โปรแกรมสร้างผังภาพและจำลองการทำงานวงจรตรรก

LOGIC CIRCUIT EDITOR AND SIMULATION PROGRAM

สุพรรณ กุลพานิชย์

SUPHAN GULPANICH

อาจารย์ที่ปรึกษา

รศ.ดร. สิทธิชัย โภไคยอุดม

ADVISOR

Assc. Prof. Dr. SITTHICHAI POOKAIYAUDOM

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2531

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

แบบฟอร์มการให้คะแนนการสอบวิทยานิพนธ์

สำหรับนักศึกษาระดับมหาบัณฑิต

ชื่อนักศึกษา นายสุพรรพ กุลพาพิชย์ เลขประจำตัว 29.0012

ชื่อเรื่องวิทยานิพนธ์ โปรแกรมสร้างผังภาพและจำลองการทำงานวงจรตรรก

(Logic Circuit Editor and Simulation Program)

ชื่ออาจารย์ผู้ควบคุมการสอบ	ลายมือชื่อ	ผลการสอบ
รศ.ดร.สิทธิชัย โภไคยอุดม		ดี
รศ.กิตติ ตีระเศรษฐ		ดี
รศ.ประทีป บัญญัตินพรัตน์		ผ่าน
ผศ.ครรชิต ไมตรี		ผ่าน

วันเดือนปี ที่สอบ 30 กันยายน 2531 เวลา 10.00 น. สถานที่ ห้อง A-305

(นายสิทธิชัย โภไคยอุดม)

คณบดีบัณฑิตวิทยาลัย

สารบัญ

	หน้า
บทคัดย่อ	IV
Abstract	V
บทที่ 1 บทนำ	1
บทที่ 2 ลักษณะโครงสร้างของโปรแกรม LOGCAD	4
2.1 โครงสร้างโปรแกรมสร้างผังภาพวงจร	4
2.2 โครงสร้างโปรแกรมจำลองการทำงานของวงจรตรรก	7
บทที่ 3 โปรแกรมการสร้างรูปอุปกรณ์ และ ผังภาพวงจร	13
3.1 โปรแกรมการสร้างรูปอุปกรณ์ไลบรารี	17
3.1.1 โปรแกรมกำหนดขอบเขตรูปไลบรารี	20
3.1.2 โปรแกรมยกเลิกงานไลบรารีที่สร้าง	22
3.1.3 โปรแกรมลบเฉพาะบางส่วนของรูปไลบรารี	23
3.1.4 โปรแกรมสร้างรูปอุปกรณ์ไลบรารี	25
3.1.5 โปรแกรมบริการไฟล์ไลบรารี	27
3.1.6 โปรแกรมแสดงจุดกริด	29
3.1.7 โปรแกรมย้ายตำแหน่งจุดตัวชี้	30
3.1.8 โปรแกรมกำหนดชื่ออุปกรณ์ไลบรารี	32
3.1.9 โปรแกรมวาดรูปไลบรารีซ้ำของเดิม	33
3.1.10 โปรแกรมกำหนดตำแหน่งการย้าย	34
3.1.11 โปรแกรมกำหนดขนาดภาพ	37
3.2 โปรแกรมการสร้างผังภาพวงจร	39
3.2.1 โปรแกรมการเลือกคำสั่งหลัก	40
3.2.2 โปรแกรมการสร้างผังภาพวงจร	43
3.3 ลักษณะโครงสร้างข้อมูลของอุปกรณ์ไลบรารี และ ผังภาพวงจร	45
3.3.1 โครงสร้างข้อมูลอุปกรณ์ไลบรารี	45
3.3.2 โครงสร้างข้อมูลผังภาพวงจร	47
บทที่ 4 โปรแกรมการสร้างไฟล์ข้อมูลโหนด	55

4.1 โปรแกรม LOGNODE1 55

4.2 โปรแกรม LOGNODE 58

บทที่ 5 การใช้ TURBO PASCAL เป็นตัวแปลภาษาให้กับแบบจำลอง 63

5.1 ไฟล์ข้อมูลโหนด และ ไฟล์ข้อมูลแบบจำลอง 67

5.1.1 ไฟล์ข้อมูลโหนด 67

5.1.2 ไฟล์ข้อมูลแบบจำลอง 68

5.2 DATATXRX UNIT 69

5.3 โปรแกรม LOGNET 72

บทที่ 6 โปรแกรมจำลองการทำงานวงจรตรรก 74

6.1 โปรแกรมหลัก 75

6.2 โปรแกรมควบคุมการแสดงคำสั่ง 77

6.3 โปรแกรมคำสั่งไฟล์ข้อมูล 78

6.4 โปรแกรมการกำหนดข้อมูลเข้าวงจร 80

6.5 โปรแกรมการแสดงผล 86

6.5.1 โปรแกรมกำหนดตำแหน่งในวงจรเพื่อการตรวจสอบ 89

6.6 โปรแกรมการประมวลผล และ การเลื่อนภาพแผนภูมิเวลา 91

6.7 โปรแกรมการพิมพ์ภาพแผนภูมิเวลา 100

บทที่ 7 ตัวอย่างการใช้งานโปรแกรม LOGCAD 110

7.1 การสร้างรูปจาก LOGCAD EDITOR และ การสร้างไฟล์-ข้อมูลโหนด 110

7.1.1 การใช้งานโปรแกรมการสร้างไฟล์อุปกรณ์ไลบรารี 111

7.1.2 การใช้งานโปรแกรมการสร้างไฟล์ผังภาพวงจร 113

7.1.3 การใช้งานโปรแกรมสร้างไฟล์ข้อมูลโหนด 115

7.2 การสร้างรูปจาก ORCAD EDITOR และ การสร้างไฟล์-ข้อมูลโหนด 116

7.2.1 การใช้งานโปรแกรมการสร้างผังภาพวงจร-
จาก ORCAD EDITOR 116
7.2.2 การใช้งานโปรแกรมสร้างไฟล์ข้อมูลโหนด 116
7.3 การสร้างแบบจำลอง และ การแปลภาษา 118
7.3.1 การเขียนแบบจำลอง 118
7.3.2 การใช้งานโปรแกรม LOGNET 120
7.3.3 การใช้งานโปรแกรม PRESIM เพื่อการแปลภาษา 120
7.4 การจำลองการทำงานของวงจร (LOGCAD SIMULATOR) 121
7.4.1 การกำหนดชื่อไฟล์เพื่อจำลองการทำงาน 121
7.4.2 การกำหนดข้อมูลเข้าวงจร 121
7.4.3 การโพรบตำแหน่งต่าง ๆ เพื่อการแสดงภาพแผน-
ภูมิเวลา 122
7.4.4 การประมวลผล และ การเลื่อนภาพแผนภูมิเวลา 122
7.4.5 การพิมพ์ภาพแผนภูมิเวลา 123

บทที่ 8 บทสรุป 137

หนังสืออ้างอิง 139
ภาคผนวก 1
ภาคผนวก 2

หัวข้อวิทยานิพนธ์ โปรแกรมสร้างผังภาพและจำลองการทำงานวงจรตรรก
นักศึกษา นาย สุพรรณ กุลพาณิชย์
อาจารย์ที่ปรึกษา รศ.ดร. สิทธิชัย โภไคยอุดม
ระดับการศึกษา วิศวกรรมศาสตรมหาบัณฑิตทางวิศวกรรมไฟฟ้า
ปีการศึกษา 2531

บทคัดย่อ

การออกแบบและทดสอบการทำงานของวงจรตรรกนั้น จำเป็นต้องใช้อุปกรณ์เครื่องมือวัด และต้องเสียเวลามากในการทดลองตลอดจนแก้ไขวงจรทางด้านฮาร์ดแวร์เพื่อให้ได้ผลลัพธ์ที่ต้องการออกมา ในปัจจุบันไมโครคอมพิวเตอร์มีราคาถูก มีประสิทธิภาพสูงและถูกประยุกต์ใช้ในงานทุก ๆ ด้าน วิทยานิพนธ์ฉบับนี้ขอเสนอการนำไมโครคอมพิวเตอร์มาช่วยในการออกแบบวงจรตรรก โดยการออกแบบสร้างโปรแกรมจำลองการทำงานของวงจรตรรก ซึ่งจะทำให้สามารถออกแบบและทดสอบการทำงานของวงจรตรรกใดๆ ได้โดยไม่ต้องใช้อุปกรณ์และเครื่องมือวัดอื่น ๆ เลย ผู้ใช้จะเป็นผู้สร้างผังภาพวงจรและกำหนดฟังก์ชั่นการทำงานตามที่ต้องการ จากนั้นไมโครคอมพิวเตอร์จะจำลองการทำงานของผังภาพวงจรดังกล่าว และแสดงผลในรูปของแผนภูมิเวลา บนส่วนแสดงผลของไมโครคอมพิวเตอร์ทางจอภาพหรือเครื่องพิมพ์ก็ได้ นอกจากนี้ผู้ใช้ยังสามารถทำการแก้ไขวงจรตรรกและจำลองการทำงานของวงจรใหม่เพื่อให้ผลลัพธ์เป็นไปตามต้องการได้โดยง่าย

Thesis Title Logic Circuit Editor and Simulation Program
Name suphan GULPANICH
Thesis Advisor sitthichai POOKAIYAUDOM, Ph.D.
Level of study MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
Academic Year 1988

Abstract

The design and investigate the operation of logic circuit requirte some certain equipment and instruments, and of course,it takes quite longer time for experiment and modify the hardwired circuit until the required result is obtained. Nowadays, microcomputer are less expensive but greater in efficiency,and is widely applied to any task. This thesis presents one application of microcomputer for logic circuit aided-design. The logic circuit simulation program is designed and created up so that any logic circuit can be designed and easy to investigate the result without any equipment or instruments needed. User will only define the schematic diagram and function of the required logic circuit, then the microcomputer simulate the operation of the overall circuit and display the result in term of timing diagram through the CRT monitor or printer. Moreover, user can modify the logic circuit using edit mode and re-simulate the operation of modified circuit until the desired result is obtained easily.

บทที่ 1
บทนำ

การพัฒนาวงจรตรรกในอดีตที่ผ่านมา สามารถตรวจสอบการใช้งานของวงจรได้จากการทดลองกับอุปกรณ์จริงเท่านั้น จึงทำให้การพัฒนาเป็นไปได้อย่างล่าช้า เพราะเหตุว่าในระหว่างที่ทำการทดสอบหรือทดลองอยู่นั้นอาจจะเกิดความผิดพลาดหรือบกพร่องอันเนื่องมาจาก เครื่องมือวัด อุปกรณ์การทดลอง หรือผู้ออกแบบได้ทุกเมื่อ ดังนั้นกว่าจะได้วงจรที่ใช้งานจริงจะต้องอาศัยเวลาในการตรวจสอบอยู่นาน เพื่อเป็นการหลีกเลี่ยงอุปสรรคต่าง ๆ เหล่านี้ วิทยานิพนธ์ฉบับนี้ขอเสนอการประยุกต์ใช้ไมโครคอมพิวเตอร์ช่วยในการออกแบบวงจรตรรก ด้วยโปรแกรมจำลองการทำงานของวงจรตรรก ซึ่งผู้ออกแบบสามารถสร้างผังภาพวงจร (SCHEMATIC DIAGRAM) และ กำหนดฟังก์ชันการทำงานที่จัดอยู่ในรูปของโปรแกรมให้สอดคล้องกับผังภาพวงจรนั้น ๆ ลงในส่วนของไมโครคอมพิวเตอร์เพื่อทำการประมวลผล จากนั้นนำข้อมูลที่ได้แสดงบนส่วนแสดงผลของไมโครคอมพิวเตอร์ให้อยู่ในรูปของ แผนภูมิเวลา (TIMING DIAGRAM)

เนื่องจากโปรแกรมจำลองการทำงานของวงจรตรรกเป็นวิธีการ ทางซอฟต์แวร์ (SOFT WARE) ที่ใช้บนไมโครคอมพิวเตอร์ IBM PC/XT/AT ซึ่งมีหน่วยความจำขนาด 256 K เป็นอย่างน้อย ในส่วนของการแสดงกำหนดให้ใช้ได้กับจอภาพ EGA หรือ MONOCHROME ทั้งนี้เพื่อให้ได้ความละเอียด และ พื้นที่ในการแสดงผล ที่สูงขึ้น นอกจากนี้ในการประมวลผลจะนำข้อมูลที่ได้เก็บลงยังหน่วยความจำของคอมพิวเตอร์ ดังนั้นการแสดงภาพที่เป็นแผนภูมิเวลาผู้ใช้สามารถกำหนดได้ว่าจะตรวจสอบ ณ.จุดใด และ ขั้นหรือสเต็ป (STEP) ที่เท่าไรของการจำลองการทำงานวงจรตลอดจนการเลื่อนภาพแผนภูมิเวลาเพื่อแสดงในสเต็ปต่าง ๆ ก็ทำได้เช่นกัน จากคุณสมบัติดังกล่าวจึงเป็นการแสดงผลในลักษณะเดียวกันกับการใช้งานของ เครื่องลอจิกอะนาไลเซอร์ (LOGIC ANALYZER) ที่ทำการตรวจสอบการทำงานของวงจรจริง การใช้งานลักษณะเช่นนี้จะทำให้สิ้นเปลืองค่าใช้จ่ายที่สูง ดังนั้นโปรแกรมจำลองการทำงานวงจรตรรกจึงเป็นวิธีหนึ่งที่ใช้ในการตรวจสอบการทำงานของวงจรได้อย่างประหยัดทั้งในเรื่องของเวลาและค่าใช้จ่ายในกรณีที่ผู้ใช้โปรแกรมมีเครื่องคอมพิวเตอร์เพื่อจุดประสงค์อื่นอยู่แล้ว

วิทยานิพนธ์ฉบับนี้กล่าวถึงโปรแกรมจำลองการทำงานของวงจรตรรก หรือมีชื่อเรียกอีกชื่อหนึ่งว่า LOGCAD สามารถแบ่งรายละเอียดออกเป็นบทได้ดังนี้

บทที่ 2 กล่าวถึงลักษณะโครงสร้างของโปรแกรม LOGCAD ว่าโครงสร้างหลักของโปรแกรมนั้นประกอบด้วยส่วน

- โปรแกรมสร้างผังภาพวงจรตรรก (LOGCAD EDITOR)
- โปรแกรมจำลองการทำงานของวงจรตรรก (LOGCAD SIMULATOR)

บทที่ 3 กล่าวถึงโปรแกรมการสร้างรูปอุปกรณ์ และผังภาพวงจร เพราะการสร้างรูปอุปกรณ์ หรือ ผังภาพวงจรจะต้องสร้างจากคำสั่งหลายคำสั่ง ซึ่งแต่ละคำสั่งจะมีหน้าที่แตกต่างกันออกไปตามลักษณะการใช้งาน ดังนั้นในบทนี้จะได้อธิบายถึงรายละเอียดของคำสั่งต่าง ๆ เหล่านี้

ในการสร้างผังภาพวงจร นอกจากจะใช้โปรแกรม LOCAD EDITOR ซึ่งเป็นโปรแกรมที่ได้พัฒนาขึ้นมาสำหรับวิทยานิพนธ์ฉบับนี้โดยเฉพาะแล้ว ยังสามารถใช้โปรแกรม ORCAD ที่เป็นโปรแกรมจากต่างประเทศได้อีกโปรแกรมหนึ่งสาเหตุที่เลือกโปรแกรม ORCAD ก็เพราะเหตุผลสองประการ

ประการแรก ORCAD เป็นโปรแกรมที่มีประสิทธิภาพ ความคล่องตัวในการใช้งานค่อนข้างสูงและเหมาะสำหรับการสร้างรูปผังภาพวงจรตรรกโดยเฉพาะ

ประการที่สอง ORCAD ยังขาดส่วนที่ใช้ในการจำลองการทำงานของวงจร ซึ่งนับว่าเป็นสิ่งที่จำเป็นสำหรับผู้ออกแบบวงจรตรรกอย่างมาก

บทที่ 4 โปรแกรมการสร้างไฟล์ข้อมูลโหนด กล่าวถึงวิธีที่จะทำให้ได้ไฟล์ข้อมูลโหนดของผังภาพวงจรที่สร้างจากโปรแกรม LOGCAD และ ORCAD

บทที่ 5 เป็นการกล่าวถึงการใช้ TURBO PASCAL เป็นตัวแปลภาษาให้กับแบบจำลอง (MODEL) ที่เขียนเป็นสมการบูลลีน (BOOLEAN EQUATION) ของวงจรตรรก

บทที่ 6 โปรแกรมจำลองการทำงานของวงจรตรรก กล่าวถึงวิธีการทำงานของโปรแกรมตลอดจนส่วนประกอบอื่น ๆ ที่เกี่ยวข้องในการจำลองการทำงาน ผลสุดท้ายที่ได้จะเป็นการแสดงสภาวะ ณ.จุดต่าง ๆ ของวงจรให้อยู่ในรูปของแผนภูมิเวลายังส่วนแสดงผลของคอมพิวเตอร์ ทั้งที่เป็นจอภาพ และเครื่องพิมพ์

บทที่ 7 ตัวอย่างการใช้งานของโปรแกรม LOGCAD

บทที่ 8 บทสรุปผลงานวิจัยของวิทยานิพนธ์นี้

ภาคผนวก 1 คำสงวนที่ห้ามใช้ในการกำหนดชื่อตัวแปรสำหรับการสร้างแบบจำลอง

ภาคผนวก 2 ข้อมูลอุปกรณ์ไลบรารี และ ข้อมูลผังภาพวงจร ของโปรแกรม LOG-

CAD EDITOR

ภาคผนวก 3 เป็นรายละเอียดของโปรแกรม

ขอบเขตของวิทยานิพนธ์

โปรแกรมจำลองการทำงานนี้จะจำลองได้เฉพาะ วงจรที่เป็นวงจรตรรกเท่านั้น และไม่มีการกำหนดค่าของการหน่วงเวลา (PROPAGATE DELAY TIME) ให้กับอุปกรณ์ทุกประเภทที่นำมาประกอบกันเป็นผังภาพวงจร.

