล้วจึงเพิ่มค่าในแกน y ทีละ 500 Twip

จากรูปที่ 3.10 รีเลย์ "OR NOT" ที่ตำแหน่ง (2,1) ขึ้นกับรีเลย์ "LOAD" ที่ตำแหน่ง (1,1) มันจึงเพิ่มค่าในแกน y 500 Twip ส่วนแกน x ค่าคงเดิมไว้



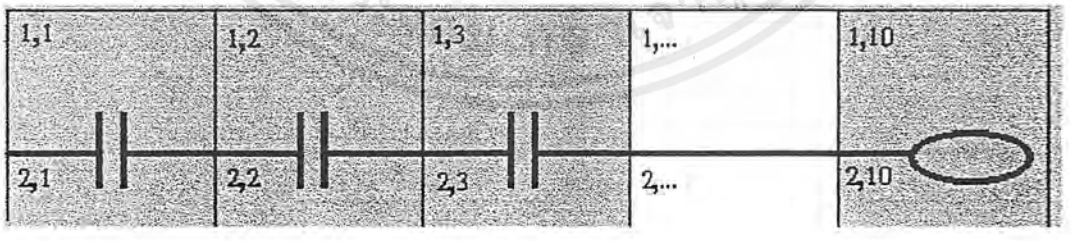
รูปที่3.10 สภาวะ Or Not

7. เส้นตรงในแนวแกน X

เนื่องจากสภาวะ OUT จะถูกกำหนดไว้ที่ Twip 10,000 เสมอ ในแกน x และจะแปรค่าไปในแกน y ดังนั้น ถ้าสภาวะรีเลย์มีเพียงไม่กี่ช่องจะทำให้ภาพที่ได้ขาดหายไป ดังนั้นจึงต้องใช้เส้นตรงแกน x ในการต่อเชื่อม เพื่อความสวยงาม

หลักการ เมื่อที่ตำแหน่งแกน y ใด ๆ มี "OUT" , "TIMER" หรือ "COUNTER" อยู่ให้ลากเส้นย้อนกลับไปในแกน x ณ ตำแหน่งสุดท้ายที่มีภาพรีเลย์อยู่

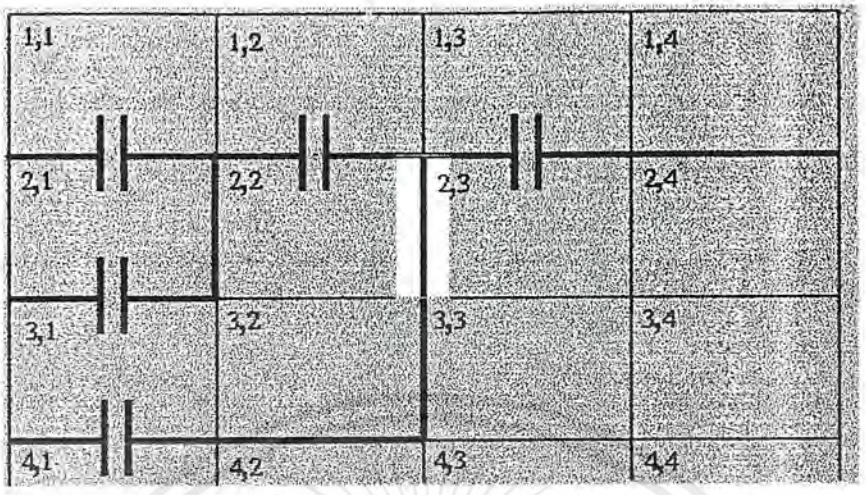
จากรูปที่ 3.11 ที่ตำแหน่ง (1,10) มีรีเลย์ "OUT" และตำแหน่งสุดท้ายที่มีรีเลย์คือ (1,3) ดังนั้นจึงเกิดการลากเส้นจาก (1,9) มาสิ้นสุดที่ (1,4)



รูปที่ 3.11 เส้นตรง แกน X

8. เส้นตรงในแกน Y

ใช้เชื่อมต่อในสภาวะที่การ OR ของ รีเลย์ และยังเชื่อมไปถึงกัน โดยอาศัยการจดจำค่า x และ y สุดท้ายของเหตุการณ์ที่เกิดการ AND และ OR

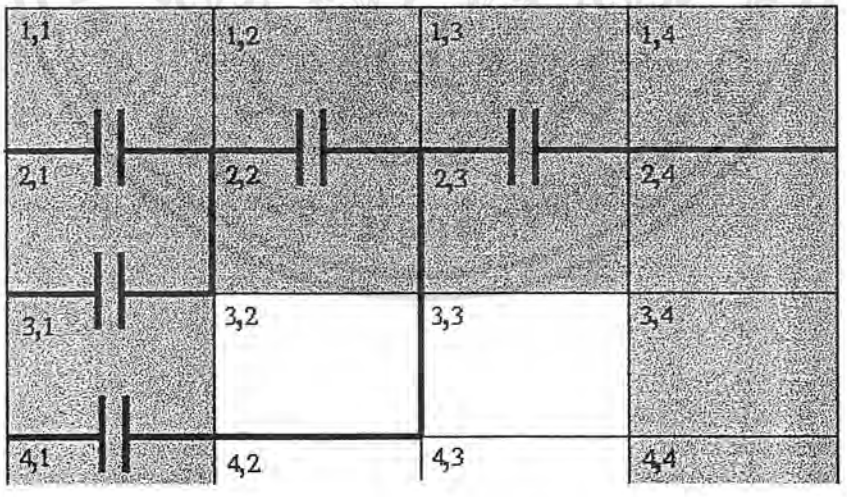


รูปที่3-12 เส้นตรงแกน Y

9. เส้นตรง x to y

ใช้เชื่อมต่อสภาวะที่การ "OR" ในทุกครั้ง

หลักการ สร้างอะเรย์ในการจดจำค่าว่ามีการ "OR" หรือไม่ ถ้ามีจะต้องมีเส้น x to y ตามมาเสมอ

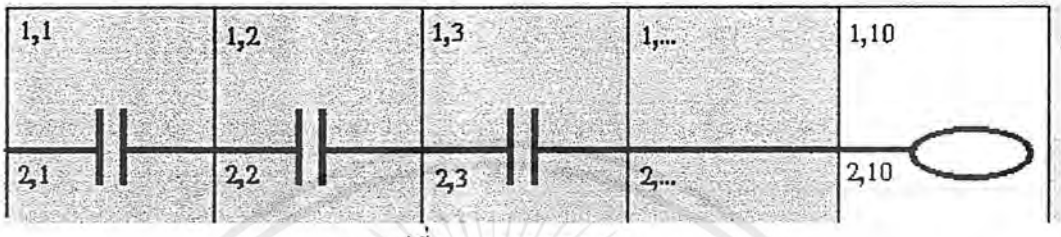


รูปที่ 3.13 เส้นตรง แกน X to Y

10. สภาวะ out

จะถูกกำหนดไว้ที่ Twip ในแกน y ที่ 5,000 เสมอ เพื่อความสวยงามและสะดวกในการตรวจเช็ค

หลักการ รับค่าสภาวะ "OUT" โดยการคลิก "OUT" จากฟอร์ม Operate มากำหนดให้แสดงภาพรีเลย์ "OUT" โดยสร้างอะเรย์จดจำว่ารีเลย์ "OUT" ขึ้นกับรีเลย์ "LOAD" ตัวใดและจดจำค่าแกน y ของรีเลย์ "LOAD" ตัวนั้นไว้ ส่วนในแกน x จะกำหนดให้อยู่ในตำแหน่งสุดท้ายเท่านั้น



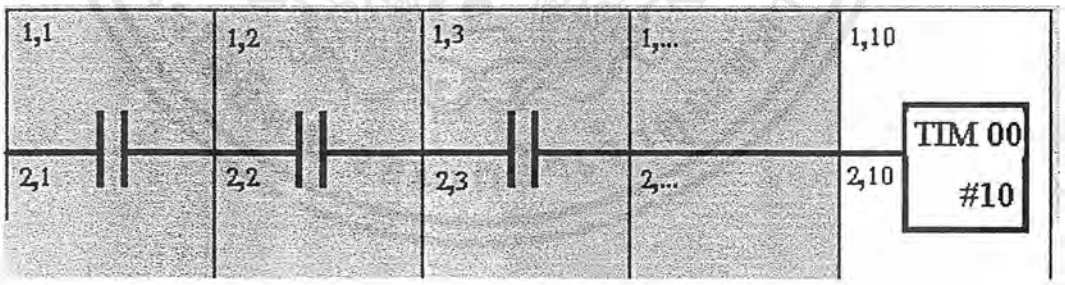
รูปที่3.14 OUT PUT

จากรูปที่ 3.14 รีเลย์ "LOAD" อยู่ที่ตำแหน่ง (1,1) ดังนั้นอะเรย์จะจดจำค่าที่ตำแหน่ง y ที่ 1 ดังนั้น จะได้ภาพรีเลย์ "OUT" ที่ตำแหน่ง (1,10)

11. Timmer

ถูกกำหนดใน Twip ในแกน y ที่ 10,000 เช่น เดียวกับ OUT

หลักการ รับค่าสภาวะ "TIMER" โดยการคลิก "TIMER" จากฟอร์ม Operate มากำหนดให้แสดงภาพรีเลย์ "TIMER" โดยสร้างอะเรย์จดจำว่ารีเลย์ "TIMER" ขึ้นกับรีเลย์ "LOAD" ตัวใดและจดจำค่าแกน y ของรีเลย์ "LOAD" ตัวนั้นไว้ ส่วนในแกน x จะกำหนดให้อยู่ในตำแหน่งสุดท้ายเท่านั้น



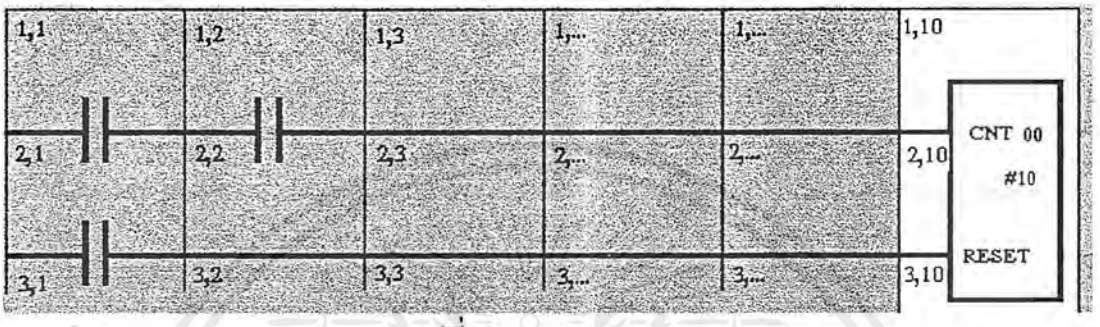
รูปที่ 3.15 TIMER

จากรูปที่ 3.15 รีเลย์ "LOAD" อยู่ที่ตำแหน่ง (1,1) ดังนั้นอะเรย์จะจดจำค่าที่ตำแหน่ง y ที่ 1 ดังนั้น จะได้ภาพรีเลย์ "TIMER" ที่ตำแหน่ง (1,10)

12. COUNTER

ถูกกำหนดใน Twip ในแกน y ที่ 10,000 เช่น เดียวกับ OUT

หลักการ รับค่าสภาวะ "COUNTER" โดยการคลิก "COUNTER" จากฟอร์ม Operate มา กำหนดให้แสดงภาพรีเลย์ "COUNTER" โดยสร้างอะเรย์จดจำว่ารีเลย์ "COUNTER" ขึ้นกับรีเลย์ "LOAD" ตัวใด และจดจำค่าแกน y ของรีเลย์ "LOAD" ตัวนั้นไว้ ส่วนในแกน x จะกำหนดให้อยู่ในตำแหน่งสุดท้ายเท่านั้น

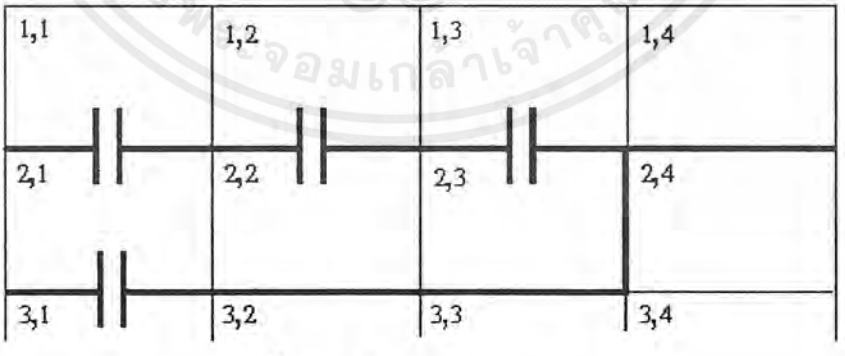


รูปที่3.16 COUNTER

จากรูปที่ 3.16 รีเลย์ "LOAD" อยู่ที่ตำแหน่ง (1,1) ดังนั้นอะเรย์จะจดจำค่าที่ตำแหน่ง y ที่ 1 ดังนั้น จะได้ภาพรีเลย์ "COUNTER" ที่ตำแหน่ง (1,10)

3.2.3 การกำหนดภาพให้เป็นตัวแปร

ภาพที่เราวางใน Dynamic Array จะไม่มีค่าอะไรเลย หากภาพเหล่านั้นวางไว้ได้โดยไม่สามารถนำมาคำนวณได้ ซึ่งที่มาของการคำนวณนี้ก็ได้มาจาก การกำหนดค่าให้ภาพเหล่านั้นก่อนโดยเก็บเป็นตัวแปร



รูปที่ 3.17 ตัวอย่าง Ladder Diagram

จากรูป จะสามารถเก็บตัวแปรได้ ในที่นี้กำหนด ชื่อตัวแปรว่า Component Incolumn (1000,10) As String

จากรูป

Component Incolumn (1,1) = รีเลย์ "LOAD"

Component Incolumn (1,2) = รีเลย์ "AND"

Component Incolumn (1,3) = รีเลย์ "AND"

Component Incolumn (1,4) = "Line In X Axis"

Component Incolumn (2,1) = "OR"

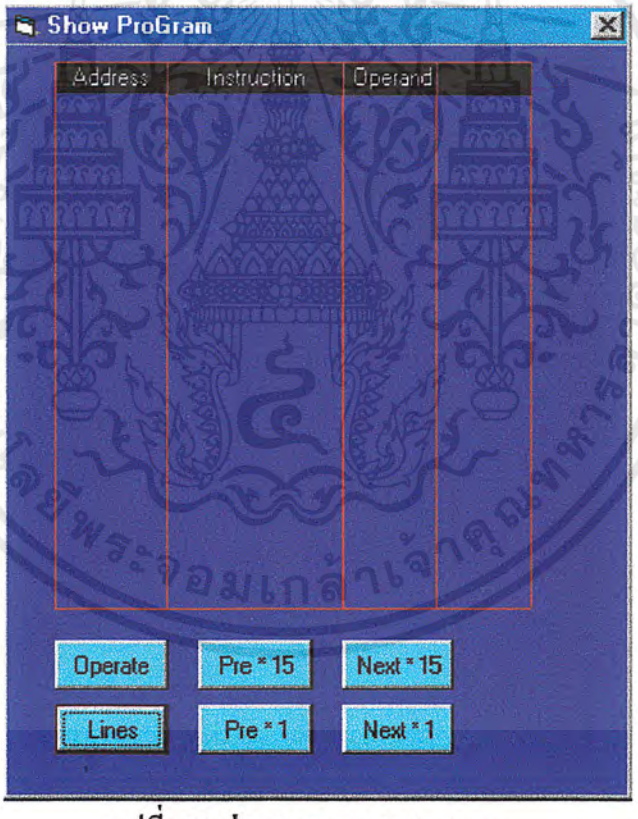
Component Incolumn (2,2) = "Line in X Axis"

Component Incolumn (2,3) = "Line from x to y"

Component Incolumn (2,4) = "Line from x to y"

3.3ส่วนภาษา Boolean

เป็นส่วนที่รับค่ามาจาก frm Operator เช่นเดียวกับส่วน Ladder Diagram มีลักษณะการทำงานดังนี้



รูปที่3.18 ส่วนแสดงภาษา Boolean

3.3.1 ส่วน Address กำหนดค่า array I = I + 1 ซึ่งก่อนใช้งานจะ Clear ค่า I ให้อยู่ที่ 0 เสมอ

3.3.2 Instruction

- สภาวะ LOAD รับค่าจากการเหตุการณ์ คลิก LOAD

- สภาวะ AND รับค่าจากการเหตุการณ์ คลิก AND
- สภาวะ OR รับค่าจากการเหตุการณ์ คลิก OR
- สภาวะ LOAD NOT รับค่าจากการเหตุการณ์ คลิก LD NOT
- สภาวะ AND NOT รับค่าจากการเหตุการณ์ คลิก AND NOT
- สภาวะ OR NOT รับค่าจากการเหตุการณ์ คลิก OR NOT
- สภาวะ OUT รับค่าจากการเหตุการณ์ คลิก OUT

3.3.3 ส่วนDATA

จะรับค่ามาจากส่วน INPUT BOX

ทั้งหมดที่กล่าวมาข้างต้นเป็นเพียงวิธีการโดยย่อ ซึ่งวิธีการโดยละเอียดสามารถดูได้จาก Flow Chart ที่ภาคผนวก

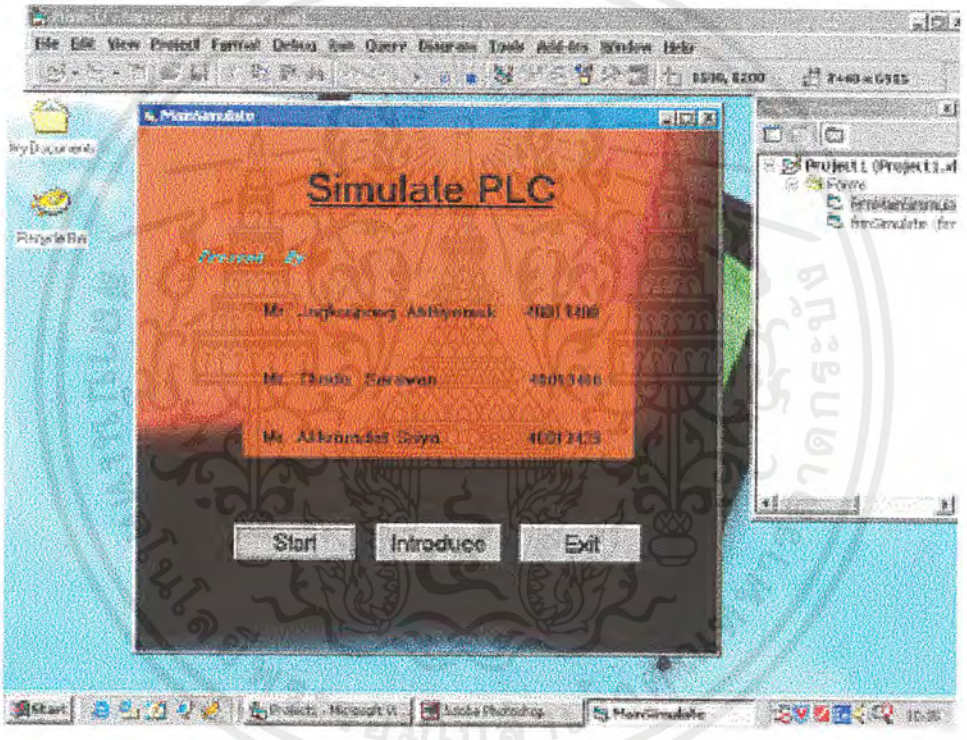
บทที่ 4

การทดลอง และการใช้งาน

4.1 หน้าต่างหลัก

เมื่อเปิดโปรแกรมขึ้นมาจะพบหน้าต่างหลัก สำหรับเข้าไปสู่หน้าต่างอื่น ๆ ดังนี้

- เมื่อ คลิก start จะปรากฏหน้าต่าง Operation ขึ้นมารับค่าอินพุทท์ เข้าไปในโปรแกรม
- เมื่อ คลิก Exit จะเป็นการออกจากโปรแกรม



รูปที่ 4.1 หน้าต่างหลัก

4.2 ส่วน Operation

เมื่อ คลิก start จะปรากฏหน้าต่าง Operate สำหรับรับข้อมูลอินพุทท์ ดังนี้

1. การใช้ปุ่ม "LD" เมื่อคลิกปุ่ม "LD" จะปรากฏ INPUT BOX ขึ้นมาเพื่อให้ใส่ค่า DATA กับ รีเลย์ตัวนั้นโดยป้อนค่าผ่านทางคีย์บอร์ด (Key Board)

ตัวอย่าง

ถ้าพิมพ์ 0 DATA จะมีค่า LD 0000

หรือพิมพ์ 100 DATA จะมีค่า LD 0100

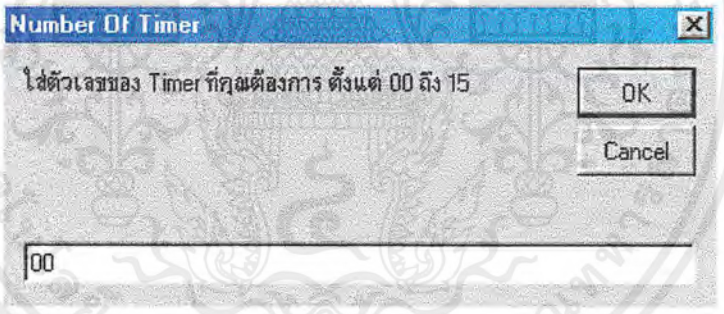
ส่วนปุ่ม AND , OR , LD_NOT , AND_NOT , OR_NOT , OUT ก็จะมีลักษณะการใช้เช่นเดียวกับ LOAD

2. AND_LD และ OR_LD จะไม่มี INPUT BOX ขึ้นมารับค่า

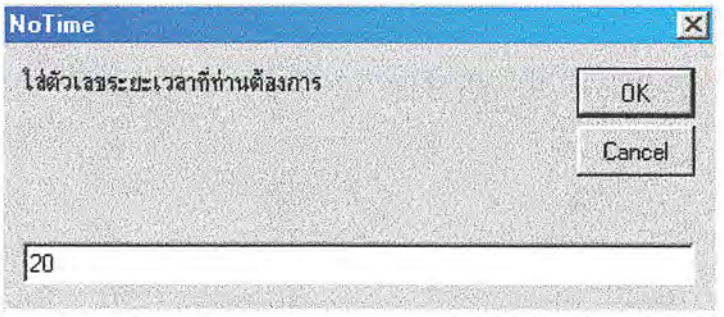
3. การใช้ปุ่ม TIMER

เมื่อคลิกปุ่ม "TIMER" จะปรากฏ INPUT BOX ขึ้นมารับค่า DATA ในโครงงานนี้กำหนดให้มี TIMER ขั้นต้น 16 TIMER ตั้งแต่ TIMER 00 - TIMER 15 เมื่อพิมพ์ค่า DATA เรียบร้อยแล้วจะปรากฏ INPUT BOX ขึ้นมาอีกครั้งหนึ่งเพื่อรับค่าเวลาที่ต้องการหน่วงเวลา

ดังลำดับภาพ



รูปที่4.2 INPUT BOX รับ DATA ของ TIMER

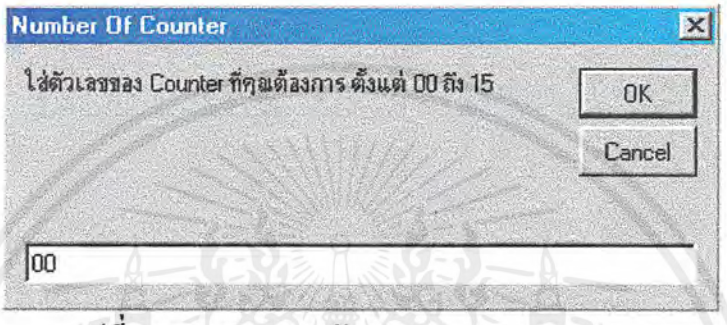


รูปที่4.3 INPUT BOX รับ ค่าเวลา ของ TIMER

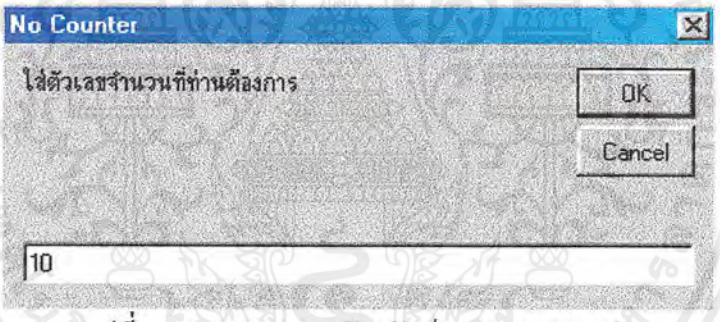
4. การใช้ปุ่ม COUNTER

เมื่อคลิกปุ่ม “COUNTER” จะปรากฏ INPUT BOX ขึ้นมารับค่า DATA ในโครงงานนี้กำหนดให้มี COUNTER ขั้นต้น 16 COUNTER ตั้งแต่ COUNTER 00 – COUNTER 15 เพื่อป้อน DATA เรียบร้อยแล้ว จะปรากฏ INPUT BOX ขึ้นมาอีกครั้งเพื่อรับจำนวน PULSE ที่จะป้อนในการนับ

ดังลำดับภาพ



รูปที่4.4 INPUT BOX รับ DATA ของ COUNTER



รูปที่4.5 INPUT BOX รับ พัลส์ ของ COUNTER

5. การใช้ปุ่ม DELETE

เมื่อคลิกปุ่ม “DELETE” จะเป็นการลบค่าสุดท้ายที่ป้อนเข้าไปในโปรแกรม ในทุก ๆ ส่วนของโปรแกรมโดยปกติจะใช้เมื่อป้อนค่าผิดพลาด

6. ปุ่ม NEW

เป็นการเริ่มต้นโปรแกรมใหม่ทั้งหมด เมื่อ คลิก “NEW”

7. ปุ่ม EXIT

เป็นการออกจากโปรแกรม

8. ปุ่ม Preview Page

เป็นการแสดงหน้าต่าง LADDER DIAGRAM ในหน้าย้อนหลังถัด ๆ มา

9. ปุ่ม Next Page

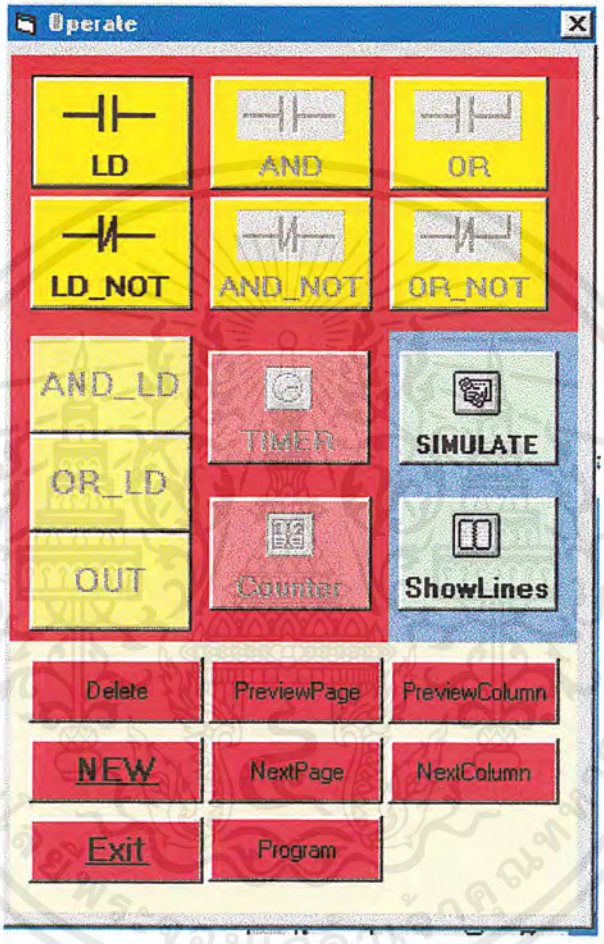
เป็นการแสดงหน้าต่าง LADDER DIAGRAM ในหน้าถัดไป

10. ปุ่ม Preview Column

เป็นการแสดงบรรทัด LADDER DIAGRAM ที่บรรทัดย้อนหลังถัด ๆ มา

11. ปุ่ม Next Column

เป็นการแสดงบรรทัด LADDER DIAGRAM ที่บรรทัดถัดไป



รูปที่4.6 หน้าต่าง Operation

4.3 ฟอร์มแลดเดอร์ไดอะแกรม

เป็นฟอร์มที่ใช้แสดงผล LADDER DIAGRAM ว่าเป็นไปตามความต้องการหรือไม่

การเข้าไปในส่วน LADDER DIAGRAM นี้ทำได้โดยการคลิกที่ปุ่ม “SHOW LINE” ซึ่งในฟอร์ม “SHOW LINE” นี้จะประกอบด้วยส่วนประกอบดังนี้

ส่วนที่ 1 Picture Box

จะเป็นส่วนที่ใช้ในการแสดงภาพ LADDER DIAGRAM

ส่วนที่ 2 Show Line

เมื่อคลิกที่ปุ่ม "SHOW LINE" จะปรากฏภาพ LADDER DIAGRAM ในส่วน Picture Box ตามที่ได้ป้อนอินพุทท์ไว้

ส่วนที่ 3 Preview Page

เป็นปุ่มที่ใช้ดูในหน้าถัดมากรณีที่ดู LADDER DIAGRAM ในหน้าอื่น ๆ อยู่ แล้วต้องการย้อนกลับมาที่หน้าที่ผ่าน ๆ มา

ส่วนที่ 4 Next Column

เป็นปุ่มที่ใช้ดูบรรทัดต่าง ๆ ในกรณีที่เราต้องการดู LADDER DIAGRAM ในช่วงต่าง ๆ ในกรณีที่ต้องการดูเพียงบรรทัดหรือสองบรรทัด

ส่วนที่ 5 Next Page

เป็นปุ่มที่ใช้ในกรณีที่ต้องการดู LADDER DIAGRAM ในหน้าถัดไป

ส่วนที่ 6 Back to Operate

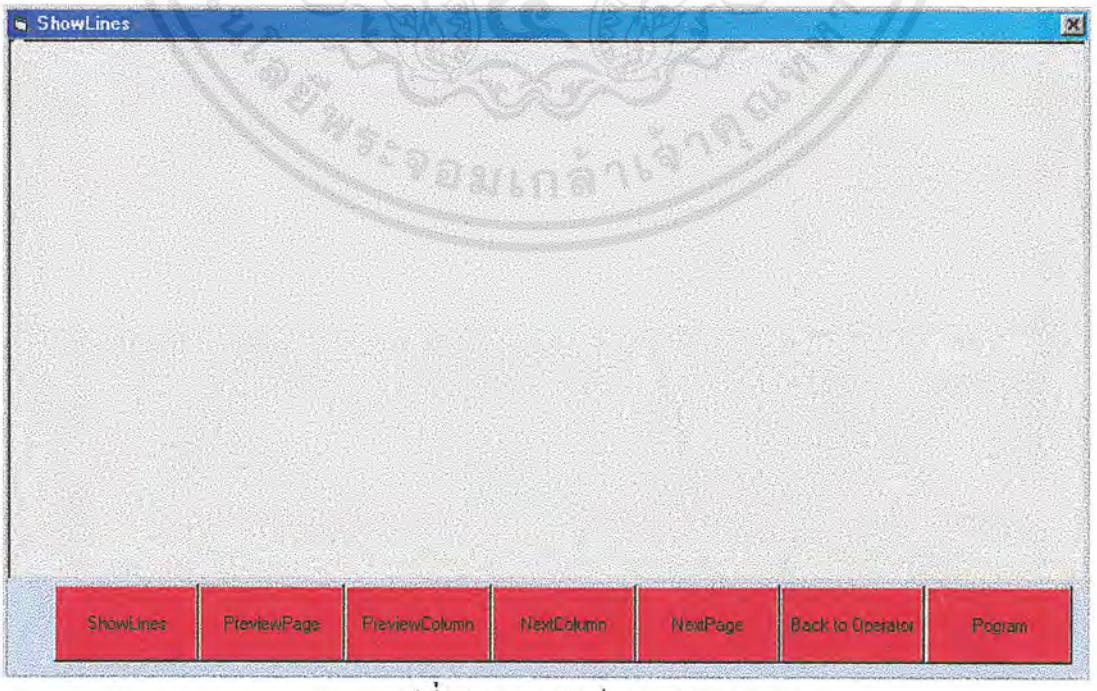
เป็นปุ่มที่นำกลับสู่หน้าต่าง Operate

ส่วนที่ 7 Program

เป็นปุ่มที่ใช้ในกรณีต้องการดู Program ในภาษา Boolean

ส่วนที่ 8 Preview Column

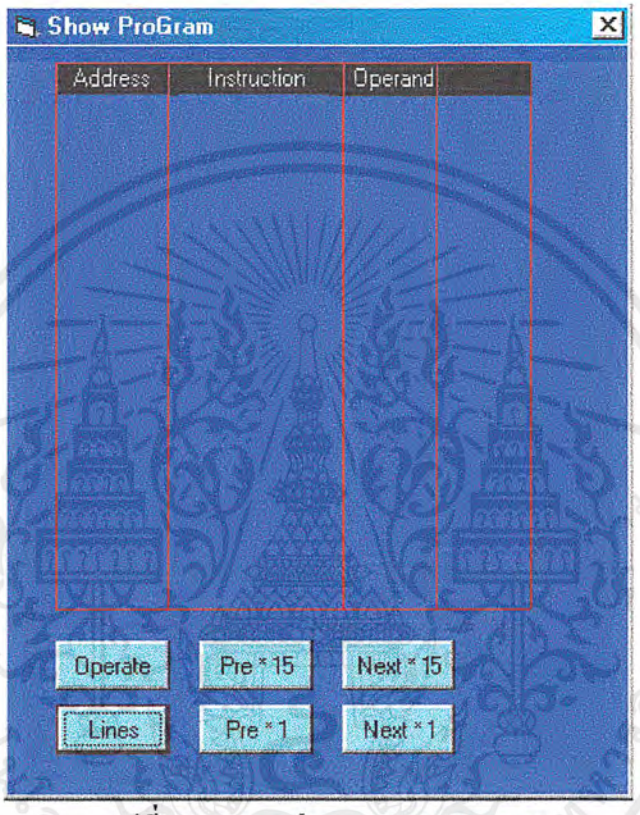
เป็นปุ่มที่ใช้ดูในบรรทัดถัดมาในกรณีที่ดู LADDER DIAGRAM อยู่แล้วต้องการย้อนกลับไปเพียงบรรทัดเดียว



รูปที่4.7 แสดงฟอร์ม Ladder Diagram

4.4 ส่วนภาษา Boolean

หลังจากป้อนคำสั่งโปรแกรมเสร็จแล้ว ถ้าต้องการตรวจสอบด้วยภาษา Boolean สามารถทำได้โดยการ click Program จะปรากฏตารางภาษา Boolean ขึ้นมา ในฟอร์ม Show Program ประกอบด้วยส่วนต่าง ๆ ดังนี้



รูปที่ 4.8 แสดงส่วนภาษา BOOLEAN

ส่วนที่ 1 ส่วนแสดงผลประกอบด้วย

Address กำหนดตำแหน่ง

Instruction กำหนดว่าเป็นรีเลย์ประเภทใด

Operamd กำหนด DATA ให้รีเลย์

ส่วนที่ 2 ปุ่ม Operate

เป็นปุ่มที่จะนำกลับไปสู่ฟอร์ม Operate

ส่วนที่ 3 ปุ่ม Line

เป็นปุ่มที่จะนำไปสู่ฟอร์ม Show Line เมื่อดู LADDER DIAGRAM

ส่วนที่ 4 ปุ่ม "Pre*15"

เป็นปุ่มใช้สำหรับเลื่อนขึ้นที่ละ 15 บรรทัด ใช้สำหรับการดูโปรแกรมด้านบน

ส่วนที่ 5 ปุ่ม "Pre*1"

เป็นปุ่มใช้สำหรับเลื่อนขึ้นทีละ 1 บรรทัด ใช้สำหรับดูโปรแกรมในบรรทัดที่เหนือไป 1 บรรทัด

ส่วนที่ 6 ปุ่ม "Next*15"

เป็นปุ่มใช้สำหรับเลื่อนลงทีละ 15 บรรทัด ใช้สำหรับดูโปรแกรมด้านล่าง

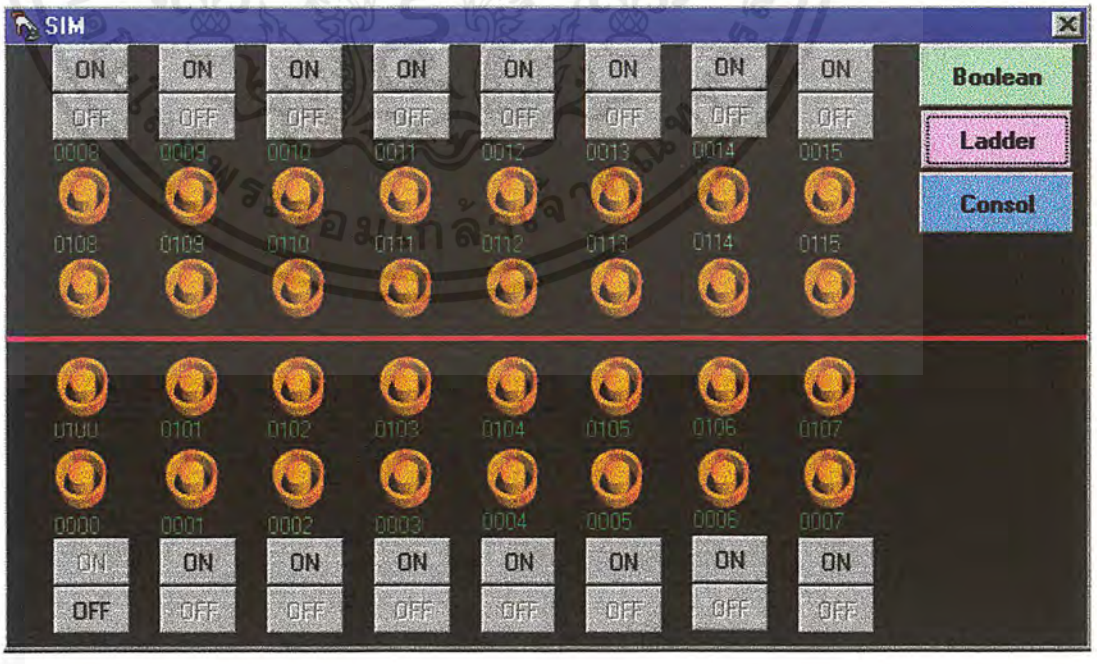
ส่วนที่ 7 ปุ่ม "Next*1"