บทที่2

ลักษณะโครงสร้างของโปรแกรม LOGCAD

LOGCAD เป็นโปรแกรมที่ใช้ในการออกแบบวงจรตรรกจะ ประกอบด้วยโปรแกรมหลัก 2 ส่วนใหญ่ ๆ มีโครงสร้างดังนี้

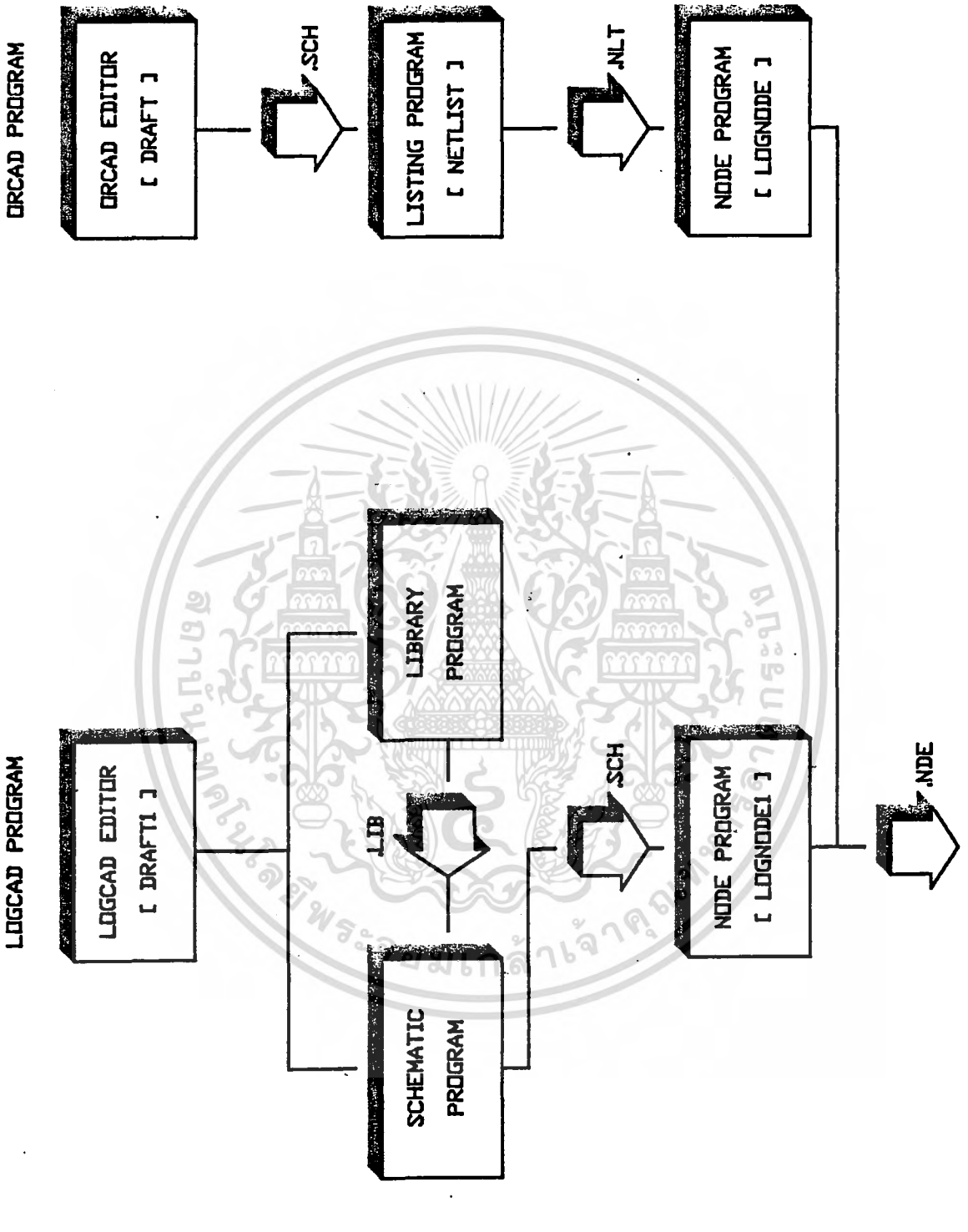
- โครงสร้างโปรแกรมการสร้างผังภาพวงจร (LOGCAD EDITOR)
- โครงสร้างโปรแกรมจำลองการทำงานวงจร (LOGCAD SIMULATOR)

2.1 โครงสร้างโปรแกรมการสร้างผังภาพวงจร (LOGCAD EDITOR)

เป็นส่วนของโปรแกรมที่ใช้ในการสร้างรูปของอุปกรณ์ ผังภาพวงจร และไฟล์ข้อมูลโหนดอัน เป็นจุดต่อของอุปกรณ์สำหรับการประกอบกัน เป็นผังภาพวงจรอีกส่วนหนึ่งด้วย ในส่วนการสร้างผังภาพวงจรตรรกจะมีขีดความสามารถของโปรแกรมดังนี้

ขีดความสามารถของโปรแกรม LOGCAD EDITOR

ประเภทของรายการ	จำนวนสูงสุด
1. เส้นที่ใช้ประกอบเป็นรูปอุปกรณ์	50
2. พื้นที่ในการสร้างผังภาพวงจร	900 x 600
3. ไฟล์ที่กำหนดให้เป็นรูปอุปกรณ์	ไม่จำกัด
4. ไฟล์อุปกรณ์ที่จะเลือกลงสู่หน่วยความจำเพื่อเรียกใช้สำหรับประกอบเป็นผังภาพวงจร	50
5. สร้างผังภาพวงจรได้จากโปรแกรม LOGCAD และโปรแกรม ORCAD	--
6. การแสดงผลสามารถแสดงได้ทั้งจอภาพ EGA และ MONOCHROME	--



รูปที่ 2-1 โครงสร้างของโปรแกรมการสร้างผังภาพและโหนดของวงจรตรรก

จากรูปที่ 2-1 จะเห็นได้ว่า LOGCAD EDITOR นั้นประกอบด้วย 2 ส่วนดังนี้

2.1.1 DRAFT1

เป็นโปรแกรมที่ใช้สำหรับสร้างไฟล์ข้อมูลไลบรารีของอุปกรณ์ และ ไฟล์ข้อมูลผังภาพวงจร

- LIBRARY เป็นส่วนของโปรแกรม DRAFT1 ที่ใช้ในการสร้างรูปของอุปกรณ์แบบต่าง ๆ ซึ่งเป็นมาตรฐานทางอิเลคทรอนิคส์ทั่ว ๆ ไปและในการสร้างรูปอุปกรณ์ต่าง ๆ เหล่านี้สามารถสร้างได้จากคำสั่งควบคุมการสร้างภาพพื้นฐาน ดังเช่น เส้นตรง เส้นโค้ง วงกลม และ ข้อความ เป็นต้น นอกจากนี้ในบางกรณีที่ผู้ใช้ต้องการเปลี่ยนแปลงแก้ไขเส้นต่าง ๆ ที่เกิดจากการลงตำแหน่งผิดพลาด หรือต้องการเพิ่มเติมบางส่วนของรูปอุปกรณ์ก็ดีจะทำได้โดยง่าย และในระหว่างที่มีการสร้างรูปอุปกรณ์อยู่ ผู้ใช้สามารถทำการย่อ-ขยาย ขนานของอุปกรณ์ได้ ทั้งนี้เพื่อให้เกิดความเหมาะสมต่อการสร้างรูปตลอดจนการที่จะนำรูปเหล่านี้ไปประกอบเป็นผังภาพวงจรต่อไป ข้อมูลที่ได้จากการสร้างรูปอุปกรณ์นี้จะถูกเก็บลงสู่ไฟล์ข้อมูล (DATA FILE) ที่มีชื่อหลังจุดเป็น LIB (FILENAME.LIB) ในส่วนของรายละเอียดอื่น ๆ จะได้กล่าวในบทที่ 3

- SCHEMATIC เป็นส่วนของโปรแกรม DRAFT1 ที่ใช้สำหรับสร้างผังภาพวงจร ลักษณะและขั้นตอนการใช้งานคล้าย ๆ กับโปรแกรมในส่วนของ LIBRARY แต่มีข้อแตกต่างตรงที่จะนำข้อมูลจากไฟล์ข้อมูลไลบรารี ที่สร้างเสร็จเรียบร้อยแล้วนั้นมาประกอบกันเป็นผังภาพวงจร ดังนั้นในการสร้างผังภาพวงจรหนึ่ง ๆ จะประกอบไปด้วยอุปกรณ์ที่เป็นข้อมูลไลบรารีอยู่หลายประเภทด้วยกัน และจำนวนของข้อมูลไลบรารีมากน้อยต่างกันนั้นจะขึ้นอยู่กับลักษณะของผังภาพวงจรที่ได้กำหนดขึ้น ว่าเป็นวงจรประเภทใด

ภายหลังจากที่สร้างผังภาพวงจรเสร็จเรียบร้อยแล้ว ข้อมูลนี้จะถูกเก็บลงสู่ไฟล์ข้อมูลเช่นกันที่มีชื่อหลังจุดเป็น SCH (FILENAME.SCH) ในส่วนของรายละเอียดอื่น ๆ จะได้กล่าวในบทที่ 3

2.1.2 LOGNODE1

เป็นโปรแกรมที่ใช้ในการอ่านไฟล์ข้อมูลที่สร้างมาจากโปรแกรม LOGCAD EDITOR (FILENAME.SCH) และเลือกเอาเฉพาะข้อมูลที่เป็น อินพุท เอาท์พุท และ จุดเชื่อมต่อ หรือรวมเรียกข้อมูลเหล่านี้ว่า โหนด (NODE) ออกมาเพื่อใช้เป็นข้อมูลหลักสำหรับโปรแกรมจำลองการทำงานของวงจร (LOGCAD SIMULATOR)

โหนด เป็นข้อมูลซึ่งนับได้ว่ามีความสำคัญอย่างยิ่ง เพราะทุกขั้นตอนของโปรแกรม LOGCAD SIMULATOR ส่วนใหญ่แล้วจะอาศัยโหนด เป็นข้อมูลหลักในการตรวจการทำงาน ดัง เช่นขั้นตอน การคำนวน การแสดงผล ตลอดจนการกำหนดข้อมูล เข้า (INPUT DATA) ของผังภาพวงจรที่ต้องการจะจำลองการทำงาน ข้อมูลหลักที่สร้างโดยโปรแกรม LOG-NODE1 จะ เป็นข้อมูลที่ถูก เก็บลงสู่ไฟล์และมีชื่อหลังจุด เป็น NDE (FILENAME.NDE) ไฟล์ข้อมูลลักษณะ เช่นนี้ถูก เรียกว่า **ไฟล์ข้อมูลโหนด** (NODE FILE)

สำหรับโปรแกรม ORCAD นั้นสามารถดำ เนินการได้ในลักษณะ เช่น เดียวกัน แต่ใน การสร้างไลบรารีของอุปกรณ์ดู เหมือนจะมีความจำ เป็นน้อย ทั้งนี้ เพราะว่าโปรแกรม ORCAD ได้บรรจุ เอาไลบรารีมาตรฐานของอุปกรณ์ไว้ซึ่งครอบคลุมทุกฟังก์ชั่นการทำงานอยู่ แล้วด้วย เหตุนี้ ผู้ใช้จึงสามารถ เรียกไลบรารีมาตรฐาน เหล่านี้ออกมาใช้งานได้ทุก เมื่อ ในระหว่างที่มีการสร้างผังภาพวงจร และในบางกรณี ผู้ใช้มีความจำ เป็นที่จะสร้างไลบรารี ของอุปกรณ์ เพิ่ม เติมซึ่งนอก เหนือไปจากไลนรารีมาตรฐานที่มีอยู่ก็ทำได้ เช่นกัน

สิ่งที่แตกต่างอีกประการหนึ่งระหว่างโปรแกรม LOGCAD กับ โปรแกรม ORCAD ก็ คือ ไฟล์ข้อมูลโหนดของโปรแกรม LOGCAD จะได้มาต้องผ่านการทำงานของโปรแกรม LOGNODE1 ตามที่ได้กล่าวมาแล้ว แต่สำหรับโปรแกรม ORCAD แล้วต้องผ่านการทำงาน ถึง 2 ขั้นตอนด้วยกัน

ขั้นตอนแรก ผ่านการทำงานของโปรแกรม NETLIST โดยการอ่านจากไฟล์ข้อมูล ผังภาพวงจร (FILENAME.SCH) ที่สร้างจากโปรแกรม DRAFT จากนั้นจะนำผลลัพธ์ที่ได้ ไปผ่านการทำงานในขั้นตอนที่สองต่อไป โดยผลลัพธ์ที่ได้ในขั้นตอนแรกจะ เป็นไฟล์ข้อมูล แบบข้อความ (TEXT FILE) ที่บอกถึงรายละ เอียดต่าง ๆ ของผังภาพวงจร เช่น ข้อมูล เกี่ยวกับชื่อของแต่ละฟังก์ชัน ข้อมูลอินพุท เอาท์พุท จุด เชื่อมต่อ และอื่น ๆ เป็นต้น ข้อมูล เหล่านี้จะถูก เก็บลงสู่ไฟล์ข้อมูลแบบข้อความที่มีชื่อหลังจุด เป็น NLT (FILENAME.NLT)

ขั้นตอนที่สอง เป็นขั้นตอนของการลดทอนข้อมูลจากขั้นตอนแรกให้ เหลือ เฉพาะข้อมูล หลัก เท่านั้น ทำได้โดยการผ่านโปรแกรม LOGNODE สุดท้ายจะได้ข้อมูล อินพุท เอาท์พุท และจุด เชื่อมต่อ ข้อมูล เหล่านี้จะถูก เก็บลงสู่ไฟล์ข้อมูลที่มีชื่อหลังจุด เป็น NDE (FILENAME. NDE) ซึ่ง เรียกว่าไฟล์ข้อมูลโหนด เช่น เดียวกับข้อมูลที่ได้จากโปรแกรม LOGNODE1 นั้น เอง

2.2 โครงสร้างโปรแกรมการจำลองการทำงานของวงจร (LOGCAD SIMULATOR)

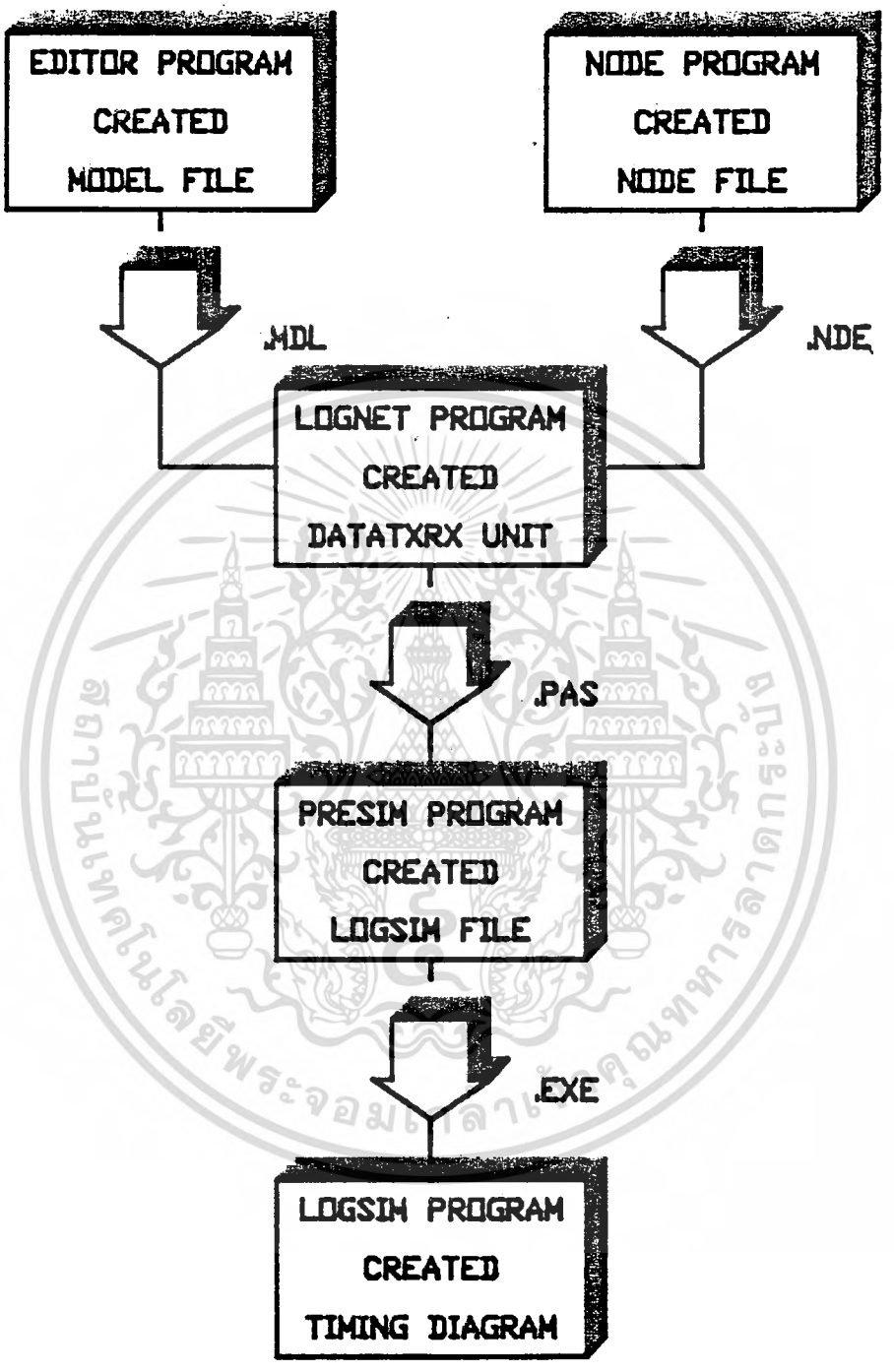
ส่วนของโปรแกรมนี้จะมีหน้าที่กำหนดข้อมูล เข้าให้กับตัวแปรโหนดอินพุท (INPUT

NODE VARIABLE) ของผังภาพวงจรที่ต้องการจำลองการทำงาน จากนั้นจะนำข้อมูลเข้าเหล่านี้ผ่านแบบจำลองที่เขียนอยู่ในรูปของโปรแกรมให้เป็นสมการบูลลีน แล้วจึงนำค่าสภาวะที่ได้จากการผ่านแบบจำลองเหล่านี้ไปเก็บยังตัวแปร โหนดเอาท์พุท (OUTPUT NODE VARIABLE) ซึ่งวิธีการนี้เป็นหลักการโดยพื้นฐานทั่ว ๆ ไป