เป็นปุ่มใช้สำหรับเลื่อนลงทีละ 1 บรรทัด ใช้สำหรับดูโปรแกรมในบรรทัดที่อยู่ต่ำลงไป 1 บรรทัด

4.5 ส่วน Simulate

เมื่อตรวจสอบว่าโปรแกรมถูกต้องตามความต้องการแล้วและต้องการตรวจสอบว่าโปรแกรมนั้นถูกต้องตามจุดประสงค์ หรือไม่สามารถกระทำได้โดย click "Simulate"

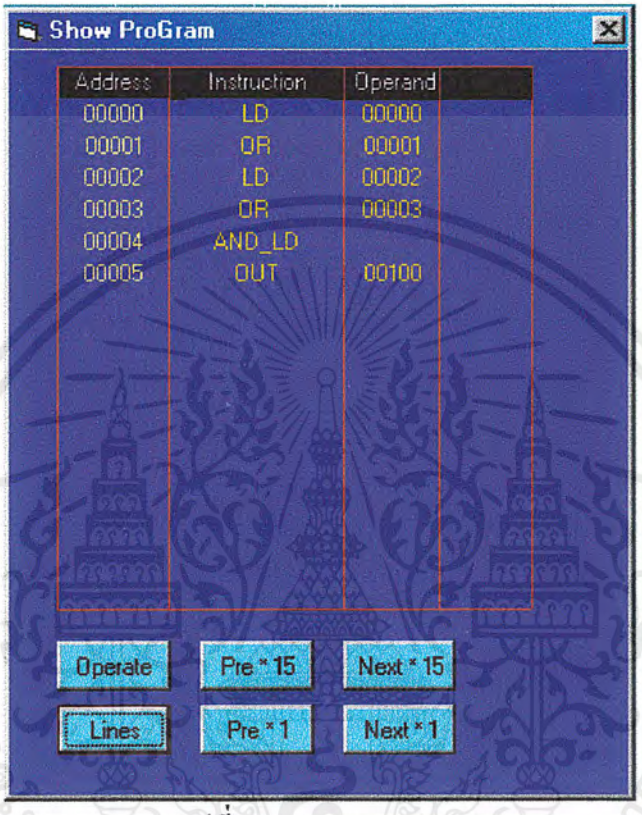
- Click ON "0000" สภาวะ Input = 1 สภาวะ Relay = 1
- Click ON "0001" สภาวะ Input = 1 สภาวะ Relay = 1
- Click OFF "0000" สภาวะ Input = 0 สภาวะ Relay = 0
- Click OFF "0001" สภาวะ Input = 0 สภาวะ Relay = 0



รูปที่ 4.9 แสดงส่วนภาษา SIMULATE

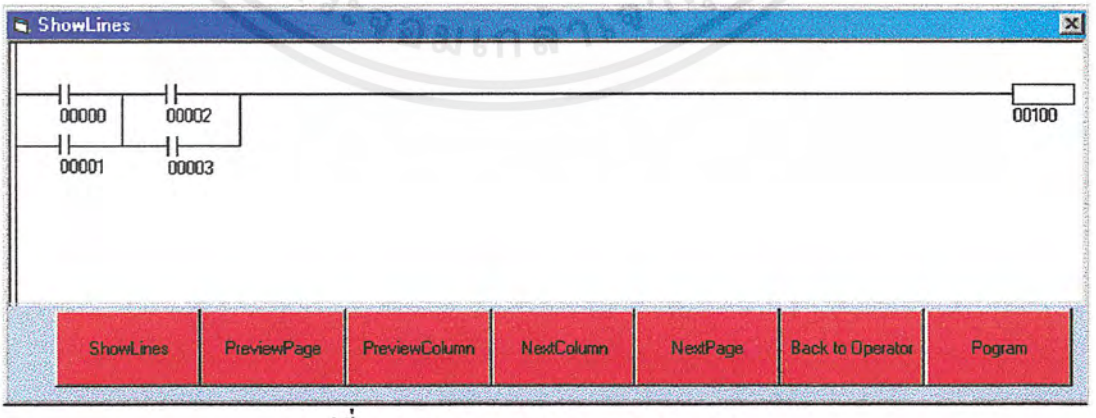
4.6 ตัวอย่างการใช้งาน คำสั่ง AND LD

1. เมื่อป้อนข้อมูลผ่าน ฟอร์ม Consol แล้วคลิกปุ่ม Program จะปรากฏ form Program ดังรูปที่ 4.10



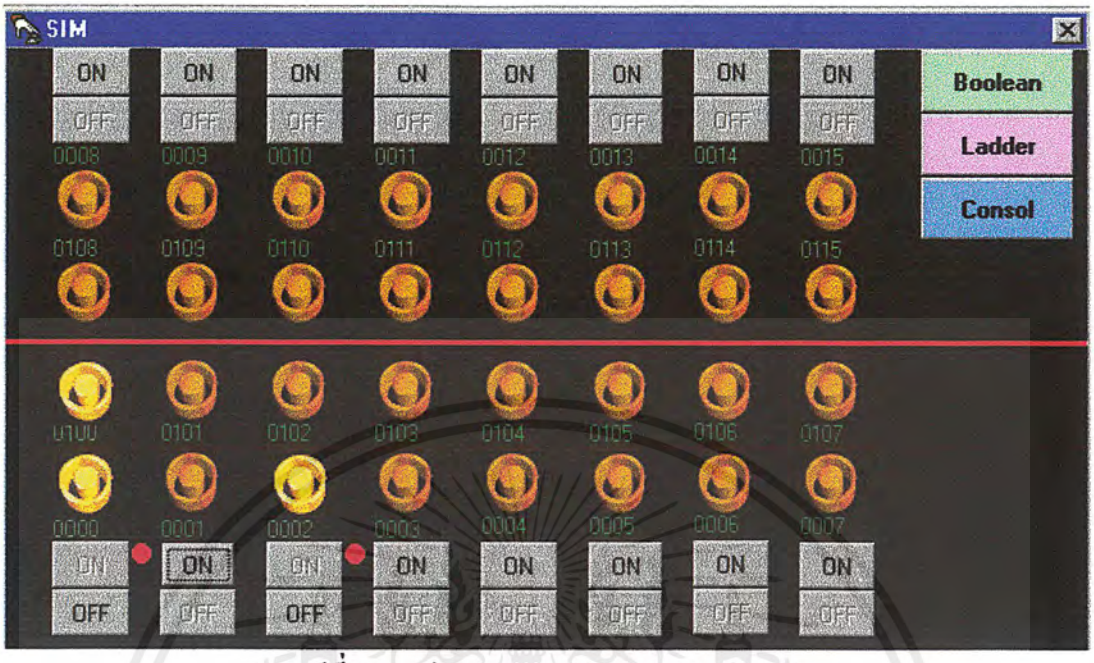
รูปที่ 4.10 Program And_LD

2.คลิกปุ่ม Ladder จะปรากฏ form Ladder ดังรูปที่ 4.11



รูปที่ 4. 11Ladder Diagram And_LD

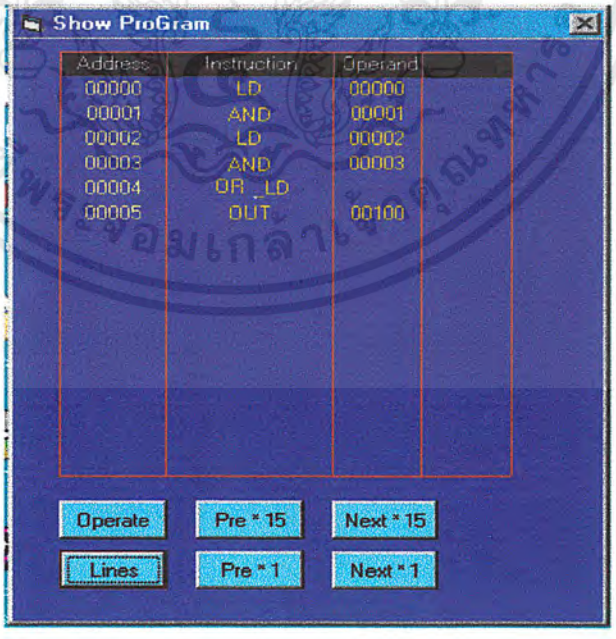
3.เมื่อคลิก SIM จะปรากฏ form Simulate เมื่อคลิกอินพุท 0000และ 0002 จะทำให้เอาท์-พุท 100 มีสภาวะ "1.



รูปที่ 4.12รูปแสดง form Simulate AND_LD

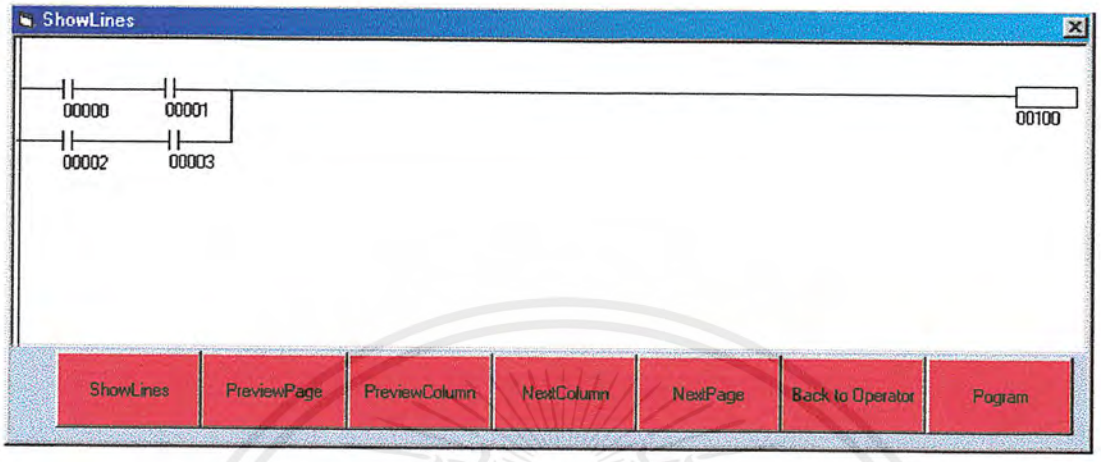
4.7 ตัวอย่างการใช้งาน คำสั่ง Or_ LD

1. เมื่อป้อนข้อมูลผ่าน ฟอร์ม Consol แล้วคลิกปุ่ม Program จะปรากฏ form Program ดังรูปที่ 4.13



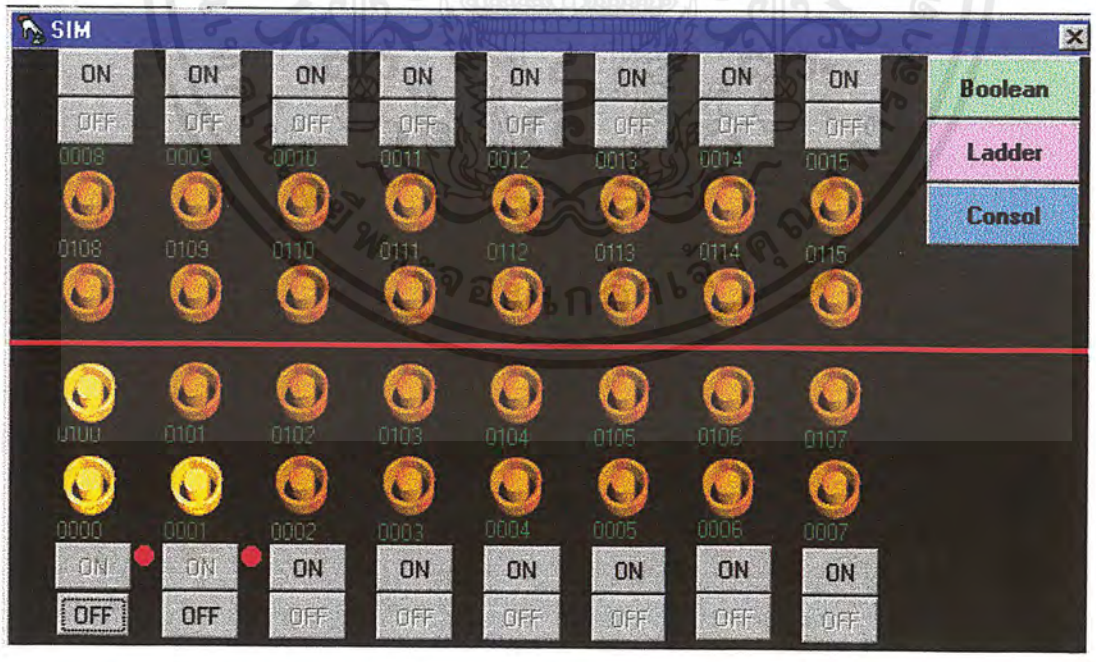
รูปที่ 4.13 Program OR_LD

2.คลิกปุ่ม Ladder จะปรากฏ form Ladder ดังรูปที่ 4.14



รูปที่ 4.14 Ladder Diagram OR_LD

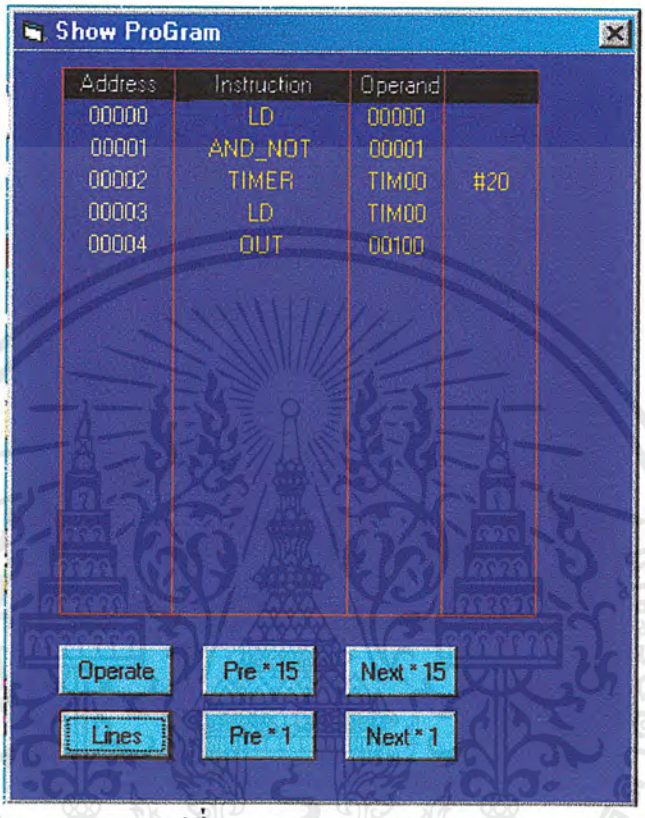
3.เมื่อคลิก SIM จะปรากฏ form Simulate และเมื่อคลิก 0000และ 0001 เอาท์พุท 100 จะมีสภาวะ "1"



รูปที่ 4.15 รูปแสดง form Simulate OR_LD

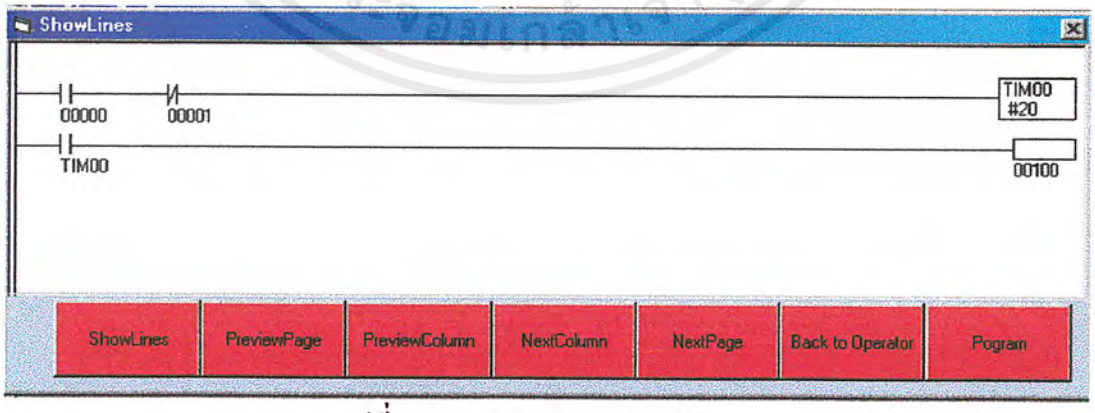
4.8 ตัวอย่างการใช้งาน คำสั่ง Timer

1. เมื่อป้อนข้อมูลผ่าน ฟอร์ม Consol แล้วคลิกปุ่ม Program จะปรากฏ form Program ดังรูปที่ 4.16



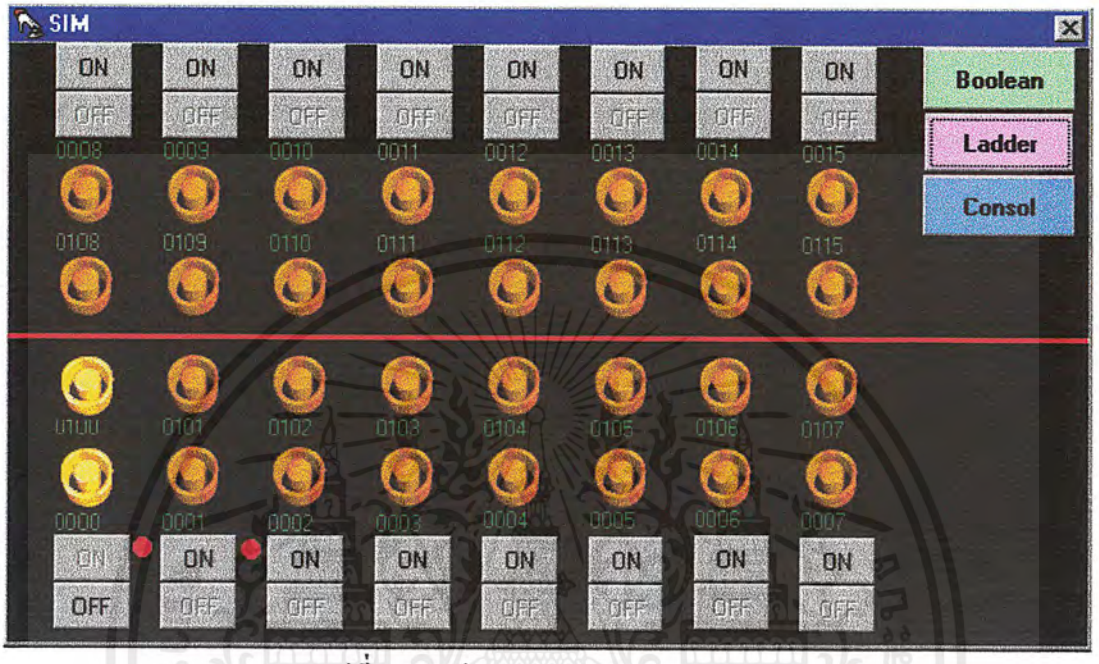
รูปที่ 4.16 Program Timer

2.คลิกปุ่ม Ladder จะปรากฏ form Ladder ดังรูปที่ 4.17



รูปที่ 4.17 Ladder Diagram Timer

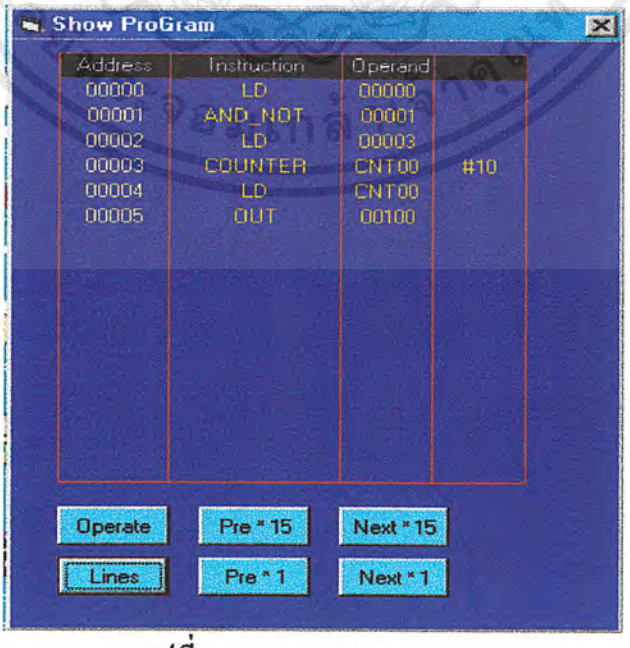
3.เมื่อคลิก SIM จะปรากฏ form Simulate และเมื่อคลิกอินพุท 0000 แล้วรอ 20 วินาที เอาท์พุท 100 จะมีสภาวะ “1”



รูปที่ 4.18 รูปแสดง form Simulate Timer

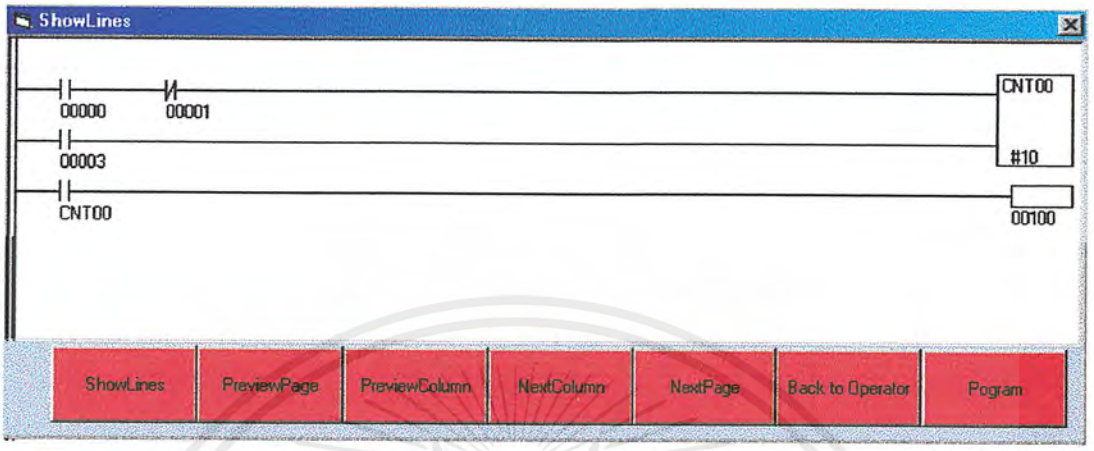
4.9 ตัวอย่างการใช้งาน คำสั่ง Counter

1. เมื่อป้อนข้อมูลผ่าน ฟอร์ม Consol แล้วคลิกปุ่ม Program จะปรากฏ form Program ดังรูปที่ 4.19



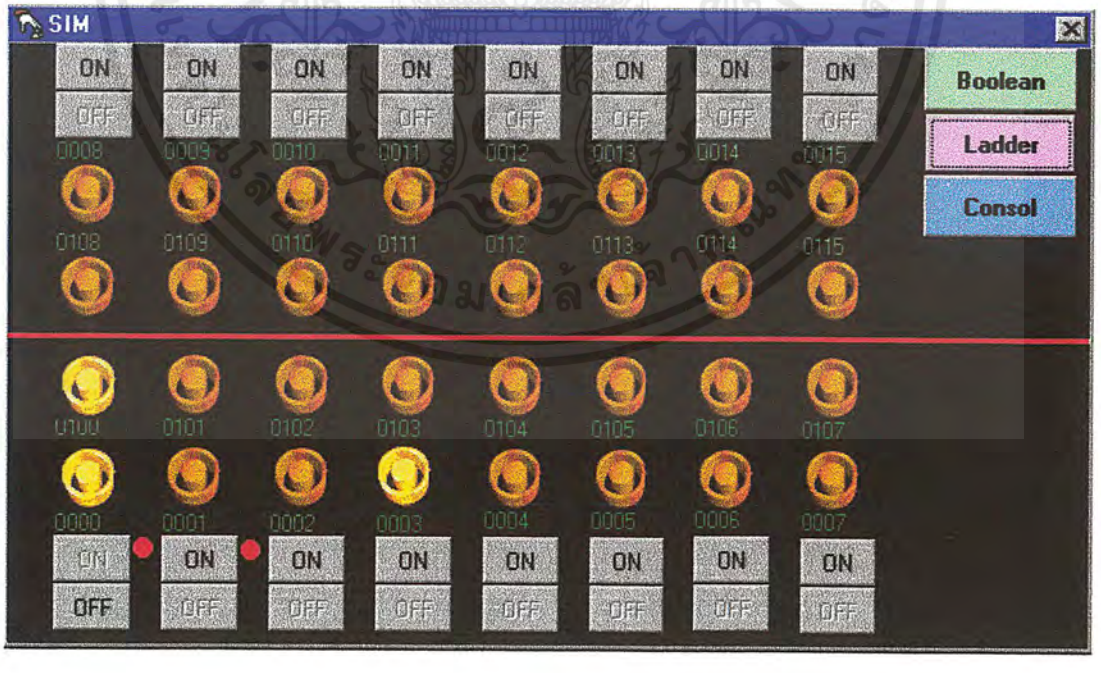
รูปที่ 4.19 Program Counter

2.คลิกปุ่ม Ladder จะปรากฏ form Ladder ดังรูปที่ 4.20



รูปที่ 4.20 Ladder Diagram Counter

3.เมื่อคลิก SIM จะปรากฏ form Simulate คลิกอินพุท 0000 จะทำให้ COUNTER ทำงาน และเมื่อป้อน Pulse ทาง อินพุท 0003 ครบ 10 ครั้ง จะทำให้ เอาท์พุท 100 มีสภาวะ "1"



รูปที่ 4.21 รูปแสดง form Simulate Counter

บทที่5

สรุปผลการทดลองและข้อเสนอแนะ

5.1 ปัญหาที่พบ

1. การ AND_LD และ OR_LD ไม่สามารถนำมาเขียนในรูป Ladder Diagram ได้แต่สามารถแก้ไขได้แล้ว
2. การ Simulate มักพบปัญหาในการจดจำค่าตัวแปร I (array)แต่สามารถแก้ไขได้แล้ว
3. โปรแกรมไม่เป็นเอกภาพ เนื่องจากมีความซับซ้อนในการถ่ายข้อมูลสูง
4. ขอบเขตของผลที่จะแสดงกว้างมาก อีกทั้งมีความหลากหลายในการนำเสนอ
5. Timer และ Counter มักพบปัญหาในการป้อน Input แต่สามารถแก้ไขได้แล้ว
6. การเขียนโปรแกรมมักพบปัญหาข้อผิดพลาดแบบ Logical error ซึ่งทำให้เสียเวลาในการหาจุดผิดพลาด เพราะไม่รู้จุดผิดพลาดที่แน่นอน
7. รูปแบบความน่าจะเป็นไม่มีรูปแบบตายตัว

5.2 วิธีการแก้ไข

1. ทำให้เกิดการวนลูปทุกครั้งที่มีการป้อนอินพุท
2. ลดขนาดโปรแกรมให้เล็กที่สุดเท่าที่ทำได้
3. กำหนดวิธีการนำเสนอที่แน่ชัด
4. แยก subโปรแกรม TIMER และ COUNTER ออกมาจาก อินพุทปกติ
5. กำหนดวิธีการใช้งานที่ตายตัว

5.3 สรุปผลการทดลอง

PLC Simulate Software ชิ้นนี้ เป็นแนวทางเริ่มต้นสำหรับการพัฒนาซอฟต์แวร์ชุดนี้ ให้มีฟังก์ชันต่าง ๆ เทียบกับ PLC ดังนั้น ฟังก์ชันต่าง ๆ ที่สร้างขึ้นได้ในระยะเวลานี้จึงยังมีไม่ครบถ้วนและยังไม่สมบูรณ์เท่าใดนัก แต่ซอฟต์แวร์ชิ้นนี้ก็สามารถทำงานเลียนแบบ PLC ได้ดีในระดับหนึ่งและยังสามารถแสดงผลได้ทั้งส่วนแลดเดอร์ไดอะแกรมและส่วนภาษาบูลีน ซึ่งจะเป็นผลดีกับผู้ที่เริ่มศึกษา PLC เพราะสามารถเห็นภาพได้อย่างชัดเจน

บรรณานุกรม

1. กิตติ ภักดีวัฒนะกุล และ จำลอง ครูอุคสาหะ , VISUAL BASIC 6 ฉบับโปรแกรมเมอร์ , ดวงกมลสมัยจำกัดม หน้า 19 –98
2. ธนพล ฉันจรัสวิชัย , เรียนรู้ MICROSOFT VISUAL BASIC 5 , เอส พี ซี บุคส์ , หน้า 1-57
3. อนิรุดิ ลิ่วหาทอง , MICROSOFT VISUAL BASIC 5 ,ซีเอ็ดยูเคชั่นจำกัด(มหาชน), หน้า 151-276
4. กฤษฎา วิศวธีรานนท์ , PC ตัวควบคุมซีเควันซ์ หลักการและเทคนิคการประยุกต์ , บริษัท บุญชัยวิศวกรรม , หน้า 1-57
5. สุเธียร เกียรติสุนทร , หลักการทำงานและเทคนิคการประยุกต์การใช้งาน PC/PLC , ภาค-วิชาระบบควบคุมคณะวิศวกรรมศาสตร์ , หน้า 5 - 42

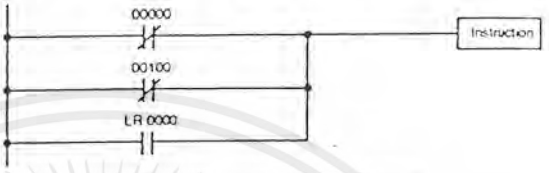
ภาคผนวก

ภาคผนวก ก

คู่มือการใช้งาน พีแอลซีรุ่น C200HS

OR and OR NOT

When two or more conditions lie on separate instruction lines which run in parallel and then join together, the first condition corresponds to a LOAD or LOAD NOT instruction; the other conditions correspond to OR or OR NOT instructions. The following example shows three conditions which correspond (in order from the top) to a LOAD NOT, an OR NOT, and an OR instruction. Again, each of these instructions requires one line of mnemonic code.



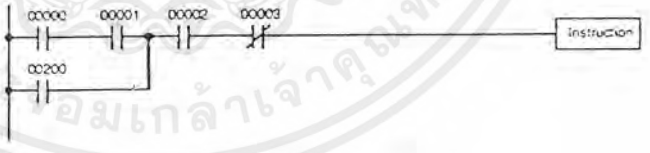
Address	Instruction	Operands
00000	LD	00000
00001	OR NOT	00100
00002	OR	LR 0000
00003	Instruction	

The instruction will have an ON execution condition when any one of the three conditions is ON, i.e., when IR 00000 is OFF, when IR 00100 is OFF, or when LR 0000 is ON.

OR and OR NOT instructions can be considered individually, each taking the logical OR between its execution condition and the status of the OR instruction's operand bit. If either one of these were ON, an ON execution condition will be produced for the next instruction.

Combining AND and OR Instructions

When AND and OR instructions are combined in more complicated diagrams, they can sometimes be considered individually, with each instruction performing a logic operation on the execution condition and the status of the operand bit. The following is one example. Study this example until you are convinced that the mnemonic code follows the same logic flow as the ladder diagram.



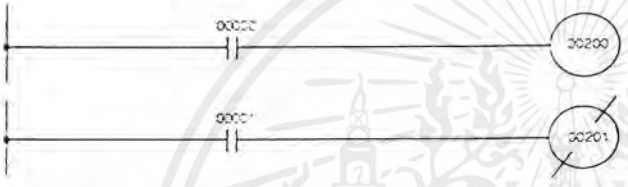
Address	Instruction	Operands
00000	LD	00000
00001	AND	00001
00002	OR	00200
00003	AND	00002
00004	AND NOT	00003
00005	nstruction	

Here, an AND is taken between the status of IR 00000 and that of IR 00001 to determine the execution condition for an OR with the status of IR 00200. The result of this operation determines the execution condition for an AND with the status of IR 00002, which in turn determines the execution condition for an AND with the inverse (i.e., and AND NOT) of the status of IR 00003.

In more complicated diagrams, however, it is necessary to consider logic blocks before an execution condition can be determined for the final instruction, and that's where AND LOAD and OR LOAD instructions are used. Before we consider more complicated diagrams, however, we'll look at the instructions required to complete a simple "input-output" program.

4-4-4 OUTPUT and OUTPUT NOT

The simplest way to output the results of combining execution conditions is to output it directly with the OUTPUT and OUTPUT NOT. These instructions are used to control the status of the designated operand bit according to the execution condition. With the OUTPUT instruction, the operand bit will be turned ON as long as the execution condition is ON and will be turned OFF as long as the execution condition is OFF. With the OUTPUT NOT instruction, the operand bit will be turned ON as long as the execution condition is OFF and turned OFF as long as the execution condition is ON. These appear as shown below. In mnemonic code, each of these instructions requires one line.



Address	Instruction	Operands
00000	LD	00000
00001	OUT	00200

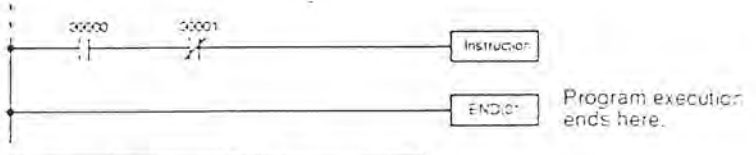
Address	Instruction	Operands
00000	LD	00001
00001	OUT NOT	00201

In the above examples, IR 00200 will be ON as long as IR 00000 is ON and IR 00201 will be OFF as long as IR 00001 is ON. Here, IR 00000 and IR 00001 will be input bits and IR 00200 and IR 00201 output bits assigned to the Units controlled by the PC, i.e., the signals coming in through the input points assigned IR 00000 and IR 00001 are controlling the output points assigned IR 00200 and IR 00201, respectively.

The length of time that a bit is ON or OFF can be controlled by combining the OUTPUT or OUTPUT NOT instruction with TIMER instructions. Refer to Examples under *5-14-1 TIMER – TIM* for details.

4-4-5 The END Instruction

The last instruction required to complete a simple program is the END instruction. When the CPU cycles the program, it executes an instruction up to the first END instruction before returning to the beginning of the program and beginning execution again. Although an END instruction can be placed at any point in a program, which is sometimes done when debugging, no instructions past the first END instruction will be executed until it is removed. The number following the END instruction in the mnemonic code is its function code, which is used when inputting most instruction into the PC. These are described later. The END instruction requires no operands and no conditions can be placed on the same instruction line with it.



Address	Instruction	Operands
00000	LD	00000
00001	AND NOT	00001
00002	Instruction	
00003	END(01)	---

If there is no END instruction anywhere in the program, the program will not be

Now you have all of the instructions required to write simple input-output programs. Before we finish with ladder diagram basic and go onto inputting the program into the PC, let's look at logic block instruction (AND LOAD and OR LOAD), which are sometimes necessary even with simple diagrams.

4-4-6 Logic Block Instructions

Logic block instructions do not correspond to specific conditions on the ladder diagram; rather, they describe relationships between logic blocks. The AND LOAD instruction logically ANDs the execution conditions produced by two logic blocks. The OR LOAD instruction logically ORs the execution conditions produced by two logic blocks.

AND LOAD

Although simple in appearance, the diagram below requires an AND LOAD instruction.

Address	Instruction	Operands
00000	LD	00000
00001	OR	00001
00002	LD	00002
00003	OR NOT	00003
00004	AND LD	---

The two logic blocks are indicated by dotted lines. Studying this example shows that an ON execution condition will be produced when: either of the conditions in the left logic block is ON (i.e., when either IR 00000 or IR 00001 is ON), and when either of the conditions in the right logic block is ON (i.e., when either IR 00002 is ON or IR 00003 is OFF).

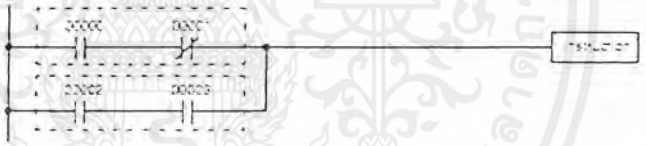
The above ladder diagram cannot, however, be converted to mnemonic code using AND and OR instructions alone. If an AND between IR 00002 and the results of an OR between IR 00000 and IR 00001 is attempted, the OR NOT between IR 00002 and IR 00003 is lost and the OR NOT ends up being an OR NOT between just IR 00003 and the result of an AND between IR 00002 and the first OR. What we need is a way to do the OR (NOT)'s independently and then combine the results.

To do this, we can use the LOAD or LOAD NOT instruction in the middle of an instruction line. When LOAD or LOAD NOT is executed in this way, the current execution condition is saved in a special buffer and the logic process is restarted. To combine the results of the current execution condition with that of a previous "unused" execution condition, an AND LOAD or an OR LOAD instruction is used. Here "LOAD" refers to loading the last unused execution condition. An unused execution condition is produced by using the LOAD or LOAD NOT instruction for any but the first condition on an instruction line.

Analyzing the above ladder diagram in terms of mnemonic instructions, the condition for IR 00000 is a LOAD instruction and the condition below it is an OR instruction between the status of IR 00000 and that of IR 00001. The condition at IR 00002 is another LOAD instruction and the condition below is an OR NOT instruction, i.e. an OR between the status of IR 00002 and the inverse of the status of IR 00003. To arrive at the execution condition for the instruction at the right, the logical AND of the execution conditions resulting from these two blocks will have to be taken. AND LOAD does this. The mnemonic code for the ladder diagram is shown below. The AND LOAD instruction requires no operands of its own, because it operates on previously determined execution conditions. Here too, dashes are used to indicate that no operands needs designated or input.

OR LOAD

The following diagram requires an OR LOAD instruction between the top logic block and the bottom logic block. An ON execution condition will be produced for the instruction at the right either when IR 00000 is ON and IR 00001 is OFF, or when IR 00002 and IR 00003 are both ON. The operation of the OR LOAD instruction and its mnemonic code is exactly the same as that for an AND LOAD instruction, except that the current execution condition is ORed with the last unused execution condition.



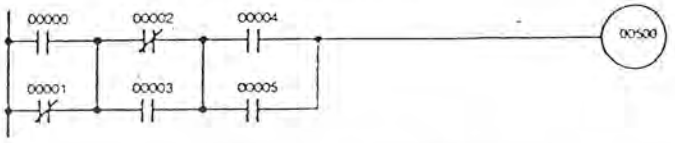
Address	Instruction	Operands
00000	LD	00000
00001	AND NOT	00001
00002	LD	00002
00003	AND	00003
00004	OR LD	---

Naturally, some diagrams will require both AND LOAD and OR LOAD instructions.

Logic Block Instructions in Series

To code diagrams with logic block instructions in series, the diagram must be divided into logic blocks. Each block is coded using a LOAD instruction to code the first condition, and then AND LOAD or OR LOAD is used to logically combine the blocks. With both AND LOAD and OR LOAD there are two ways to achieve this. One is to code the logic block instruction after the first two blocks and then after each additional block. The other is to code all of the blocks to be combined, starting each block with LOAD or LOAD NOT, and then to code the logic block instructions which combine them. In this case, the instructions for the last pair of blocks should be combined first, and then each preceding block should be combined, working progressively back to the first block. Although either of these methods will produce exactly the same result, the second method, that of coding all logic block instructions together, can be used only if eight or fewer blocks are being combined, i.e. if seven or fewer logic block instructions are required.

The following diagram requires AND LOAD to be converted to mnemonic code because three pairs of parallel conditions lie in series. The two options for coding the programs are also shown.



Address	Instruction	Operands
00000	LD	00000
00001	OR NOT	00001
00002	LD NOT	00002
00003	OR	00003
00004	AND LD	—
00005	LD	00004
00006	OR	00005
00007	AND LD	—
00008	OUT	00500

Address	Instruction	Operands
00000	LD	00000
00001	OR NOT	00001
00002	LD NOT	00002
00003	OR	00003
00004	LD	00004
00005	OR	00005
00006	AND LD	—
00007	AND LD	—
00008	OUT	00500

Again, with the method on the right, a maximum of eight blocks can be combined. There is no limit to the number of blocks that can be combined with the first method.

The following diagram requires OR LOAD instructions to be converted to mnemonic code because three pairs of series conditions lie in parallel to each other.

The first of each pair of conditions is converted to LOAD with the assigned bit operand and then ANDed with the other condition. The first two blocks can be coded first, followed by OR LOAD, the last block, and another OR LOAD; or the three blocks can be coded first followed by two OR LOADs. The mnemonic codes for both methods are shown below.

Address	Instruction	Operands
00000	LD	00000
00001	AND NOT	00001
00002	LD NOT	00002
00003	AND NOT	00003
00004	OR LD	—
00005	LD	00004
00006	AND	00005
00007	OR LD	—
00008	OUT	00501

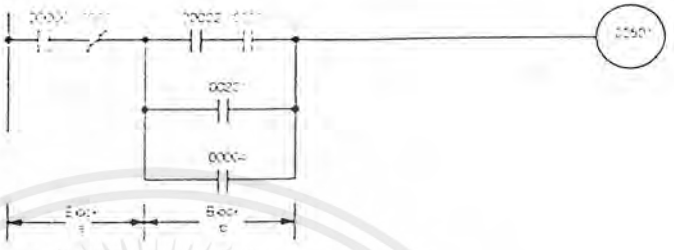
Address	Instruction	Operands
00000	LD	00000
00001	AND NOT	00001
00002	LD NOT	00002
00003	AND NOT	00003
00004	LD	00004
00005	AND	00005
00006	OR LD	—
00007	OR LD	—
00008	OUT	00501

Again, with the method on the right, a maximum of eight blocks can be combined. There is no limit to the number of blocks that can be combined with the first method.

Combining AND LOAD and OR LOAD

Both of the coding methods described above can also be used when using AND LOAD and OR LOAD, as long as the number of blocks being combined does not exceed eight.

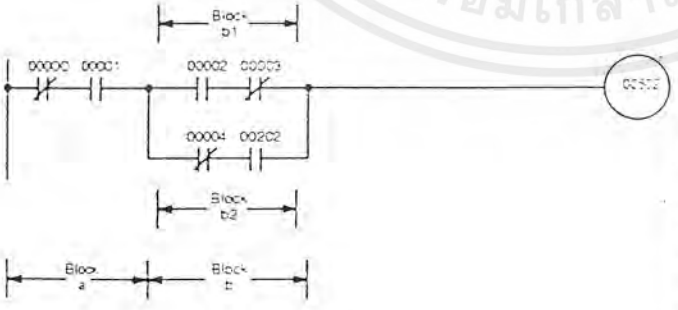
The following diagram contains only two logic blocks as shown. It is not necessary to further separate block b components because it can be coded directly using only AND and OR.



Address	Instruction	Operands
00000	LD	00000
00001	AND NOT	00001
00002	LD	00002
00003	AND	00003
00004	OR	00201
00005	OR	00004
00006	AND LD	—
00007	OUT	00501

Although the following diagram is similar to the one above, block b in the diagram below cannot be coded without separating it into two blocks combined with OR LOAD. In this example, the three blocks have been coded first and then OR LOAD has been used to combine the last two blocks, followed by AND LOAD to combine the execution condition produced by the OR LOAD with the execution condition of block a.

When coding the logic block instructions together at the end of the logic blocks they are combining, they must, as shown below, be coded in reverse order, i.e., the logic block instruction for the last two blocks is coded first, followed by the one to combine the execution condition resulting from the first logic block instruction and the execution condition of the logic block third from the end, and on back to the first logic block that is being combined.



Address	Instruction	Operands
00000	LD NOT	00000
00001	AND	00001
00002	LD	00002
00003	AND NOT	00003
00004	LD NOT	00004
00005	AND	00202
00006	OR LD	—
00007	AND LD	—
00008	OUT	00502

Complicated Diagrams

When determining what logic block instructions will be required to code a diagram, it is sometimes necessary to break the diagram into large blocks and then continue breaking the large blocks down until logic blocks that can be coded without logic block instructions have been formed. These blocks are then coded, combining the small blocks first, and then combining the larger blocks. Either AND LOAD or OR LOAD is used to combine the blocks, i.e., AND LOAD or OR LOAD always combines the last two execution conditions that existed, regardless of whether the execution conditions resulted from a single condition, from logic blocks, or from previous logic block instructions.

Any one TC number cannot be defined twice, i.e., once it has been used as the definer in any of the timer or counter instructions, it cannot be used again. Once defined, TC numbers can be used as many times as required as operands in instructions other than timer and counter instructions.

TC numbers run from 000 through 511. No prefix is required when using a TC number as a definer in a timer or counter instruction. Once defined as a timer, a TC number can be prefixed with TIM for use as an operand in certain instructions. The TIM prefix is used regardless of the timer instruction that was used to define the timer. Once defined as a counter, a TC number can be prefixed with CNT for use as an operand in certain instructions. The CNT is also used regardless of the counter instruction that was used to define the counter.

TC numbers can be designated as operands that require either bit or word data. When designated as an operand that requires bit data, the TC number accesses a bit that functions as a 'Completion Flag' that indicates when the time/count has expired, i.e., the bit, which is normally OFF, will turn ON when the designated SV has expired. When designated as an operand that requires word data, the TC number accesses a memory location that holds the present value (PV) of the timer or counter. The PV of a timer or counter can thus be used as an operand in CMP(20), or any other instruction for which the TC area is allowed. This is done by designating the TC number used to define that timer or counter to access the memory location that holds the PV.

Note that "TIM 000" is used to designate the TIMER instruction defined with TC number 000, to designate the Completion Flag for this timer, and to designate the PV of this timer. The meaning of the term in context should be clear, i.e., the first is always an instruction, the second is always a bit operand, and the third is always a word operand. The same is true of all other TC numbers prefixed with TIM or CNT.

An SV can be input as a constant or as a word address in a data area. If an IR area word assigned to an Input Unit is designated as the word address, the Input Unit can be wired so that the SV can be set externally through thumbwheel switches or similar devices. Timers and counters wired in this way can only be set externally during RUN or MONITOR mode. All SVs, including those set externally, must be in BCD.

5-14-1 TIMER – TIM

Definer Values

N: TC number
(000 through 511)

Ladder Symbol

Operand Data Areas

SV: Set value (word, BCD)
IR, AR, DM, HR, LR, #

Limitations

SV is between 000.0 and 999.9. The decimal point is not entered.

Each TC number can be used as the definer in only one TIMER or COUNTER instruction.

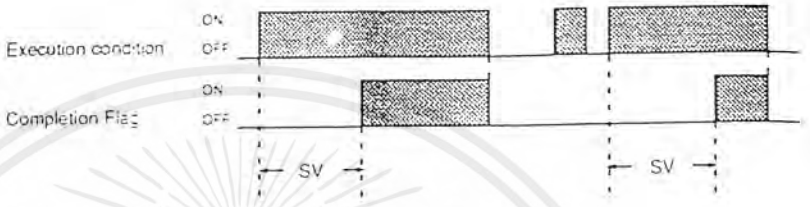
TC 000 through TC 015 should not be used in TIM if they are required for TIMH(15). Refer to *5-14-2 HIGH-SPEED TIMER – TIMH(15)* for details.

Description

A timer is activated when its execution condition goes ON and is reset (to SV) when the execution condition goes OFF. Once activated, TIM measures in units of 0.1 second from the SV.

If the execution condition remains ON long enough for TIM to time down to zero, the Completion Flag for the TC number used will turn ON and will remain ON until TIM is reset (i.e., until its execution condition is goes OFF).

The following figure illustrates the relationship between the execution condition for TIM and the Completion Flag assigned to it.



Precautions

Timers in interlocked program sections are reset when the execution condition for IL(02) is OFF. Power interruptions also reset timers. If a timer that is not reset under these conditions is desired, SR area clock pulse bits can be counted to produce timers using CNT. Refer to *5-14-4 COUNTER – CNT* for details.

Program execution will continue even if a non-BCD SV is used, but timing will not be accurate.

Flags

ER: SV is not in BCD.

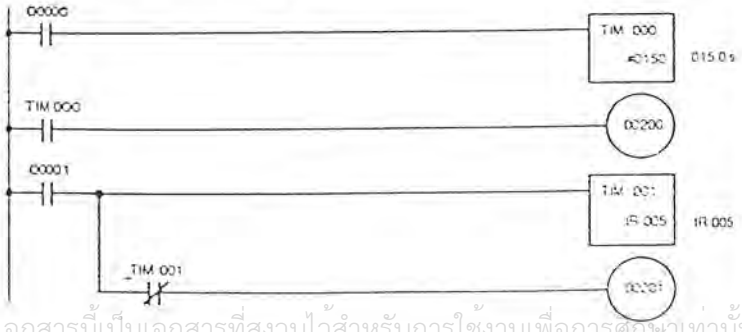
Indirectly addressed DM word is non-existent. (Content of *DM word is not BCD, or the DM area boundary has been exceeded.)

Examples

All of the following examples use OUT in diagrams that would generally be used to control output bits in the IR area. There is no reason, however, why these diagrams cannot be modified to control execution of other instructions.

Example 1: Basic Application

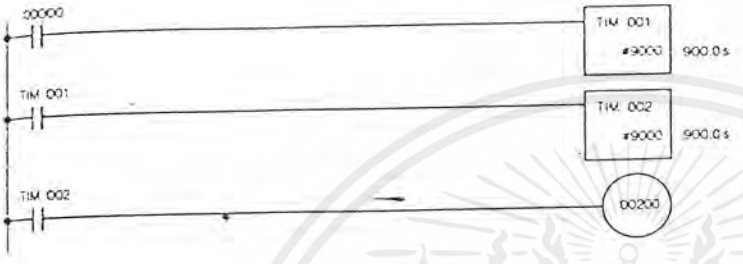
The following example shows two timers, one set with a constant and one set via input word 005. Here, 00200 will be turned ON after 00000 goes ON and stays ON for at least 15 seconds. When 00000 goes OFF, the timer will be reset and 00200 will be turned OFF. When 00001 goes ON, TIM 001 is started from the SV provided through IR word 005. Bit 00201 is also turned ON when 00001 goes ON. When the SV in 005 has expired, 00201 is turned OFF. This bit will also be turned OFF when TIM 001 is reset, regardless of whether or not SV has expired.



Address	Instruction	Operands
00000	LD	00000
00001	TIM	000
		# 0150
00002	LD	TIM 000
00003	OUT	00200
00004	LD	00001
00005	TIM	001
		005
00006	AND NOT	TIM 001
00007	OUT	00200

Example 2: Extended Timers

There are two ways to achieve timers that operate for longer than 999.9 seconds. One method is to program consecutive timers, with the Completion Flag of each timer used to activate the next timer. A simple example with two 900.0-second (15-minute) timers combined to functionally form a 30-minute timer.



Address	Instruction	Operands
00000	LD	00000
00001	TIM	001
		# 9000
00002	LD	TIM 001
00003	TIM	002
		# 9000
00004	LD	TIM 002
00005	OUT	00200

In this example, 00200 will be turned ON 30 minutes after 00000 goes ON.

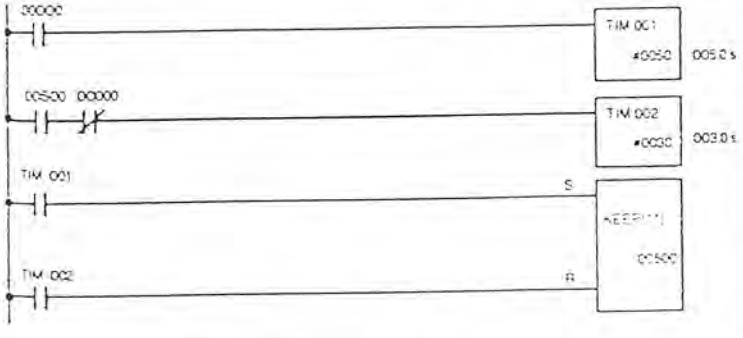
TIM can also be combined with CNT or CNT can be used to count SR area clock pulse bits to produce longer timers. An example is provided in *5-14-4 COUNTER – CNT*.

Example 3: ON/OFF Delays

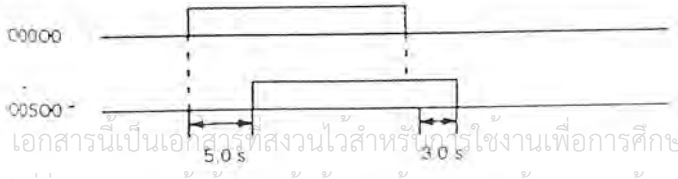
TIM can be combined with KEEP(11) to delay turning a bit ON and OFF in reference to a desired execution condition. KEEP(11) is described in *5-9-4 KEEP – KEEP(11)*.

To create delays, the Completion Flags for two TIM are used to determine the execution conditions for setting and reset the bit designated for KEEP(11). The bit whose manipulation is to be delayed is used in KEEP(11). Turning ON and OFF the bit designated for KEEP(11) is thus delayed by the SV for the two TIM. The two SV could naturally be the same if desired.

In the following example, 00500 would be turned ON 5.0 seconds after 00000 goes ON and then turned OFF 3.0 seconds after 00000 goes OFF. It is necessary to use both 00500 and 00000 to determine the execution condition for TIM 002; 00000 in an inverse condition is necessary to reset TIM 002 when 00000 goes ON and 00500 is necessary to activate TIM 002 (when 00000 is OFF).

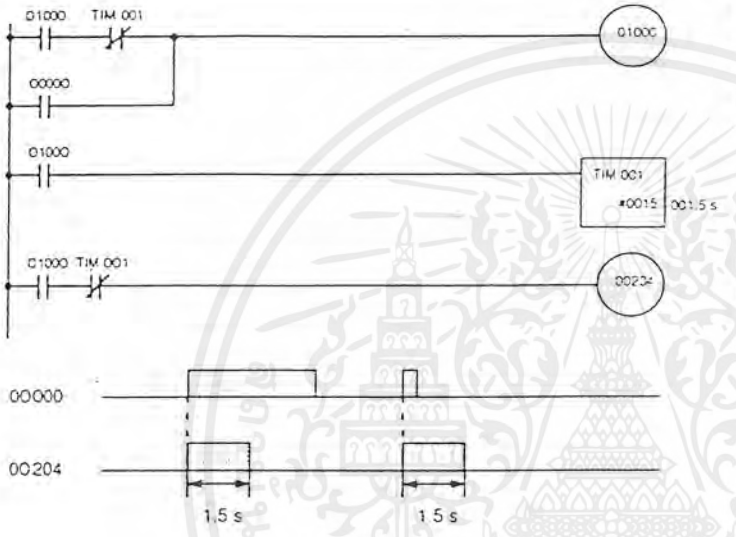


Address	Instruction	Operands
00000	LD	00000
00001	TIM	001
		# 0050
00002	LD	00500
00003	AND NOT	00000
00004	TIM	* 002
		# 0030
00005	LD	TIM 001
00006	LD	TIM 002
00007	KEEP(11)	00500



Example 4: One-Shot Bits

The length of time that a bit is kept ON or OFF can be controlled by combining TIM with OUT or OUT NO. The following diagram demonstrates how this is possible. In this example, 00204 would remain ON for 1.5 seconds after 00000 goes ON regardless of the time 00000 stays ON. This is achieved by using 01000 as a self-maintaining bit activated by 00000 and turning ON 00204 through it. When TIM 001 comes ON (i.e., when the SV of TIM 001 has expired), 00204 will be turned OFF through TIM 001 (i.e., TIM 001 will turn ON which, as an inverse condition, creates an OFF execution condition for OUT 00204).



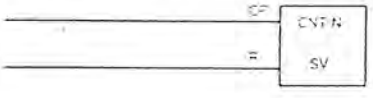
Address	Instruction	Operands	
00000	LD		01000
00001	AND NOT	TIM	001
00002	OR		00000
00003	OUT		01000
00004	LD		01000
00005	TIM		001
		#	0015
00006	LD		01000
00007	AND NOT	TIM	001
00008	OUT		00204

The following one-shot timer may be used to save memory.

Address	Instruction	Operands	
00000	LD		00000
00001	OR		00100
00002	TIM		001
		#	0015
00003	AND NOT	TIM	001
00004	OUT		00100

5-14-4 COUNTER – CNT

Ladder Symbol



Definer Values

N: TC number
(000 through 511)

Operand Data Areas

SV: Set value (word, BCD)
IR, AR, DM, HR, LR, #

Limitations

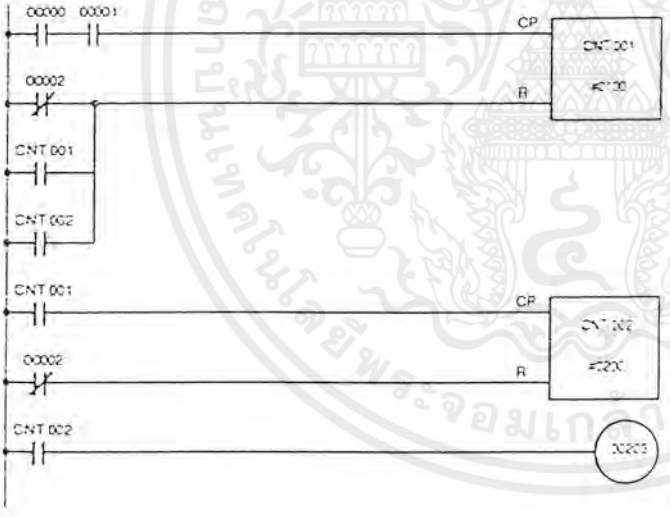
Each TC number can be used as the definer in only one TIMER or COUNTER instruction.

Example 2:
Extended Counter

Counters that can count past 9,999 can be programmed by using one CNT to count the number of times another CNT has counted to zero from SV.

In the following example, 00000 is used to control when CNT 001 operates. CNT 001, when 00000 is ON, counts down the number of OFF to ON changes in 00001. CNT 001 is reset by its Completion Flag, i.e., it starts counting again as soon as its PV reaches zero. CNT 002 counts the number of times the Completion Flag for CNT 001 goes ON. Bit 00002 serves as a reset for the entire extended counter, resetting both CNT 001 and CNT 002 when it is OFF. The Completion Flag for CNT 002 is also used to reset CNT 001 to inhibit CNT 001 operation, once SV for CNT 002 has been reached, until the entire extended counter is reset via 00002.

Because in this example the SV for CNT 001 is 100 and the SV for CNT 002 is 200, the Completion Flag for CNT 002 turns ON when 100 x 200 or 20,000 OFF to ON changes have been counted in 00001. This would result in 00203 being turned ON.



Address	Instruction	Operands	
00000	LD		00000
00001	AND		00001
00002	LD NOT		00002
00003	OR	CNT	001
00004	OR	CNT	002
00005	CNT		001
		#	0100
00006	LD	CNT	001
00007	LD NOT		00002
00008	CNT		002
		#	0200
00009	LD	CNT	002
00010	OUT		00203

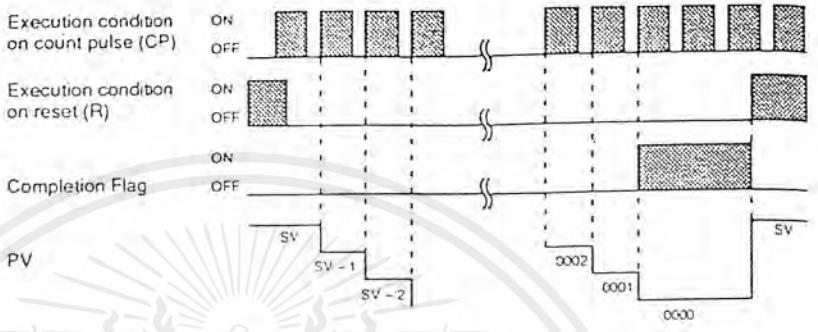
CNT can be used in sequence as many times as required to produce counters capable of counting any desired values.

Example 3:
Extended Timers

CNT can be used to create extended timers in two ways: by combining TIM with CNT and by counting SR area clock pulse bits.

In the following example, CNT 002 counts the number of times TIM 001 reaches zero from its SV. The Completion Flag for TIM 001 is used to reset TIM 001 so that it runs continuously and CNT 002 counts the number of times the Completion Flag for TIM 001 goes ON (CNT 002 would be executed once each time between when the Completion Flag for TIM 001 goes ON and TIM 001 is reset by its Completion Flag). TIM 001 is also reset by the Completion Flag for CNT 002 so that the extended timer would not start again until CNT 002 was reset by 00001, which serves as the reset for the entire extended timer.

Changes in execution conditions, the Completion Flag, and the PV are illustrated below. PV line height is meant only to indicate changes in the PV.



Precautions

Program execution will continue even if a non-BCD SV is used, but the SV will not be correct.

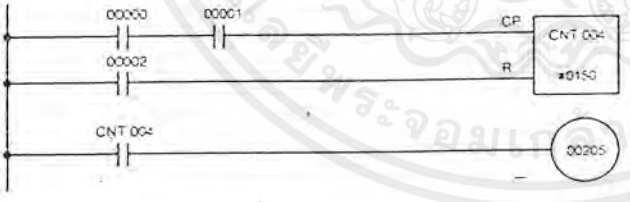
Flags

ER: SV is not in BCD.

Indirectly addressed DM word is non-existent. (Content of *DM word is not BCD, or the DM area boundary has been exceeded.)

Example 1: Basic Application

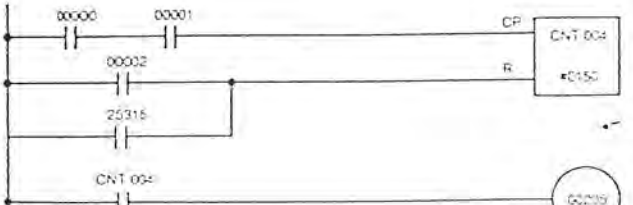
In the following example, the PV will be decremented whenever both 00000 and 00001 are ON provided that 00002 is OFF and either 00000 or 00001 was OFF the last time CNT 004 was executed. When 150 pulses have been counted down (i.e., when PV reaches zero), 00205 will be turned ON.



Address	Instruction	Operands
00000	LD	00000
00001	AND	00001
00002	LD	00002
00003	CNT	0004
		# 0150
00004	LD	CNT 004
00005	OUT	00205

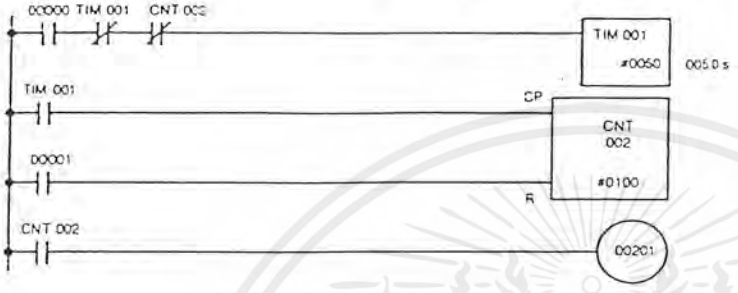
Here, 00000 can be used to control when CNT is operative and 00001 can be used as the bit whose OFF to ON changes are being counted.

The above CNT can be modified to restart from SV each time power is turned ON to the PC. This is done by using the First Cycle Flag in the SR area (25315) to reset CNT as shown below.



Address	Instruction	Operands
00000	LD	00000
00001	AND	00001
00002	LD	00002
00003	OR	25315
00004	CNT	004
		# 0150
00005	LD	CNT 004
00006	OUT	00205

Because in this example the SV for TIM 001 is 5.0 seconds and the SV for CNT 002 is 100, the Completion Flag for CNT 002 turns ON when 5 seconds x 100 times, i.e., 500 seconds (or 8 minutes and 20 seconds) have expired. This would result in 00201 being turned ON.



Address	Instruction	Operands
00000	LD	00000
00001	AND NOT	TIM 001
00002	AND NOT	CNT 002
00003	TIM	001
		# 0050
00004	LD	TIM 001
00005	LD	00001
00006	CNT	002
		# 0100
00007	LD	CNT 002
00008	OUT	00201

In the following example, CNT 001 counts the number of times the 1-second clock pulse bit (25502) goes from OFF to ON. Here again, 00000 is used to control the times when CNT is operating.

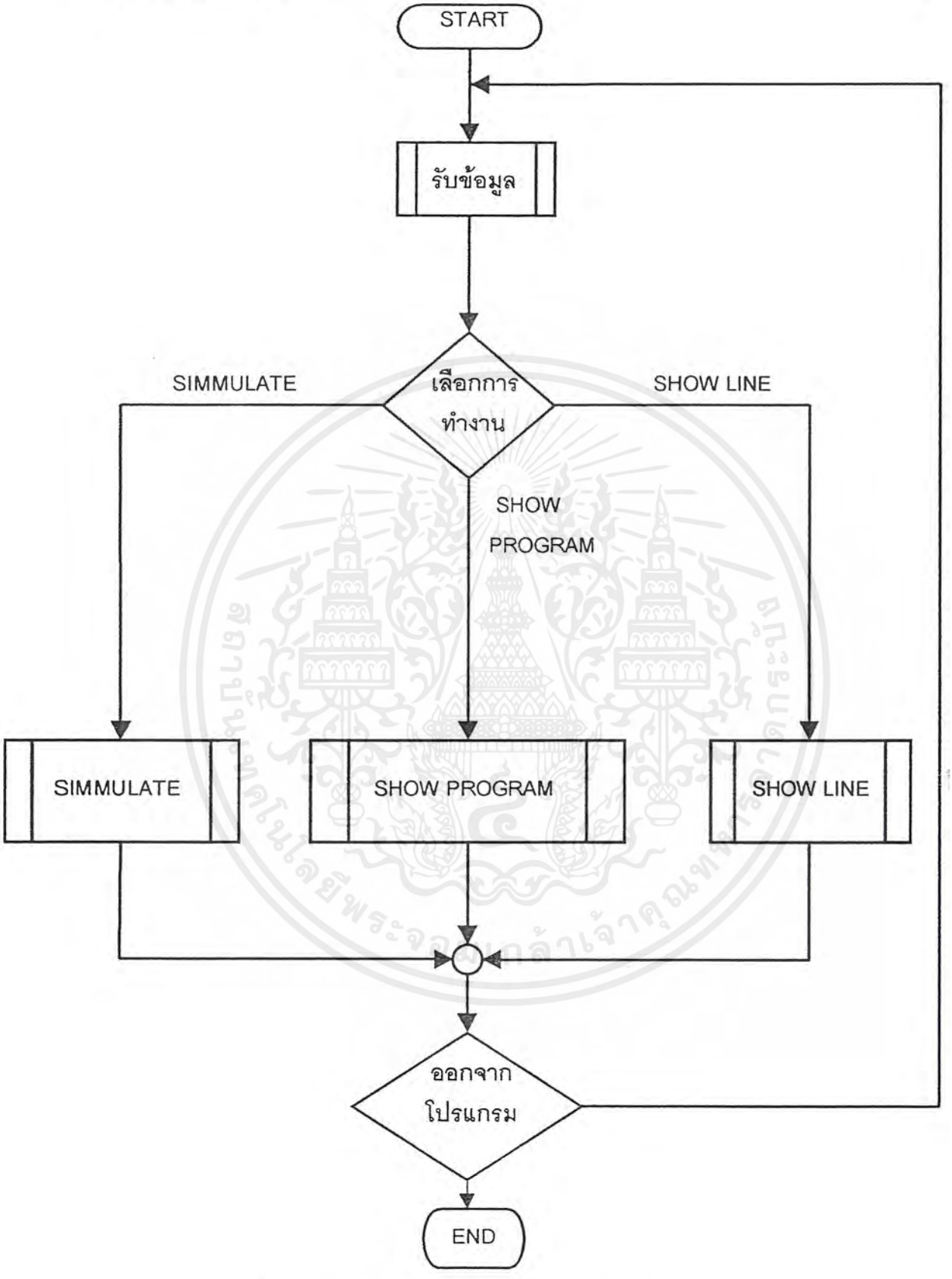
Because in this example the SV for CNT 001 is 700, the Completion Flag for CNT 002 turns ON when 1 second x 700 times, or 11 minutes and 40 seconds have expired. This would result in 00202 being turned ON.

Address	Instruction	Operands
00000	LD	00000
00001	AND	25502
00002	LD NOT	00001
00003	CNT	001
		# 0700
00004	LD	CNT 001
00005	OUT	00202

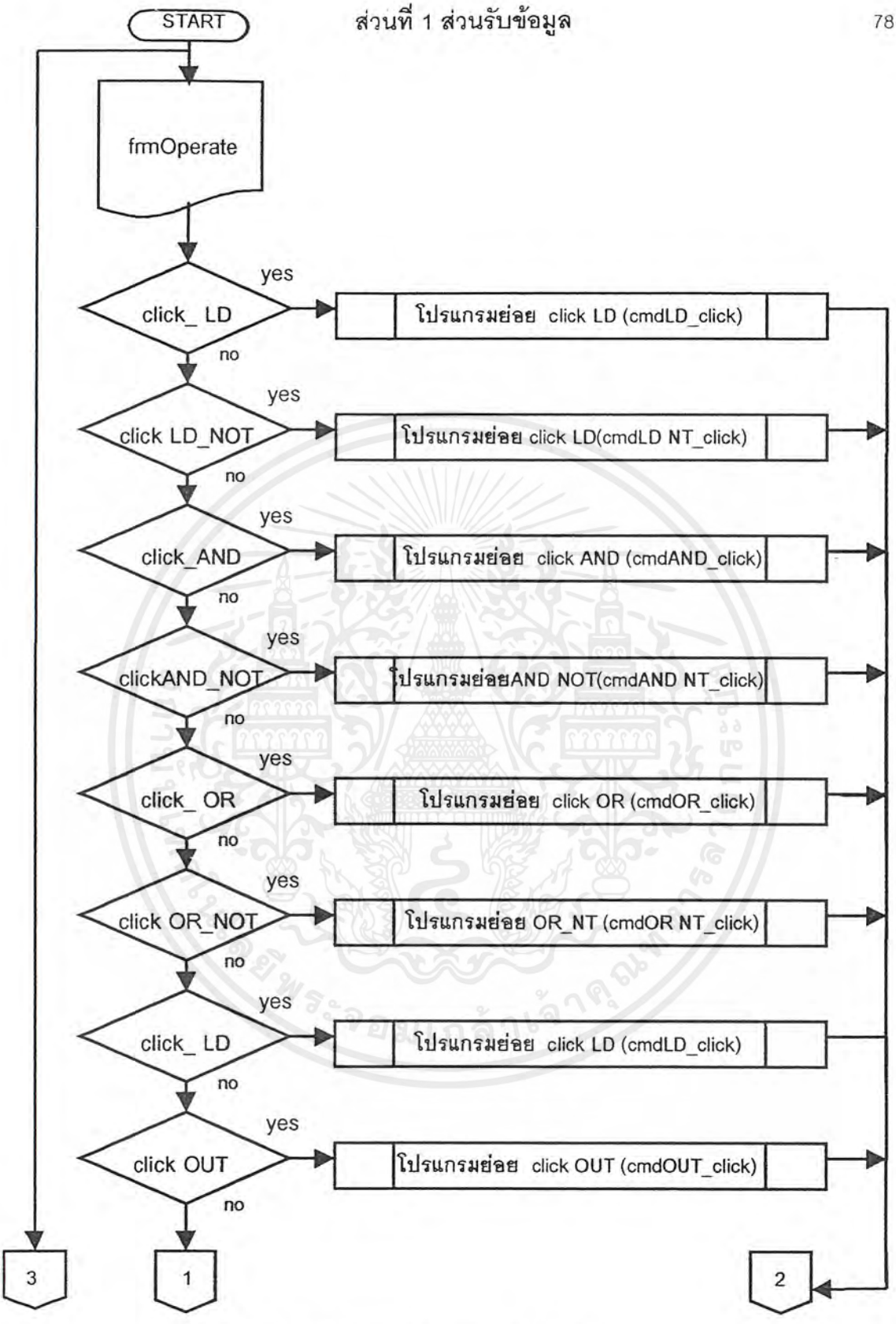
⚠ Caution The shorter clock pulses will not necessarily produce accurate timers because their short ON times might not be read accurately during longer cycles. In particular, the 0.02-second and 0.1-second clock pulses should not be used to create timers with CNT instructions.