สำหรับในการสร้างผังภาพวงจรหนึ่ง ๆ ส่วนที่ประกอบกันเป็นผังภาพวงจรนอกจากจะเป็นตัวอุปกรณ์แล้วยังประกอบด้วย อินพุท เอาท์พุท และจุดเชื่อมต่อ ซึ่งสิ่งเหล่านี้มีความสัมพันธ์กันในส่วนของการสร้างแบบจำลองตามที่ได้กล่าวมาแล้ว ดังนั้นในการจำลองการทำงานของวงจรจึงสามารถเห็นสภาวะการทำงาน คล้ายกับความเป็นจริงได้ทุกจุด ทุกตำแหน่งของผังภาพวงจรที่เป็น อินพุท เอาท์พุท และ จุดเชื่อมต่อให้อยู่ในรูปของแผนภูมิเวลา ซึ่งง่ายและสะดวกต่อการทำความเข้าใจ ในส่วนของการจำลองการทำงานจะมีขีดความสามารถของโปรแกรมดังนี้

ขีดความสามารถของโปรแกรม LOGCAD SIMULATOR

ประเภทของรายการ	จำนวนสูงสุด
1. จุดเชื่อมต่อ (NODE)	255
2. สเต็ป (STEP)	255
3. ตำแหน่งที่ต้องการแสดงผล (PROBE)	20
4. สัญญาณนาฬิกา (CLOCK)	20
5. การแสดงผลสามารถแสดงได้ทั้งจอภาพ EGA และ MONOCHROME	--
6. การแสดงภาพแผนภูมิเวลาสามารถแสดงได้ทางจอภาพและเครื่องพิมพ์	--



รูปที่ 2-2 โครงสร้างของโปรแกรมในการเตรียมข้อมูลสำหรับการจำลองการทำงาน

จากโครงสร้างของโปรแกรมในรูป 2-2 สามารถจำแนกออกเป็นส่วน ๆ ได้ดังนี้

2.2.1 แบบจำลอง (MODEL)

ผู้ใช้จะต้องเป็นผู้กำหนดสมการบูลลีนหรือเงื่อนไขต่าง ๆ ในลักษณะของโปรแกรมขึ้นมาให้สอดคล้องกับสภาพความเป็นจริงตามผังภาพวงจรที่ต้องการจำลองการทำงาน หรือกล่าวได้ว่า จะต้องเขียนสมการบูลลีนแสดงความสัมพันธ์ระหว่างโหนดทั้งหมดของผังภาพวงจรขึ้นมา

ในการเขียนแบบจำลองนี้จะเขียนได้จาก โปรแกรมอิดิเตอร์อื่น ๆ ได้ไม่ว่าจะเป็น WORDSTAR หรือ SIDEKICK หรือ TURBO PASCAL เป็นต้น สิ่งสำคัญลักษณะของแบบจำลองจะต้องเขียนให้ถูกตามรูปแบบหรือข้อกำหนดของ TURBO PASCAL ทั้งนี้เพราะว่าการจำลองการทำงานในแต่ละครั้งที่มีการเปลี่ยนผังภาพวงจร จะต้องมีการแปลภาษา (COMPILED) ในส่วนของแบบจำลองที่เขียนเป็นสมการบูลลีนนี้ใหม่ทุกครั้งเสมอ และการแปลภาษานั้นได้อาศัยตัวแปลภาษา (COMPILER) ของ TURBO PASCAL ช่วยในการแปล สิ่งนี้จึงเป็นข้อกำหนดสำหรับการสร้างแบบจำลอง

ตัวอย่าง การสร้างแบบจำลองตามสมการบูลลีน A EXCLUSIVE OR B

```
BEGIN
  N1  :=  INPUTA AND (NOT INPUTB);
  N2  :=  (NOT INPUTA) AND INPUTB;
  OUTPUT :=  N1 OR N2;
END;
```

จากที่ได้อาศัยตัวแปลภาษาของ TURBO PASCAL ช่วยในการแปล สมการบูลลีนด้วยเทคนิคและวิธีการเช่นนี้จึงทำให้การพัฒนาโปรแกรม LOGCAD ไม่ต้องสร้างตัวแปลภาษาขึ้นใช้เองให้เป็นการยุ่งยาก ในขณะเดียวกันผู้ใช้สามารถที่จะสร้างฟังก์ชั่นการทำงานอย่างไรก็ได้ตามความต้องการ เพราะนอกจากจะมีฟังก์ชั่นพื้นฐานอย่าง AND OR NOT แล้วยังสามารถเขียนให้มีการตรวจสอบเงื่อนไขการทำงานได้อีกด้วย ดังนั้นการใช้งานจึงกล่าวได้ว่าสามารถทำได้ครอบคลุมแทนทุกฟังก์ชั่นการทำงาน ยกเว้นฟังก์ชั่นบางฟังก์ชั่นที่ต้องอาศัยการหน่วงเวลาเข้ามาเกี่ยวข้อง ซึ่งฟังก์ชั่นลักษณะเช่นนี้โปรแกรม LOGCAD ยังไม่ได้มีการพัฒนา ตัวอย่างเช่น ฟังก์ชั่นของโมโนสเตเบิ้ลมัลติไวเบเตอร์ (MONOSTABLE

MULTIVIBRATOR) เป็นต้น

2.2.2 LOGNODE

เป็นโปรแกรมที่ใช้สร้างไฟล์ข้อมูลโหนดจากการอ่านไฟล์ข้อมูลที่มีชื่อหลังจุดเป็น NLT ซึ่งกล่าวมาแล้วในข้อ 2.1.2

2.2.3 LOGNET

เป็นโปรแกรมที่ใช้ในการสร้างไฟล์ใหม่ให้เป็นยูนิตหรือกล่าวอีกลักษณะหนึ่งว่า เป็นการสร้างยูนิตระดับไฟล์ และรูปแบบของการสร้างไฟล์จะประกอบด้วยโปรแกรมย่อยหลาย ๆ ส่วน ในแต่ละส่วนจะได้ข้อมูลมาจากไฟล์ข้อมูลโนด กับ ไฟล์ข้อมูลแบบจำลอง รายละเอียดจะได้กล่าวในบทที่ 4

2.2.4 PRESIM

เป็น BATCH FILE ซึ่งมีหน้าที่ในการแปลภาษา จากตัวแปลภาษาของ TURBO PASCAL โดยที่ไม่ต้องเข้าไปยังตัวโปรแกรมอิดิเตอร์ของ TURBO PASCAL เลยเพราะอาศัยการแปลภาษาแบบ COMMAND LINE

2.2.5 DATATXRX UNIT

เป็นยูนิตที่ได้จากการสร้างในหัวข้อ 2.2.3 เพื่อที่จะให้โปรแกรมหลักสามารถเรียกโปรแกรมย่อยภายในยูนิตนี้ใช้งานได้ รายละเอียดจะได้กล่าวในบทที่ 4

สรุป ในบทที่ 2 นี้เป็นการกล่าวถึงขีดความสามารถสูงสุดของโปรแกรมว่าในแต่ละส่วนนั้นมีขีดความสามารถเท่าใดและอย่างใดบ้าง ตลอดจนบอกถึงลักษณะโครงสร้างหลัก ๆ ของโปรแกรม LOGCAD ว่าประกอบด้วยส่วนโปรแกรมการสร้างรูป (LOGCAD EDITOR) และ ส่วนจำลองการทำงานของวงจรตรรก (LOGCAD SIMULATOR) นอกจากนี้ยังประกอบไปด้วยส่วนประกอบย่อยอื่น ๆ อีกและในการสร้างผังภาพวงจรตรรกนอกจากจะใช้โปรแกรม LOGCAD แล้วยังสามารถใช้โปรแกรม ORCAD ได้อีกโปรแกรมหนึ่งผลสุดท้ายของโปรแกรมทั้งสองจะได้ไฟล์ข้อมูลลักษณะเดียวกันที่มีชื่อหลังจุดเป็น NDE (FILENAME.NDE) ซึ่งเป็นไฟล์ข้อมูลโหนดนั่นเอง.

บทที่ 3

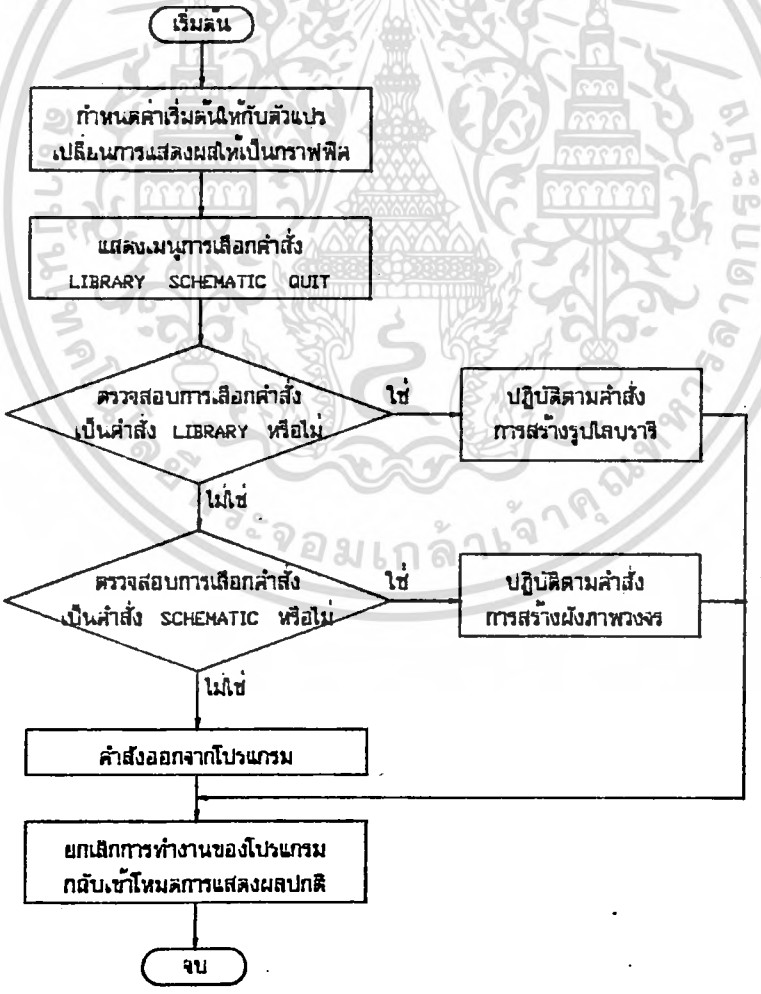
โปรแกรมการสร้างรูปอุปกรณ์ และ ผังภาพวงจร (LOGCAD EDITOR)

บทที่ 3 เป็นส่วนของโปรแกรม LOGCAD EDITOR สามารถแบ่งหัวข้อการอธิบายได้ 3 หัวข้อดังนี้

- โปรแกรมการสร้างรูปอุปกรณ์ไลบรารี
- โปรแกรมการสร้างผังภาพวงจรตรรก
- ลักษณะโครงสร้างข้อมูลการสร้างรูปอุปกรณ์ และ ผังภาพวงจร

ส่วนในการใช้งานของโปรแกรม LOGCAD EDITOR สามารถทำได้โดยการเรียกจากชื่อ DRAFT1 และการทำงานของโปรแกรมจะแบ่งออกได้เป็นส่วนสำคัญ 3 ส่วนคือ

- โปรแกรมการสร้างรูปอุปกรณ์ไลบรารี
- โปรแกรมการสร้างผังภาพวงจรตรรก
- ออกจากโปรแกรมการทำงาน

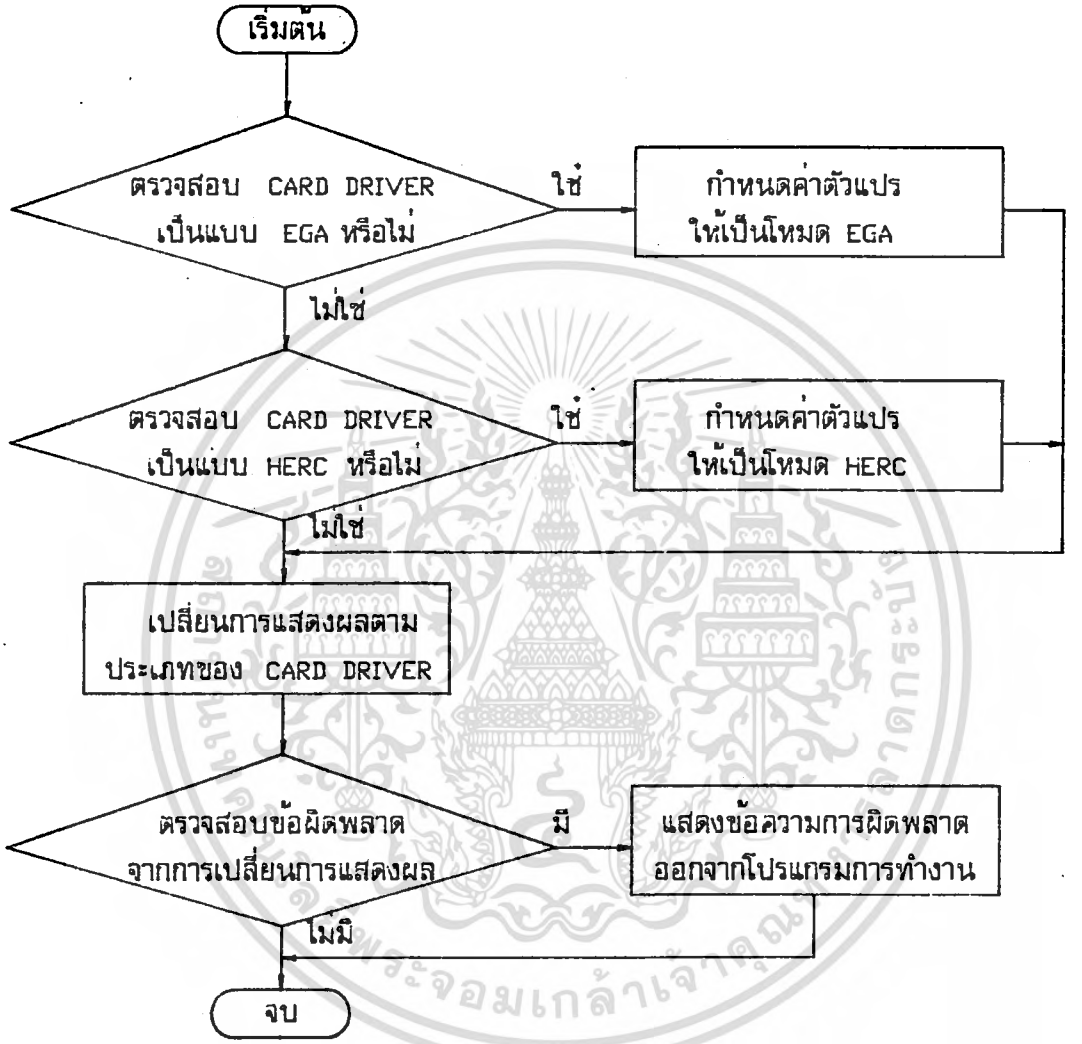


รูปที่ 3-1 แสดงแผนผังการทำงานของโปรแกรม DRAFT1

อธิบายการทำงานของโปรแกรม DRAFT1

1. กำหนดตัวแปรและจัดค่าเริ่มต้นให้กับตัวแปร เป็นขั้นตอนเริ่มต้นของโปรแกรม โดยการกำหนดตัวแปรร่วม (GLOBAL) ต่างๆ เพื่อให้ทุกส่วนของโปรแกรมย่อยสามารถเรียกใช้ได้ นอกจากนี้ยังกำหนดค่าเริ่มต้นให้กับตัวแปรก่อนจะมีการทำในขั้นตอน ต่อไป
2. กำหนดการแสดงผลให้อยู่ในแบบการแสดงผลกราฟฟิค โปรแกรม DRAFT1 ตลอดทั้งโปรแกรมจะทำงานอยู่ในโหมดกราฟฟิค ฉะนั้นก่อนเข้าสู่โปรแกรมจึงจำเป็นต้องมีการจัดให้อยู่ในโหมดกราฟฟิค โดยเริ่มจากการตรวจสอบฮาร์ดแวร์ว่าเป็นฮาร์ดแวร์ประเภทใด (EGA หรือ MONOCHROME) เพื่อที่จะได้กำหนดการแสดงผลให้เป็นไปตามกราฟฟิคประเภทนั้น ๆ
3. โปรแกรมจะแสดงเมนูของคำสั่งดังรูปที่ 3-19 เพื่อรอการเลือกจากผู้ใช้และการเลือกคำสั่งจะสมบูรณ์ได้ก็ต่อเมื่อมีการกดแป้นพิมพ์ RETURN
4. ขั้นตอนตรวจสอบการเลือกเมนู ในส่วนนี้จะแยกการตรวจสอบออกเป็นลำดับขั้นดังนี้
   - LIBRARY จะเป็นการเลือกไปยังส่วนของการสร้างรูปอุปกรณ์ไลบรารี
   - SCHEMATIC จะเป็นการเลือกไปยังส่วนการสร้างผังภาพวงจร
   - QUIT จะเป็นการสิ้นสุดโปรแกรม ยกเลิกตัวแปรที่ใช้ในโปรแกรมและกลับเข้าสู่การแสดงผลแบบปกติ

การจัดการ เปลี่ยนแปลงการแสดงผลให้อยู่ในแบบกราฟฟิคของโปรแกรม DRAFT1 โปรแกรมจะตรวจสอบ เองว่า จอภาพที่ใช้ในการแสดงผลขณะนั้น เป็นจอภาพชนิด EGA หรือ MONOCHROME มีแผนผังการทำงานดังนี้



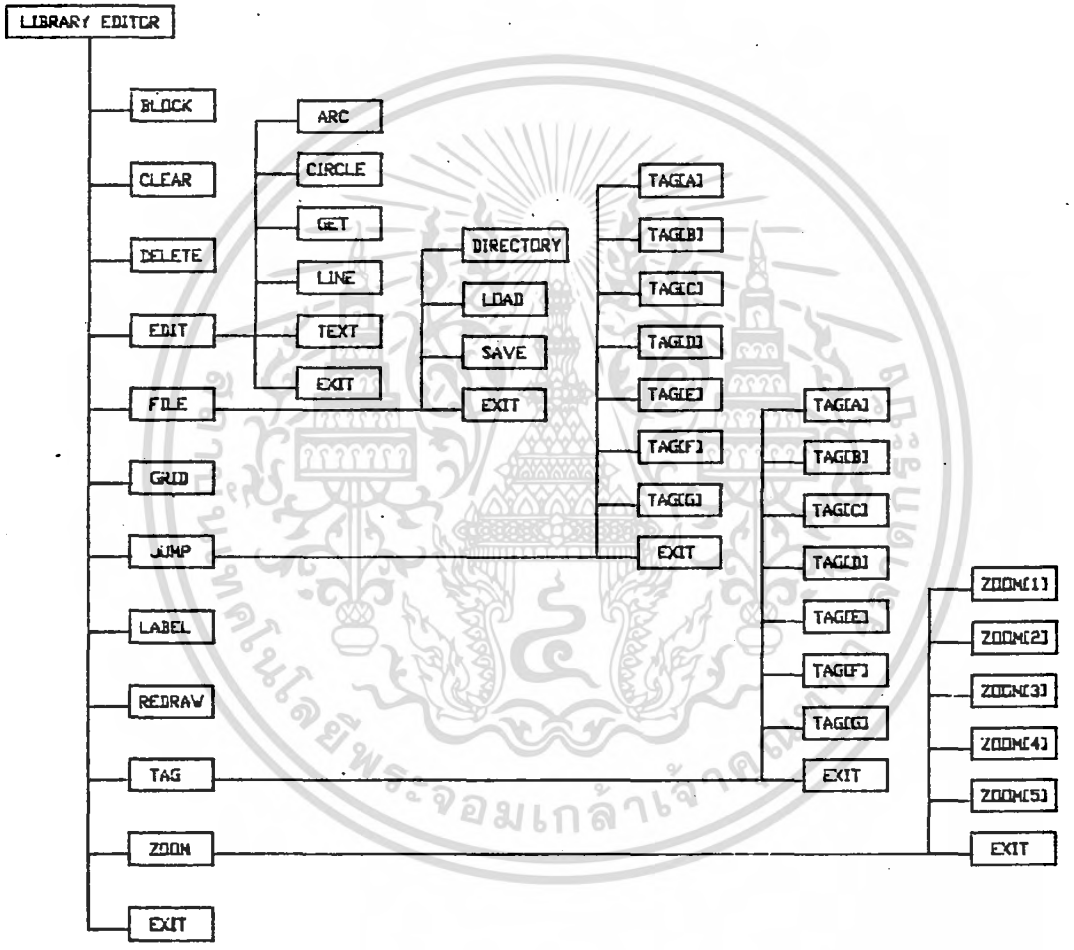
รูปที่ 3-2 แผนผังแสดงการทำงานโปรแกรมกำหนดการแสดงผล เป็นกราฟฟิค

อธิบายแผนผังแสดงการทำงานโปรแกรมกำหนดการแสดงผลเป็นกราฟฟิค

1. ตรวจสอบฮาร์ดแวร์ที่เป็นไดร์เวอร์กราฟฟิคของจอภาพ ด้วยซอฟต์แวร์ว่าใช้กับจอภาพประเภทใด โดยที่ผู้ใช้ไม่จำเป็นต้องควบคุมการแสดงผลของจอภาพแต่อย่างใด
2. ภายหลังจากการตรวจสอบแล้ว จะทำการกำหนดโหมดการแสดงผลโดยแบ่งตามชนิดของจอภาพได้ดังนี้
   - EGA จะกำหนดให้อยู่ในโหมดและจัดขนาดต่าง ๆ อยู่ในแบบของ EGAHI ความละเอียดจอภาพ 640 x 350 จุด
   - MONOCHROME จะกำหนดให้เป็นโหมดและจัดขนาดต่าง ๆ อยู่ในแบบ HER-MONOHI ความละเอียดจอภาพ 720 x 348 จุด
3. เปลี่ยนแปลงการแสดงผลให้เป็นกราฟฟิคตามชนิดของฮาร์ดแวร์ที่ได้ตรวจสอบมาแล้ว
4. ตรวจสอบข้อผิดพลาด ภายหลังจากการเปลี่ยนแปลงการแสดงผลเป็นกราฟฟิค ถ้าฮาร์ดแวร์ไม่อยู่ในขอบเขตที่กำหนด (EGA & MONOCHROME) โปรแกรมจะแสดงข้อผิดพลาดให้ผู้ใช้งานทราบ ในขณะเดียวกันก็จะออกจากโปรแกรม DRAFT1 กลับเข้าสู่การควบคุมของโปรแกรมจัดระบบ (OS)

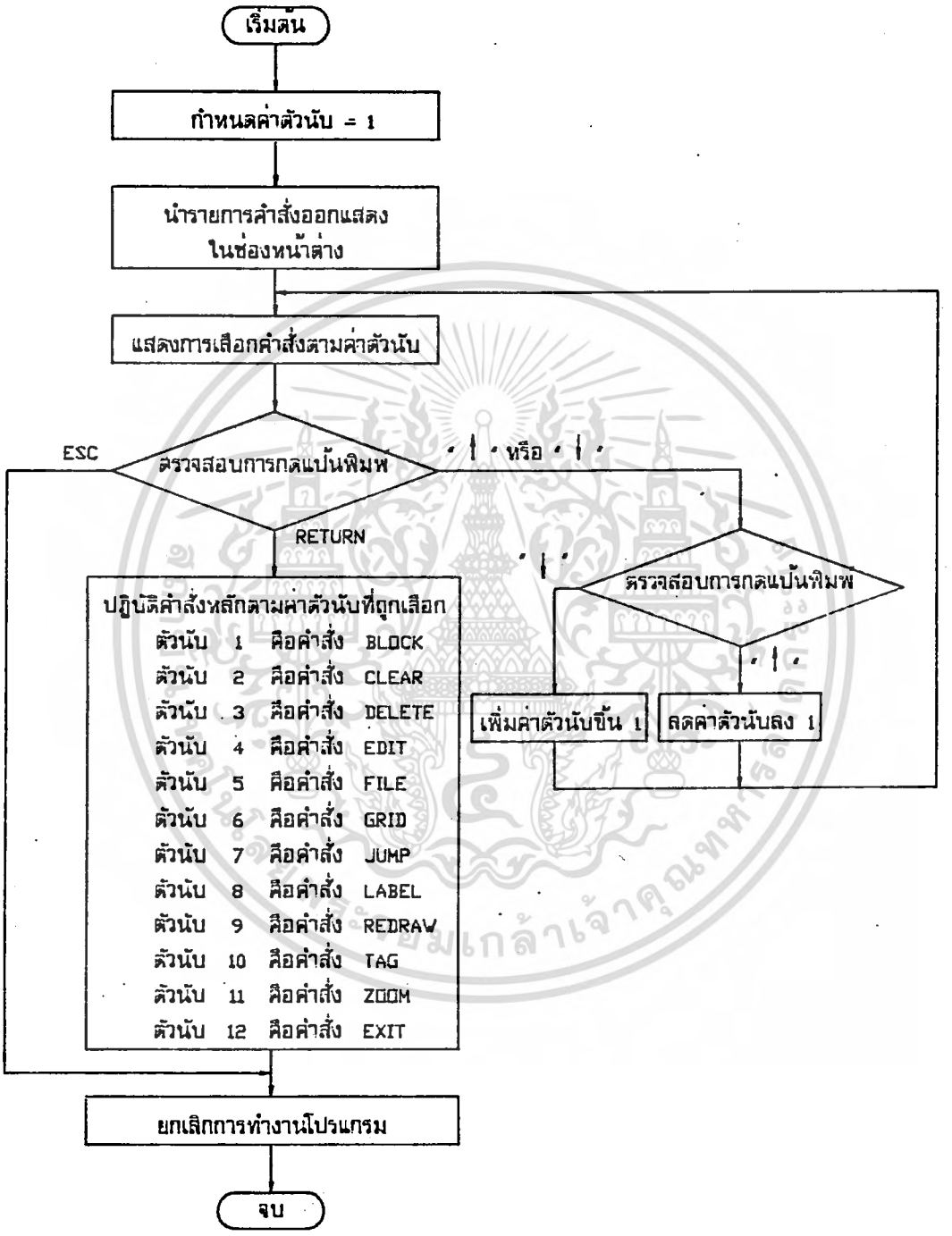
3.1 โปรแกรมการสร้างรูปอุปกรณ์ไลบรารี (LIBRARY EDITOR)

มีทำหน้าที่ในการสร้างไลบรารีของอุปกรณ์ประเภทต่าง ๆ ขึ้นมาใหม่หรือบางกรณีก็แก้ไขอุปกรณ์ไลบรารีที่สร้างไว้แล้วเหล่านี้ให้ถูกต้องและเหมาะสมมากกว่าเดิม ดังนั้นคำสั่งต่าง ๆ ที่ใช้ในการสร้าง หรือ แก้ไขรูปจึงต้องมีเพียงพอกับความต้องการ เพื่อจุดประสงค์ดังกล่าว คำสั่งเหล่านี้สามารถแสดงเป็นรากของคำสั่ง (ROOT COMMAND MENU) ดังรูปที่ 3-3



รูปที่ 3-3 แผนผังแสดงรากของคำสั่ง (ROOT COMMAND MENU) ในส่วน LIBRARY

โปรแกรมการ เลือกคำสั่งหลักจะสามารถ เลือกคำสั่งจากการใช้แป้นพิมพ์ลูกศรซึ่งมีลำดับการเลือกดังนี้



รูปที่ 3-4 แผนผังแสดงการทำงานของโปรแกรมการ เลือกคำสั่งหลัก

อธิบายแผนผังการทำงานของโปรแกรมการเลือกคำสั่งหลัก

การทำงานของโปรแกรมเลือกคำสั่งหลักนั้นสามารถเลือกได้ถึง 12 คำสั่งและในการเลือกคำสั่งต่าง ๆ ทำได้โดยกดแป้นพิมพ์ลูกศร ขึ้น หรือ ลง ซึ่งมีผลทำให้ตัวนับมีการเปลี่ยนแปลง จนกระทั่งมีการกดแป้นพิมพ์ RETURN ค่าของตัวนับสุดท้ายมีค่าเท่าใดนั้นหมายถึงการเลือกคำสั่งนั้น ๆ ขั้นตอนการทำงานของโปรแกรมมีดังนี้

1. นำรายการคำสั่งแสดงในช่องหน้าต่าง ดังรูปที่ 3-20 รายการคำสั่งมีดังนี้

   BLOCK CLEAR DELETE EDIT FILE GRID JUMP

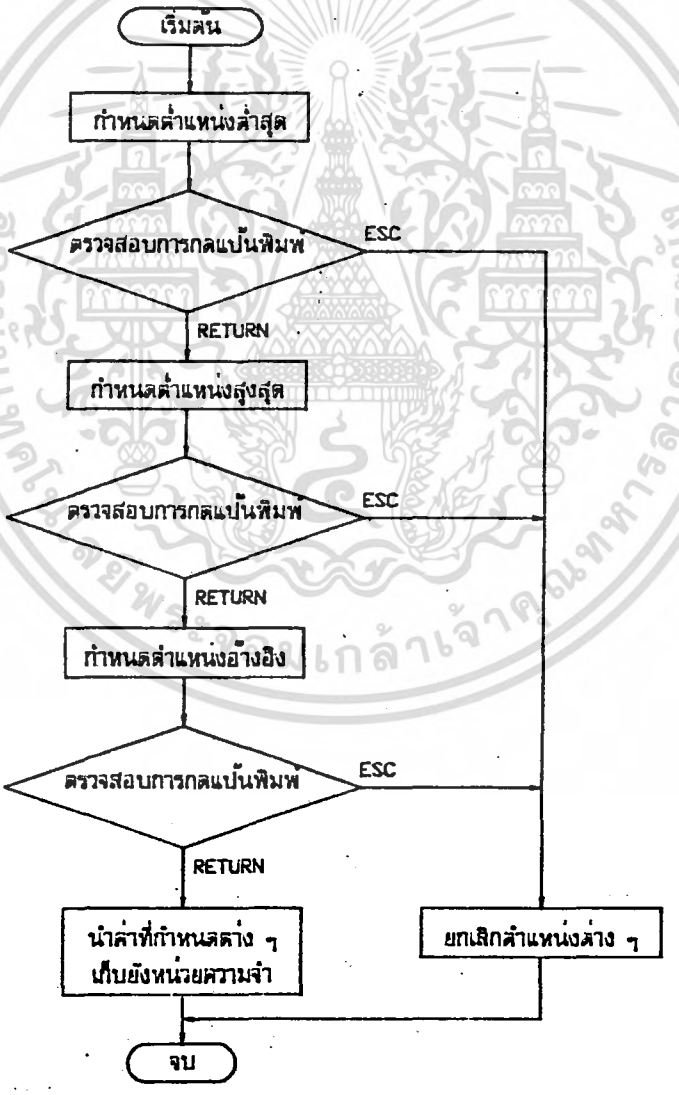
   LABEL REDRAW TAG ZOOM EXIT

2. ตรวจสอบการกดแป้นพิมพ์ ในการกดแป้นพิมพ์เพื่อให้โปรแกรมสามารถปฏิบัติตามความต้องการได้จะมีเพียง 4 แป้นพิมพ์เท่านั้น
   - แป้นพิมพ์ลูกศรขึ้น จะเป็นการลดค่าของตัวนับลงไป 1 และการลดค่านี้จะไม่ให้ตัวนับมีค่าน้อยไปกว่า 1 ซึ่งถือว่าเป็นค่าต่ำสุดของตัวนับ
   - แป้นพิมพ์ลูกศรลง จะเป็นการเพิ่มค่าของตัวนับขึ้นไป 1 และการเพิ่มค่าจะไม่ให้ตัวนับมีค่ามากไปกว่า 12 ซึ่งถือว่าเป็นค่าสูงสุดของตัวนับ
   - แป้นพิมพ์ RETURN จะมีผลทำให้ไปปฏิบัติคำสั่งตามค่าลำดับตัวนับ
   - แป้นพิมพ์ ESC ยกเลิกการทำงานโปรแกรมย่อย และ กลับสู่การทำงานโปรแกรม DRAFT1
3. ค่าของตัวนับ จะกำหนดให้ปฏิบัติคำสั่งต่าง ๆ ดังนี้
   - ค่าตัวนับ = 1 คือคำสั่ง LIBRARY/BLOCK
   - ค่าตัวนับ = 2 คือคำสั่ง LIBRARY/CLEAR
   - ค่าตัวนับ = 3 คือคำสั่ง LIBRARY/DELETE
   - ค่าตัวนับ = 4 คือคำสั่ง LIBRARY/EDIT
   - ค่าตัวนับ = 5 คือคำสั่ง LIBRARY/FILE
   - ค่าตัวนับ = 6 คือคำสั่ง LIBRARY/GRID
   - ค่าตัวนับ = 7 คือคำสั่ง LIBRARY/JUMP
   - ค่าตัวนับ = 8 คือคำสั่ง LIBRARY/LABEL
   - ค่าตัวนับ = 9 คือคำสั่ง LIBRARY/REDRAW
   - ค่าตัวนับ = 10 คือคำสั่ง LIBRARY/TAG
   - ค่าตัวนับ = 11 คือคำสั่ง LIBRARY/ZOOM
   - ค่าตัวนับ = 12 คือคำสั่ง LIBRARY/EXIT

เมื่อปฏิบัติตามคำสั่งต่าง ๆ ยกเว้นคำสั่ง EXIT แล้วโปรแกรมจะกลับไปเริ่มที่รายการคำสั่งหลัก (LIBRARY EDITOR) ใหม่ แต่ถ้าเป็นคำสั่ง EXIT จะเป็นการยกเลิกโปรแกรมการสร้างรูปอุปกรณ์ LIBRARY และกลับเข้าสู่การทำงานของโปรแกรม DRAFT1

3.1.1 โปรแกรมการทำคำสั่งกำหนดขอบเขตของรูปไลบรารี (LIBRARY/BLOCK COMMAND)

จะเป็นส่วนของการกำหนดขอบเขตให้กับรูปอุปกรณ์ไลบรารี ซึ่งข้อมูลรูปที่จะถูกนำเก็บลงไฟล์ไลบรารีนั้นจะต้องอยู่ภายในขอบเขตที่กำหนดนี้เท่านั้น ดังนั้นการสร้างรูปอุปกรณ์ไลบรารีแต่ละครั้งจึงต้องมีการกำหนดขอบเขตให้กับรูป การทำงานของโปรแกรมมีขั้นตอนตามแผนผังดังรูปที่ 3-5



รูปที่ 3-5 แสดงแผนผังการทำงานของโปรแกรมกำหนดขอบเขตรูปไลบรารี

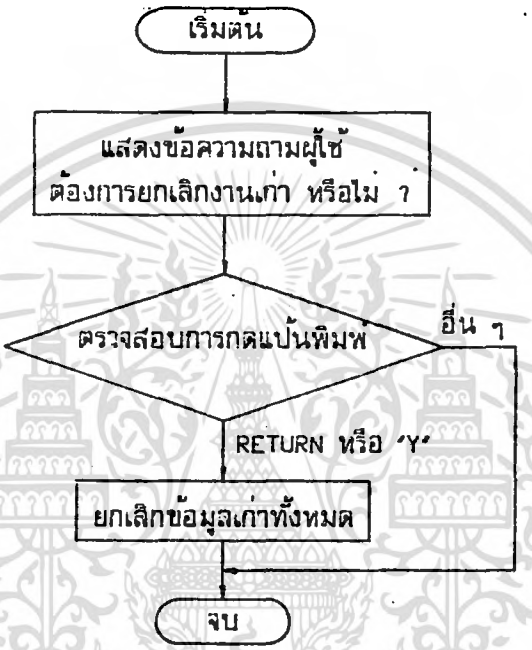
อธิบายแผนผังการทำงานของโปรแกรมกำหนดขอบเขตรูปไลบรารี

ในการกำหนดตำแหน่งทำได้โดยการจุดเครื่องหมาย ฌ. ตำแหน่งที่ต้องการ 2 จุดในลักษณะมุมทะแยง และภายหลังจากที่กำหนดจุดเหล่านี้แล้วลำดับต่อไปเป็นการตรวจสอบแป้นพิมพ์เฉพาะ ESC และ RETURN เท่านั้น กรณีที่เป็นแป้นพิมพ์ RETURN โปรแกรมจะรับรู้การกำหนดจุดแต่ถ้าเป็นแป้นพิมพ์ ESC โปรแกรมจะยกเลิกจุดทั้งหมดที่กำหนดและกลับสู่การทำงานของโปรแกรม LIBRARY EDITOR