ภาคผนวก ข

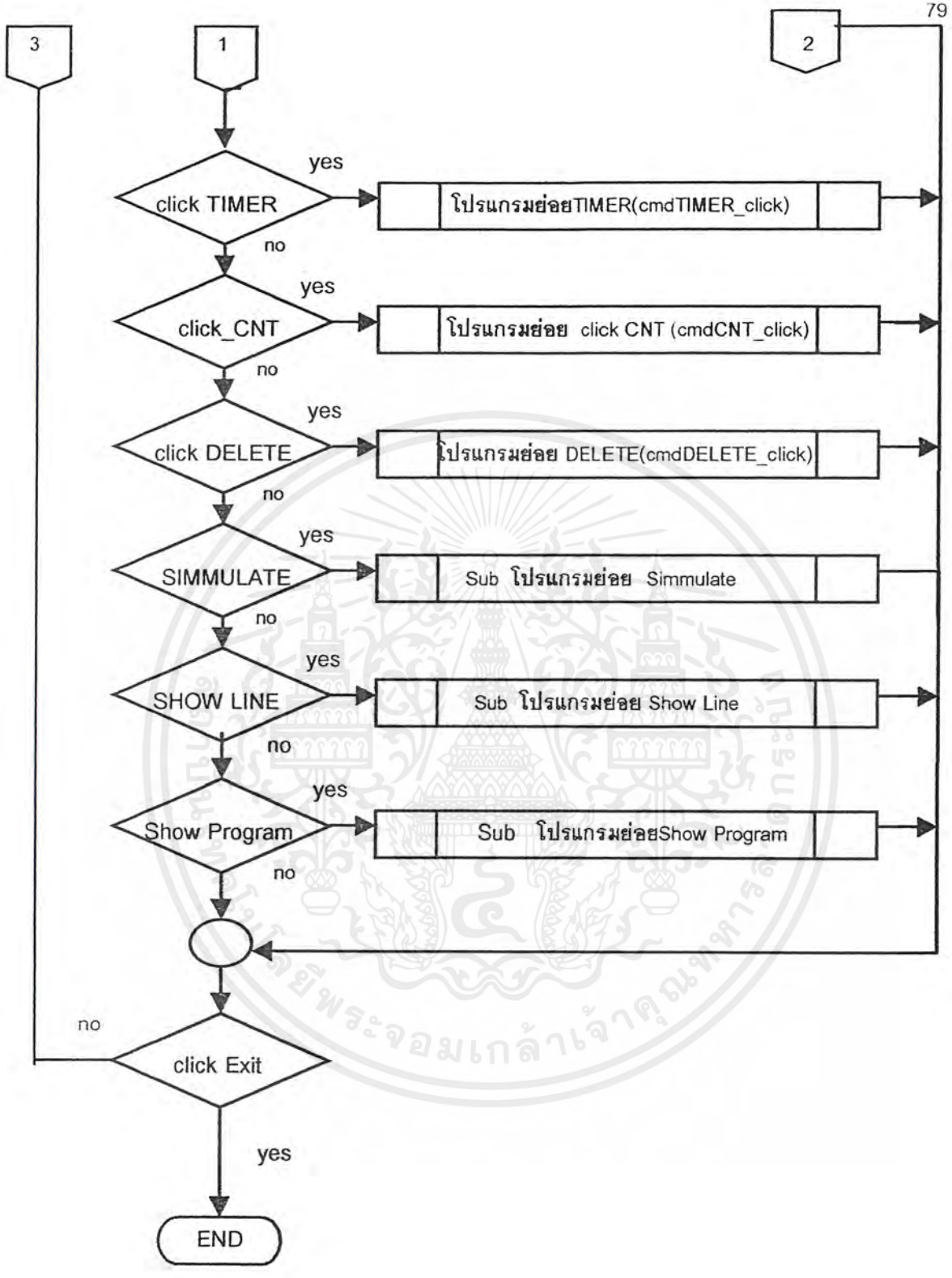
กราฟแสดงผลของโปรแกรม



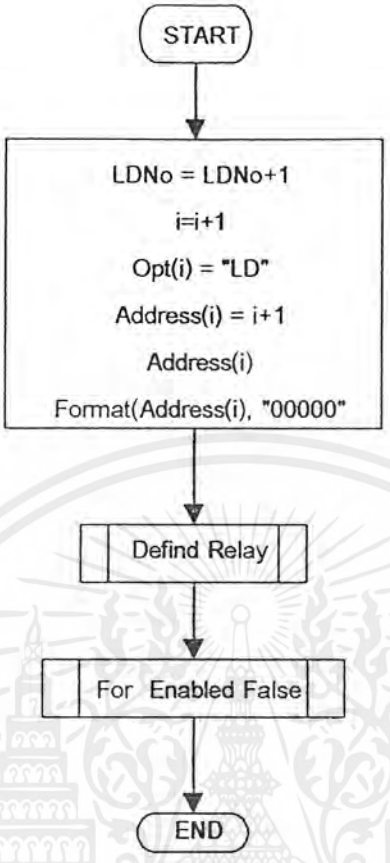
รูปที่ ข.1 กราฟแสดงส่วนการทำงานหลักของโปรแกรม



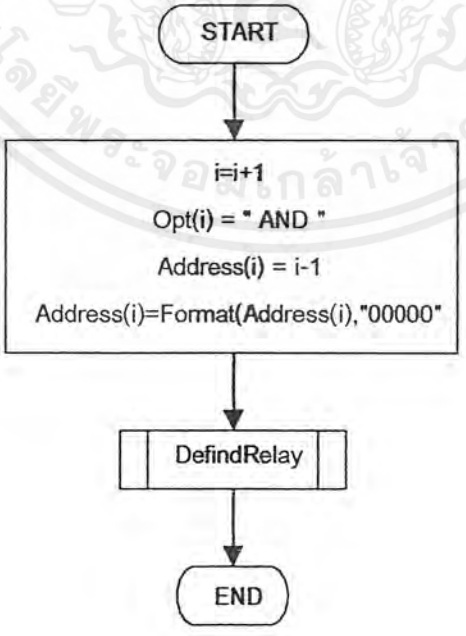
รูปที่ ข.2 กราฟแสดงการรับข้อมูล



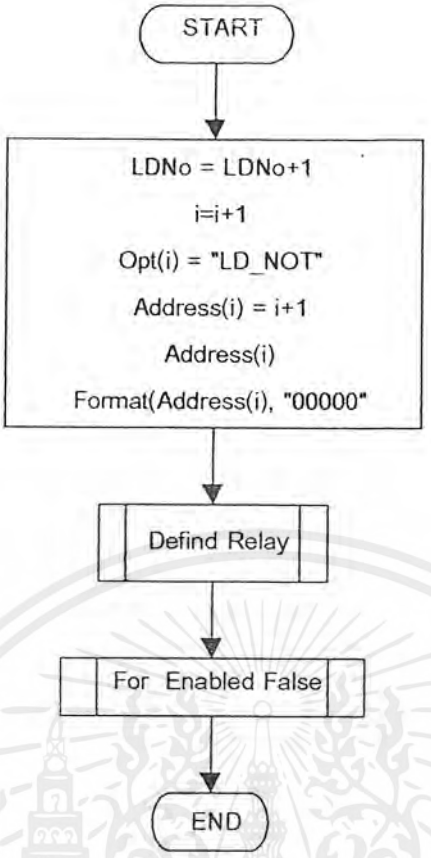
รูปที่ ข.3 กราฟแสดงการรับข้อมูล(ต่อ)



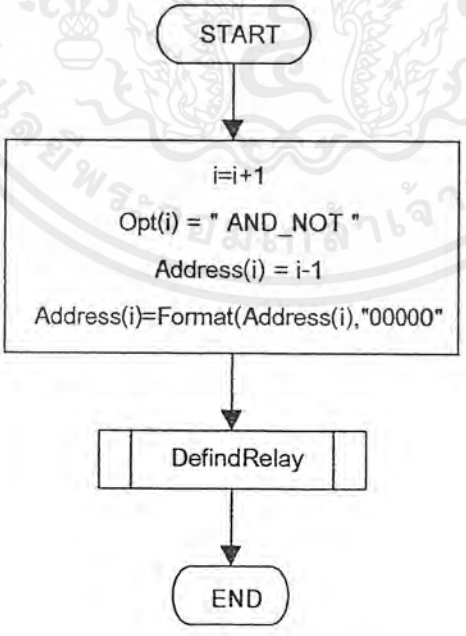
รูปที่ ข.4 กราฟแสดงการประมวลผล cmdLD_click



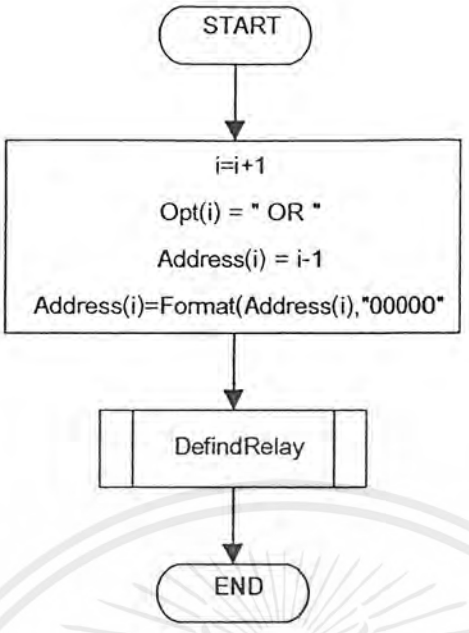
รูปที่ ข.5 กราฟแสดงการประมวลผล cmdAND_click



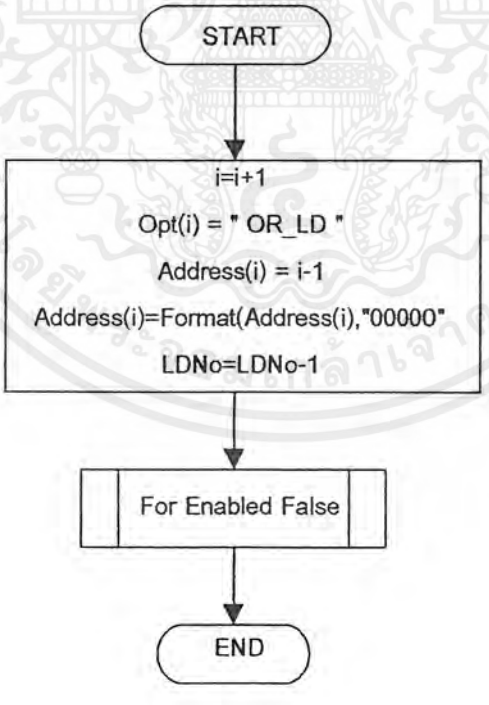
รูปที่ ข.6 กราฟแสดงการประมวลผล cmdLDNt_click



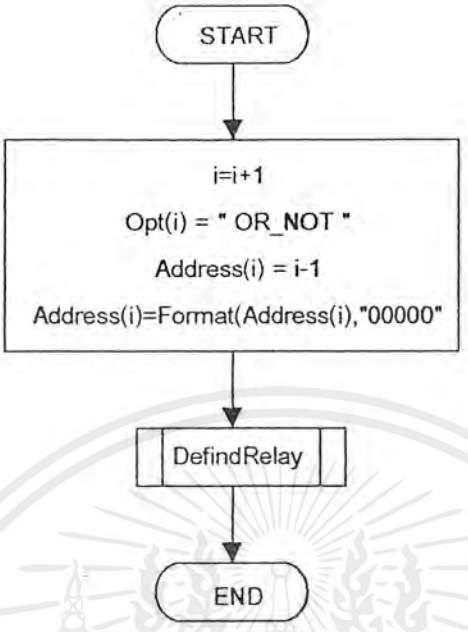
รูปที่ ข.7 กราฟแสดงการประมวลผล cmdAndNt_click



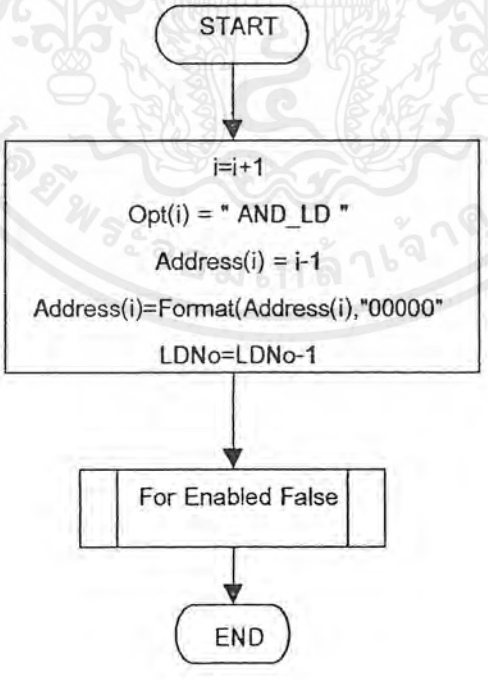
รูปที่ ข.8 กราฟแสดงการประมวลผล cmdOR_click



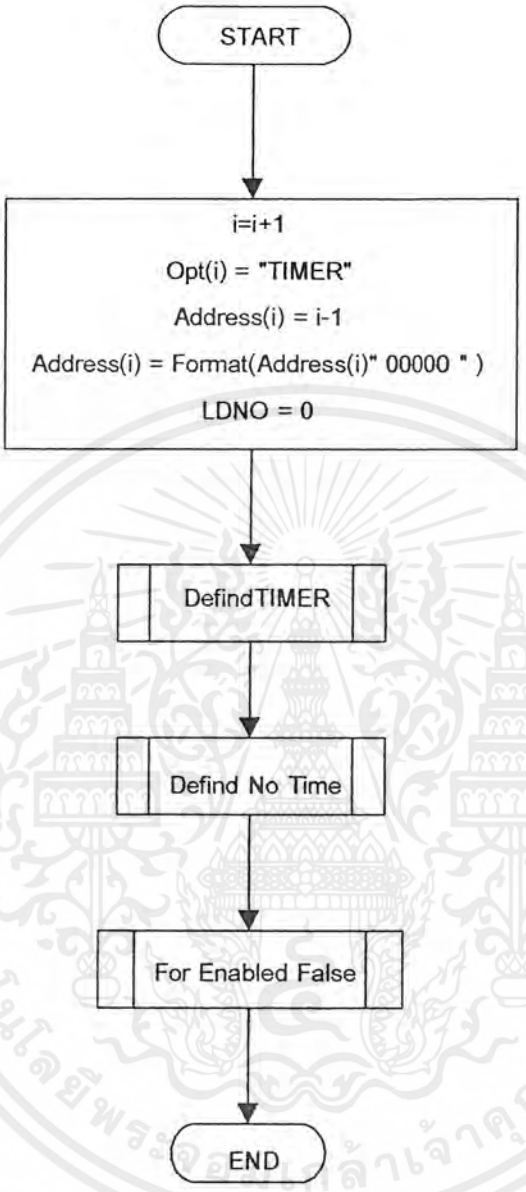
รูปที่ ข.9 กราฟแสดงการประมวลผล cmdOrLD_click



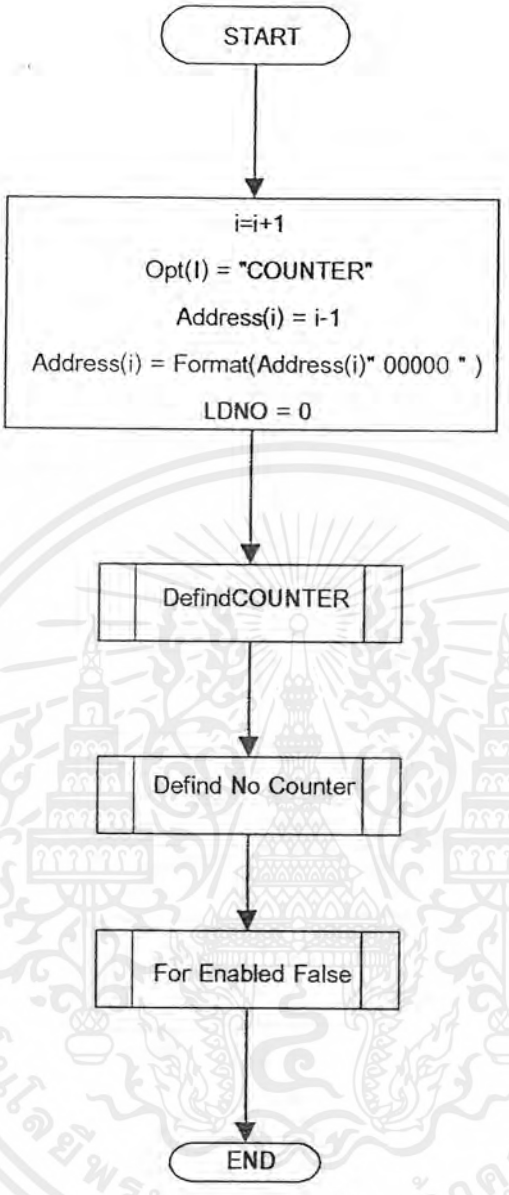
รูปที่ ข.10 กราฟแสดงการประมวลผล cmdOrNt_click



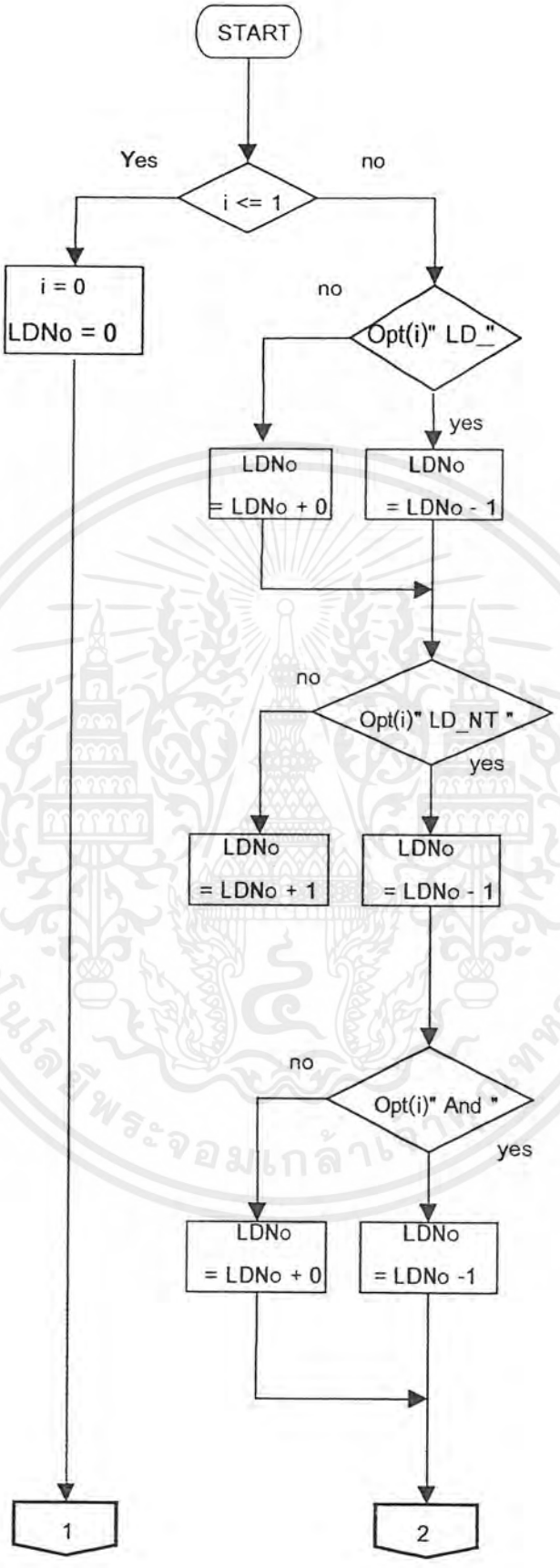
รูปที่ ข.11 กราฟแสดงการประมวลผล cmdAndLD_click



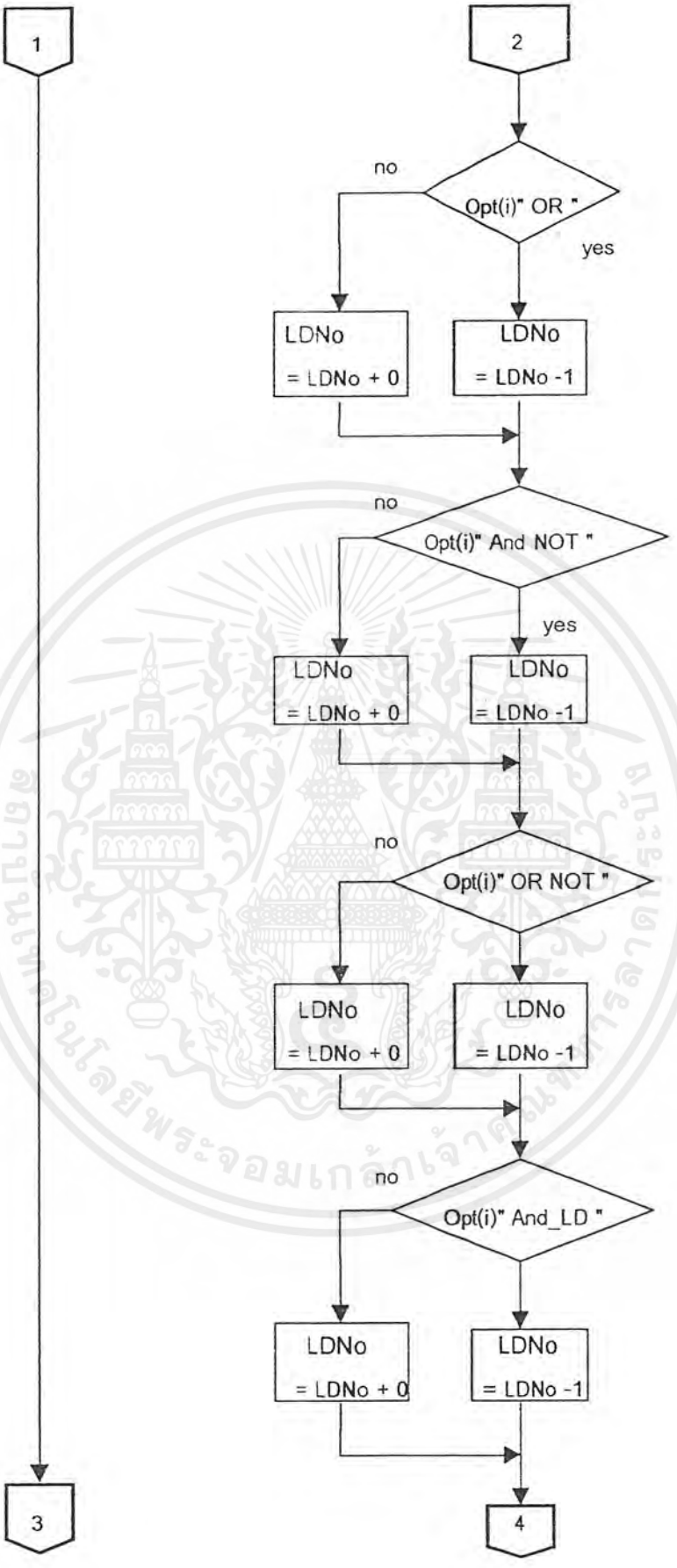
รูปที่ ข.12 กราฟแสดงการประมวลผล cmdTIMER_click



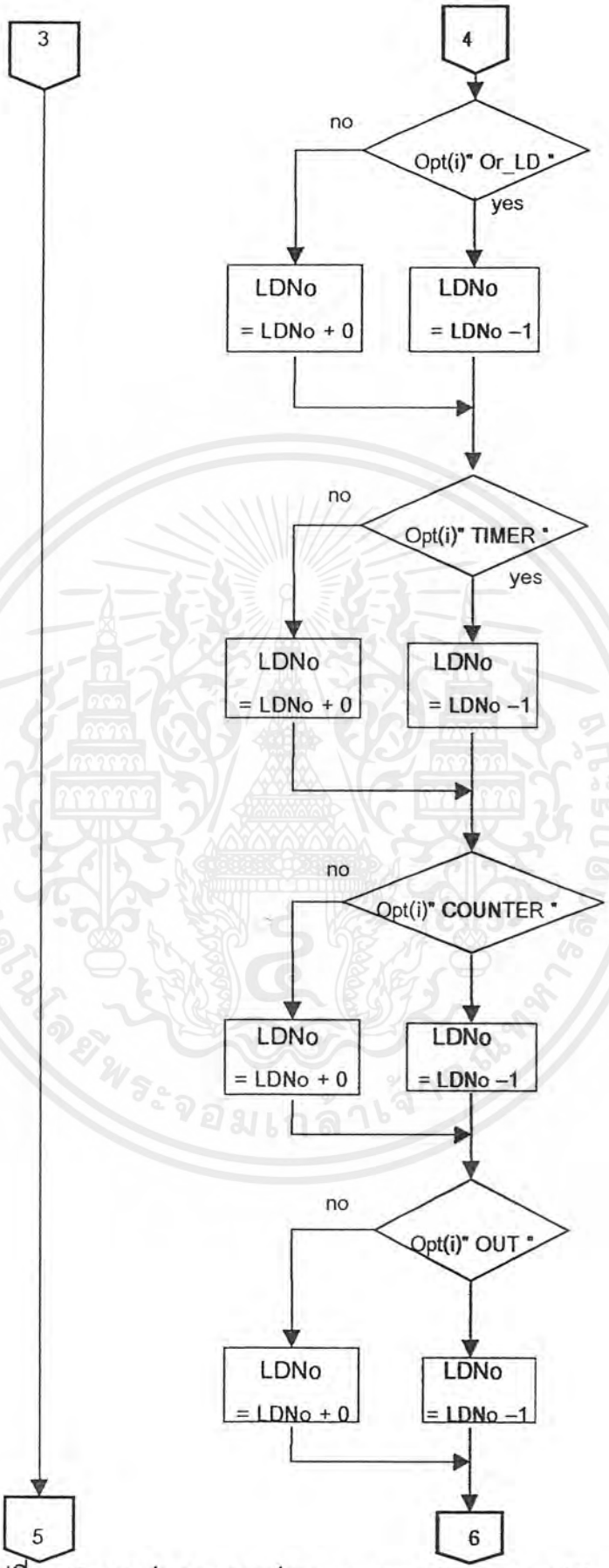
รูปที่ ข.13 กราฟแสดงการประมวลผล cmdCounter_click



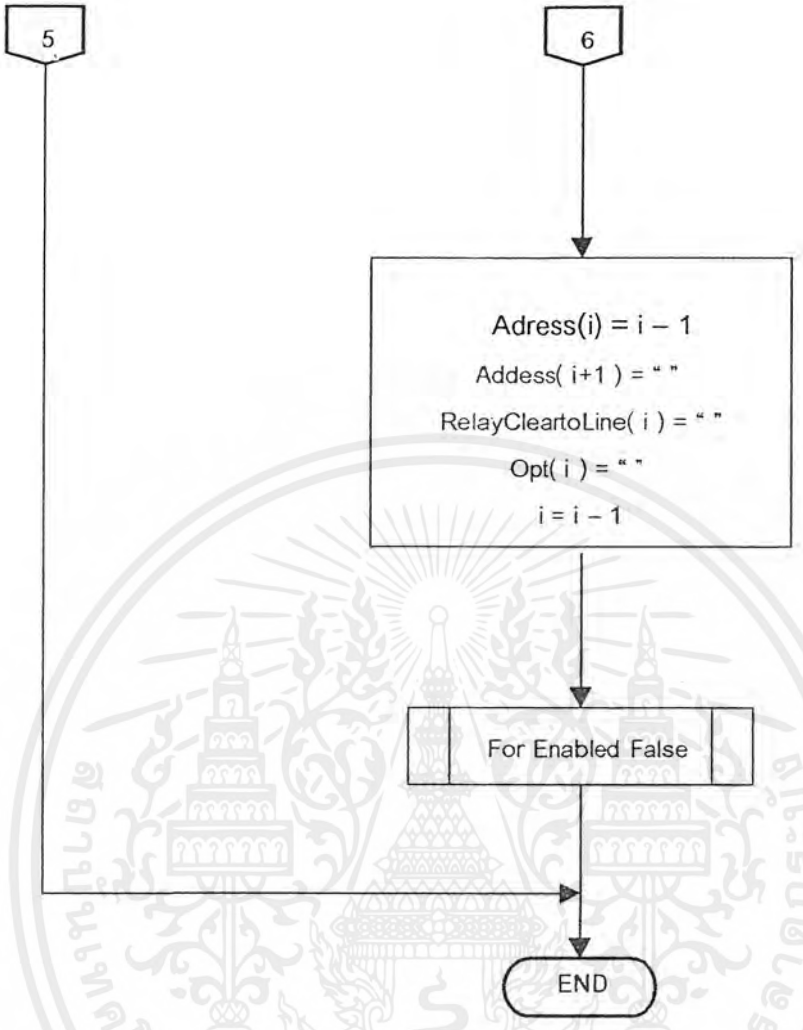
รูปที่ ข.14 กราฟแสดงการประมวลผล cmdDelete_click(1)



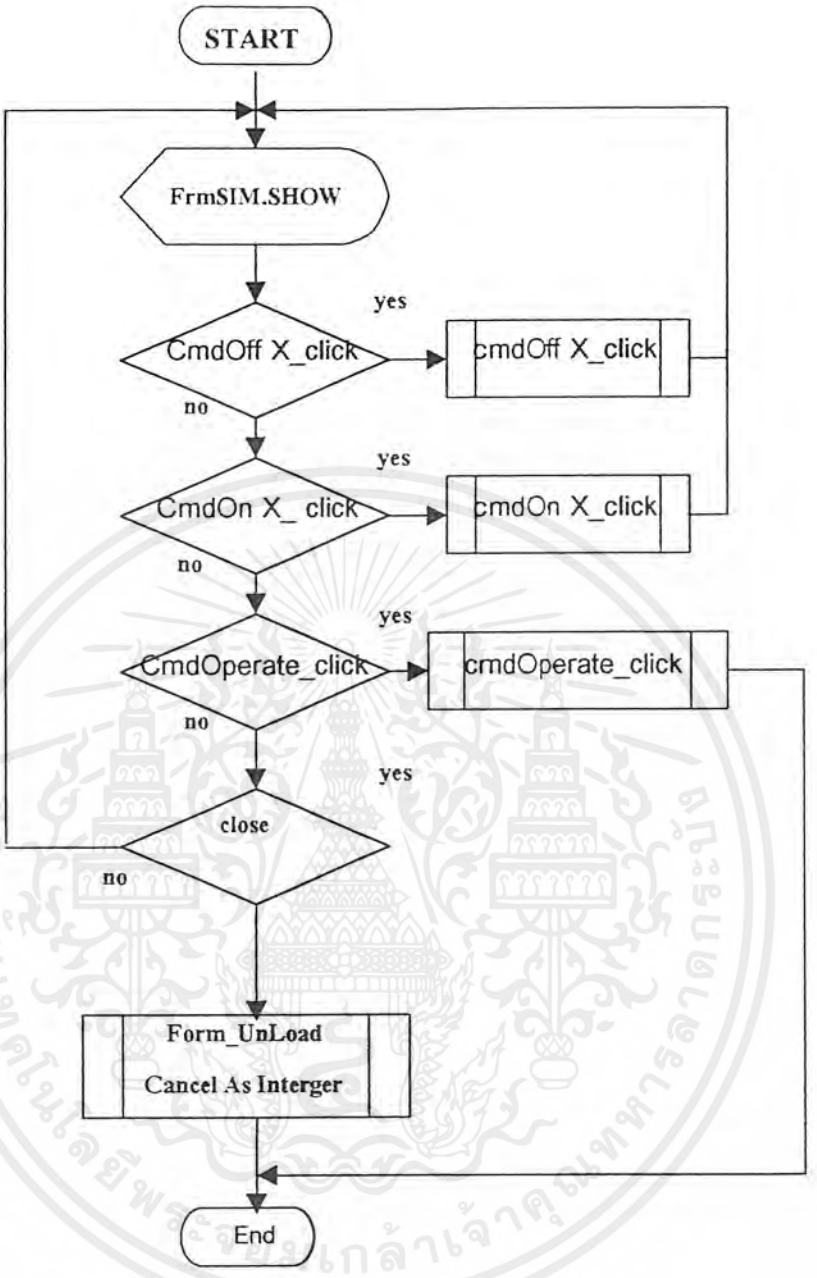
รูปที่ ข.15 กราฟแสดงการประมวลผลcmdDelete_click(2)



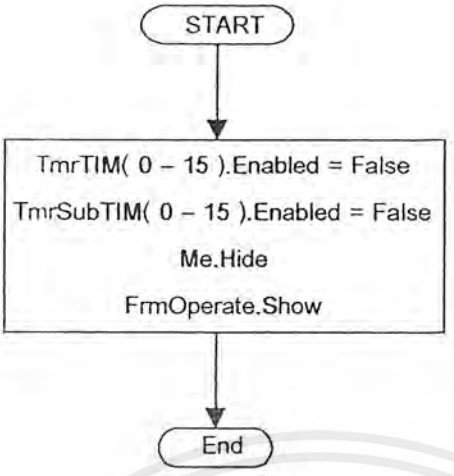
รูปที่ ข.16 กราฟแสดงการประมวลผลcmdDelete_click(3)



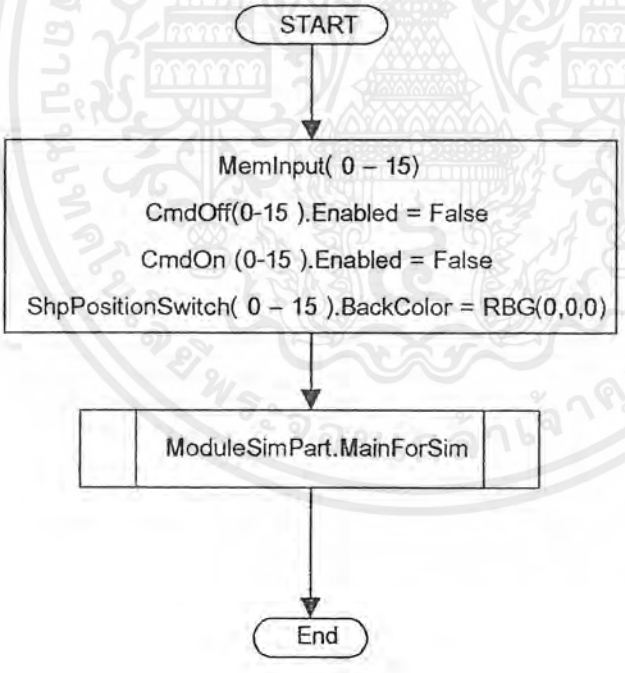
รูปที่ ข.17 กราฟแสดงการประมวลผล cmdDelete_click(4)



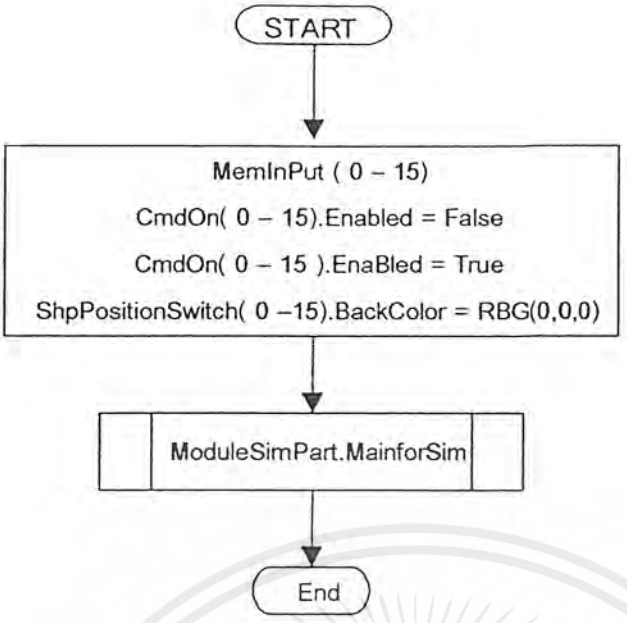
รูปที่ ข.18 กราฟแสดงการประมวลผล ฟอร์ม SIM



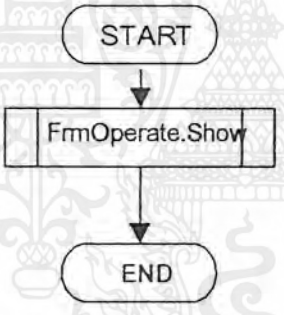
รูปที่ ข.19 กราฟแสดงการประมวลผล cmdGotoOperate_click



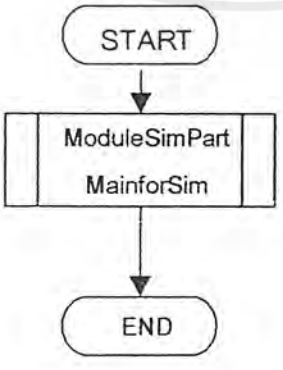
รูปที่ ข.20 กราฟแสดงการประมวลผล cmdOff (0 – 15)_click



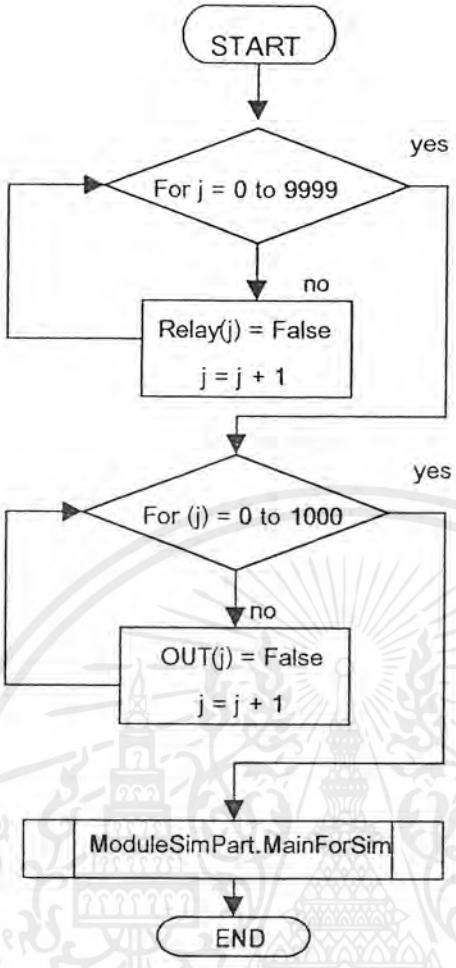
รูปที่ ข.21 กราฟแสดงการประมวลผล cmdOn(0 – 15)_click



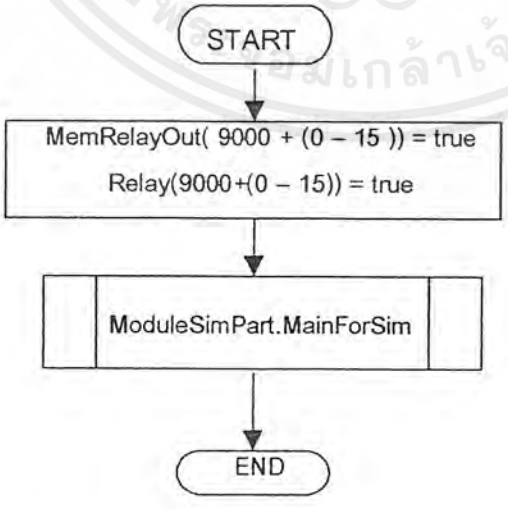
รูปที่ ข.22 กราฟแสดงการประมวลผล Form.Unload



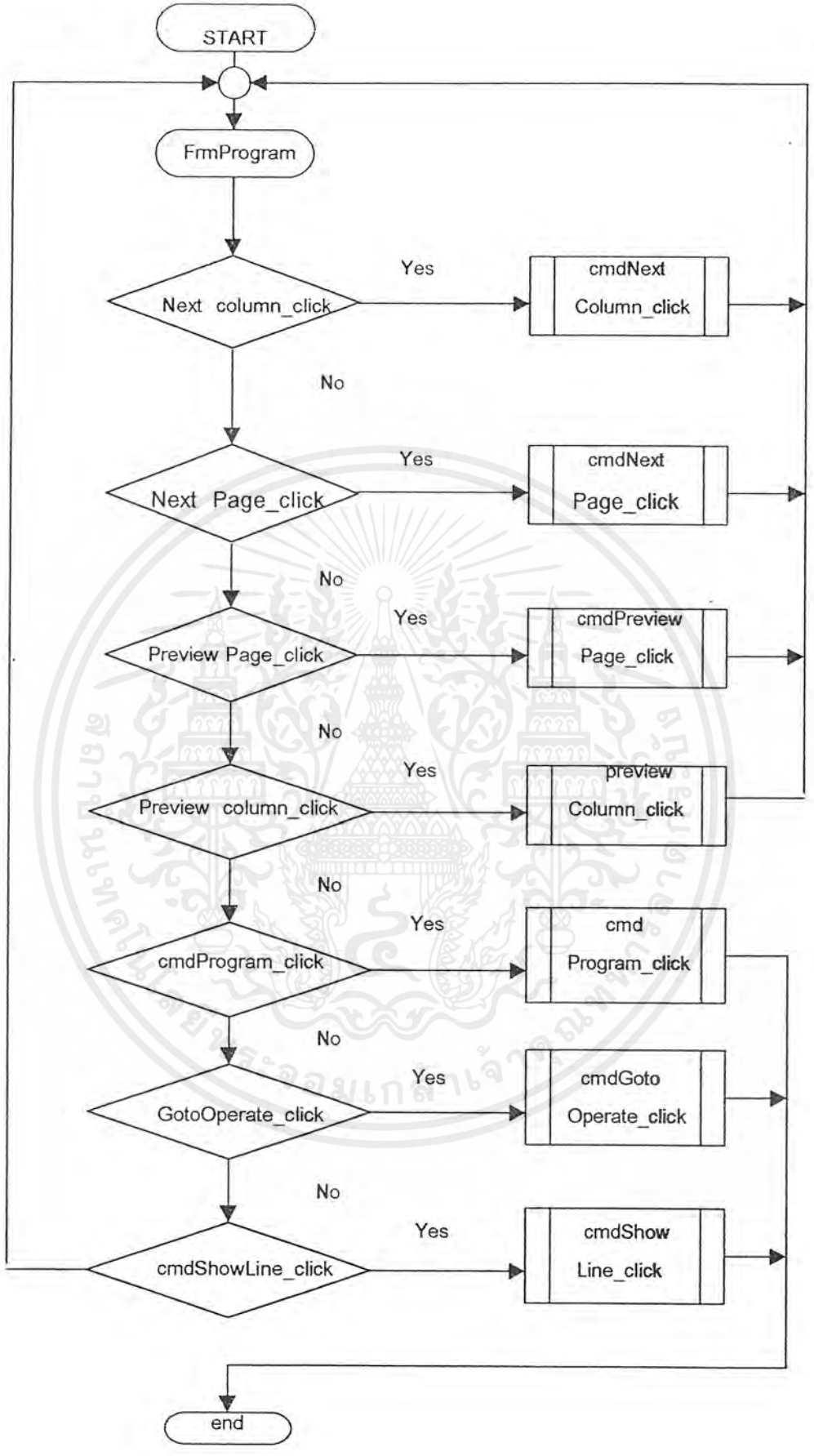
รูปที่ ข.23 กราฟแสดงการประมวลผล tmrSubTIM(00-15)



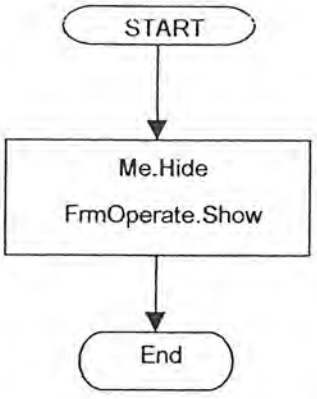
รูปที่ ข.24 กราฟแสดงการประมวลผล Form.load



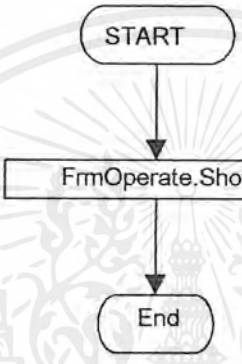
รูปที่ ข.25 กราฟแสดงการประมวลผล tmrTIM(00-15)