1. การเลือกตำแหน่งต่ำสุดโดยเลื่อน CURSOR ไปยังตำแหน่งที่ต้องการและกดแป้นพิมพ์ RETURN โปรแกรมจะเก็บค่าจุดนี้ไว้เป็นจุดบอกตำแหน่งขอบเขตต่ำสุด
2. การเลือกตำแหน่งสูงสุดโดยเลื่อน CURSOR ไปยังตำแหน่งที่ต้องการและกดแป้นพิมพ์ RETURN โปรแกรมจะเก็บค่าจุดนี้ไว้เป็นจุดบอกตำแหน่งขอบเขตสูงสุด
3. การเลือกตำแหน่งจุดอ้างอิง โดยเลื่อน CURSOR ไปยังตำแหน่งจุดที่ต้องการและกดแป้นพิมพ์ RETURN โปรแกรมก็จะเก็บค่าจุดนี้ไว้เป็นจุดอ้างอิง

3.1.2 โปรแกรมยกเลิกงานไลบรารีที่สร้าง (LIBRARY/CLEAR COMMAND)

เป็นส่วนของการลบภาพอุปกรณ์เก่าที่แสดงอยู่บนจอภาพออกไป เพื่อเริ่มต้นการสร้างอุปกรณ์ใหม่ โดยโปรแกรมจะมีการถามกลับมาว่าต้องการลบจริงหรือไม่ เพื่อให้ผู้ใช้แน่ใจว่าต้องการลบหรือยกเลิกจริง ๆ เพราะบางครั้งผู้ใช้ไม่มีความประสงค์ที่จะเลือกคำสั่งนี้ การทำงานของโปรแกรมมีลำดับขั้นตามแผนผังรูปที่ 3-6



รูปที่ 3-6 แสดงแผนผังการทำงานของโปรแกรมยกเลิกการสร้างรูปอุปกรณ์ไลบรารี

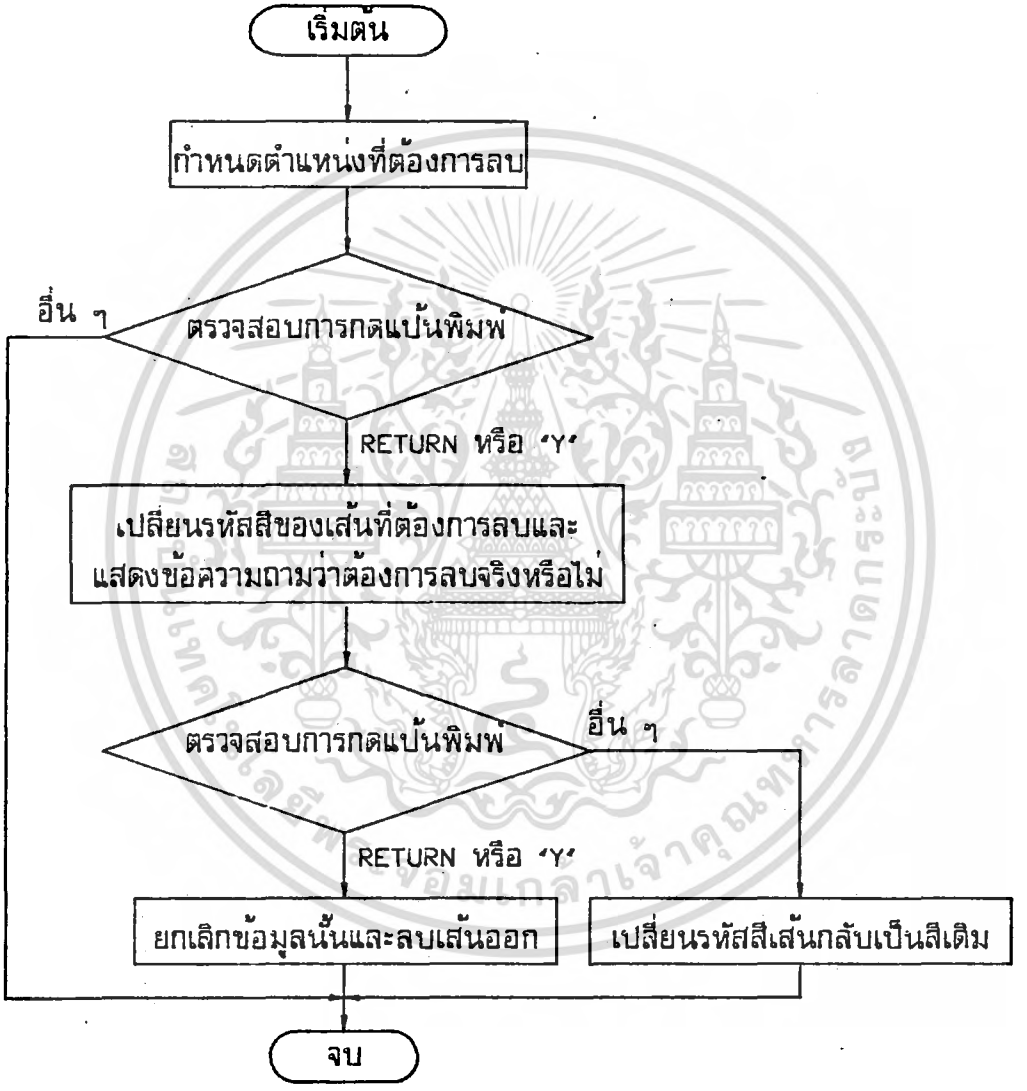
อธิบายแผนผังการทำงานของโปรแกรมยกเลิกการสร้างรูปอุปกรณ์ไลบรารีมีลำดับขั้นดังนี้

1. แสดงข้อความถามกลับมายังผู้ใช้และรอรับการกดแป้นพิมพ์ ในขั้นตอนนี้จะแสดงข้อความถามผู้ใช้ว่าจะลบรูปอุปกรณ์ไลบรารีจริง หรือไม่ และรอรับการกดแป้นพิมพ์จากผู้ใช้
2. ตรวจสอบแป้นพิมพ์ โดยการตรวจสอบว่าผู้ใช้กดแป้นพิมพ์ใดถ้าเป็น RETURN หรือ 'Y' โปรแกรมก็จะไปทำงานในส่วนยกเลิกข้อมูลรูปอุปกรณ์ไลบรารี แต่ถ้าหากไม่ใช่แล้ว ก็จะออกจากโปรแกรมส่วนนี้กลับสู่การทำงานของโปรแกรม LIBRARY EDITOR
3. ยกเลิกข้อมูลรูปอุปกรณ์ไลบรารี เป็นการยกเลิกข้อมูลที่เก็บในหน่วยความจำต่าง ๆ ที่เกี่ยวกับรูปทั้งหมดของโปรแกรม เพื่อเตรียมสำหรับการสร้างขึ้นใหม่

3.1.3 โปรแกรมลบ เฉพาะบางส่วนของรูปอุปกรณ์ไลบรารี

(LIBRARY/DELETE COMMAND)

เป็นโปรแกรมลบ เฉพาะบางส่วนของรูปอุปกรณ์ที่กำลังทำอยู่ขณะนั้น โดยจะทำการลบ เฉพาะส่วนที่ CURSOR ชี้อยู่ตลอดทั้งแนวที่มีการสร้างในแต่ละครั้ง เช่น เส้นตรง วงกลม ข้อความ เป็นต้น การทำงานของโปรแกรมมีลำดับขั้นตามแผนผังดังรูปที่ 3-7



รูปที่ 3-7 แสดงแผนผังการทำงานโปรแกรมลบ เฉพาะบางส่วนของรูป

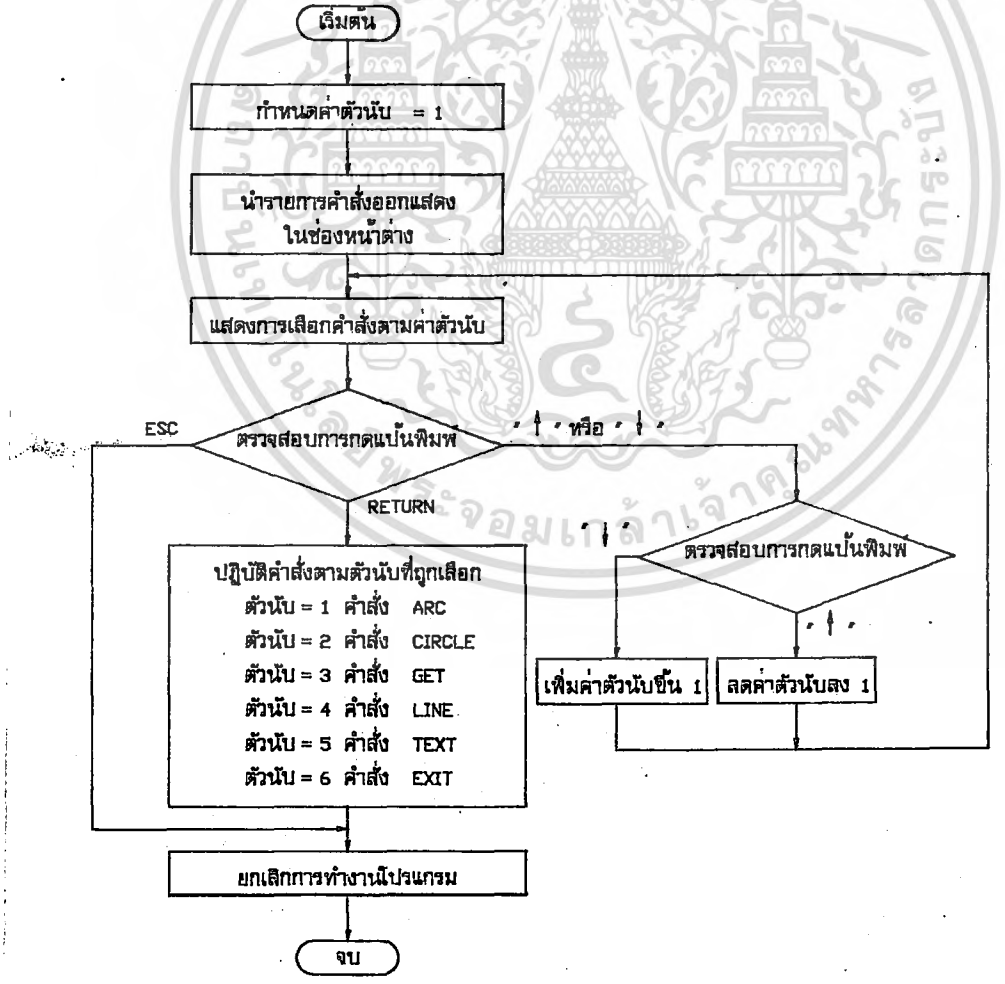
อธิบายแผนผังการทำงานของโปรแกรมลบ เฉพาะบางส่วนของรูปมีลำดับขั้นดังนี้

1. เลือกตำแหน่งของส่วนที่จะลบ สำหรับการเลือกตำแหน่งนี้ต้องให้ CURSOR ชี้ที่ปลาย จุดใดจุดหนึ่งของเส้นในกรณีที่เป็นเส้นตรงหรือเส้นโค้ง และชี้ภายในบริเวณพื้นที่ในกรณีที่เป็นวงกลมหรือข้อความ
2. ตรวจสอบแป้นพิมพ์เพื่อเลือกตำแหน่งของส่วนที่ต้องการลบ จะตรวจสอบเฉพาะแป้นพิมพ์ RETURN เพื่อที่จะทำตามโปรแกรมต่อไป ถ้าเป็นแป้นพิมพ์อื่น ๆ จะยกเลิกการทำงานของโปรแกรม และ กลับสู่การทำงานของโปรแกรม LIBRARY EDITOR
3. เปลี่ยนรหัสสีของเส้นที่ต้องการลบ และแสดงข้อความถามกลับว่าจะลบจริงหรือไม่ เพื่อความแน่ใจ
4. รอรับการกดแป้นพิมพ์จะหยุดรอจนกว่า ผู้ใช้ทำการกดแป้นพิมพ์ใดแป้นพิมพ์หนึ่ง
5. ตรวจสอบแป้นพิมพ์ ถ้าเป็นแป้นพิมพ์ RETURN หรือ 'Y' ก็จะลบข้อมูลส่วนนั้นออก แต่ถ้าไม่ใช่ก็จะเปลี่ยนรหัสสีของเส้นกลับเป็นสีเดิม และยกเลิกการทำงานของโปรแกรมย่อย กลับสู่การทำงานของโปรแกรม LIBRARY EDITOR

3.1.4 โปรแกรมการสร้างรูปอุปกรณ์ไลบรารี (LIBRARY/EDIT COMMAND)

การทำงานของโปรแกรมส่วนนี้จะ เป็นการ เลือกคำสั่งสำหรับสร้างรูปอุปกรณ์ไลบรารี คำสั่งต่าง ๆ เหล่านี้ เป็นคำสั่งพื้นฐานของการสร้างรูป ฉะนั้นในการสร้างรูปหนึ่ง ๆ ผู้ใช้สามารถ เลือกคำสั่งการสร้างรูปมาประกอบกัน เป็นรูปที่ต้องการได้ ส่วนของการสร้างรูปจะประกอบด้วยคำสั่งย่อยดังนี้

- คำสั่งการสร้าง เส้นโค้ง (LIBRARY/EDIT/ARC COMMAND)
- คำสั่งการสร้างวงกลม (LIBRARY/EDIT/CIRCLE COMMAND)
- คำสั่งการ เลือกไลบรารี (LIBRARY/EDIT/GET COMMAND)
- คำสั่งการสร้าง เส้นตรง (LIBRARY/EDIT/LINE COMMAND)
- คำสั่งการสร้างข้อความ (LIBRARY/EDIT/TEXT COMMAND)
- คำสั่งออกจากโปรแกรมย่อย (LIBRARY/EDIT/EXIT COMMAND)



รูปที่ 3-8 แสดงแผนผังการทำงานของโปรแกรมสร้างรูปอุปกรณ์

อธิบายแผนผังการทำงานของโปรแกรมสร้างรูปอุปกรณ์

การทำงานของโปรแกรมการสร้างรูปอุปกรณ์ เลือกรายการคำสั่งได้ดังรูป 3-21 รายการจะมีการเลือกเช่นเดียวกับข้อ 3-1 ที่กล่าวถึงแผนผังการทำงานของโปรแกรมเลือกคำสั่งหลัก ผลการเลือกคำสั่งจะขึ้นอยู่กับค่าลำดับตัวนับดังนี้

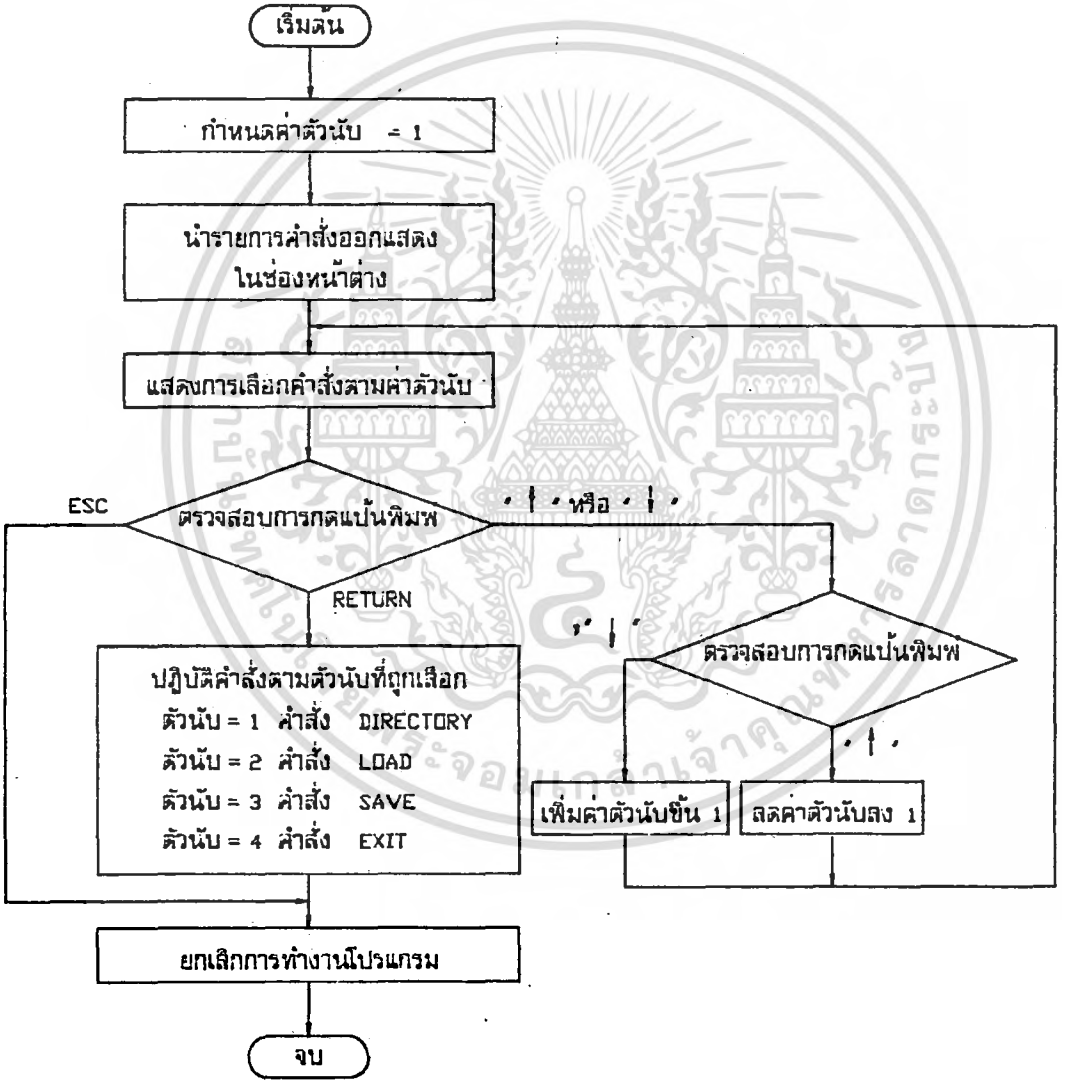
1. ARC จะเป็นการสร้างเส้นโค้ง โดยเริ่มต้นจากการกำหนดจุดศูนย์กลาง รัศมีของเส้นโค้ง มุมเริ่มต้น และมุมสุดท้าย ตามลำดับ
2. CIRCLE จะเป็นการสร้างวงกลม โดยเริ่มจากการกำหนดจุดศูนย์กลางและรัศมีของวงกลมตามลำดับ
3. GET จะเป็นการเลือกไฟล์ข้อมูลไลบรารีจากแผ่นหน่วยความจำมาเปลี่ยนแปลงแก้ไข
4. LINE จะเป็นการสร้างเส้นตรง โดยเริ่มจากการกำหนดจุดเริ่มต้นและจุดสุดท้ายตามลำดับ
5. TEXT จะเป็นการสร้างข้อมูลแบบข้อความ (TEXT) และกำหนดตำแหน่งที่ต้องการแสดงผลตามลำดับ
6. EXIT ออกจากโปรแกรมย่อย

เมื่อผ่านขั้นตอนทั้งหมดนี้แล้วก็จะสิ้นสุดโปรแกรมและกลับสู่การทำงานของโปรแกรม LIBRARY EDITOR

3.1.5 โปรแกรมบริการไฟล์ไลบรารี (LIBRARY/FILE COMMAND)

เป็นโปรแกรมทำงานเกี่ยวกับไฟล์ไลบรารี คือ การแสดงไดเรคทอรี่ การอ่านและเขียนข้อมูลกับแผ่นหน่วยความจำ ประกอบด้วยคำสั่งต่าง ๆ ดังนี้

- คำสั่งการแสดงไดเรคทอรี่ (LIBRARY/FILE/DIRECTORY COMMAND)
- คำสั่งการอ่านไฟล์ข้อมูล (LIBRARY/FILE/LOAD COMMAND)
- คำสั่งการเก็บไฟล์ข้อมูล (LIBRARY/FILE/SAVE COMMAND)
- คำสั่งออกจากโปรแกรมย่อย (LIBRARY/FILE/EXIT COMMAND)



รูปที่ 3-9 แสดงแผนผังการทำงานของโปรแกรมบริการไฟล์ไลบรารี

อธิบายแผนผังการทำงานของโปรแกรมบริการไฟล์ไลบรารี

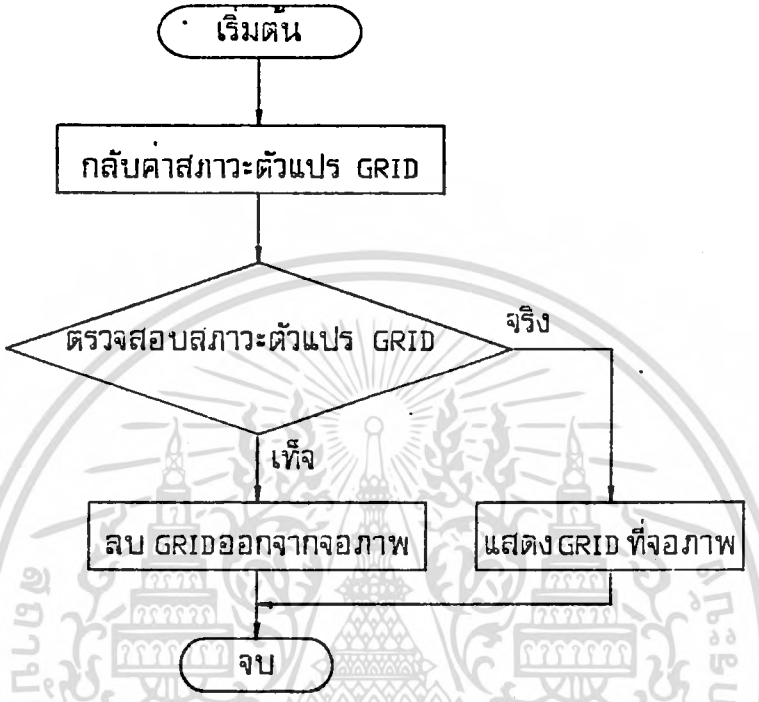
การทำงานของโปรแกรมการสร้างรูปอุปกรณ์ เลือกรายการคำสั่งได้ดังรูป 3-22 รายการจะมีการเลือกเช่นเดียวกับข้อ 3-1 ที่กล่าวถึงแผนผังการทำงานของโปรแกรมเลือกคำสั่งหลัก ผลการเลือกคำสั่งจะขึ้นอยู่กับค่าลำดับตัวนับดังนี้

1. DIRECTORY จะทำโปรแกรมแสดงรายการชื่อไฟล์ไลบรารีที่เก็บในแผ่นหน่วยความจำ
2. LOAD จะทำโปรแกรมนำข้อมูลจากไฟล์ข้อมูลไลบรารีเก็บยังหน่วยความจำในขณะเดียวกันก็แสดงรูปอุปกรณ์ไลบรารีนั้น ๆ ที่จอภาพ
3. SAVE จะทำโปรแกรมเก็บข้อมูลรูปอุปกรณ์ไลบรารีที่สร้างลงในแผ่นหน่วยความจำ
4. EXIT ออกจากโปรแกรมย่อย

เมื่อผ่านขั้นตอนทั้งหมดนี้แล้วก็จะสิ้นสุดโปรแกรมและกลับสู่การทำงานของโปรแกรม LIBRARY EDITOR

3.1.6 โปรแกรมแสดงกริด (LIBRARY/GRID COMMAND)

โดยที่การแสดงกริดจะเป็นแบบ TOGGLE ถ้ามีการเลือกคำสั่งนี้อีกครั้ง ดังรูปที่ 3-10 การทำงานของโปรแกรมมีลำดับขั้นดังนี้



รูปที่ 3-10 แสดงแผนผังการทำงานของโปรแกรมการแสดงกริด

อธิบายแผนผังการทำงานของโปรแกรมการแสดงกริดมีลำดับขั้นดังนี้

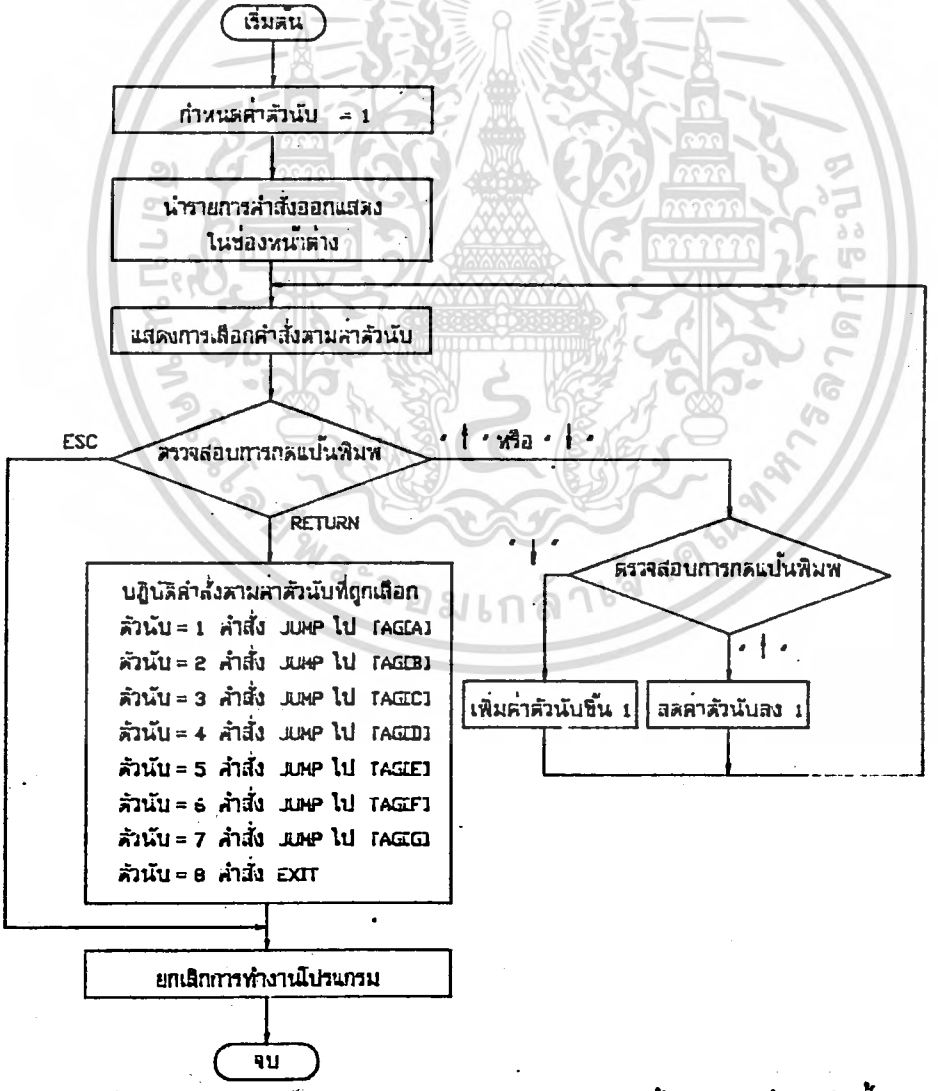
1. กลับค่าสภาวะตัวแปร GRID เพื่อที่จะทำการเปลี่ยนค่าสภาวะของตัวแปรนี้ให้เป็นตรงกันข้ามกับค่าสภาวะเดิม
2. ตรวจสอบค่าสภาวะตัวแปร GRID
   - ถ้าค่าตัวแปร GRID เป็นจริง จะทำการแสดงจุดกริดที่จอภาพ
   - ถ้าค่าตัวแปร GRID เป็นเท็จ จะทำการลบจุดกริดออกจากจอภาพ

หลังจากปฏิบัติตามลำดับขั้นเหล่านี้แล้ว จะยกเลิกการทำงานโปรแกรมวาดกริดและกลับสู่การทำงานของโปรแกรม LIBRARY EDITOR

3.1.7 โปรแกรมย้ายตำแหน่งจุดตัวชี้ (LIBRARY/JUMP)

เป็นโปรแกรมย้ายตำแหน่งของตัวชี้ (CURSOR) ไปยังตำแหน่งที่ได้กำหนดโดยคำสั่ง TAG จะประกอบด้วยคำสั่งในส่วนของโปรแกรมย่อยดังนี้

- คำสั่งย้ายตัวชี้ไปตำแหน่ง A (JUMP/TAG[A] COMMAND)
- คำสั่งย้ายตัวชี้ไปตำแหน่ง B (JUMP/TAG[B] COMMAND)
- คำสั่งย้ายตัวชี้ไปตำแหน่ง C (JUMP/TAG[C] COMMAND)
- คำสั่งย้ายตัวชี้ไปตำแหน่ง D (JUMP/TAG[D] COMMAND)
- คำสั่งย้ายตัวชี้ไปตำแหน่ง E (JUMP/TAG[E] COMMAND)
- คำสั่งย้ายตัวชี้ไปตำแหน่ง F (JUMP/TAG[F] COMMAND)
- คำสั่งออกจากโปรแกรมย่อย (JUMP/EXIT COMMAND)



รูปที่ 3-11 แสดงแผนผังการทำงานของโปรแกรมย้ายตำแหน่งจุดตัวชี้

อธิบายแผนผังการทำงานของโปรแกรมย้ายตำแหน่งจุด

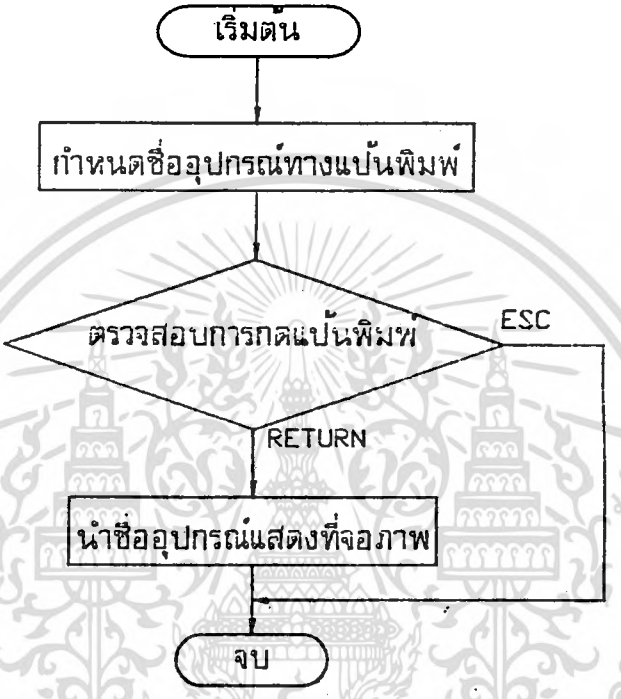
การทำงานของโปรแกรมการสร้างรูปอุปกรณ์ เลือกรายการคำสั่งได้ดังรูป 3-23 รายการจะมีการเลือกเช่นเดียวกับข้อ 3-1 ที่กล่าวถึงแผนผังการทำงานของโปรแกรมเลือกคำสั่งหลัก ผลการเลือกคำสั่งจะขึ้นอยู่กับค่าลำดับตัวนับดังนี้

1. TAG[A] จะเป็นการย้ายตัวชี้ไปยังตำแหน่ง (XA,YA)
2. TAG[B] จะเป็นการย้ายตัวชี้ไปยังตำแหน่ง (XB,YB)
3. TAG[C] จะเป็นการย้ายตัวชี้ไปยังตำแหน่ง (XC,YC)
4. TAG[D] จะเป็นการย้ายตัวชี้ไปยังตำแหน่ง (XD,YD)
5. TAG[E] จะเป็นการย้ายตัวชี้ไปยังตำแหน่ง (XE,YE)
6. TAG[F] จะเป็นการย้ายตัวชี้ไปยังตำแหน่ง (XF,YF)
7. TAG[G] จะเป็นการย้ายตัวชี้ไปยังตำแหน่ง (XG,YG)
8. EXIT ออกจากโปรแกรมย่อย

เมื่อผ่านขั้นตอนทั้งหมดนี้แล้วก็จะสิ้นสุดโปรแกรมและกลับสู่การทำงานของโปรแกรม LIBRARY EDITOR

3.1.8 โปรแกรมกำหนดชื่ออุปกรณ์ไลบรารี (LIBRARY/LABEL COMMAND)

โปรแกรมกำหนดชื่อของอุปกรณ์ไลบรารี โดยชื่อที่กำหนดให้นี้จะมีความยาวได้ไม่เกิน 15 ตัวอักษร และไม่จำเป็นต้องเหมือนกับชื่อของไฟล์ที่จะเก็บเป็นข้อมูลรูปอุปกรณ์ การทำงานของโปรแกรมมีลำดับขั้นดังนี้



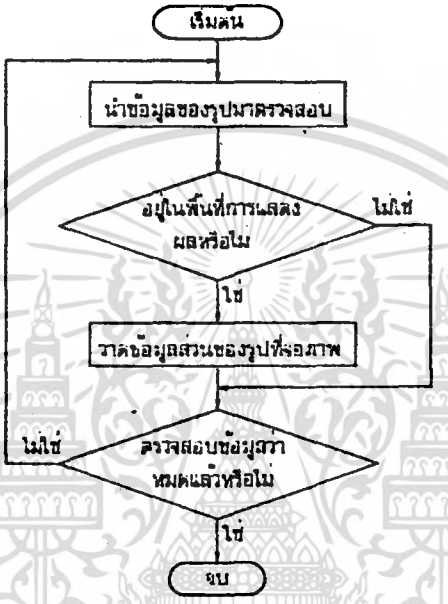
รูปที่ 3-12 แสดงแผนผังการทำงานของโปรแกรมกำหนดชื่ออุปกรณ์ไลบรารี

อธิบายการทำงานของโปรแกรมกำหนดชื่ออุปกรณ์ไลบรารี

1. กำหนดชื่ออุปกรณ์ โดยที่โปรแกรมจะเริ่มจากการแสดงข้อความถาม เพื่อให้ผู้ใช้กำหนดชื่ออุปกรณ์ตามความต้องการ และชื่อที่กำหนดนี้จะต้องมีความยาวไม่เกิน 15 ตัวอักษร
2. ตรวจสอบการกดแป้นพิมพ์ ถ้าผลการตรวจสอบปรากฏว่าเป็นแป้นพิมพ์
   - RETURN โปรแกรมจะนำชื่อที่กำหนดไปเป็นชื่อของอุปกรณ์ไลบรารี
   - ESC โปรแกรมจะยกเลิกการกำหนดชื่อ และ จะยึดถือชื่อเดิมถ้ามีการกำหนดมาแล้วครั้งหนึ่ง
3. ยกเลิกการทำงานของโปรแกรมและกลับยังส่วนของ LIBRARY EDITOR

3.1.9 โปรแกรมวาดรูปไลบรารีซ้ำของเดิม (LIBRARY/REDRAW COMMAND)

จะเป็นการวาดรูปอุปกรณ์ไลบรารีที่ทำการสร้างขณะนั้นใหม่อีกครั้งยังจอภาพ ทั้งนี้ในการสร้างรูปอาจทำให้รูปภาพดูไม่สมบูรณ์เนื่องมาจากคำสั่งบางคำสั่ง เช่นในการทำคำสั่ง DELETE จะทำให้จุดบางจุดขาดหายไป เหตุที่เป็นเช่นนี้เพราะเป็นธรรมชาติของการแสดงผลแบบกราฟฟิค ดังนั้นผู้ใช้อาจต้องการวาดรูปที่สร้างขณะนั้นขึ้นใหม่อีกครั้ง