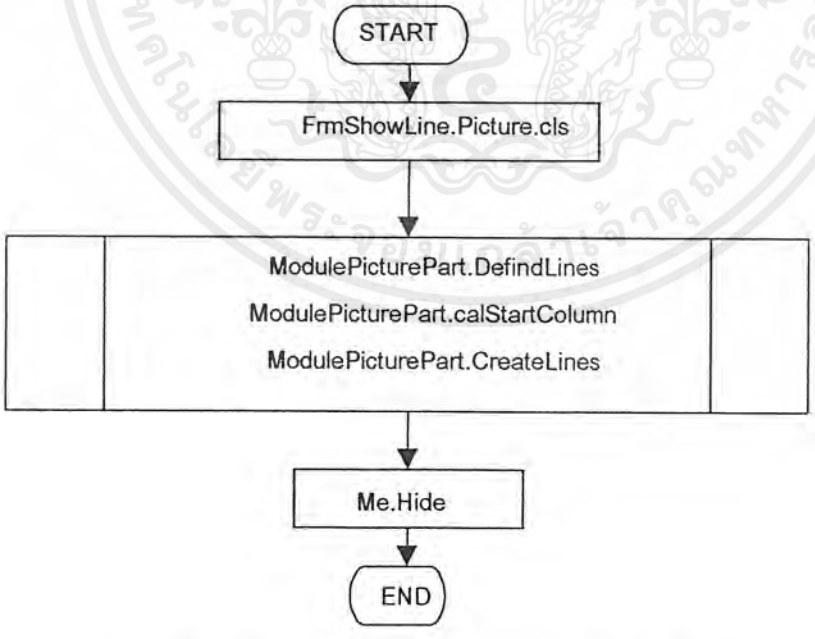
รูปที่ ข.26 กราฟแสดงการประมวลผล Form Program



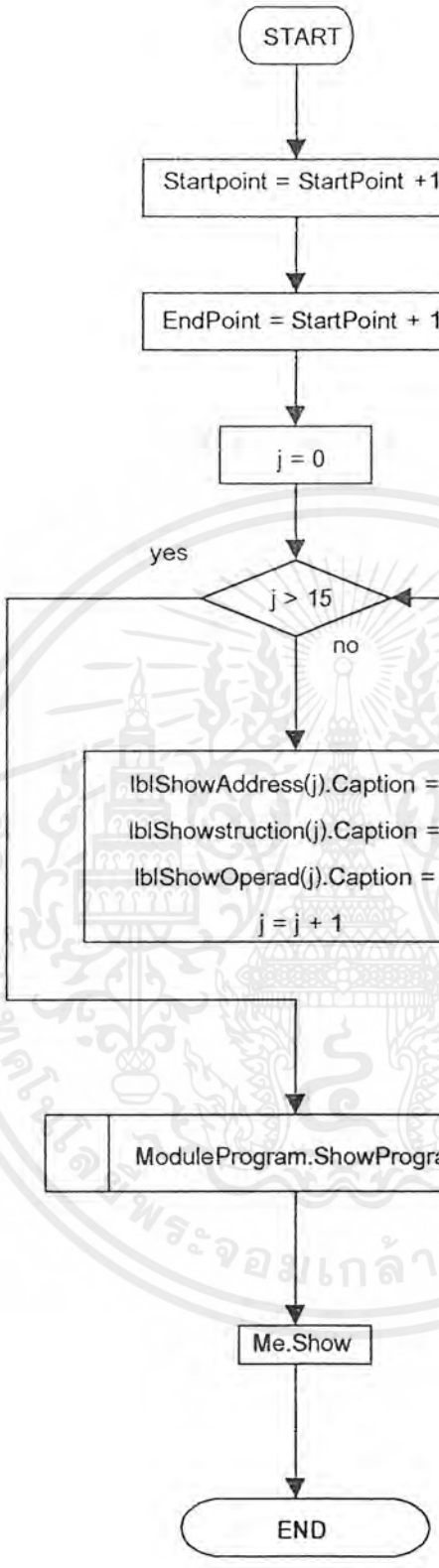
รูปที่ ข.27 กราฟแสดงการประมวลผล Operate_click



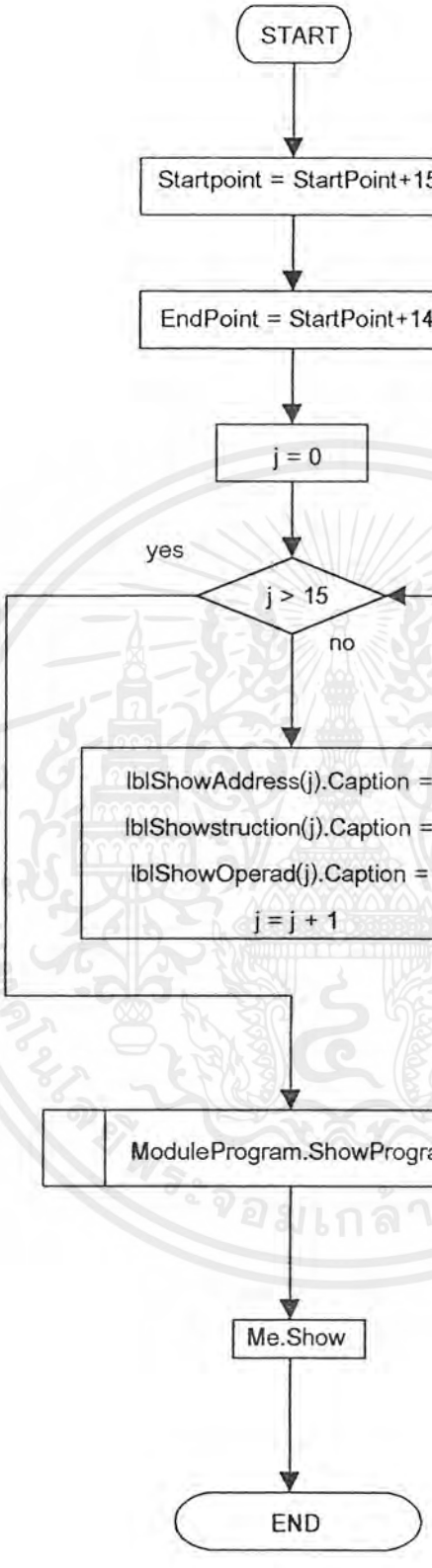
รูปที่ ข.28 กราฟแสดงการประมวลผล Form _Unload
(Cancel As Interger)



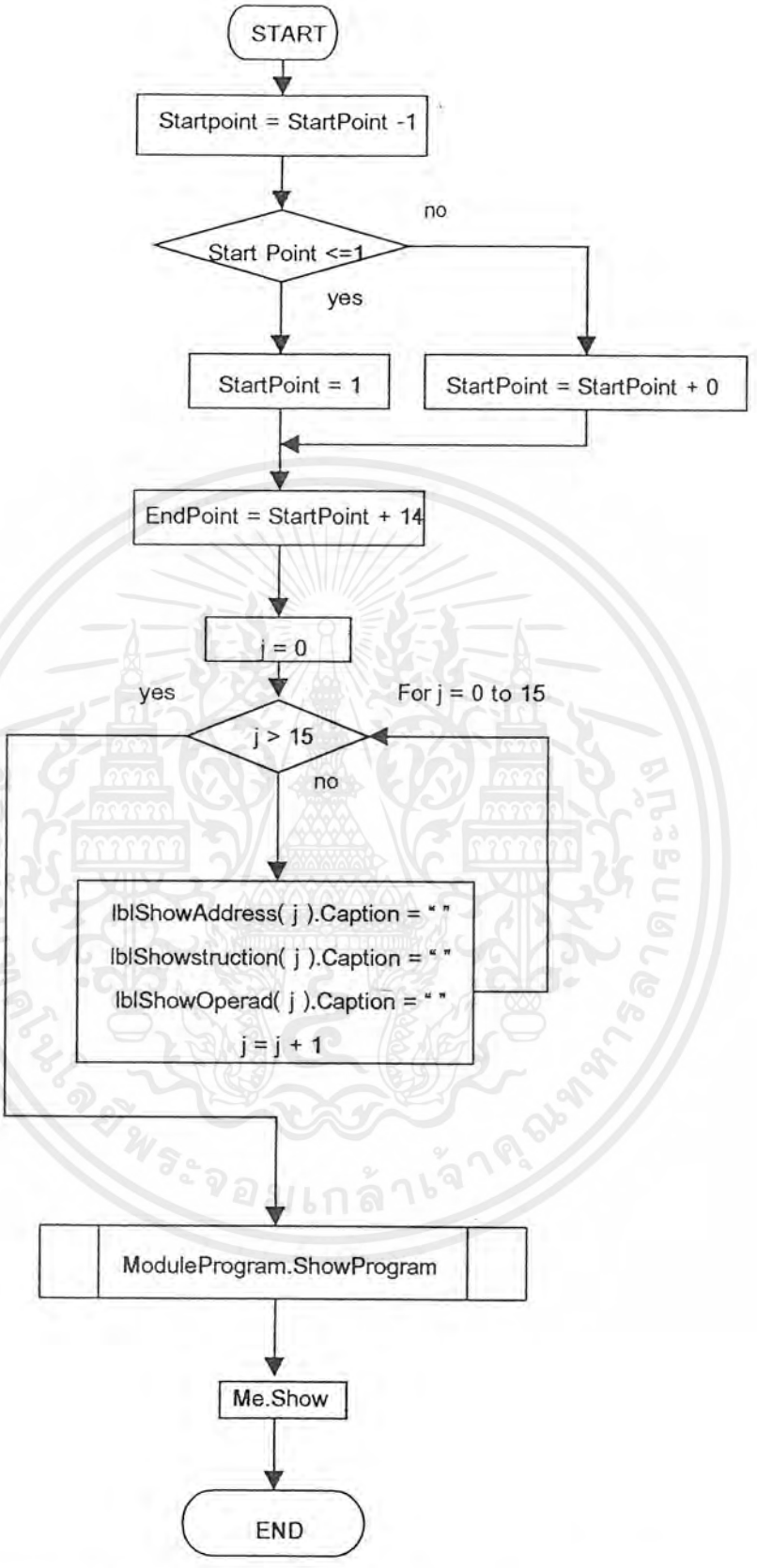
รูปที่ ข.29 กราฟแสดงการประมวลผล Show Line



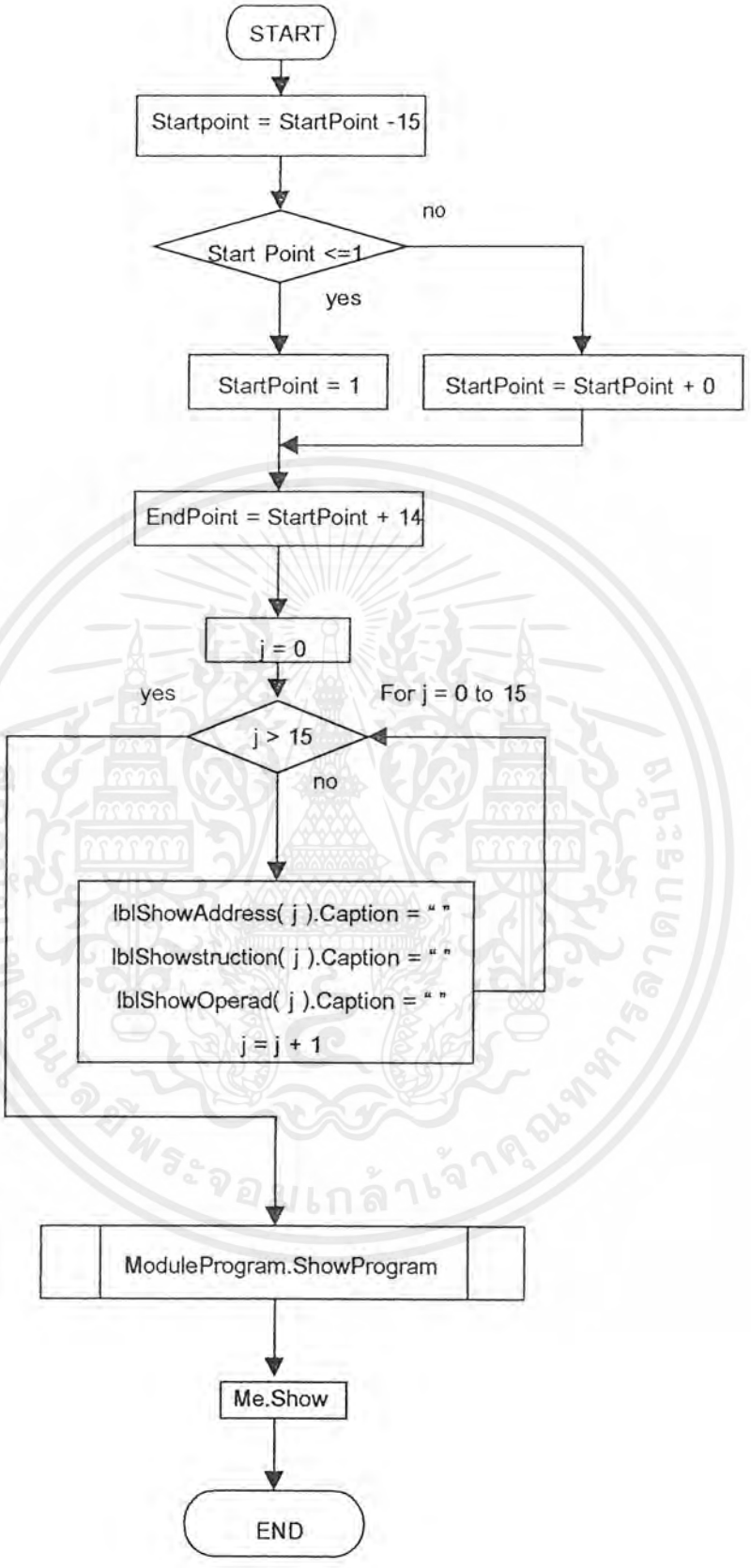
รูปที่ ข.30 กราฟแสดงการประมวลผลของ
Cmd NextColumn_click



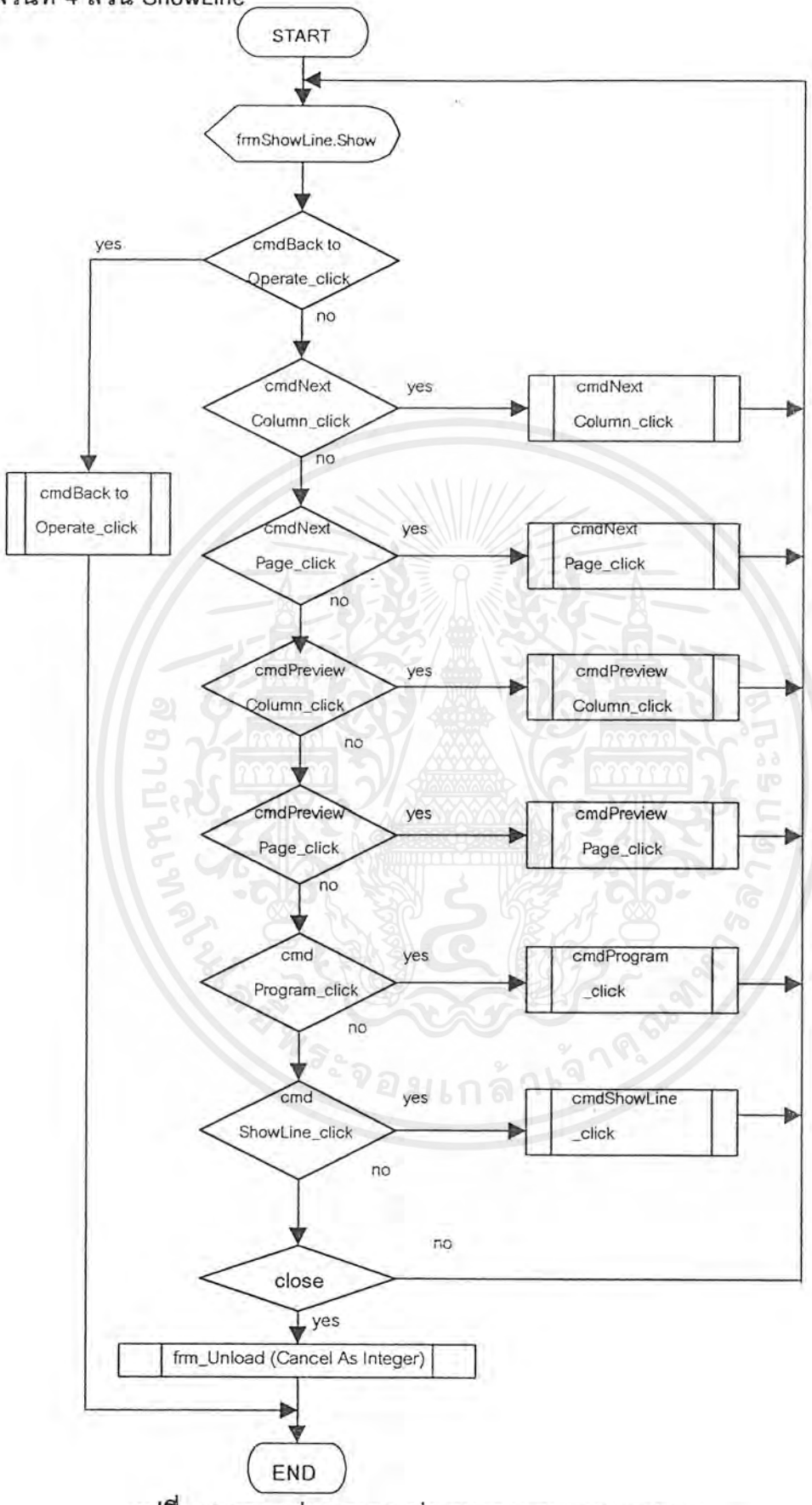
รูปที่ ข.31 กราฟแสดงการประมวลผลของcmdNextPage_click



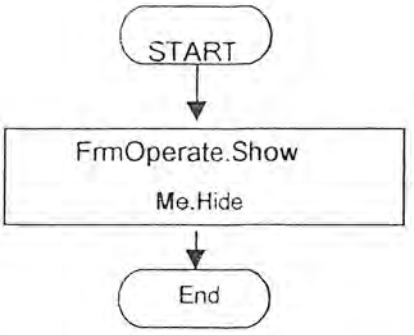
รูปที่ ข.32 กราฟแสดงการประมวลผลของCmdPreviewColumn_click



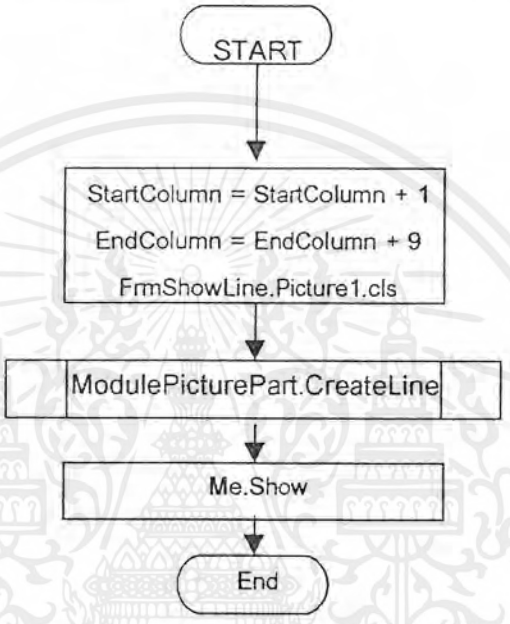
รูปที่ ข.33 กราฟแสดงการประมวลผลของcmdPreviewPage_click



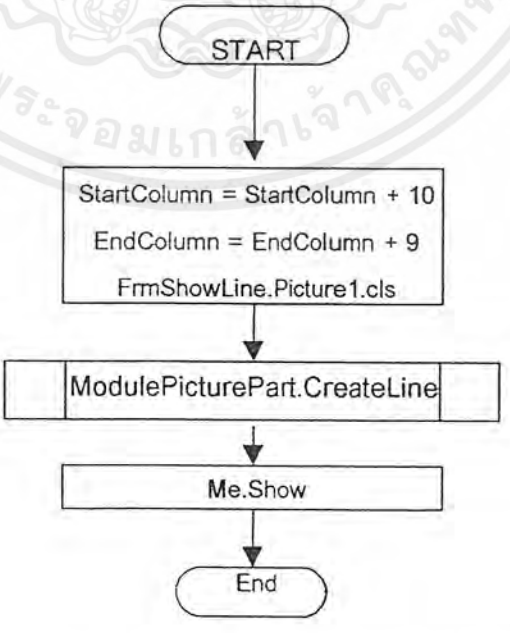
รูปที่ ข.34 กราฟแสดงการประมวลผล FormShowLine



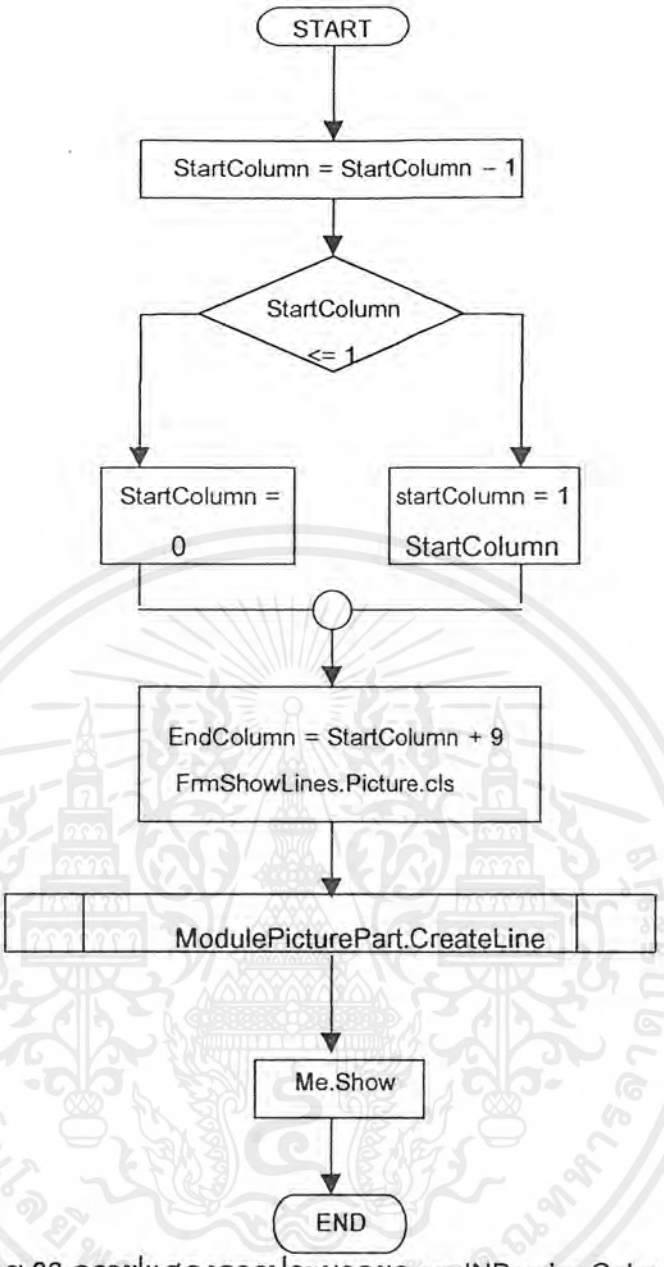
รูปที่ ข.35 กราฟแสดงการประมวลผล cmdBack to Operate_click



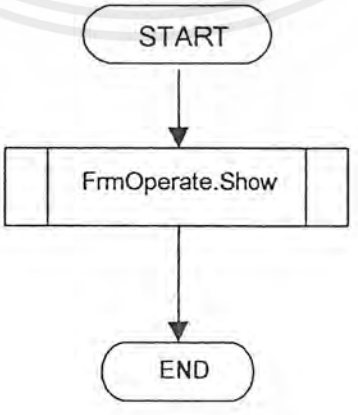
รูปที่ ข.36 กราฟแสดงการประมวลผล cmdNextColumn_click



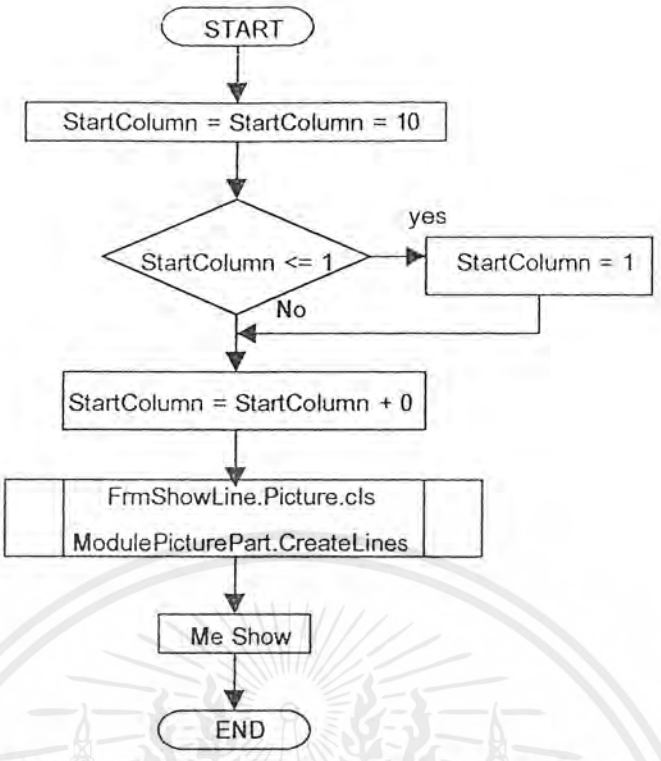
รูปที่ ข.37 กราฟแสดงการประมวลผล cmdNextPage_click



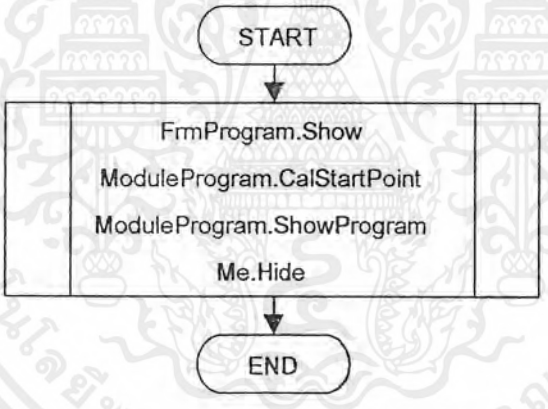
รูปที่ ข.38 กราฟแสดงการประมวลผล cmdNPreviewColumn_click



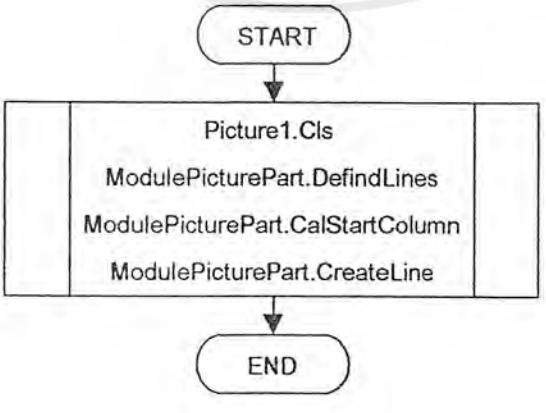
รูปที่ ข.39 กราฟแสดงการประมวลผล Form_UnLoad



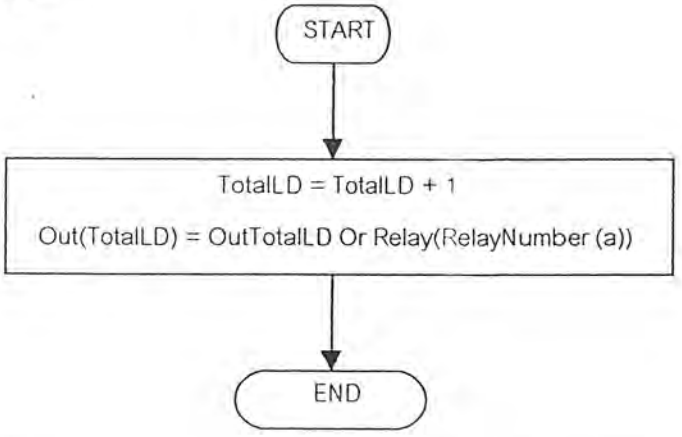
รูปที่ ข.40 กราฟแสดงการประมวลผล PreviewPage_click



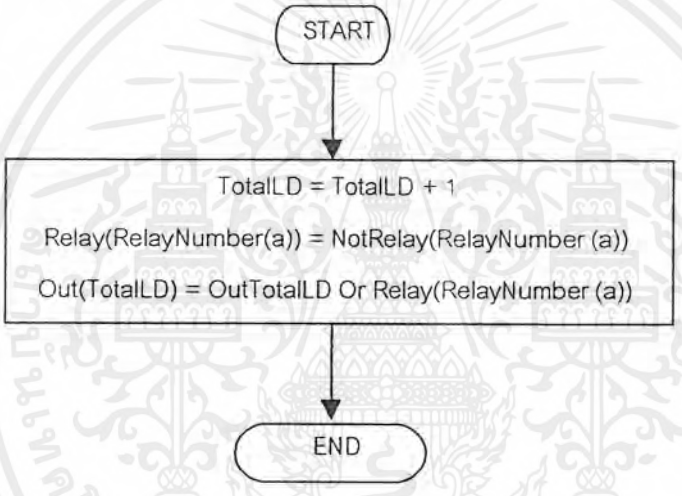
รูปที่ ข.41 กราฟแสดงการประมวลผล cmdProgram_click



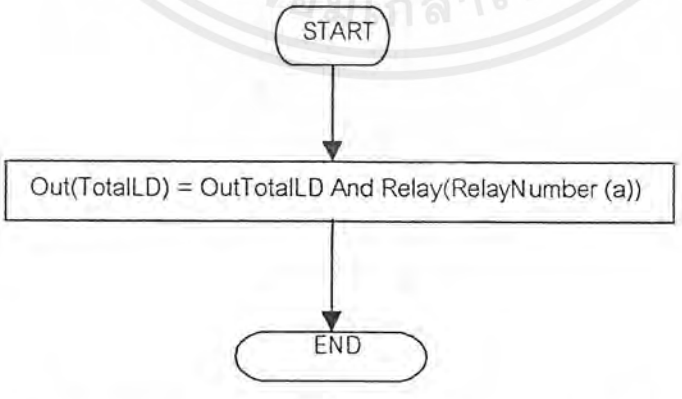
รูปที่ ข.42 กราฟแสดงการประมวลผล cmdShowLine_click



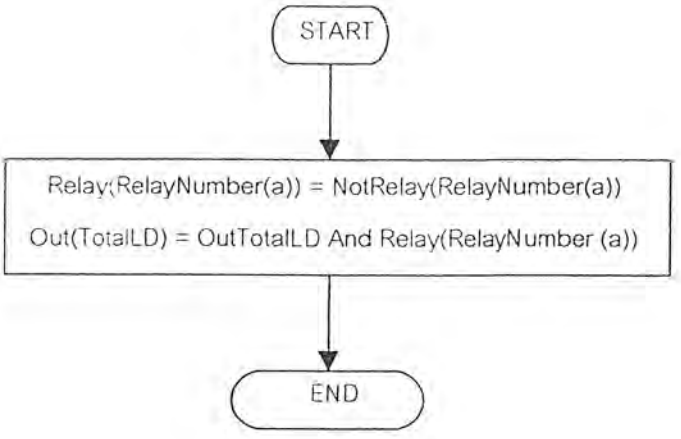
รูปที่ ข.43 กราฟแสดงการประมวลผล Sub To SimLD



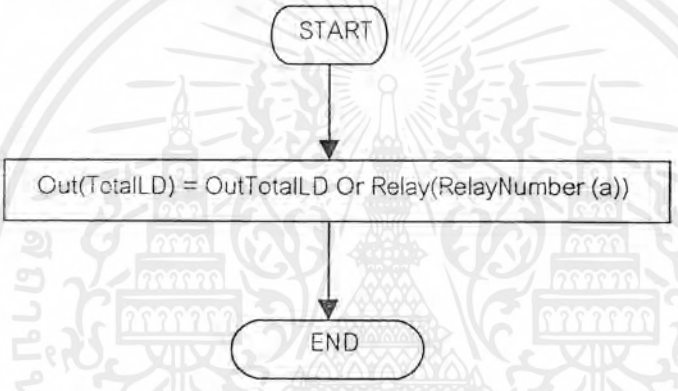
รูปที่ ข.44 กราฟแสดงการประมวลผล Sub To SimLDNot



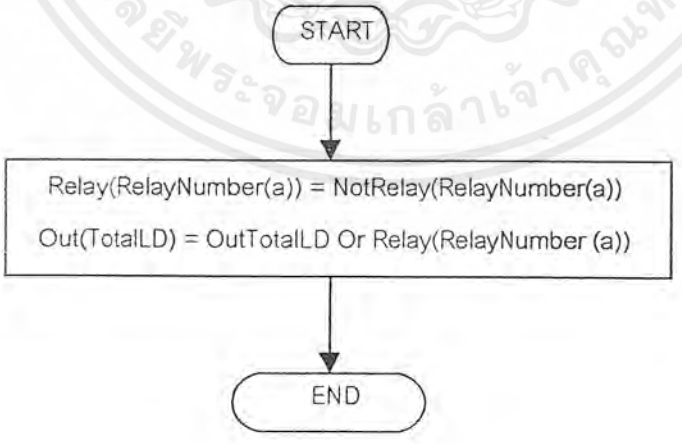
รูปที่ ข.45 กราฟแสดงการประมวลผล Sub To SimAnd



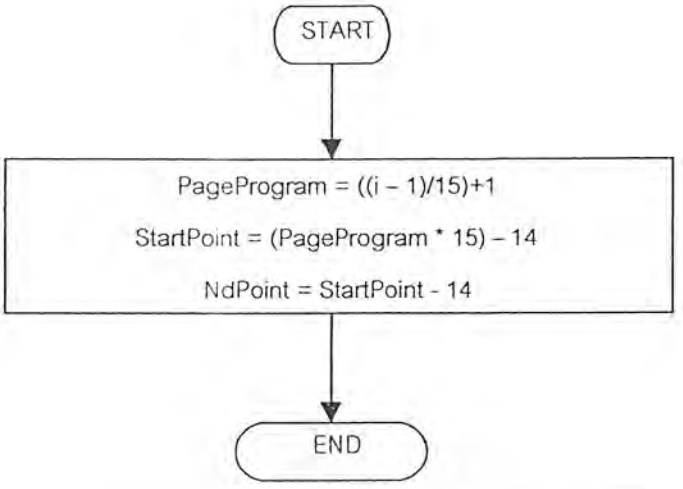
รูปที่ ข.46 กราฟแสดงการประมวลผล Sub To SimAndNot



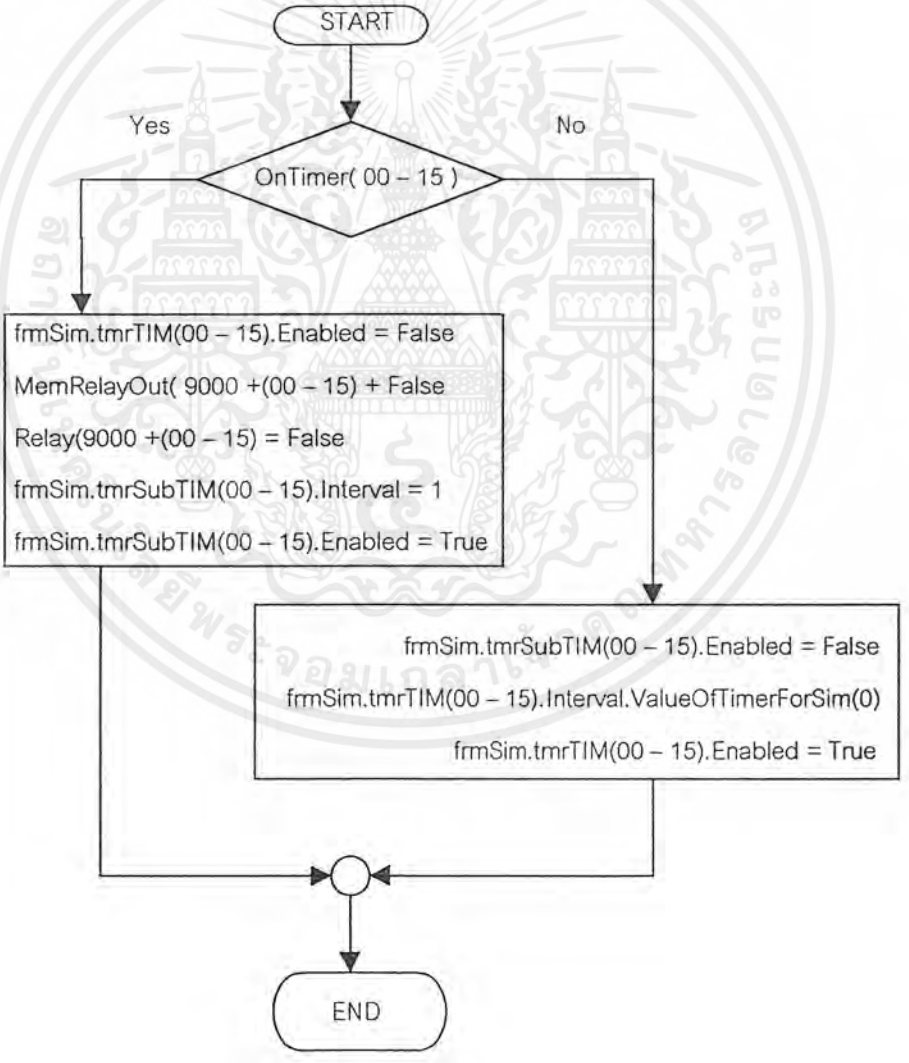
รูปที่ ข.47 กราฟแสดงการประมวลผล Sub To SimOr