รูปที่ 3-13 แสดงแผนผังการทำงานของโปรแกรมวาดรูปไลบรารีซ้ำของเดิม

อธิบายแผนผังการทำงานของโปรแกรมวาดรูปไลบรารีซ้ำของเดิมมีลำดับขั้นดังนี้

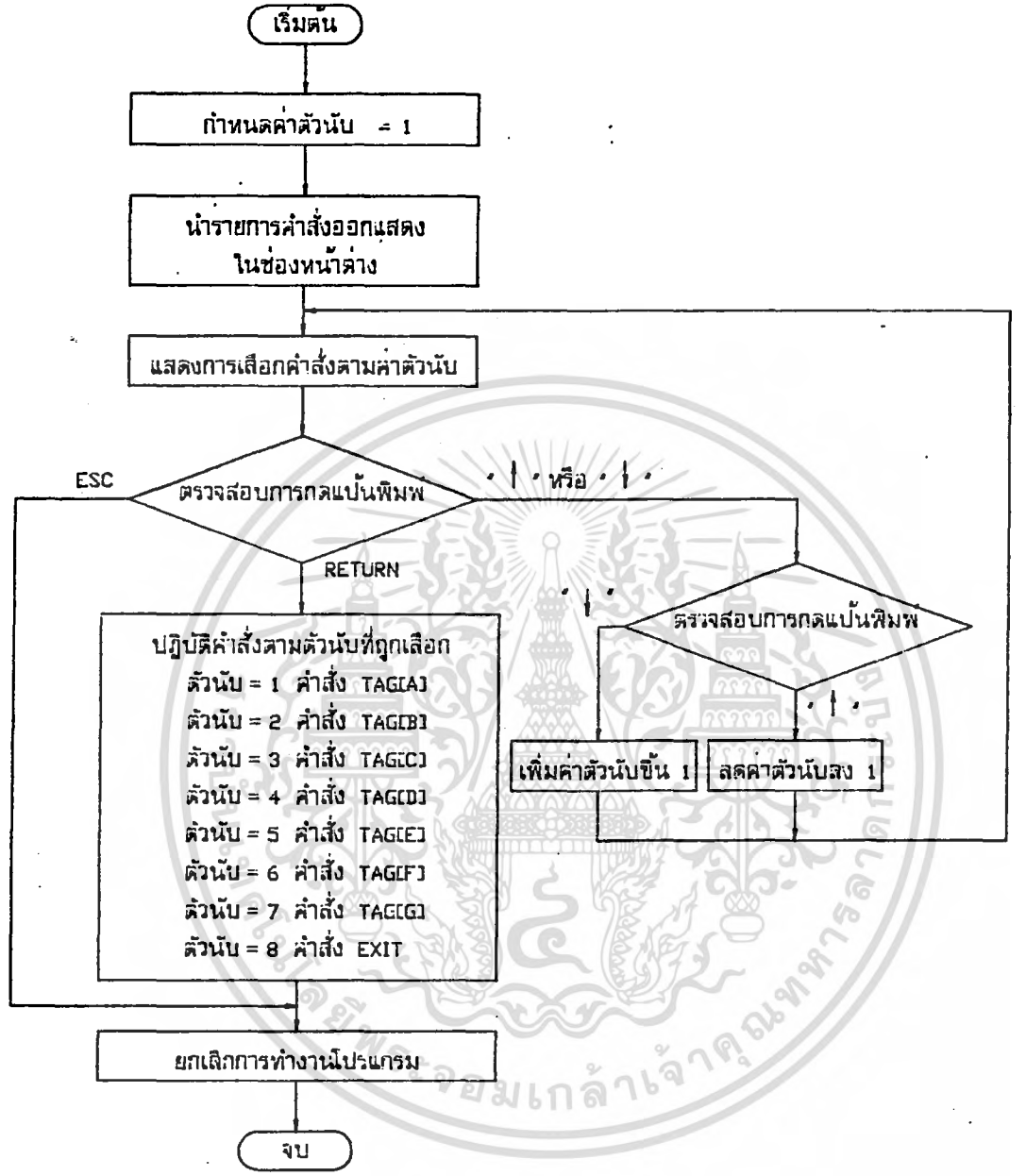
1. ตรวจสอบข้อมูลของรูปภาพ โดยเริ่มจากข้อมูลแรกจนกระทั่งถึงข้อมูลสุดท้ายว่าเป็นข้อมูลที่อยู่ในพื้นที่หน้าจอภาพที่แสดงนี้หรือไม่ ถ้าการตรวจสอบปรากฏว่าอยู่ในพื้นที่บนจอภาพจริง ข้อมูลเหล่านั้นก็จะถูกนำออกแสดงที่หน้าจอภาพ
2. ในกรณีที่เป็นข้อมูลสุดท้ายก็จะยกเลิกการทำงานของโปรแกรมส่วนนี้และกลับไปสู่การทำงานของโปรแกรม LIBRARY EDITOR

3.1.10 โปรแกรมกำหนดตำแหน่งจุดการย้าย (LIBRARY/TAG COMMAND)

เป็นโปรแกรม เก็บค่าตำแหน่งปัจจุบันของตัวชี้ (CURSOR) ลงสู่หน่วยความจำ เพื่อจะได้ทำการย้ายกลับมาตำแหน่ง เดิมโดยคำสั่ง JUMP ที่ได้อธิบายมาแล้วในข้อ 3.1.7

คำสั่งในส่วนของโปรแกรมย่อยจะประกอบด้วยคำสั่งต่าง ๆ ดังนี้

- คำสั่งกำหนดตำแหน่งตัวชี้ที่ A (LIBRARY/TAG/TAG[A] COMMAND)
- คำสั่งกำหนดตำแหน่งตัวชี้ที่ B (LIBRARY/TAG/TAG[B] COMMAND)
- คำสั่งกำหนดตำแหน่งตัวชี้ที่ C (LIBRARY/TAG/TAG[C] COMMAND)
- คำสั่งกำหนดตำแหน่งตัวชี้ที่ D (LIBRARY/TAG/TAG[D] COMMAND)
- คำสั่งกำหนดตำแหน่งตัวชี้ที่ E (LIBRARY/TAG/TAG[E] COMMAND)
- คำสั่งกำหนดตำแหน่งตัวชี้ที่ F (LIBRARY/TAG/TAG[F] COMMAND)
- คำสั่งกำหนดตำแหน่งตัวชี้ที่ G (LIBRARY/TAG/TAG[G] COMMAND)
- คำสั่งออกจากโปรแกรมย่อย (LIBRARY/TAG/EXIT COMMAND)



รูปที่ 3-14 แผนผังแสดงการทำงานของโปรแกรมกำหนดตำแหน่งจุดการย้ายของตัวชี้

อธิบายแผนผังการทำงานของโปรแกรมกำหนดตำแหน่งจุดการย้ายของตัวชี้

การทำงานของโปรแกรมในส่วนของการเลือกคำสั่ง ดังรูปที่ 3-24 เช่นเดียวกับข้อ 3.1 ผลของการเลือกคำสั่งจะขึ้นอยู่กับค่าของตัวนับดังนี้

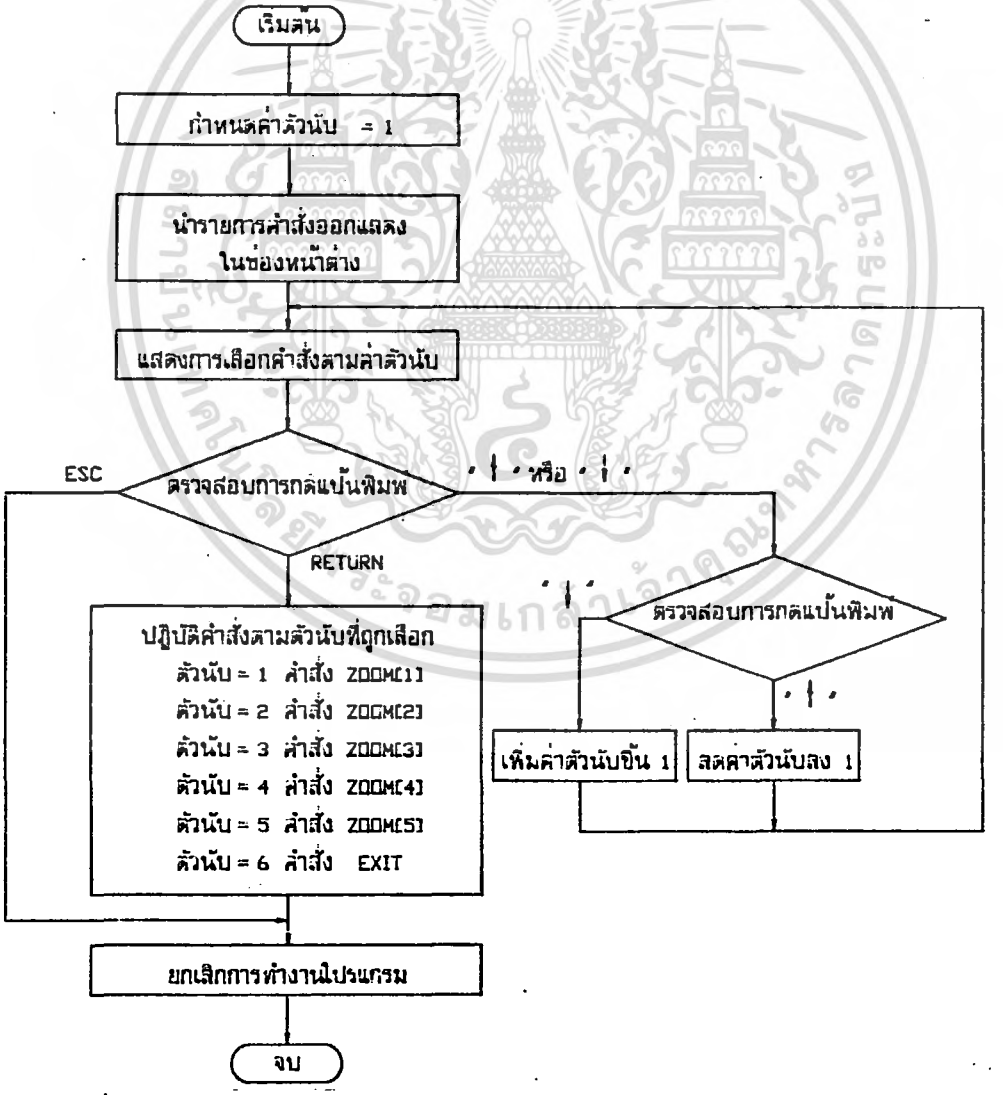
1. TAG[A] จะเก็บค่าตำแหน่งของตัวชี้ที่ตำแหน่ง (XA,YA) ลงในหน่วยความจำ
2. TAG[B] จะเก็บค่าตำแหน่งของตัวชี้ที่ตำแหน่ง (XB,YB) ลงในหน่วยความจำ
3. TAG[C] จะเก็บค่าตำแหน่งของตัวชี้ที่ตำแหน่ง (XC,YC) ลงในหน่วยความจำ
4. TAG[D] จะเก็บค่าตำแหน่งของตัวชี้ที่ตำแหน่ง (XD,YD) ลงในหน่วยความจำ
5. TAG[E] จะเก็บค่าตำแหน่งของตัวชี้ที่ตำแหน่ง (XE,YE) ลงในหน่วยความจำ
6. TAG[F] จะเก็บค่าตำแหน่งของตัวชี้ที่ตำแหน่ง (XF,YF) ลงในหน่วยความจำ
7. TAG[G] จะเก็บค่าตำแหน่งของตัวชี้ที่ตำแหน่ง (XG,YG) ลงในหน่วยความจำ
8. EXIT ออกจากโปรแกรมย่อย

เมื่อผ่านขั้นตอนทั้งหมดนี้แล้วก็จะสิ้นสุดโปรแกรมและกลับสู่การทำงานของโปรแกรม LIBRARY EDITOR

3.1.11 โปรแกรมกำหนดขนาดภาพ (LIBRARY/ZOOM COMMAND)

เป็นโปรแกรมจัดขนาดภาพแสดงผลที่หน้าจอ ในการแสดงมีอยู่ด้วยกัน 5 ขนาด ซึ่งขนาดปกติของภาพจะอยู่ที่ ZOOM (5) คำสั่งในส่วนของโปรแกรมย่อยจะประกอบด้วยคำสั่งต่าง ๆ ดังนี้

- คำสั่งการแสดงผลขนาดเล็กสุด (LIBRARY/ZOOM/ZOOM[1] COMMAND)
- คำสั่งการแสดงผลขนาดเล็ก (LIBRARY/ZOOM/ZOOM[2] COMMAND)
- คำสั่งการแสดงผลขนาดกลาง (LIBRARY/ZOOM/ZOOM[3] COMMAND)
- คำสั่งการแสดงผลขนาดใหญ่ (LIBRARY/ZOOM/ZOOM[4] COMMAND)
- คำสั่งการแสดงผลขนาดใหญ่สุด (LIBRARY/ZOOM/ZOOM[5] COMMAND)
- คำสั่งออกจากโปรแกรมย่อย (LIBRARY/ZOOM/EXIT)



รูปที่ 3-15 แสดงแผนผังการทำงานโปรแกรมจัดขนาดภาพแสดงผลหน้าจอ

อธิบายแผนผังการทำงานโปรแกรมจัดขนาดแสดงผลหน้าจอ

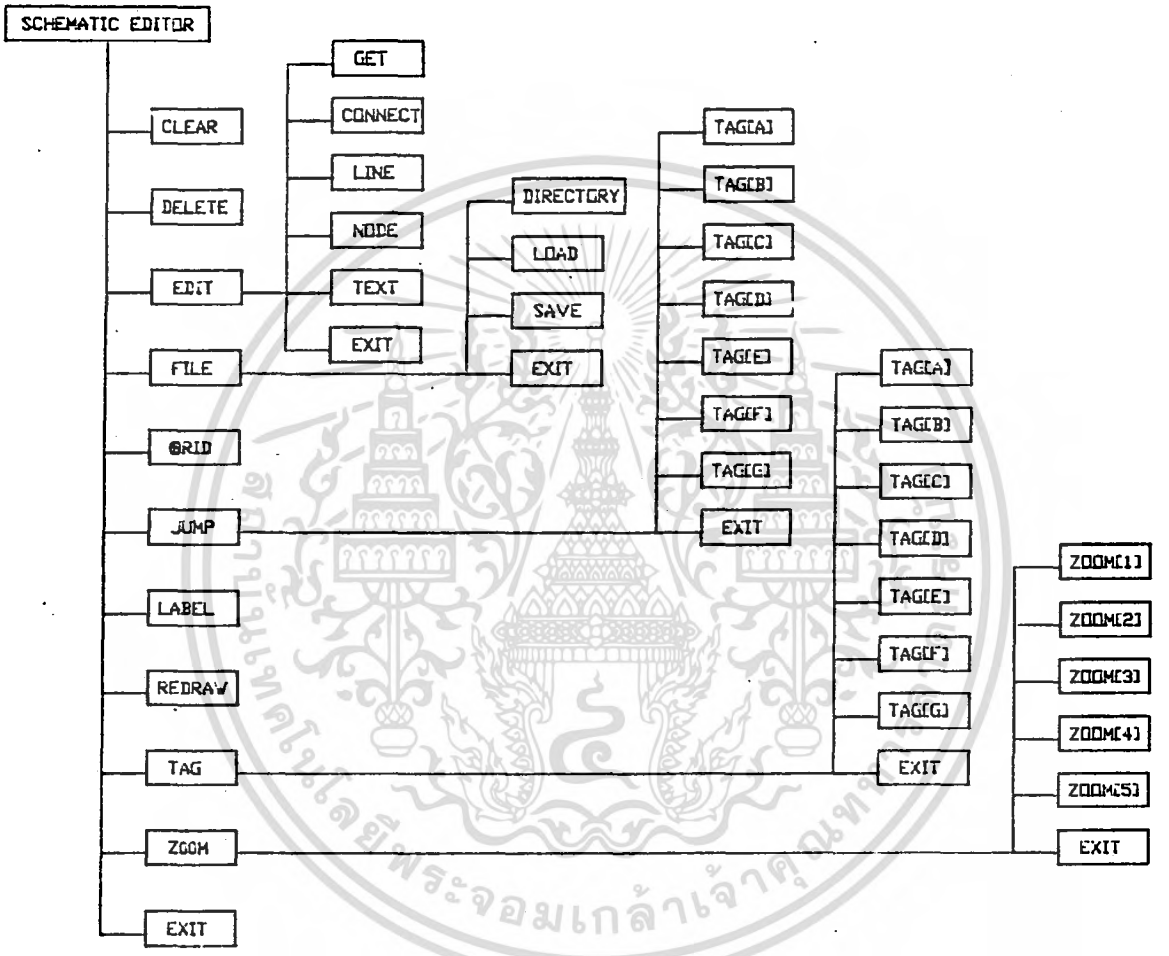
การทำงานของโปรแกรม เลือกรายการคำสั่งจัดขนาดหน้าจอดัง รูป 3-25 รายการคำสั่งจะมีการ เลือก เช่น เดียวกับข้อ 3.1 ผลของการ เลือกคำสั่งจะขึ้นอยู่กับค่าของตัวนับดังนี้

1. ZOOM[1] จะทำการจัดขนาดและการแสดงผลให้ เป็นขนาด เล็กสุด
2. ZOOM[2] จะทำการจัดขนาดและการแสดงผลให้ เป็นขนาด เล็ก
3. ZOOM[3] จะทำการจัดขนาดและการแสดงผลให้ เป็นขนาดกลาง
4. ZOOM[4] จะทำการจัดขนาดและการแสดงผลให้ เป็นขนาดใหญ่
5. ZOOM[5] จะทำการจัดขนาดและการแสดงผลให้ เป็นขนาดใหญ่สุด
6. EXIT ออกจากโปรแกรมย่อย

เมื่อผ่านขั้นตอนทั้งหมดนี้แล้วก็จะสิ้นสุดโปรแกรมและกลับสู่การทำงานของโปรแกรม LIBRARY EDITOR