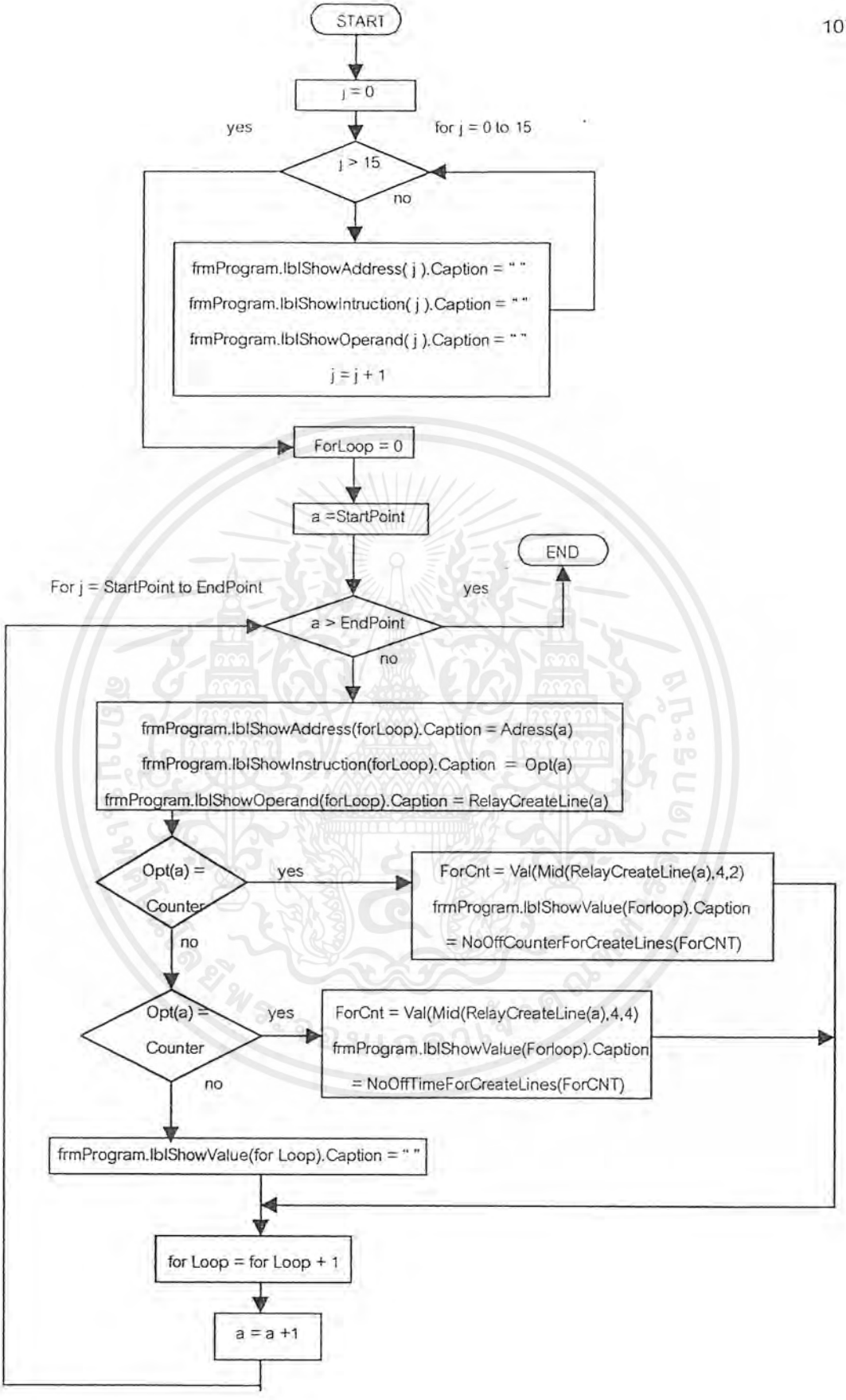
รูปที่ ข.48 กราฟแสดงการประมวลผล Sub To SimOrNot



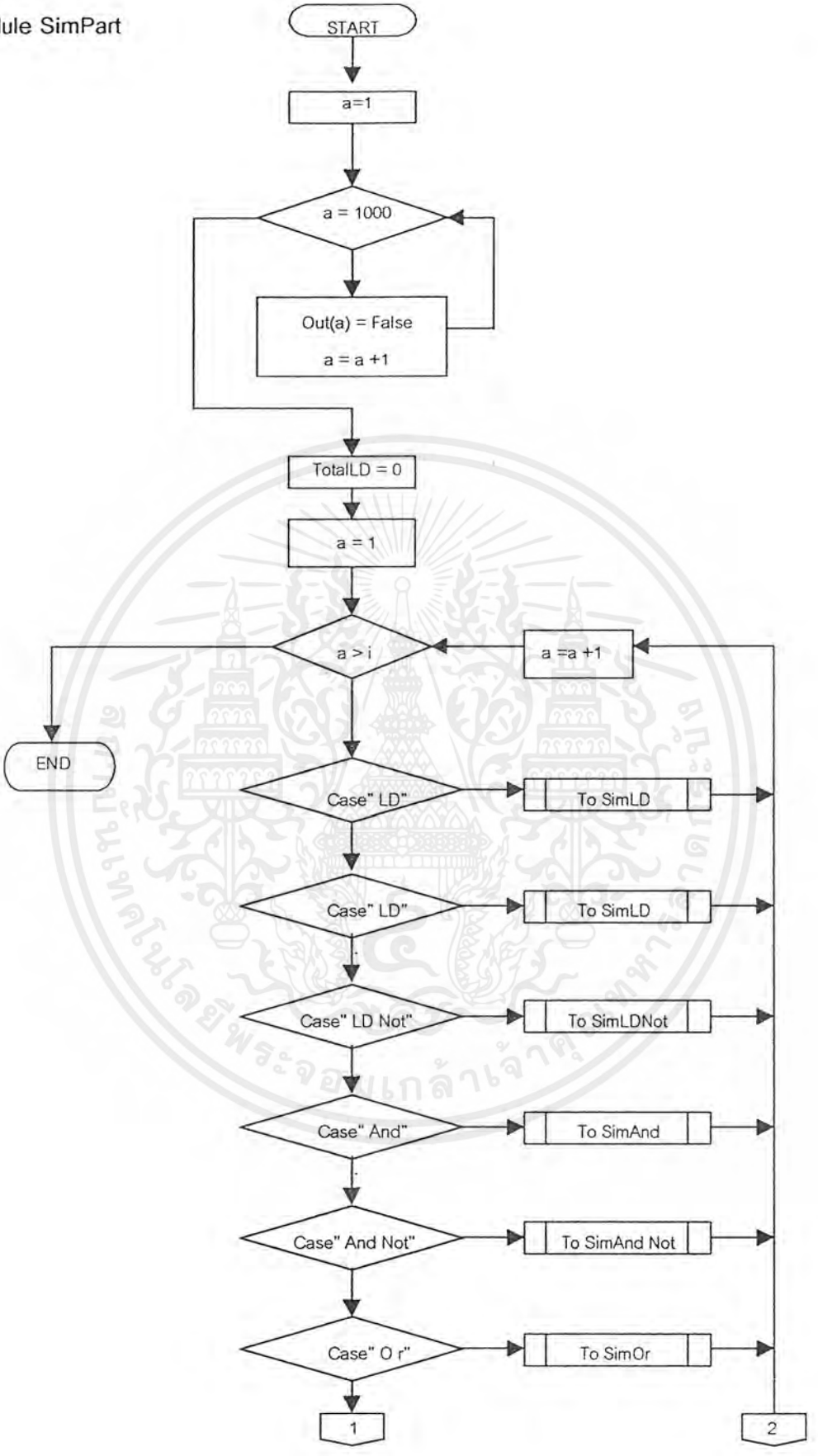
รูปที่ ข.49 กราฟแสดงการประมวลผล Sub call StartPoint



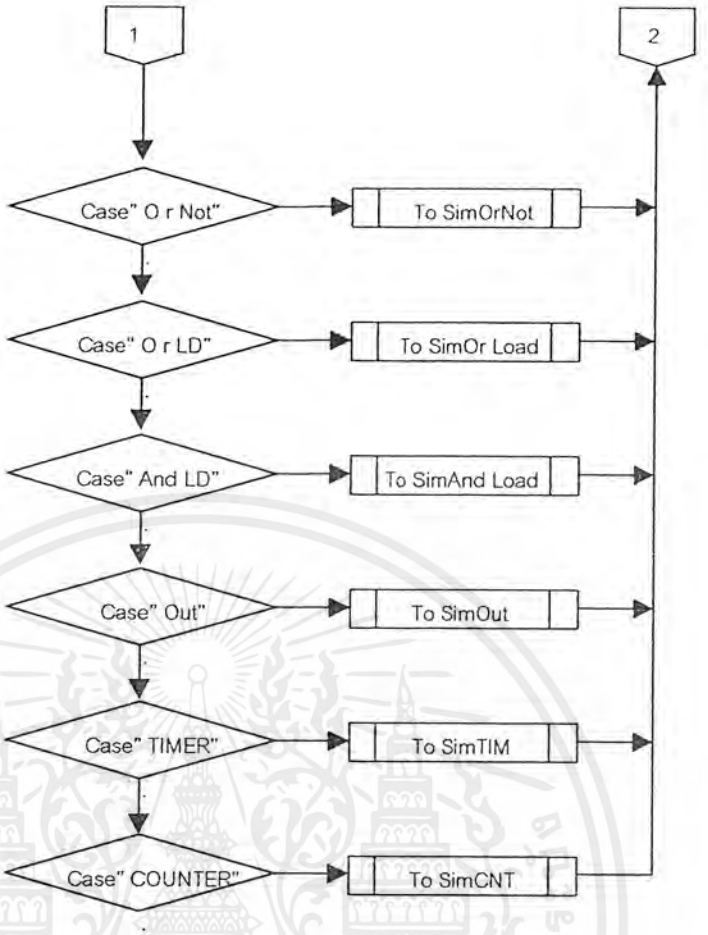
รูปที่ ข.50 กราฟแสดงการประมวลผล Sub to call Timer



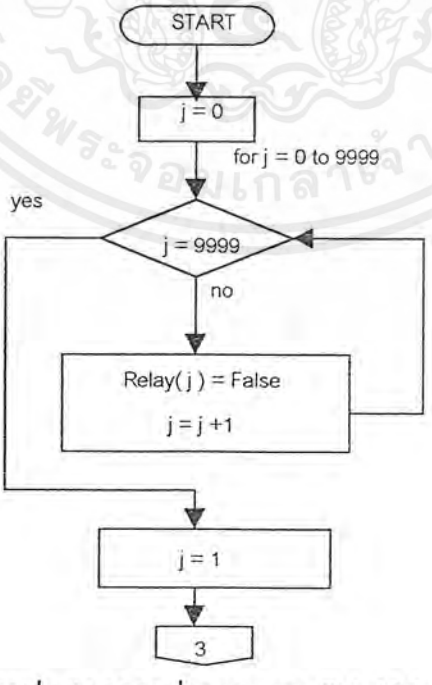
รูปที่ ข.51 กราฟแสดงการประมวลผล Sub Show Program



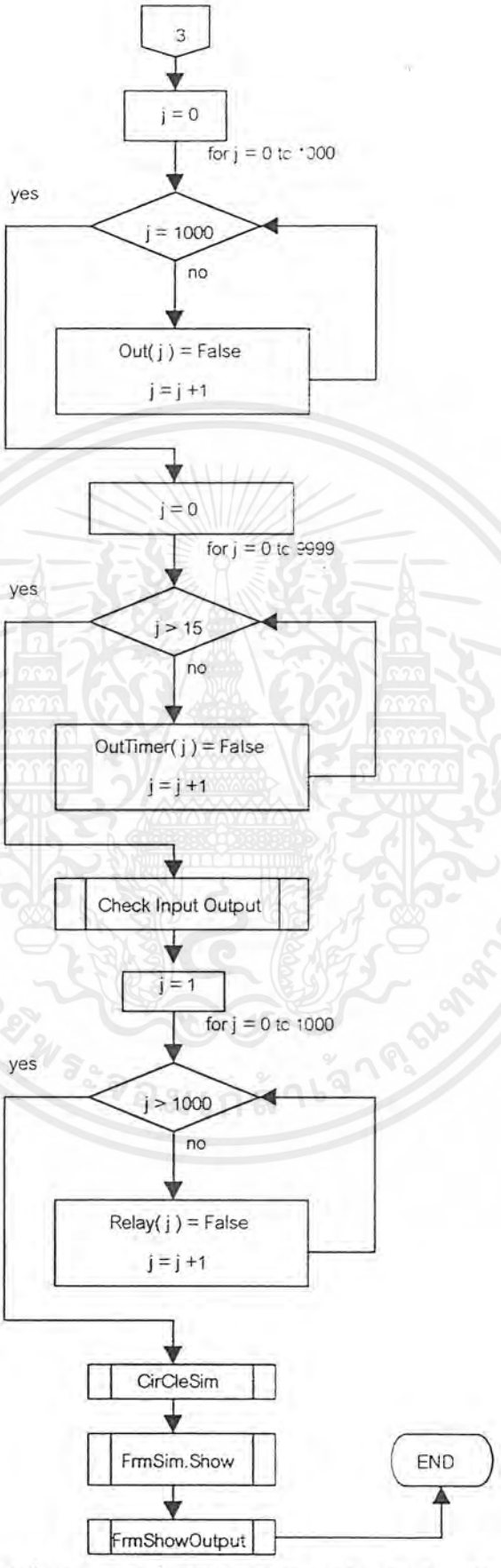
รูปที่ ข.52 กราฟแสดงการประมวลผล Circle Sim(1)



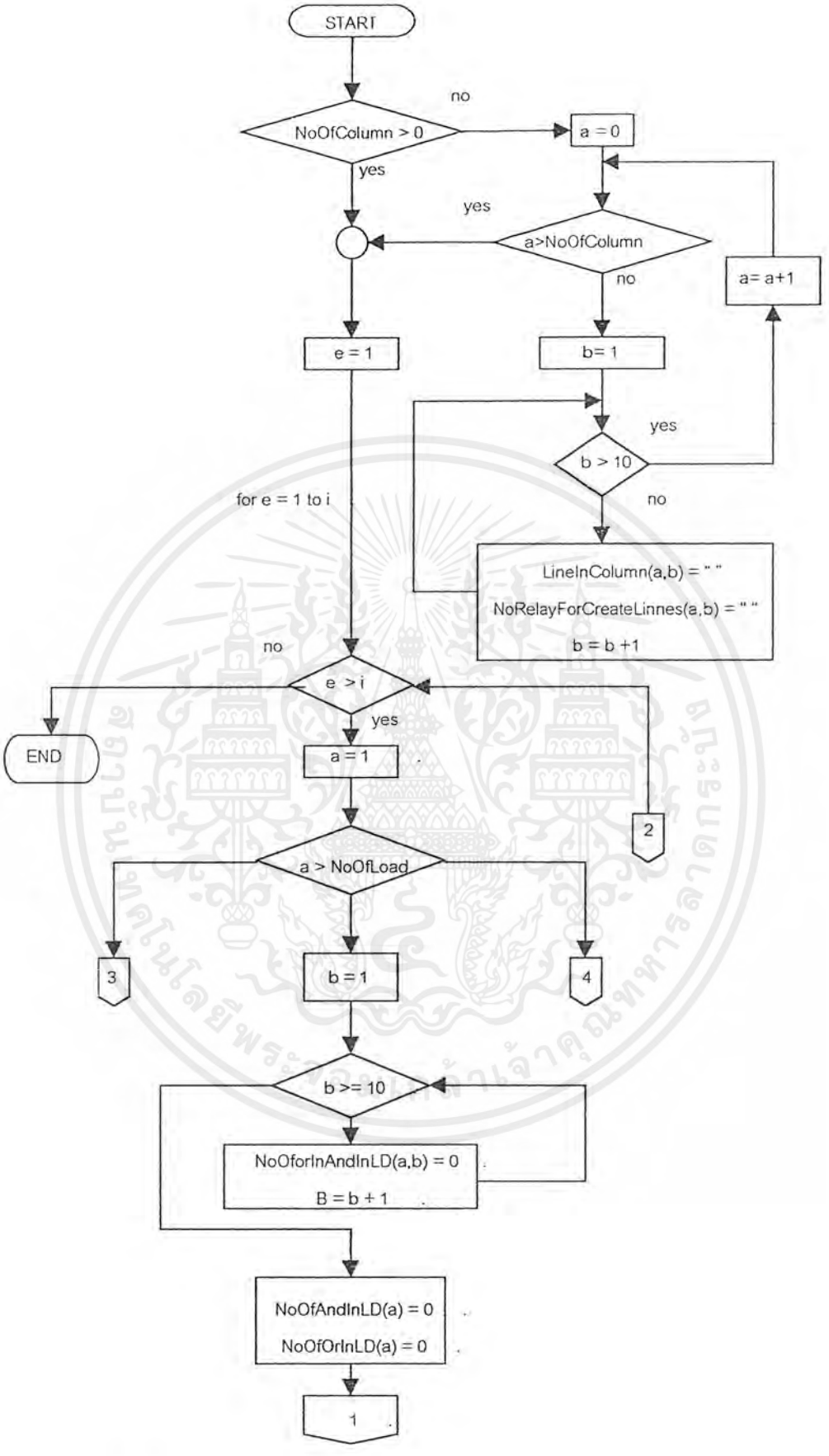
รูปที่ ข.53 กราฟแสดงการประมวลผล Circle Sim(2)



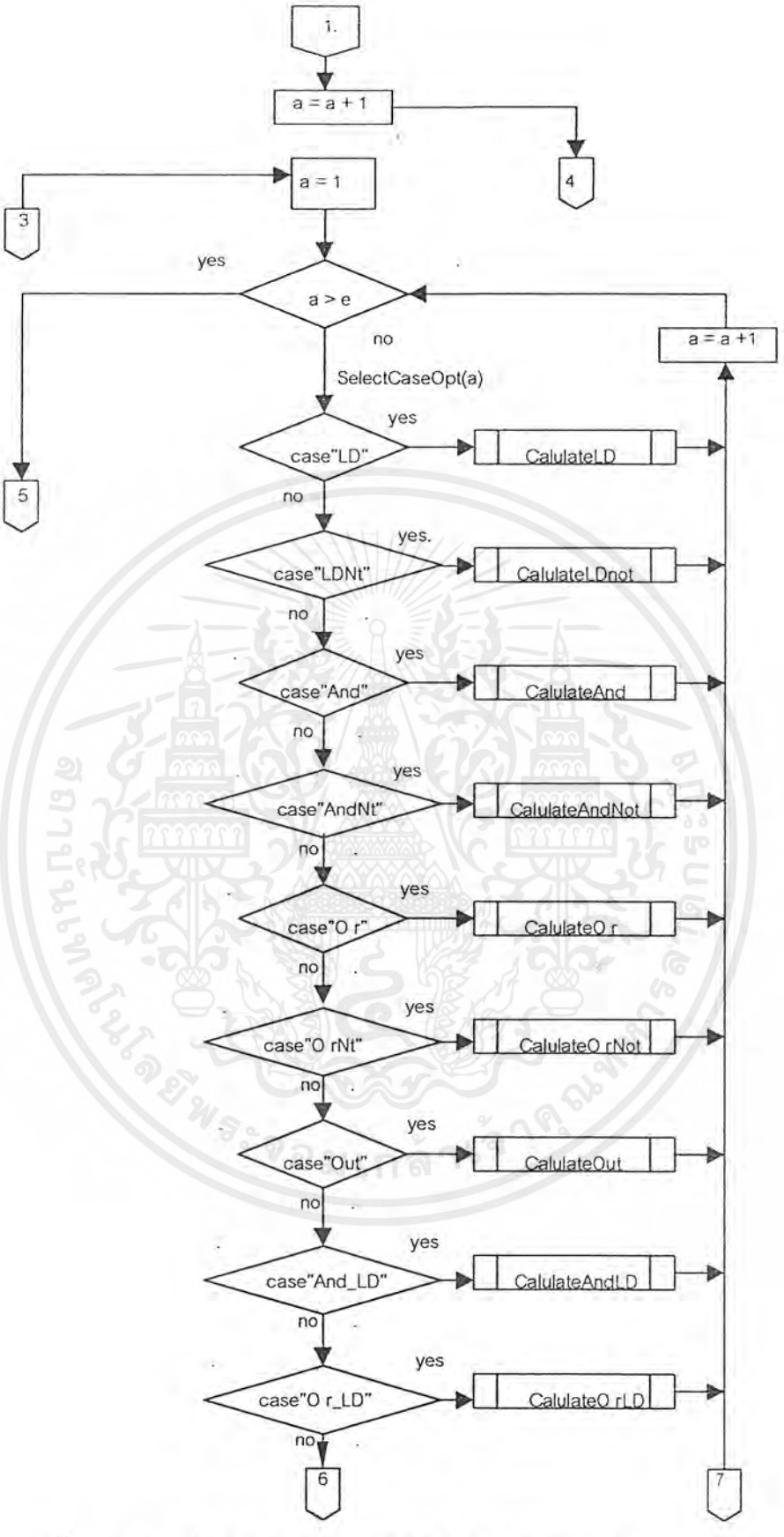
รูปที่ ข.54 กราฟแสดงการประมวลผล Sub MainForSim(1)



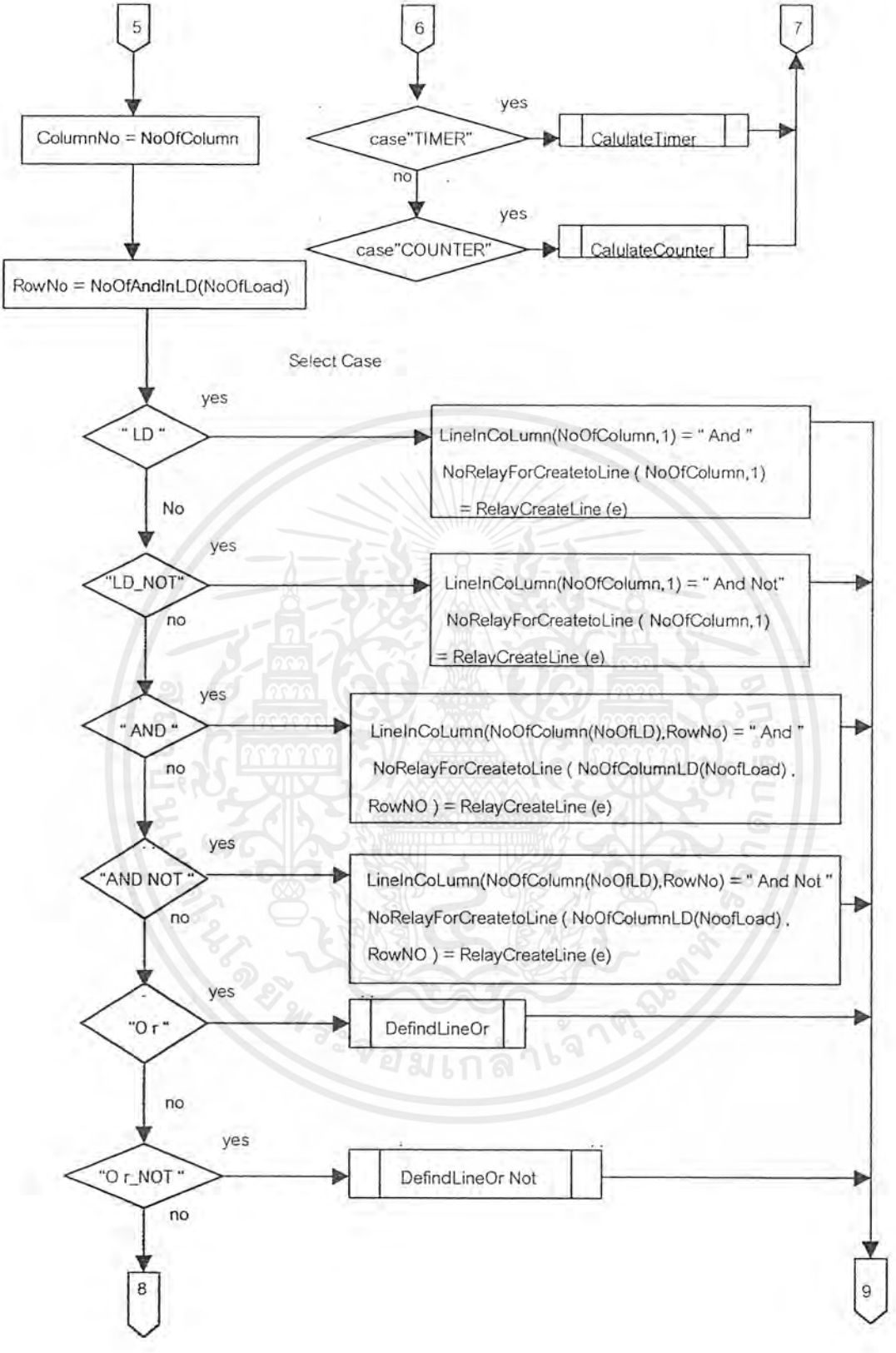
รูปที่ ข.55 กราฟแสดงการประมวลผล Sub MainForSim(2)



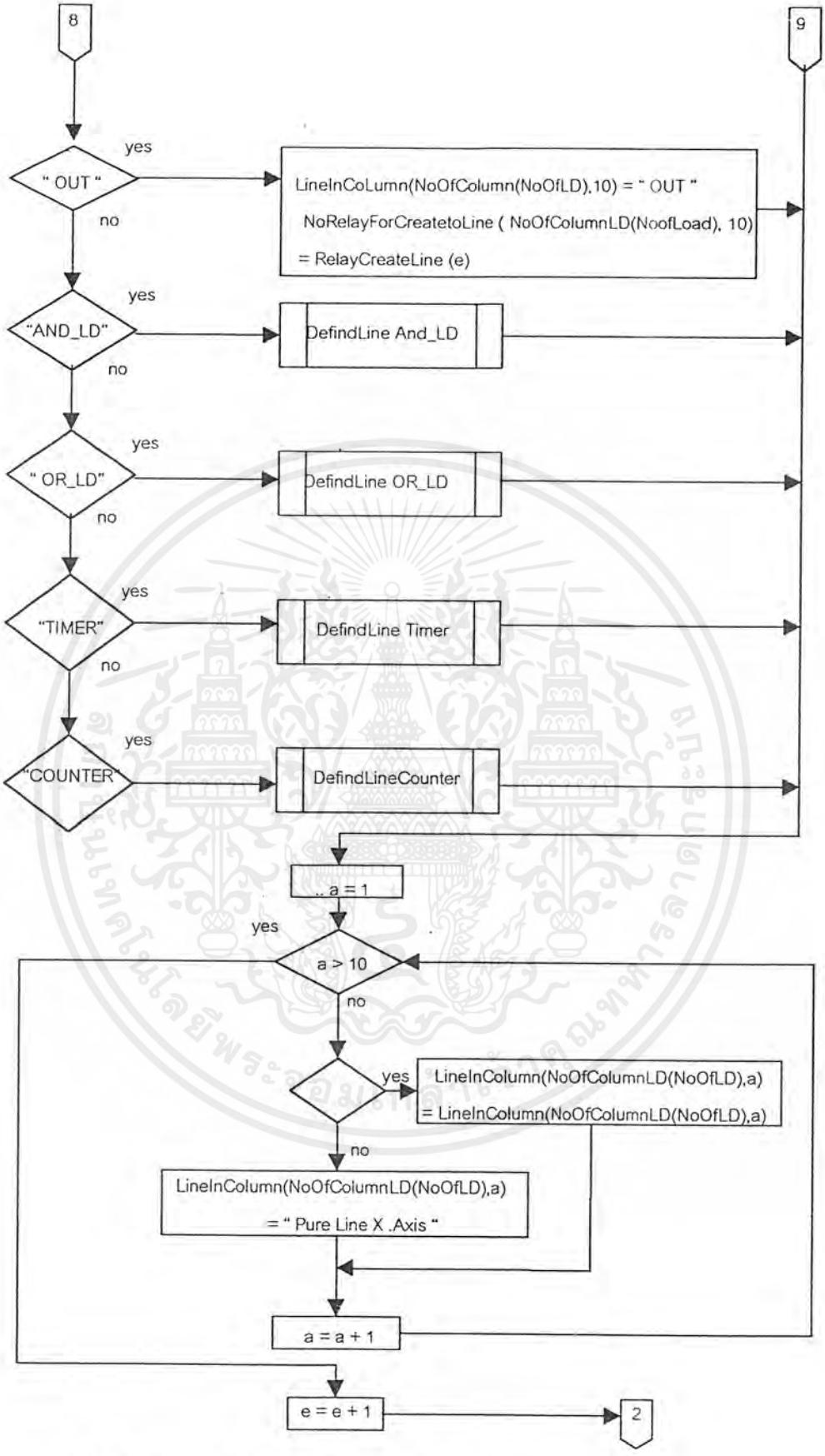
รูปที่ ข.56 กราฟแสดงการประมวลผล SubDefindLine



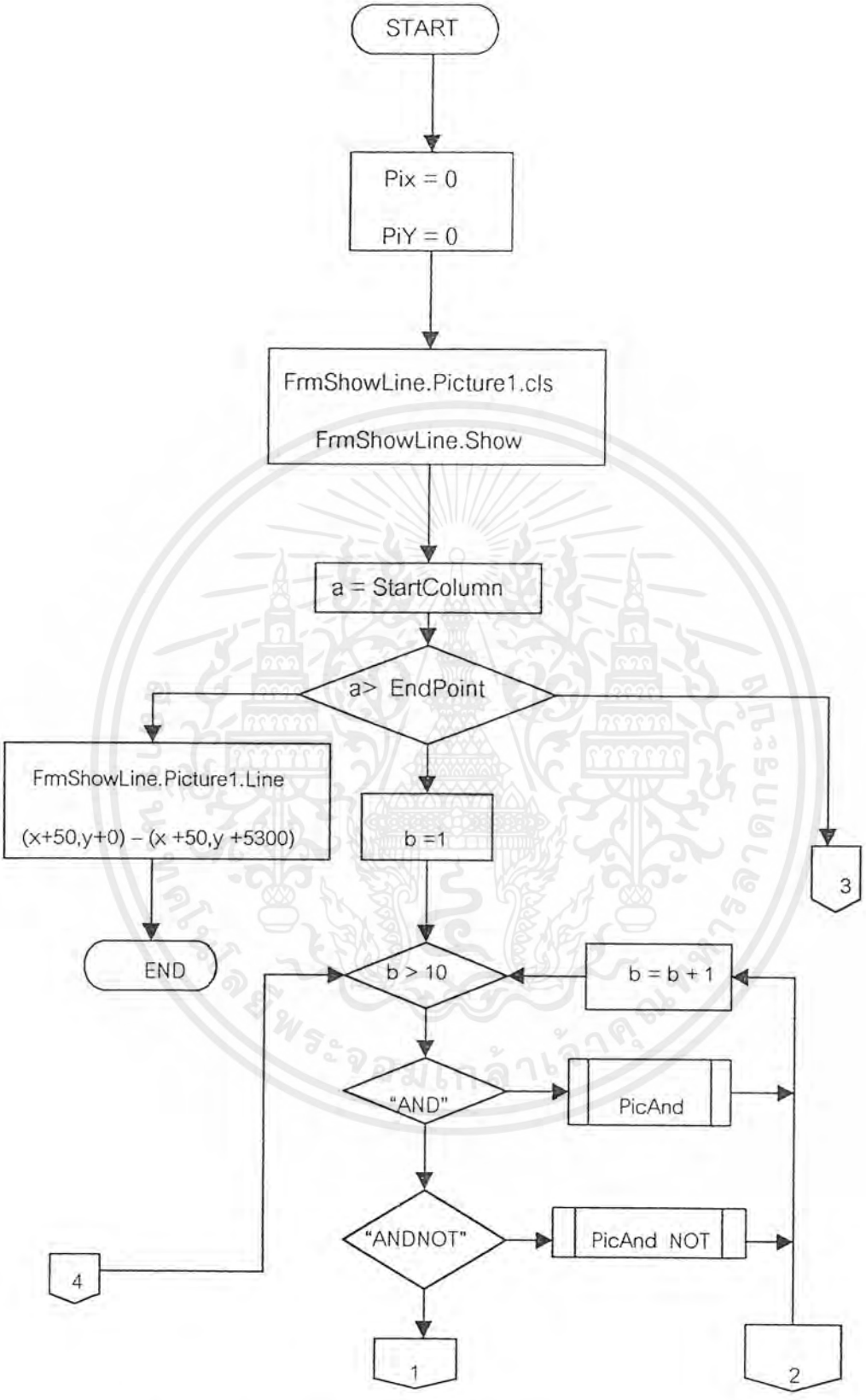
รูปที่ ข.57 กราฟแสดงการประมวลผล SubDefindLine(2)



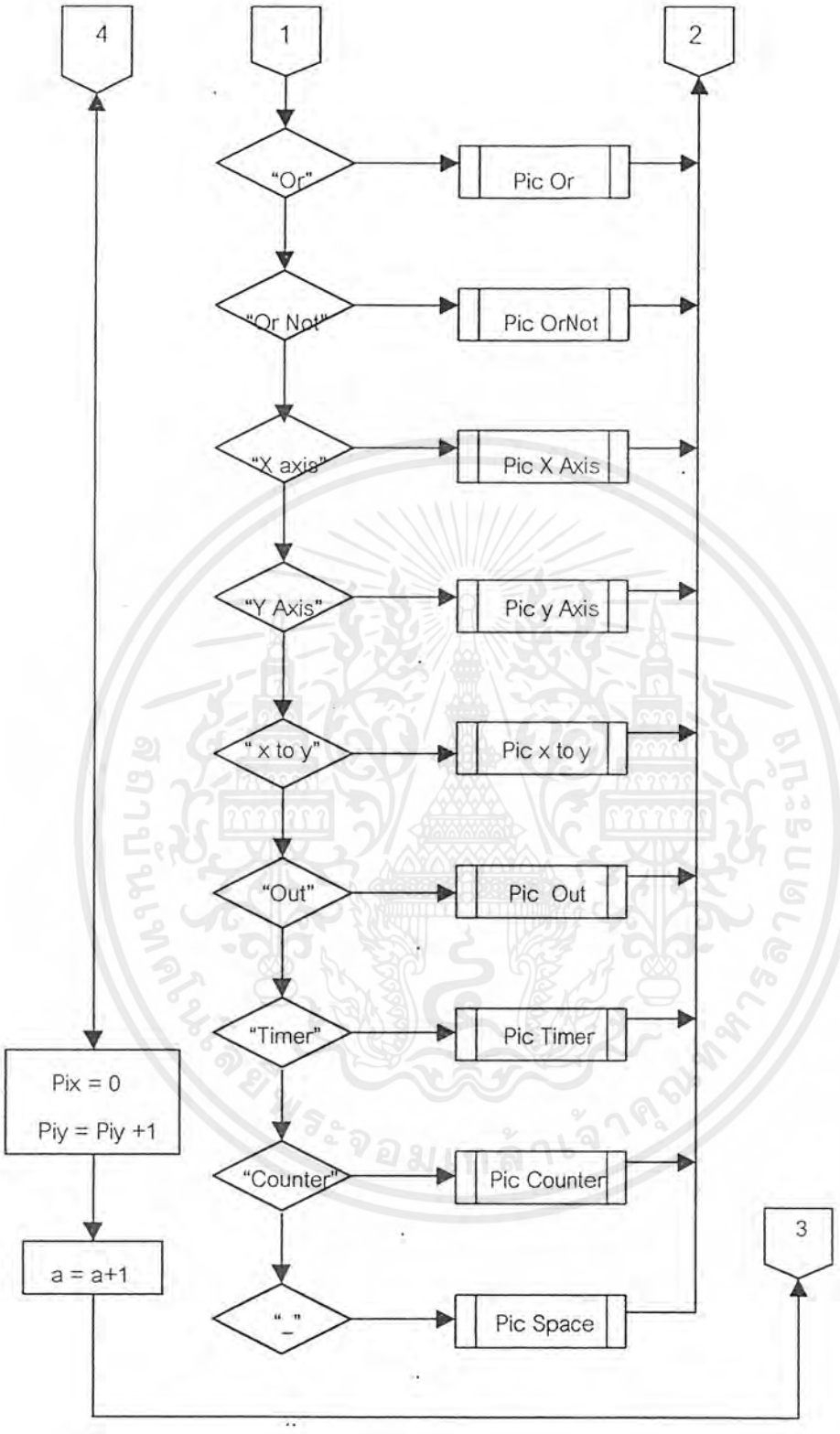
รูปที่ ข.58 กราฟแสดงการประมวลผล SubDefindLine(3)



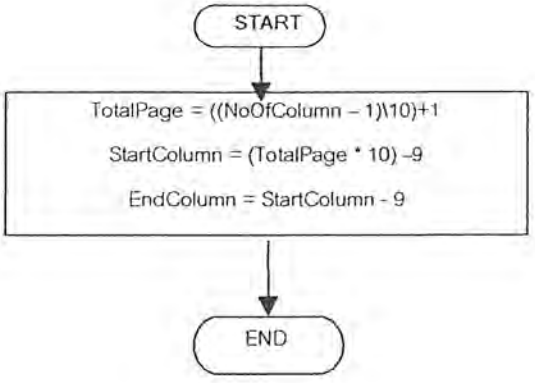
รูปที่ ข.59 กราฟแสดงการประมวลผล SubDefindLine(4)



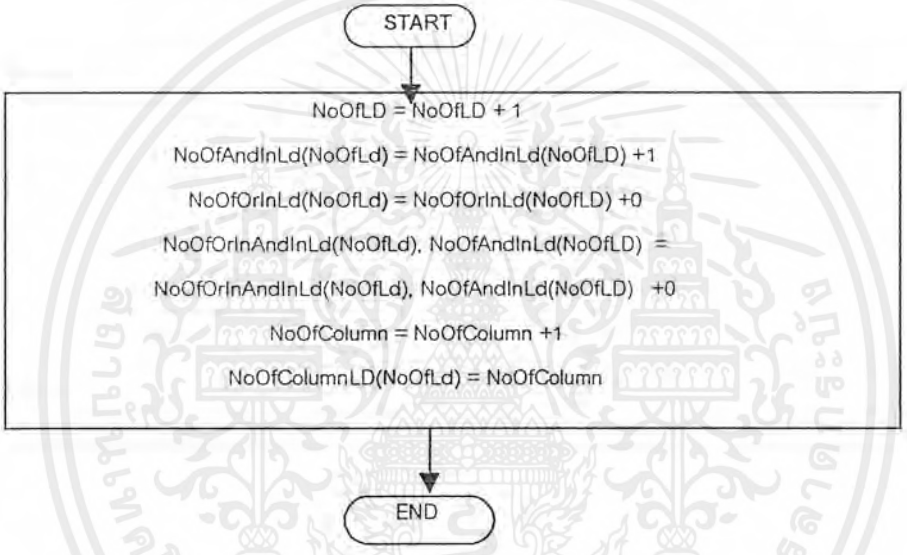
รูปที่ ข.60 กราฟแสดงการประมวลผล Create Line(1)



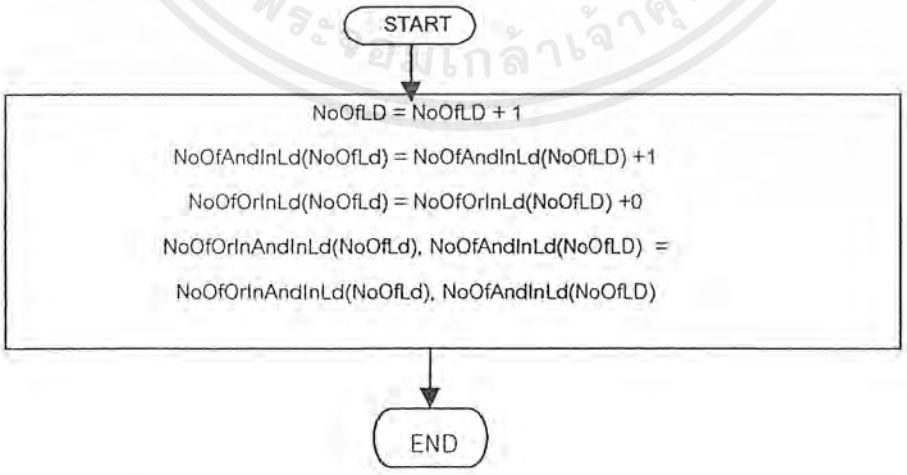
รูปที่ ข.61 กราฟแสดงการประมวลผล Create Line(2)



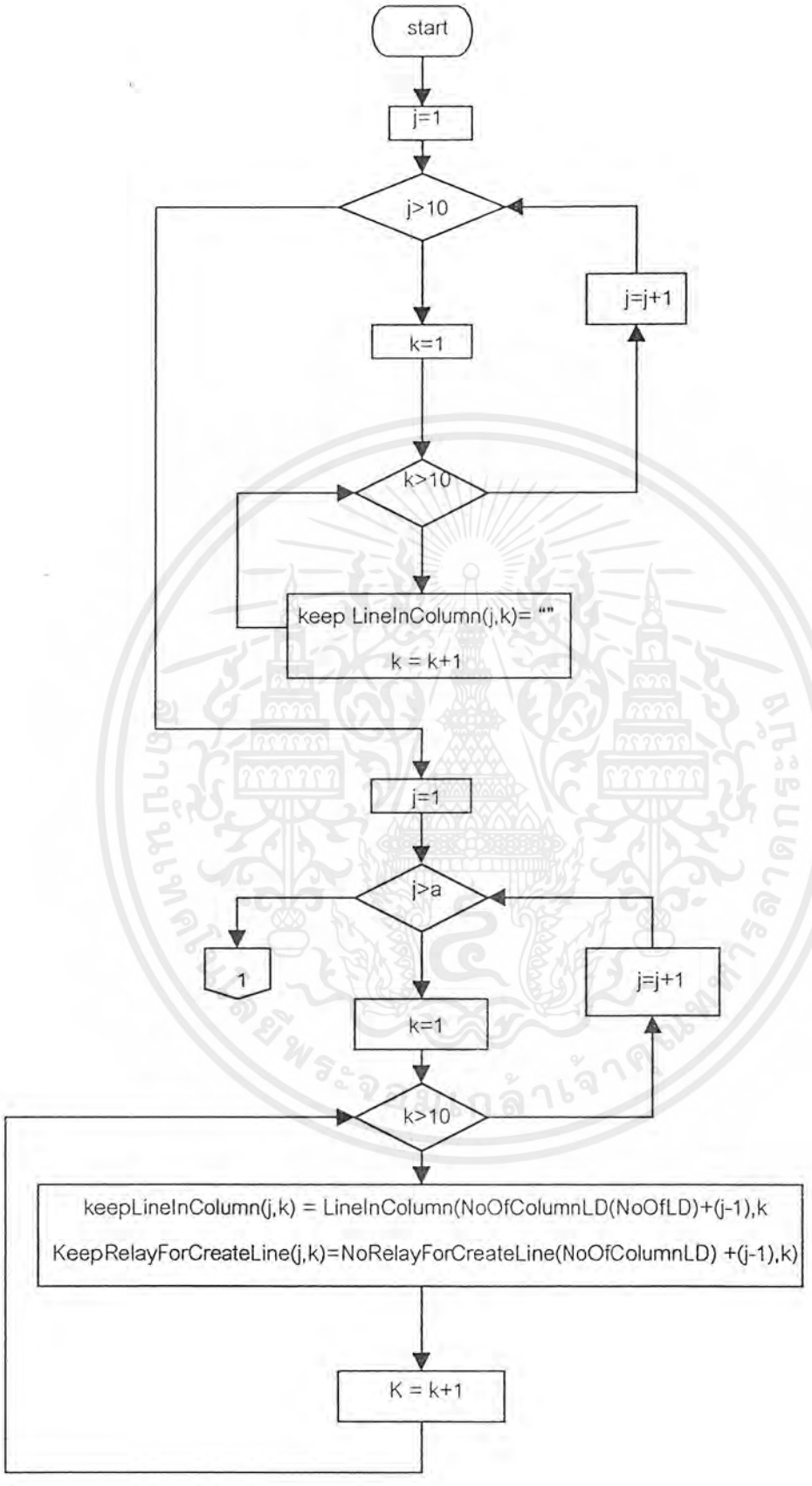
รูปที่ ข.62 กราฟแสดงการประมวลผล Cal Start Column



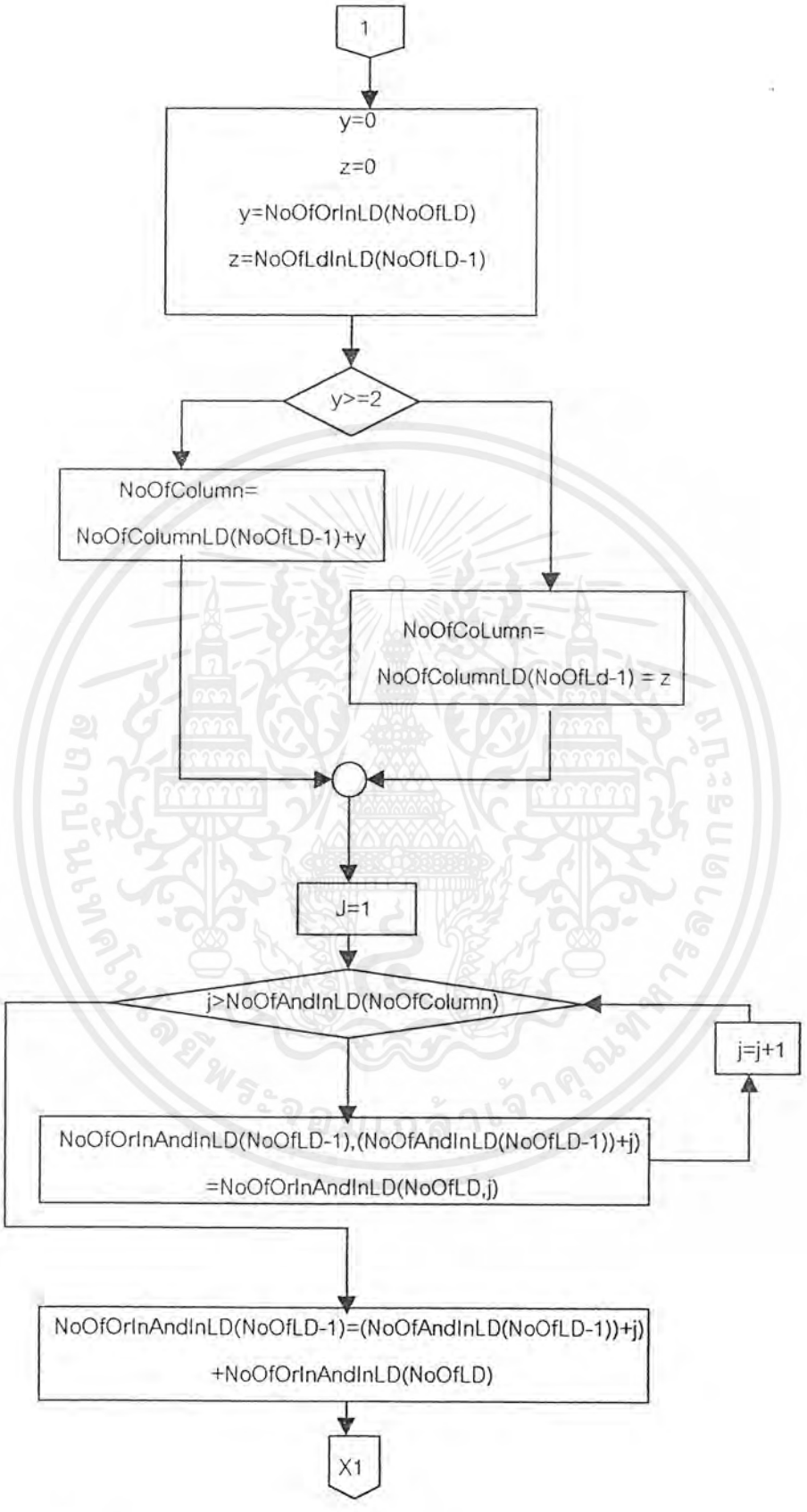
รูปที่ ข.63 กราฟแสดงการประมวลผล Calculate LD



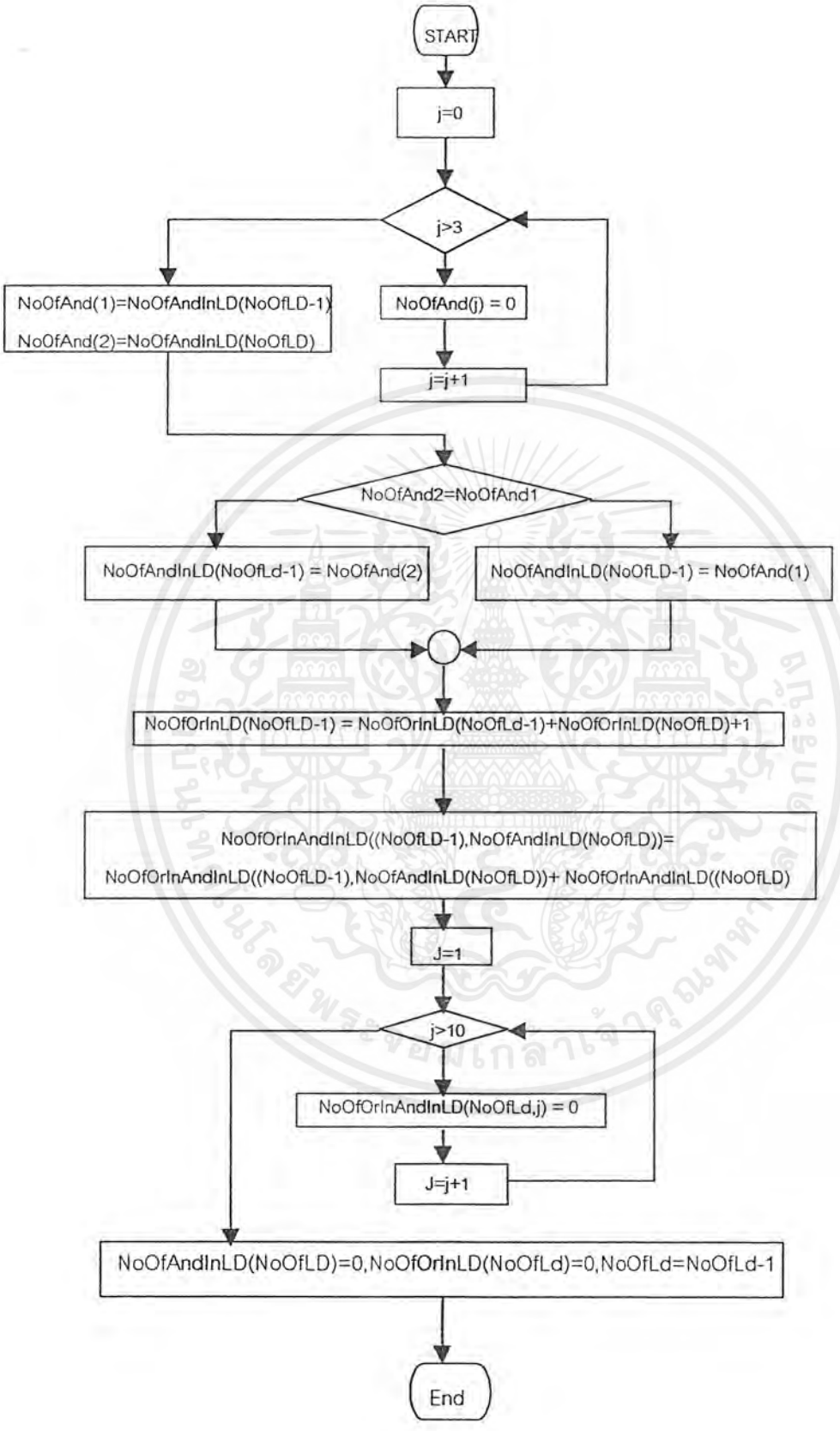
รูปที่ ข.64 กราฟแสดงการประมวลผล Calculate AND



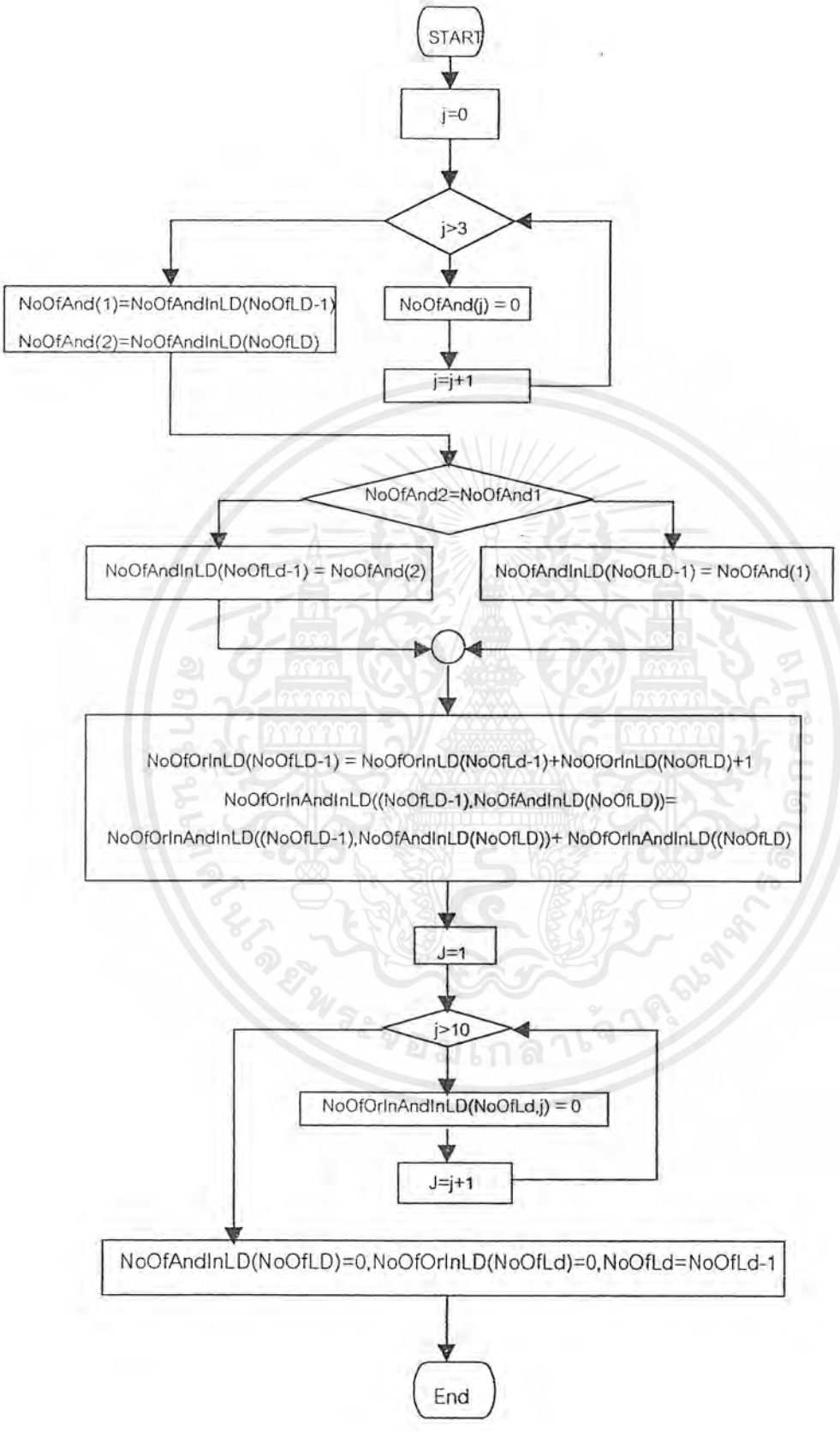
รูปที่ ข.65 กราฟแสดงการประมวลผล Calculate AND LD(1)



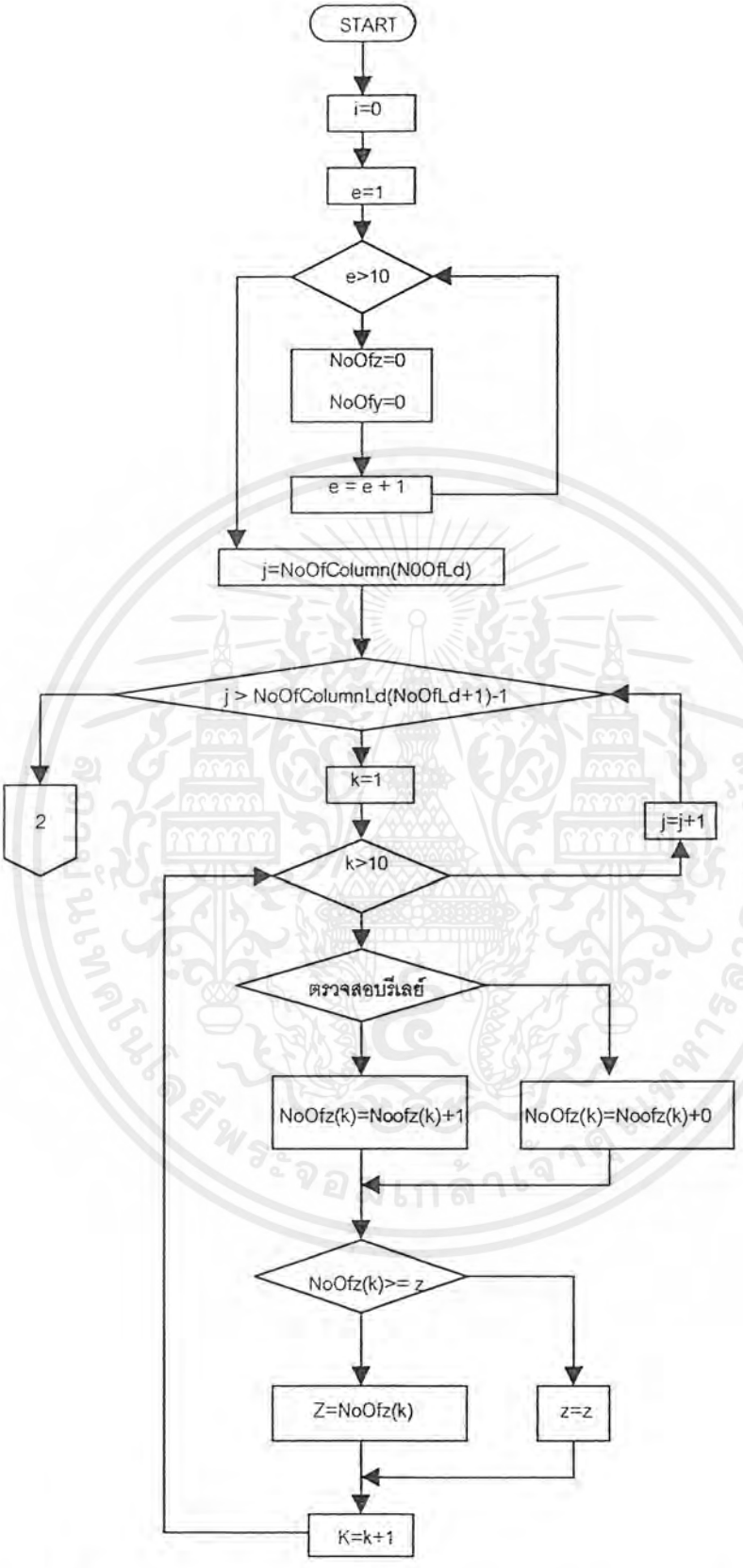
รูปที่ ข.66 กราฟแสดงการประมวลผล Calculate AND LD(2)



รูปที่ ข.67 กราฟแสดงการประมวลผล Calculate Or LD

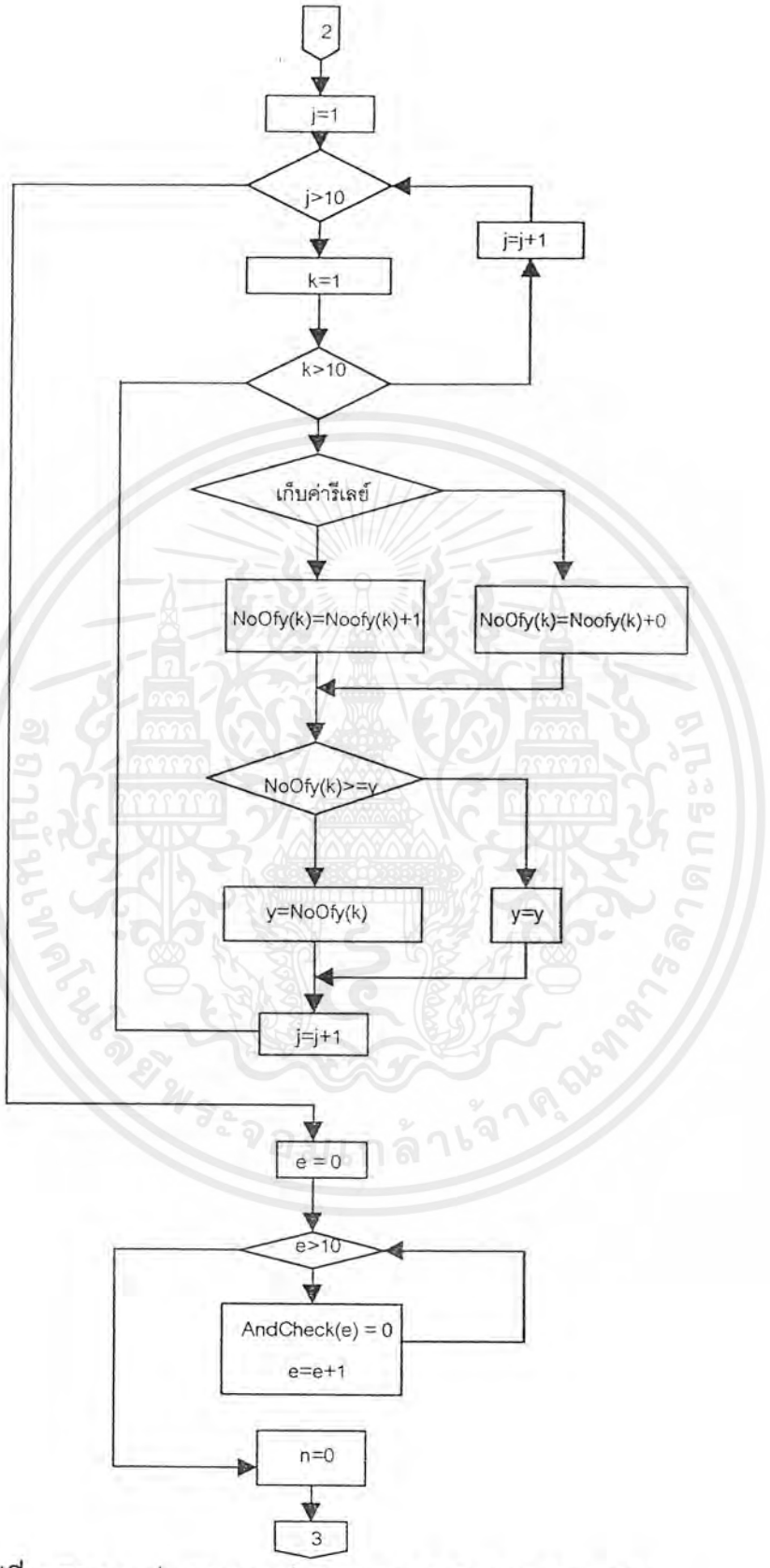


รูปที่ ข.68 กราฟแสดงการประมวลผล Calculate Counter



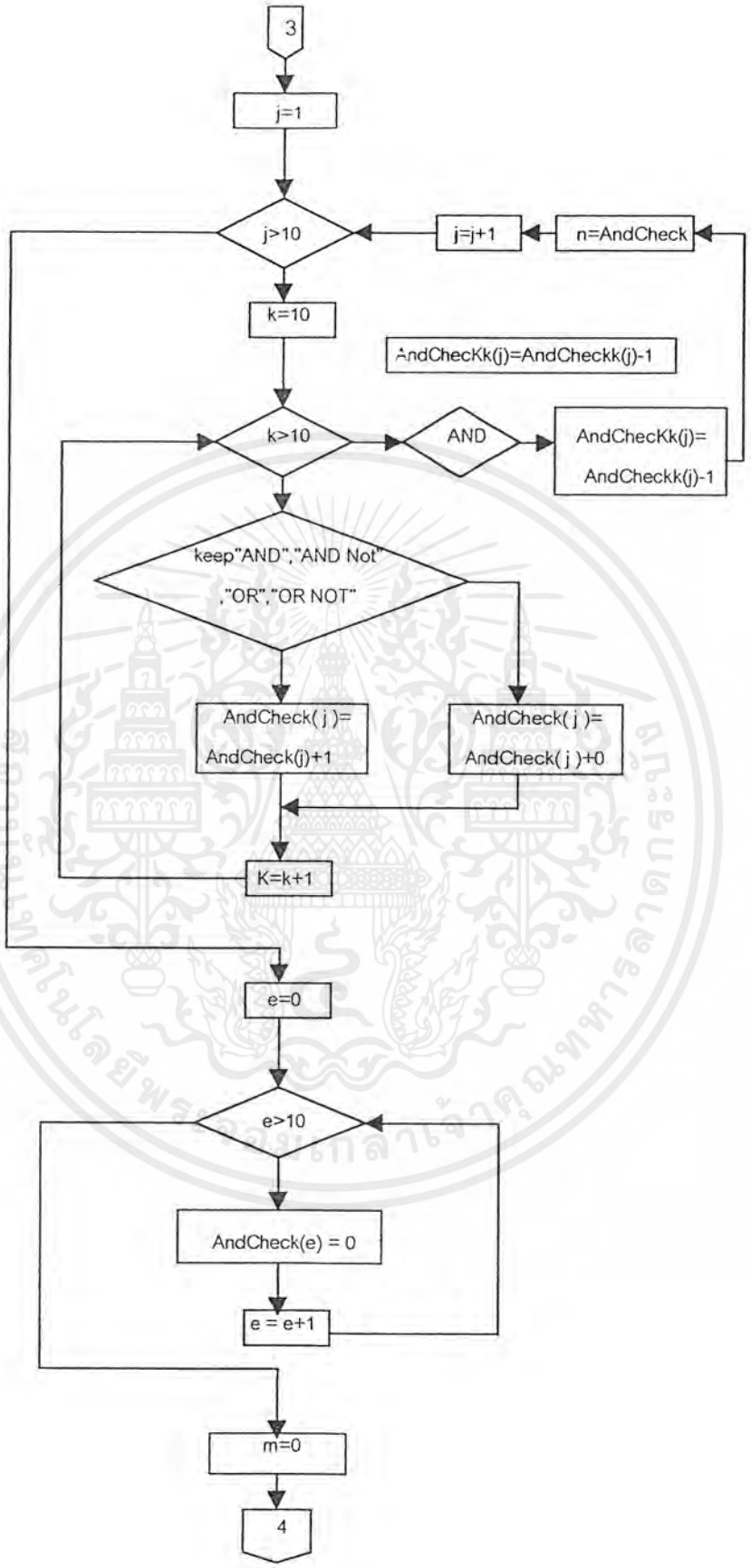
รูปที่ ข.69 กราฟแสดงการประมวลผล DefindLineAndLD(1)

ตรวจสอบรีเลย์ = Relay("AND","AND NOT","OR","OR NOT","Line X","LineY","Line X to Y"

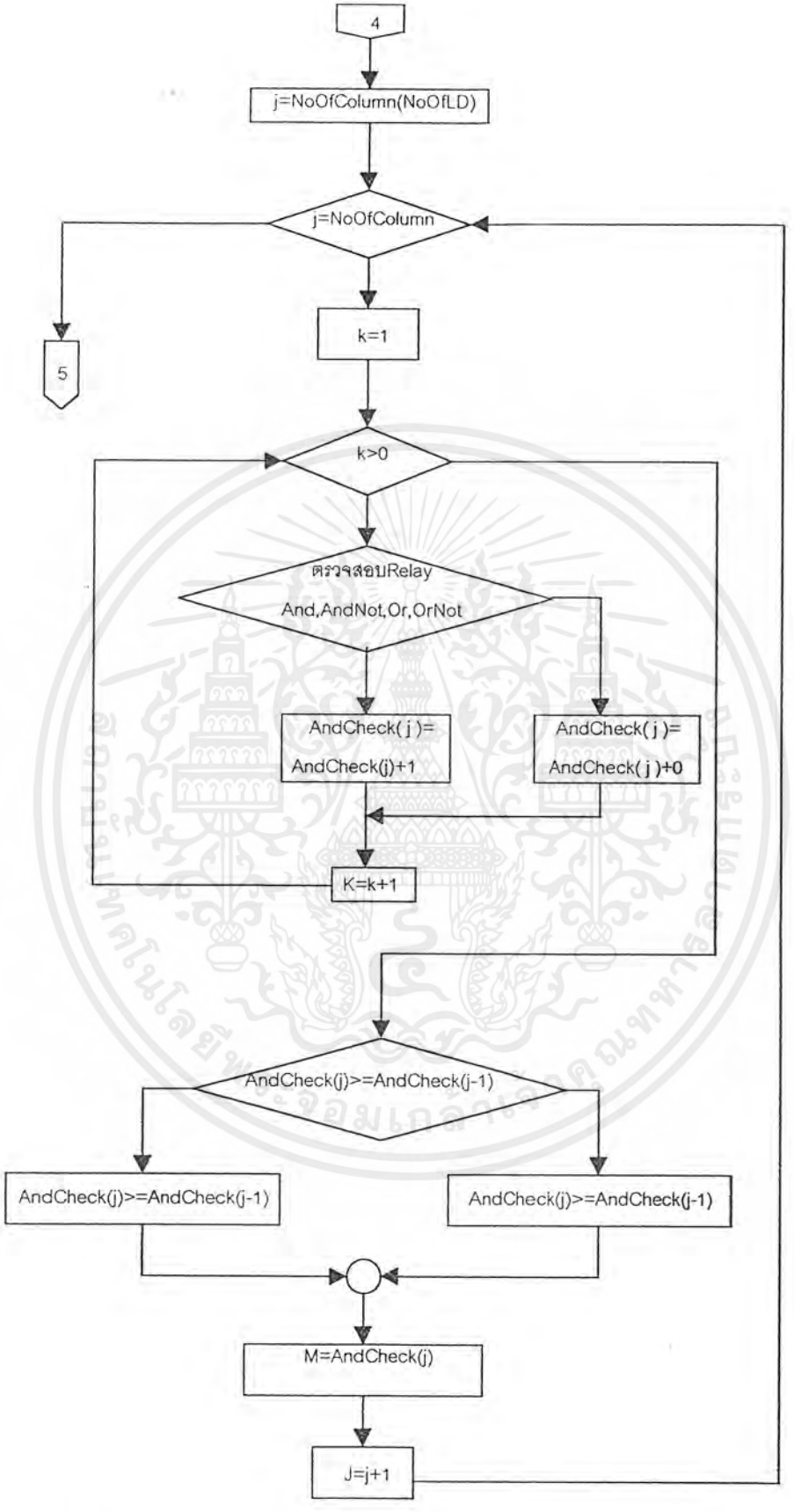


รูปที่ ข.70 กราฟแสดงการประมวลผล DefindLineAndLD(2)

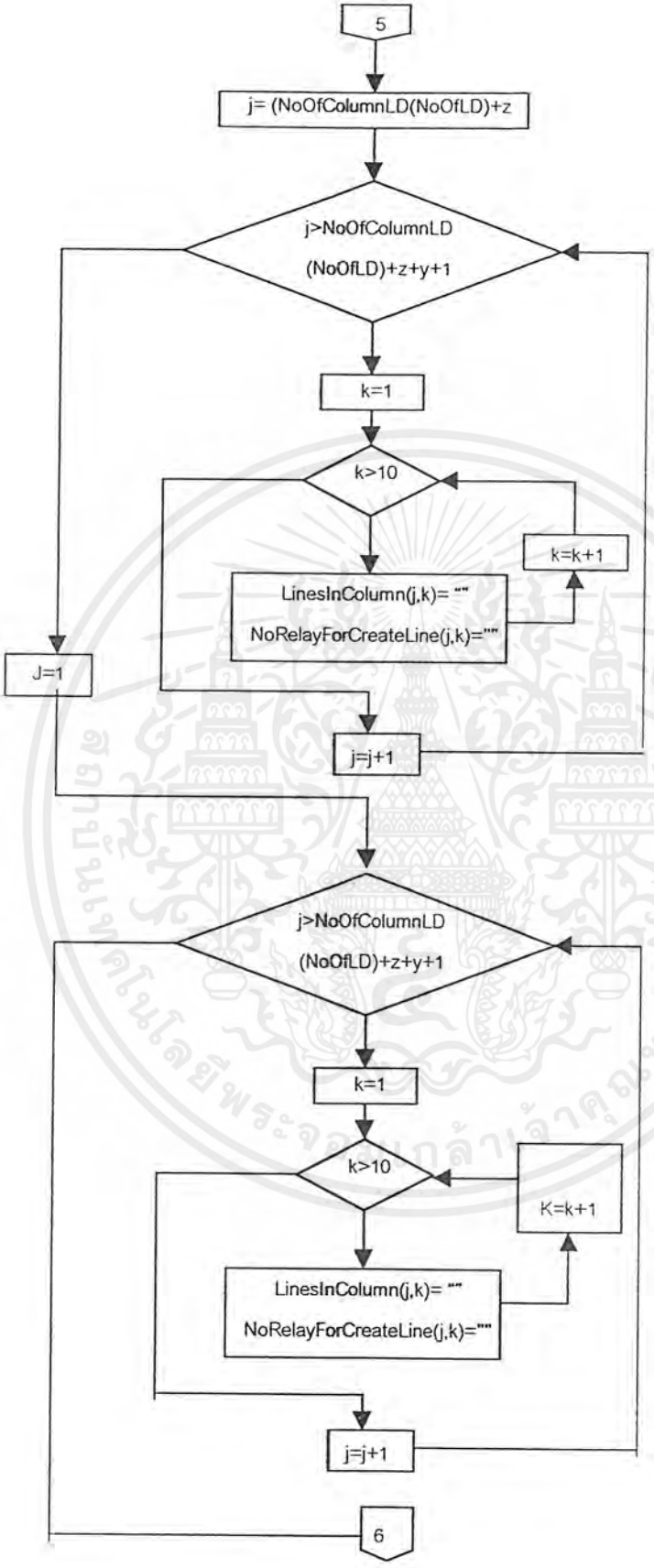
เก็บค่ารีเลย์ = Relay("AND","AND NOT","OR","OR NOT","Line X","LineY","Line X to Y"



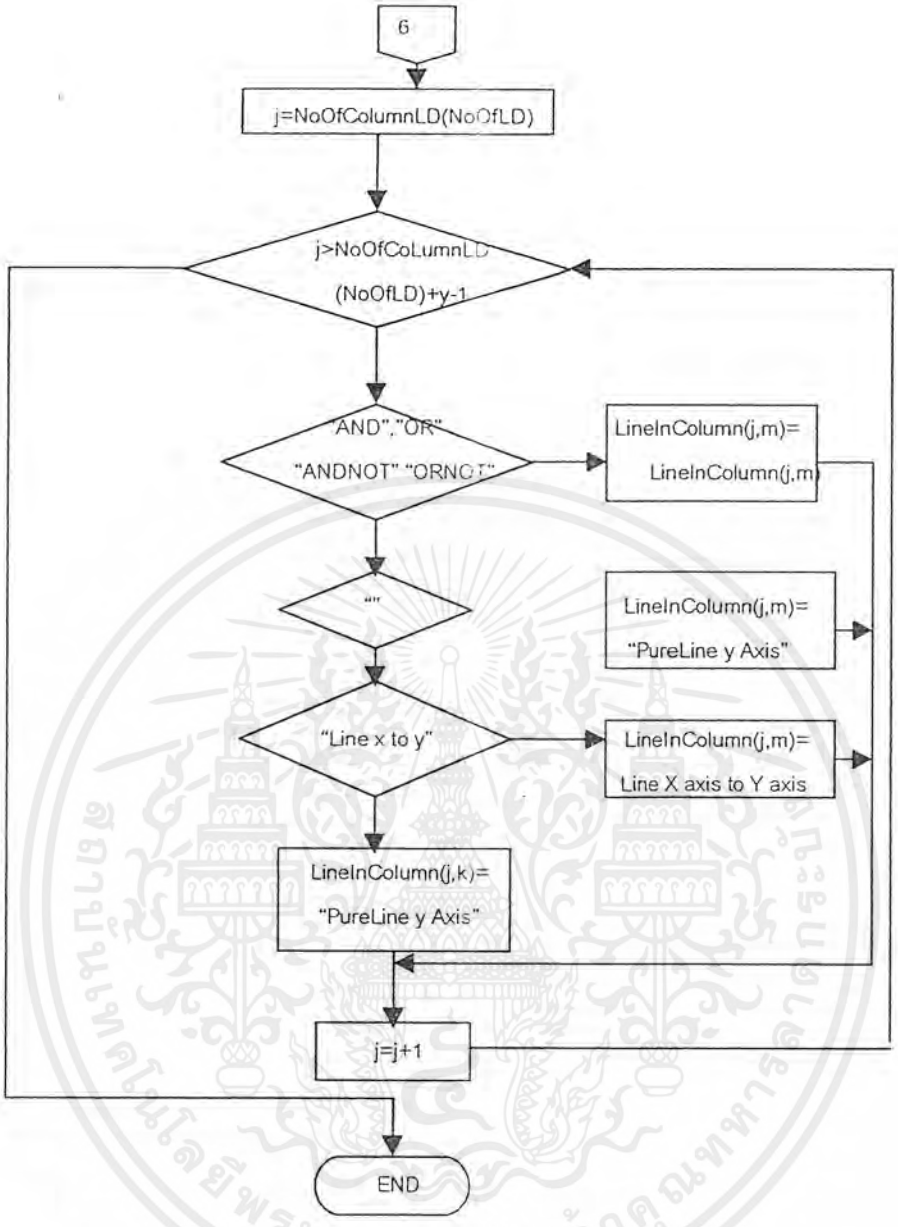
รูปที่ ข.71 กราฟแสดงการประมวลผล DefindLineAndLD(3)



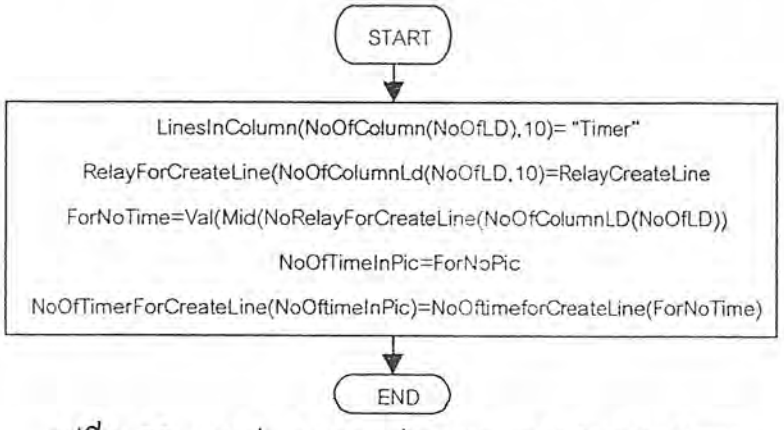
รูปที่ ข.72 กราฟแสดงการประมวลผล DefindLineAndLD(4)



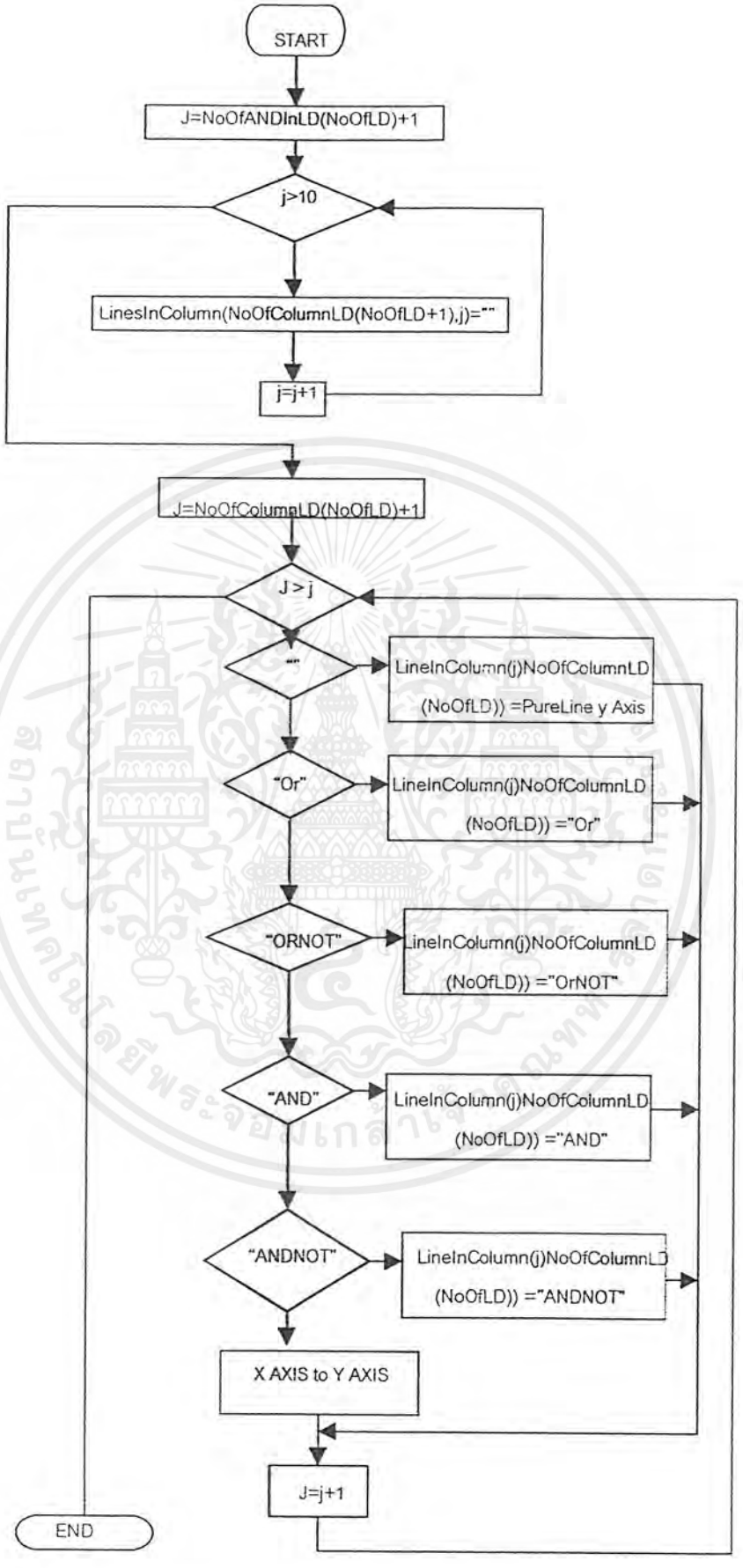
รูปที่ ข.73 กราฟแสดงการประมวลผล DefindLineAndLD(5)



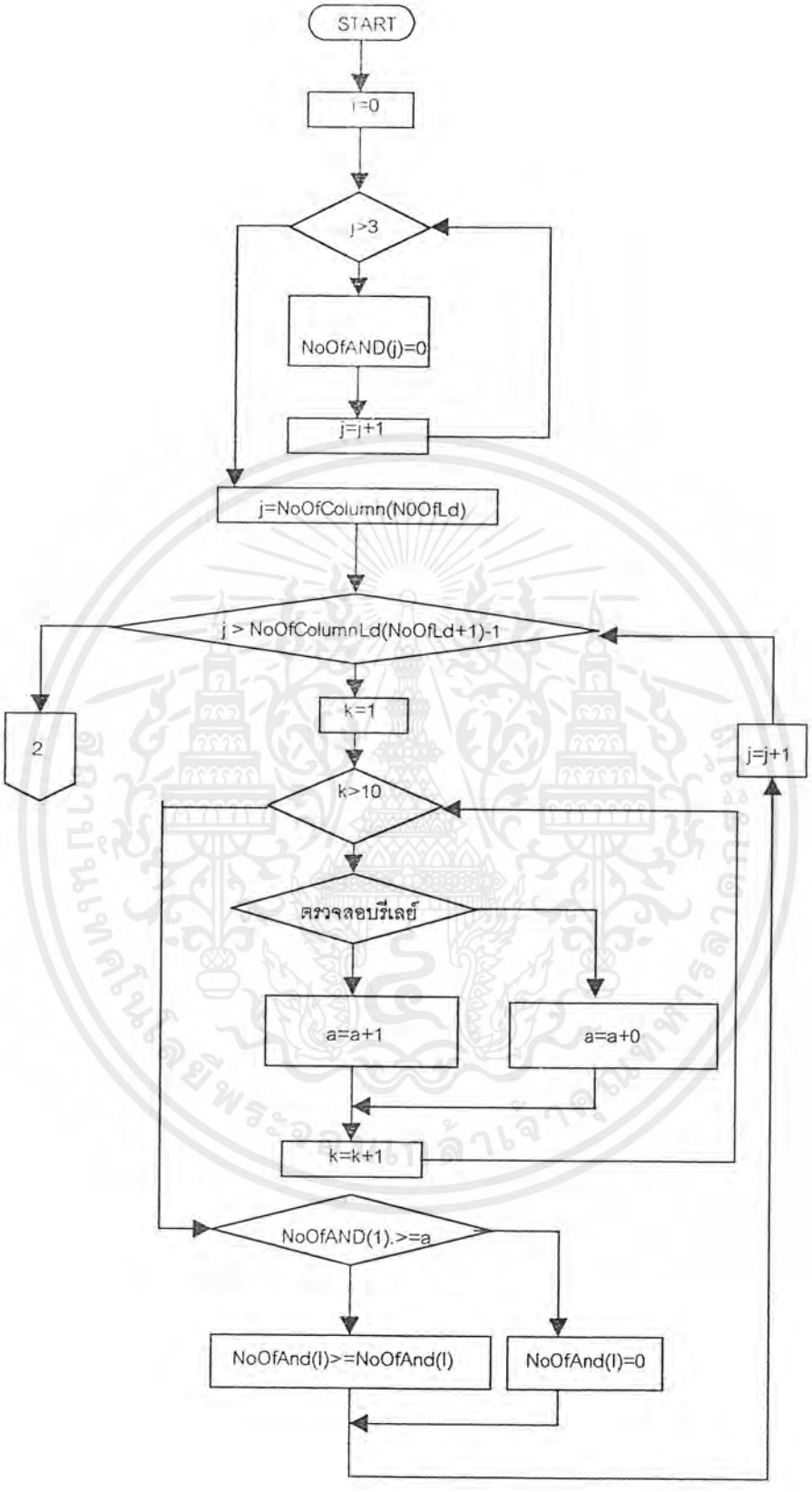
รูปที่ ข-74 กราฟแสดงการประมวลผล DefindLineAndLD(6)



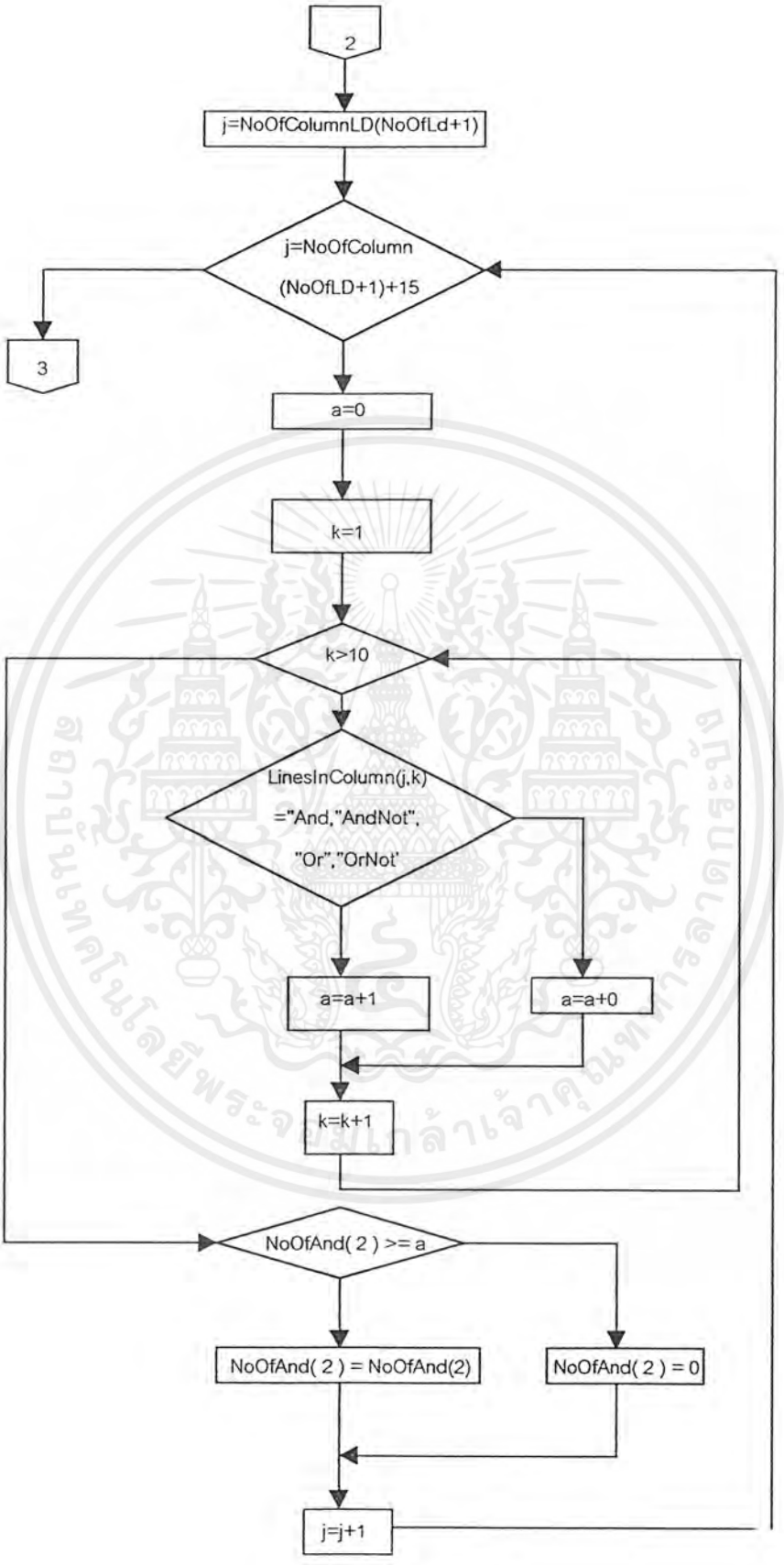
รูปที่ ข.75 กราฟแสดงการประมวลผล DefindLineTimer



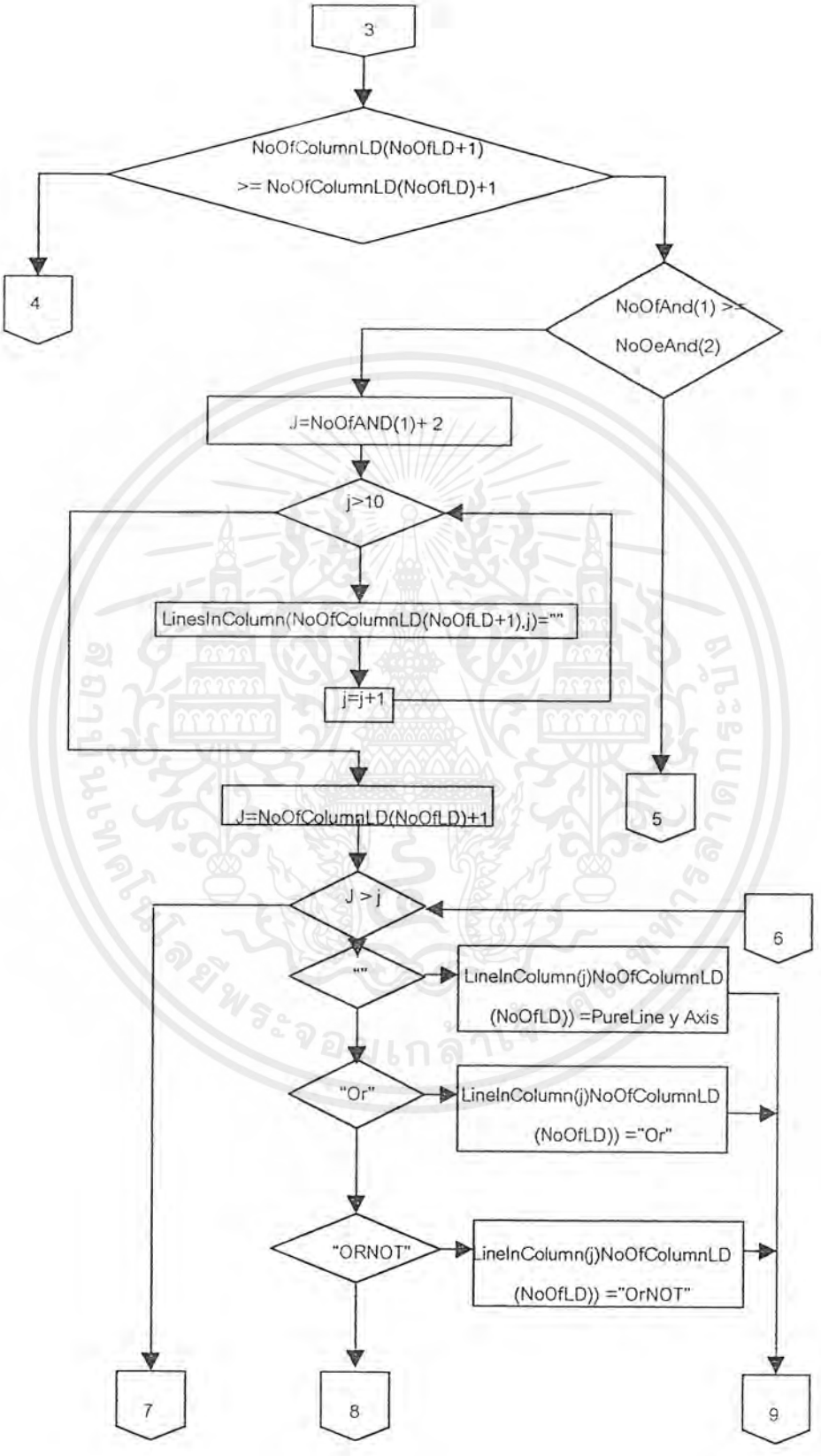
รูปที่ ข.76 กราฟแสดงการประมวลผล DefindLine OrLD



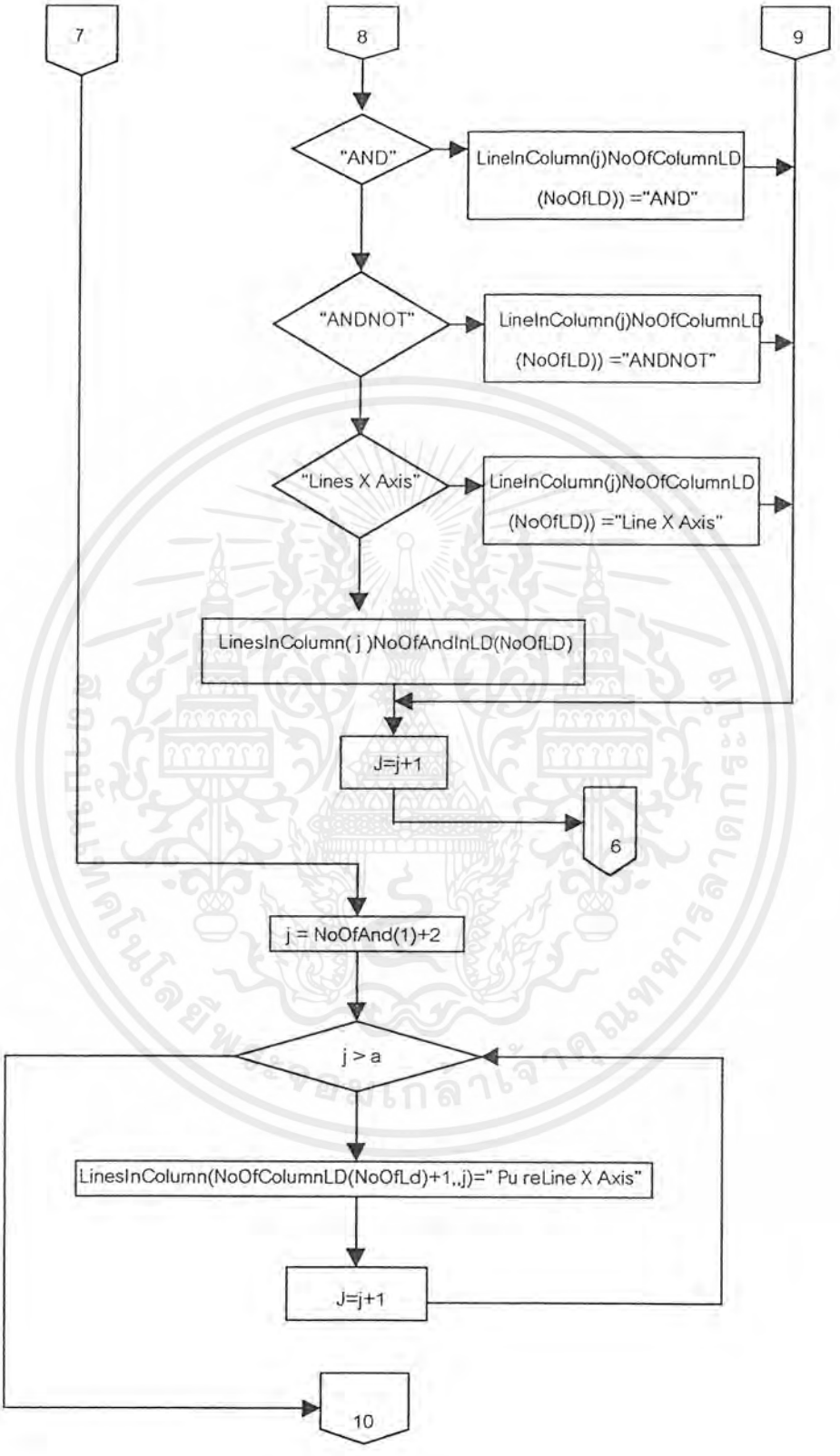
รูปที่ ข.77 กราฟแสดงการประมวลผล DefindLine COUNTER(1)



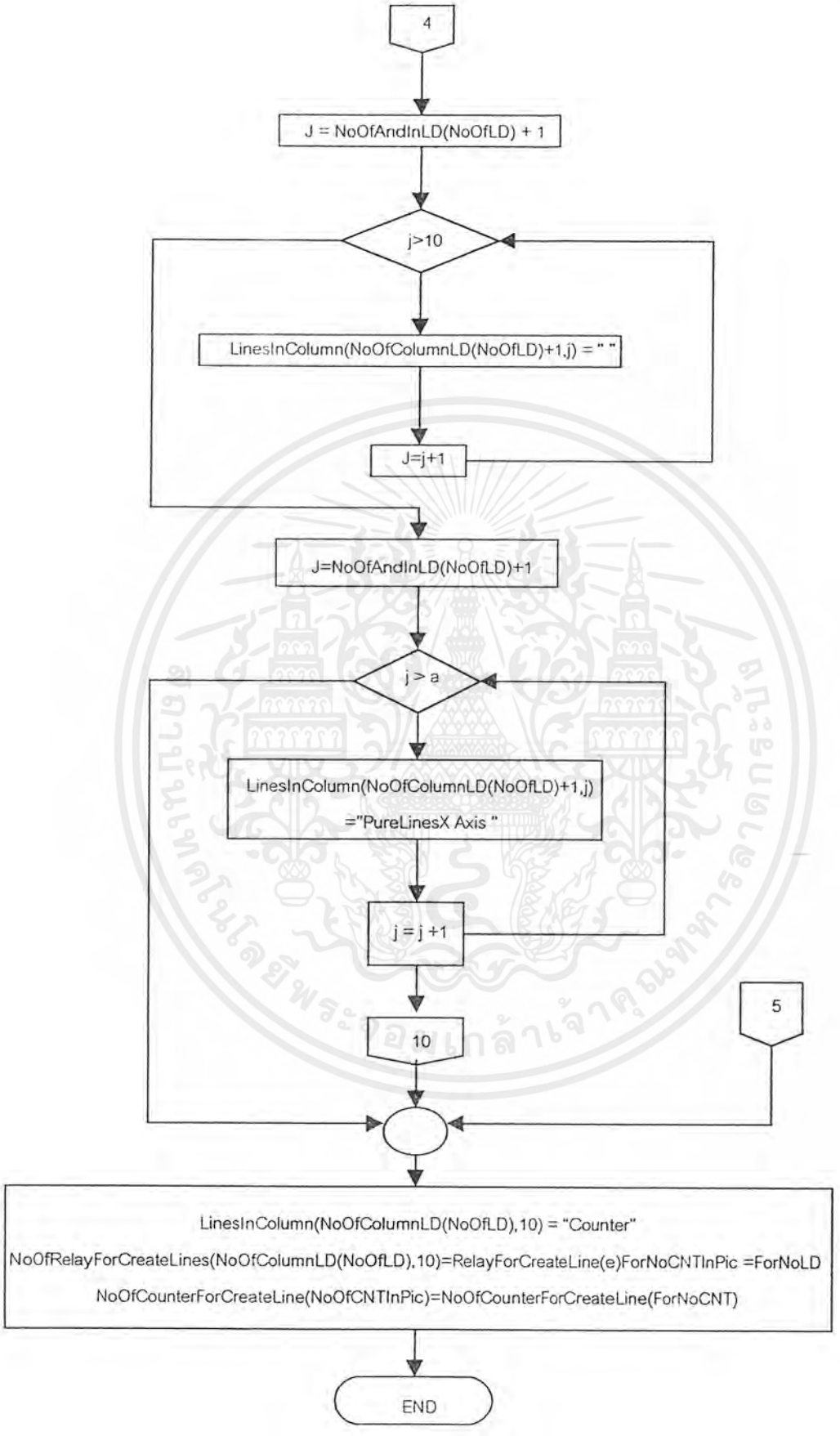
รูปที่ ข.78 กราฟแสดงการประมวลผล DefindLine COUNTER(2)



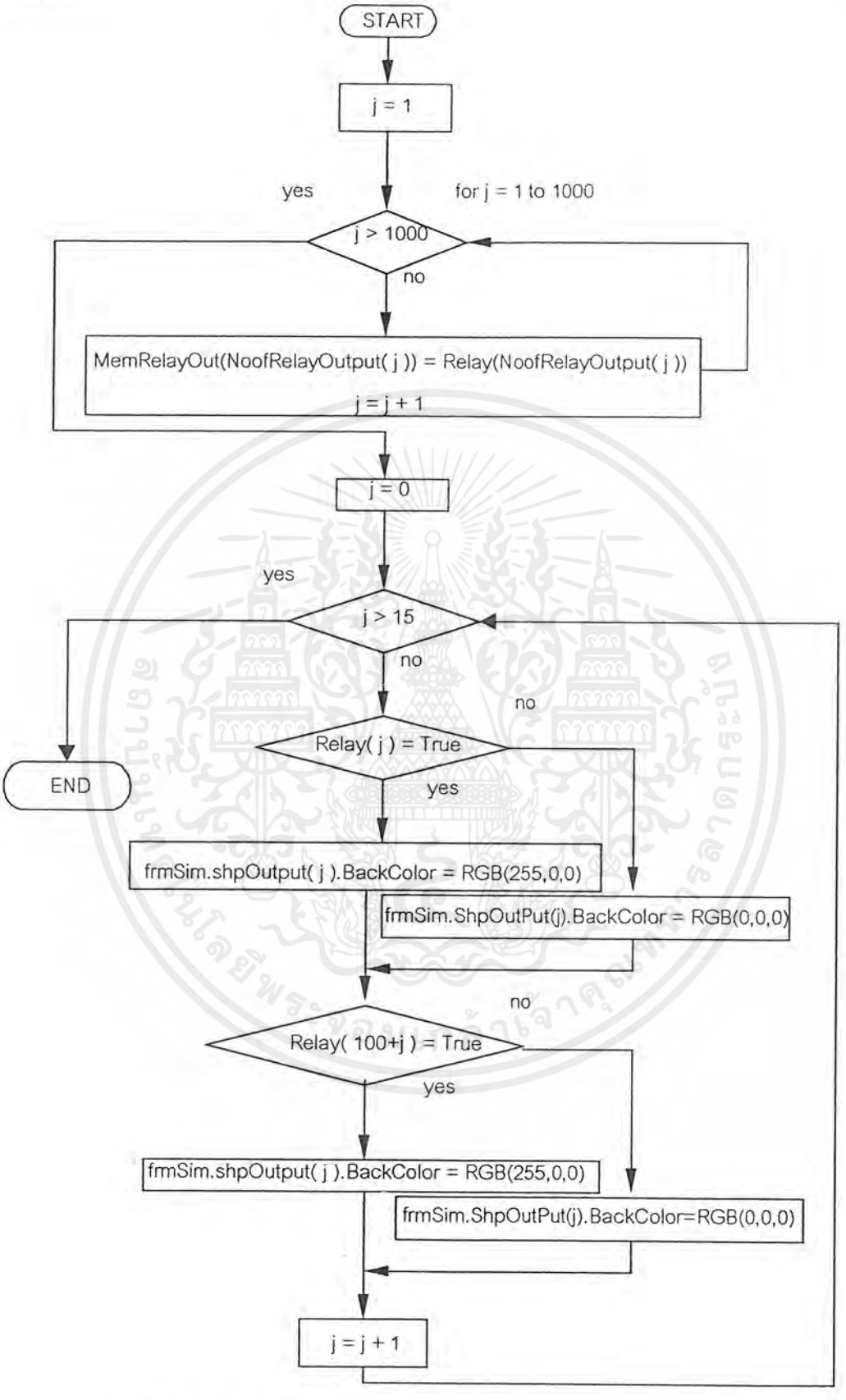
รูปที่ ข.79 กราฟแสดงการประมวลผล DefindLine COUNTER(3)



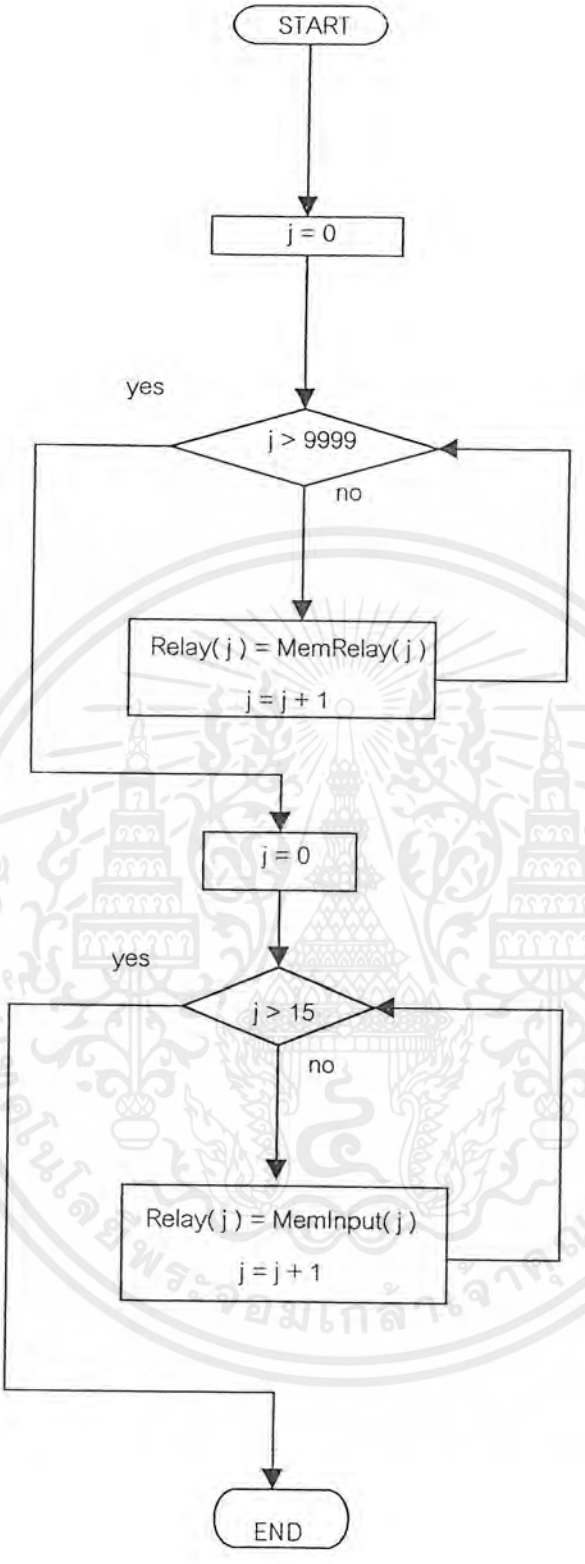
รูปที่ ข.80 กราฟแสดงการประมวลผล DefindLine COUNTER(4)



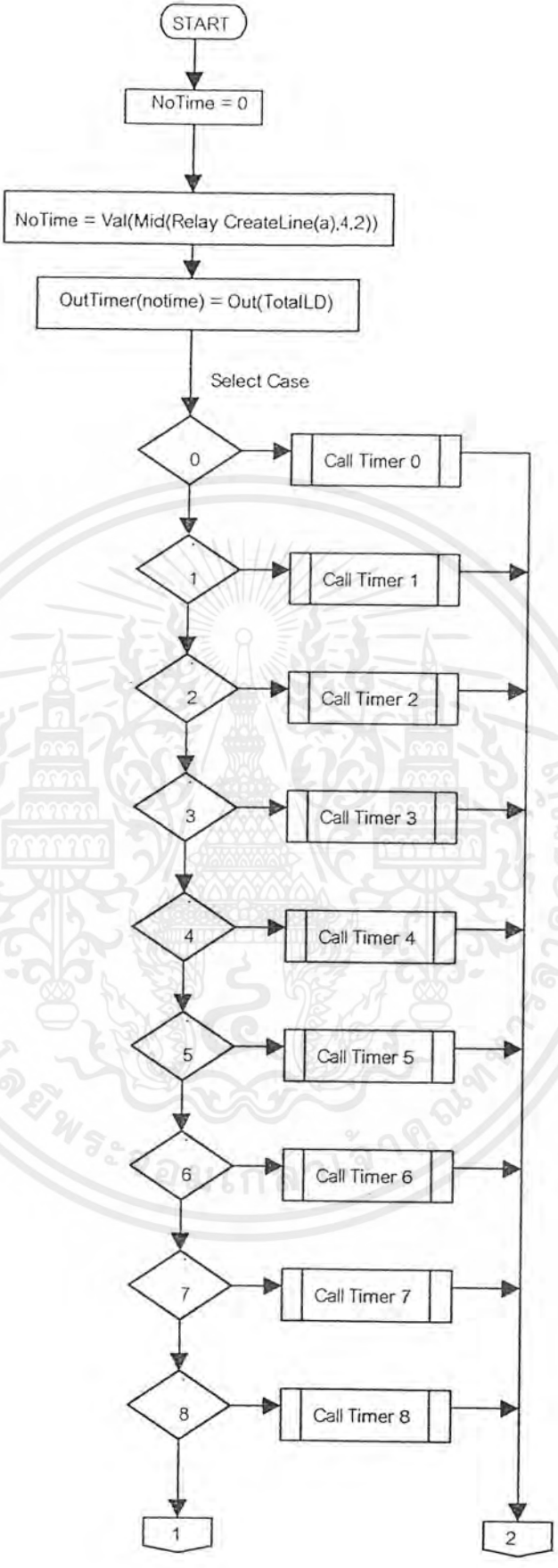
รูปที่ ข.81 กราฟแสดงการประมวลผล DefindLine COUNTER(5)



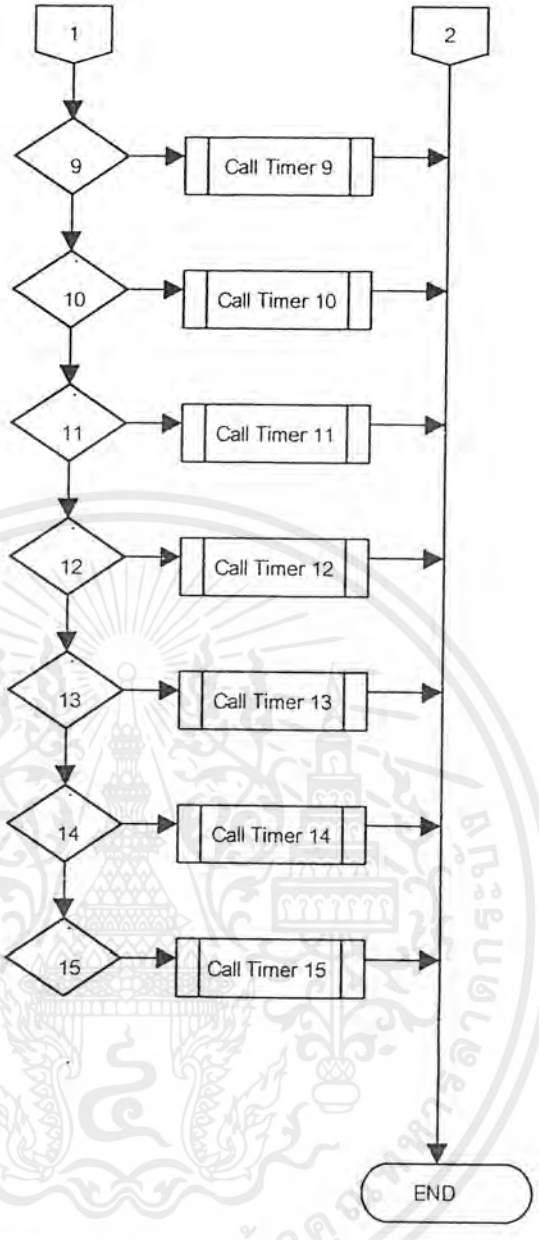
รูปที่ ข.82 กราฟแสดงการประมวลผล ShowOutput



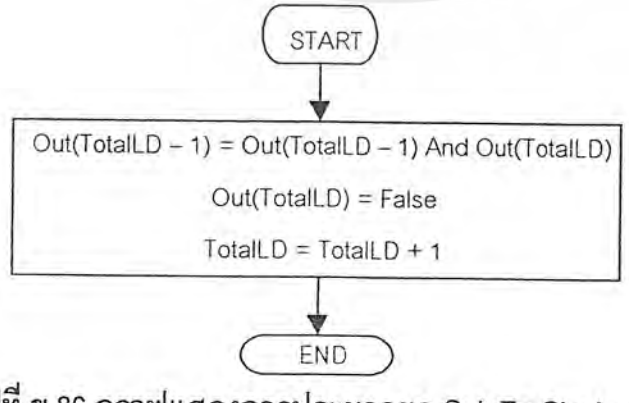
รูปที่ ข.83 กราฟแสดงการประมวลผล Sub CheckInput Output



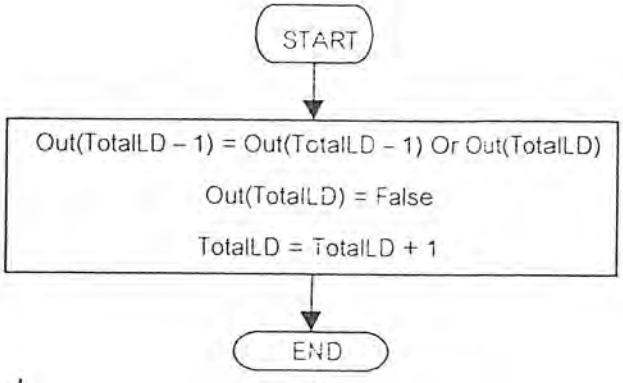
รูปที่ ข.84 กราฟแสดงการประมวลผล Sub To SimTime(1)



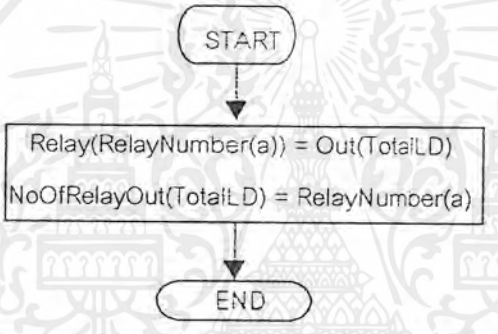
รูปที่ ข.85 กราฟแสดงการประมวลผล Sub To SimTime(2)



รูปที่ ข.86 กราฟแสดงการประมวลผล Sub To SimAndLD



รูปที่ ข.87 กราฟแสดงการประมวลผล Sub To SimOrLD



รูปที่ ข.88 กราฟแสดงการประมวลผล Sub To SimOut