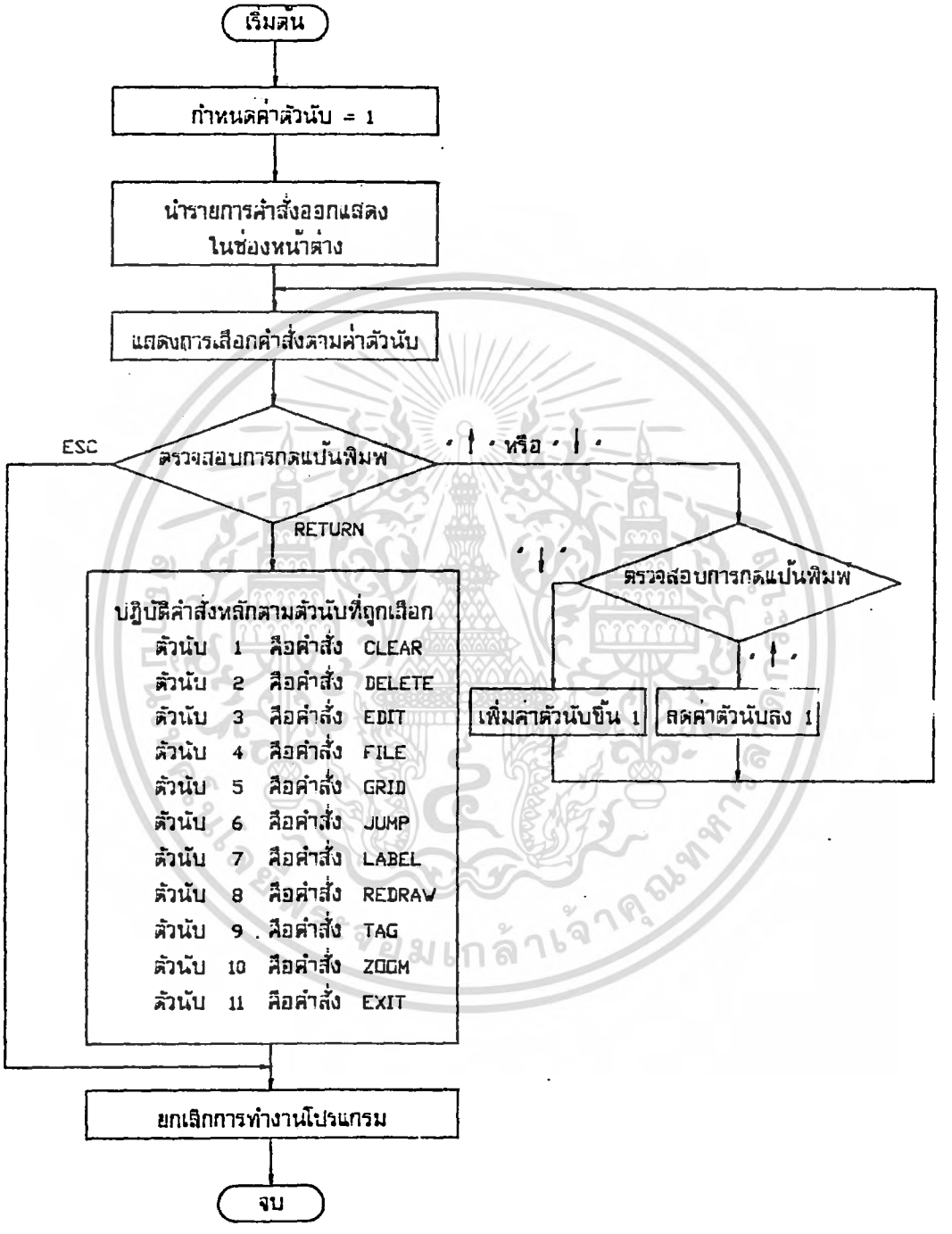
3.2 โปรแกรมการสร้างผังภาพวงจร (SCHEMATIC EDITOR)

โปรแกรมทำหน้าที่สร้างผังภาพวงจรตรรกขึ้นใหม่หรือทำการแก้ไข เพิ่ม เติมจากที่มีอยู่เดิมในแผ่นหน่วยความจำ โดยมีคำสั่งในการใช้งานดังแสดงในรูปที่ 3-26 ผู้ใช้สามารถจะเลือกคำสั่งจาก เมนูหลักที่แสดง เป็นรากของคำสั่ง ดังรูปที่ 3-16



รูปที่ 3-16 แผนผังแสดงรากของคำสั่ง (ROOT COMMAND MENU) ในส่วน SCHEMATIC

3.2.1 โปรแกรมการเลือกคำสั่งหลักจะสามารถเลือกคำสั่งจากการใช้แป้นพิมพ์ลูกศรซึ่งมีลำดับการเลือกดังนี้



รูปที่ 3-17 แผนผังแสดงการทำงานของโปรแกรมการเลือกคำสั่งหลัก

อธิบายแผนผังการทำงานของโปรแกรมการเลือกคำสั่งหลัก

การทำงานของโปรแกรมเลือกคำสั่งหลักนั้นสามารถเลือกได้ถึง 11 คำสั่งและในการเลือกคำสั่งต่าง ๆ ทำได้โดยกดแป้นพิมพ์ลูกศร ขึ้น หรือ ลง ซึ่งมีผลทำให้ตัวนับมีการเปลี่ยนแปลง จนกระทั่งมีการกดแป้นพิมพ์ RETURN ค่าของตัวนับสุดท้ายมีค่าเท่าใดนั้นหมายถึงการเลือกคำสั่งนั้น ๆ ขั้นตอนการทำงานของโปรแกรมมีดังนี้

1. นำรายการคำสั่งแสดงในช่องหน้าต่าง ดังรูปที่ 3-31 รายการคำสั่งมีดังนี้

   CLEAR DELETE EDIT FILE GRID JUMP LEBEL

   REDRAW TAG ZOOM EXIT

2. ตรวจสอบการกดแป้นพิมพ์ ในการกดแป้นพิมพ์เพื่อให้โปรแกรมสามารถปฏิบัติตามความต้องการได้จะมีเพียง 4 แป้นพิมพ์เท่านั้น
   - แป้นพิมพ์ลูกศรขึ้น จะเป็นการลดค่าของตัวนับลงไป 1 และการลดค่านี้จะไม่ให้ตัวนับมีค่าน้อยไปกว่า 1 ซึ่งถือว่าเป็นค่าต่ำสุดของตัวนับ
   - แป้นพิมพ์ลูกศรลง จะเป็นการเพิ่มค่าของตัวนับขึ้นไป 1 และการเพิ่มค่าจะไม่ให้ตัวนับมีค่ามากไปกว่า 11 ซึ่งถือว่าเป็นค่าสูงสุดของตัวนับ
   - แป้นพิมพ์ RETURN จะมีผลทำให้ไปปฏิบัติตามคำสั่งตามค่าลำดับของตัวนับ
   - แป้นพิมพ์ ESC ยกเลิกการทำงานของโปรแกรม และกลับยังส่วน DRAFT1
3. ค่าของตัวนับ จะกำหนดให้ไปปฏิบัติตามคำสั่งต่าง ๆ ดังนี้
   - ค่าตัวนับ = 1 คือคำสั่ง SCHEMATIC/CLEAR
   - ค่าตัวนับ = 2 คือคำสั่ง SCHEMATIC/DELETE
   - ค่าตัวนับ = 3 คือคำสั่ง SCHEMATIC/EDIT
   - ค่าตัวนับ = 4 คือคำสั่ง SCHEMATIC/FILE
   - ค่าตัวนับ = 5 คือคำสั่ง SCHEMATIC/GRID
   - ค่าตัวนับ = 6 คือคำสั่ง SCHEMATIC/JUMP
   - ค่าตัวนับ = 7 คือคำสั่ง SCHEMATIC/LABEL
   - ค่าตัวนับ = 8 คือคำสั่ง SCHEMATIC/REDRAW
   - ค่าตัวนับ = 9 คือคำสั่ง SCHEMATIC/TAG
   - ค่าตัวนับ = 10 คือคำสั่ง SCHEMATIC/ZOOM
   - ค่าตัวนับ = 11 คือคำสั่ง SCHEMATIC/EXIT

เมื่อปฏิบัติตามคำสั่งต่าง ๆ ยกเว้นคำสั่ง EXIT แล้วโปรแกรมจะกลับไปเริ่มที่รายการคำสั่งหลัก (SCHEMATIC EDITOR) ใหม่ แต่ถ้าเป็นคำสั่ง EXIT จะเป็นการยกเลิก

โปรแกรมการสร้างผังภาพวงจร และกลับเข้าสู่การทำงานของโปรแกรม DRAFT1

ในส่วนการสร้างผังภาพวงจรตรรก โดยส่วนรวมแล้วจะเหมือนกับการทำงานในส่วนของไลบรารี และในบางคำสั่งก็ใช้ร่วมกันอยู่ด้วยซึ่งพอจะแบ่งแยกคำสั่งที่ใช่ร่วมกันและคำสั่งที่มีการทำงานเหมือนกัน ดังนี้

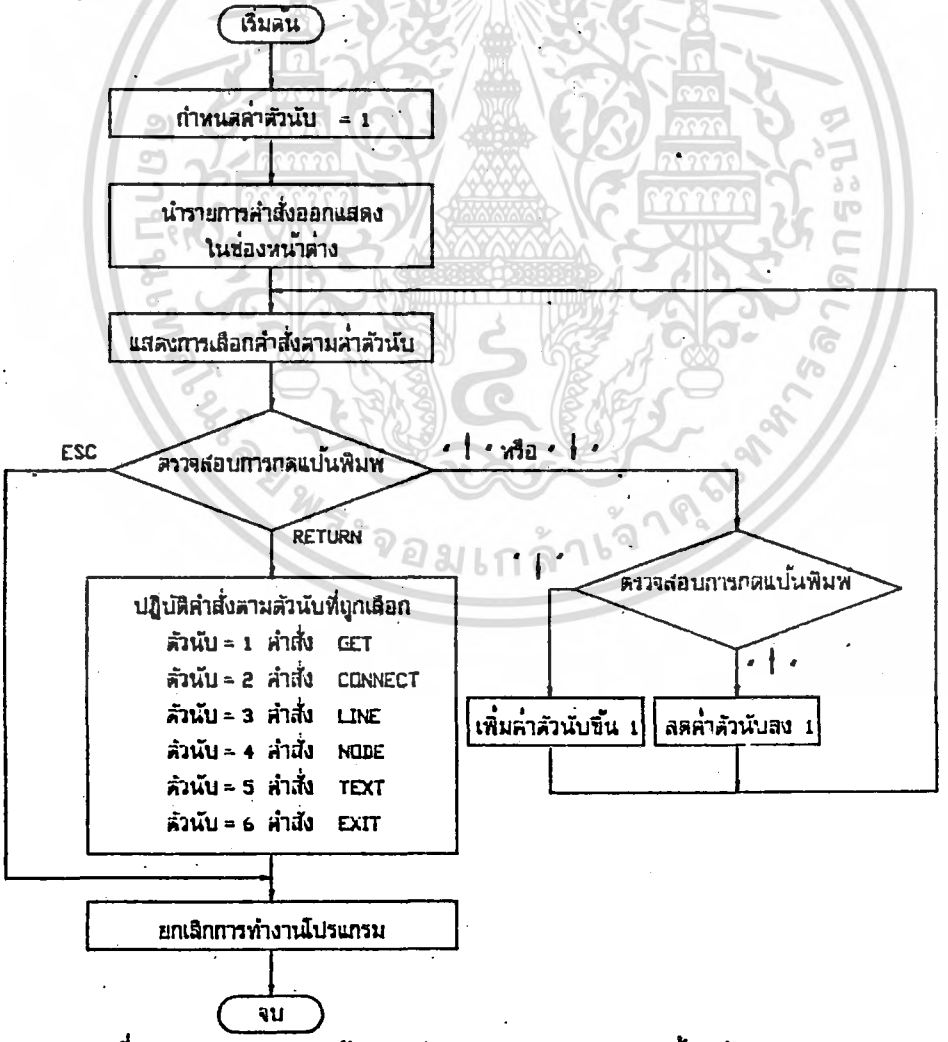
- คำสั่งที่ใช้ร่วมกัน ได้แก่ GRID JUMP TAG ZOOM
- คำสั่งที่มีการทำงานเหมือนกัน ได้แก่ CLEAR FILE LABEL DELETE REDRAW

คำสั่งต่าง ๆ เหล่านี้ได้กล่าวมาแล้วในการสร้างไลบรารีแต่จะมีบางคำสั่งที่มีการทำงานนอกเหนือไปจากที่ได้กล่าวมาคือ คำสั่ง EDIT ซึ่งจะได้กล่าวต่อไป ทุกคำสั่งที่กล่าวมานี้เมื่อเสร็จสิ้นการทำงานในคำสั่งนั้น ๆ แล้วโปรแกรมจะกลับมายังส่วนของ SCHEMATIC EDITOR

3.2.2 โปรแกรมสร้างผังภาพวงจร (SCHEMATIC/EDIT COMMAND)

เป็นโปรแกรมสร้างผังภาพวงจรโดยการอ่านไฟล์ข้อมูลไลบรารี ที่เป็นรูปอุปกรณ์ประเภทต่าง ๆ มาประกอบกัน และข้อมูลที่นำมาประกอบให้เป็นผังภาพวงจรนอกจากจะเป็นอุปกรณ์ไลบรารีแล้ว ยังรวมไปถึงข้อมูลของการลากเส้นกำหนดจุดต่อสัญญาณ ข้อมูลแบบข้อความ และโหนดอีกด้วย คำสั่งต่าง ๆ ที่ใช้ในการสร้างผังภาพวงจรมีดังนี้

- คำสั่งเลือกอุปกรณ์ไลบรารี (SCHEMATIC/EDIT/GET COMMAND)
- คำสั่งกำหนดจุดต่อของสัญญาณ (SCHEMATIC/EDIT/CONNECT COMMAND)
- คำสั่งกำหนดการลากเส้นวงจร (SCHEMATIC/EDIT/LINE COMMAND)
- คำสั่งกำหนดชื่อโหนด (SCHEMATIC/EDIT/NODE COMMAND)
- คำสั่งกำหนดข้อความ (SCHEMATIC/EDIT/TEXT COMMAND)
- คำสั่งออกจากโปรแกรมย่อย (SCHEMATIC/EDIT/EXIT COMMAND)



รูปที่ 3-18 แสดงแผนผังการทำงานของโปรแกรมสร้างผังภาพวงจร

อธิบายแผนผังการทำงานของโปรแกรมสร้างรูปผังภาพวงจร

การทำงานของโปรแกรมในส่วนของการเลือกคำสั่งดังรูปที่ 3-27 เช่นเดียวกับข้อ 3.2.1 ที่กล่าวถึงการอธิบายแผนผังการทำงานของโปรแกรมการเลือกคำสั่งหลัก และผลของการเลือกคำสั่งจะขึ้นอยู่กับค่าของลำดับตัวนับดังนี้

1. GET จะปฏิบัติโปรแกรมเลือกอุปกรณ์ไลบรารี
2. CONNECT จะปฏิบัติโปรแกรมกำหนดจุดต่อของสัญญาณ
3. LINE จะปฏิบัติโปรแกรมลากเส้นต่อในวงจร
4. NODE จะปฏิบัติโปรแกรมกำหนดชื่อโหนด
5. TEXT จะปฏิบัติโปรแกรมกำหนดข้อความ
6. EXIT ออกจากโปรแกรมการทำงาน

หลังจากที่ปฏิบัติตามคำสั่งต่าง ๆ เหล่านี้เสร็จสิ้นแล้วก็จะยกเลิกการทำงานของโปรแกรม และกลับยังส่วนของโปรแกรม SCHEMATIC DEITOR

3.3 ลักษณะโครงสร้างข้อมูลของอุปกรณ์ไลบรารี และ ผังภาพวงจร

ข้อมูลรูปภาพที่สร้างโดยโปรแกรม DRAFT 1 จะมีอยู่ 2 ส่วนคือ

- โครงสร้างข้อมูลอุปกรณ์ไลบรารี (LIBRARY DATA)
- โครงสร้างข้อมูลผังภาพวงจร (SCHEMATIC DATA)

3.3.1 โครงสร้างข้อมูลอุปกรณ์ไลบรารี

อุปกรณ์ไลบรารีโดยทั่ว ๆ ไปจะประกอบด้วยข้อมูลของเส้นตรง วงกลม ส่วนโค้ง และข้อความ ฉะนั้นการสร้างรูปอุปกรณ์ไลบรารีจึงเป็นการกำหนดเอาส่วนต่าง ๆ เหล่านี้มาประกอบกันเพื่อให้ได้เป็นรูปที่ต้องการ ข้อมูลประเภทต่าง ๆ จึงได้มีการกำหนดให้เป็นรหัสของตัวเลขที่มีความหมายต่างกันดังนี้

รหัส 1 หมายถึง กำหนดให้เป็นเส้นตรง (LINE)

รหัส 2 หมายถึง กำหนดให้เป็นวงกลม (CIRCLE)

รหัส 3 หมายถึง กำหนดให้เป็นเส้นโค้ง (ARC)

รหัส 4 หมายถึง กำหนดให้เป็นของข้อความ (TEXT)

นอกจากนี้ในแต่ละส่วนยังประกอบด้วยส่วนย่อยอื่น ๆ ที่แตกต่างกันตามประเภทของข้อมูลนั้น ๆ กันออกไป ซึ่งจะมีรูปแบบดังนี้

- ข้อมูลเส้นตรง

  ข้อมูลเส้นตรงจะประกอบด้วยจุดเริ่มต้น จุดสุดท้าย และสีของเส้นตรง ฉะนั้นโครงสร้างของข้อมูลเส้นตรงจึงเป็น

รหัสเส้นตรง	เป็น	1
จุดเริ่มต้นของเส้นตรง	เป็น	(X0,Y0)
จุดสุดท้ายของเส้นตรง	เป็น	(X1,Y1)
สีเส้นตรง	เป็น	COLOR

- ข้อมูลวงกลม

  ข้อมูลวงกลมจะประกอบด้วย จุดศูนย์กลาง รัศมี และสีของวงกลม ฉะนั้นโครงสร้างข้อมูลวงกลมจึงเป็น

รหัสวงกลม	เป็น	2
จุดศูนย์กลางของวงกลม	เป็น	(X0,Y0)
รัศมีของวงกลม	เป็น	RADIUS
สีวงกลม	เป็น	COLOR

- ข้อมูลเส้นโค้ง

ข้อมูลเส้นโค้งจะประกอบด้วยจุดศูนย์กลางของส่วนโค้ง รัศมีส่วนโค้ง มุมเริ่มต้น มุมสุดท้ายและสีของเส้น ฉะนั้นโครงสร้างข้อมูลเส้นโค้งของไลบรารีจึงเป็น

รหัสเส้นโค้ง	เป็น	3
จุดศูนย์กลาง	เป็น	(XO,YO)
มุมเริ่มต้นของส่วนโค้ง	เป็น	ANGLE 1
มุมสุดท้ายของส่วนโค้ง	เป็น	ANGLE 2
รัศมีของเส้นโค้ง	เป็น	RADIUS
สีเส้นโค้ง	เป็น	COLOR

- ข้อมูลแบบข้อความ

ข้อมูลแบบข้อความ จะประกอบด้วยจุดที่บอกตำแหน่งข้อความ ข้อความ และสีของข้อความ ฉะนั้นโครงสร้างข้อมูลแบบข้อความจึงเป็น

รหัสข้อความ	เป็น	4
จุดบอกตำแหน่งของข้อความ	เป็น	(XO,YO)
ข้อมูลข้อความ	เป็น	TEXT
สีข้อความ	เป็น	COLOR

จากข้อมูลส่วนประกอบต่าง ๆ นี้แล้วในการสร้างรูปอุปกรณ์ไลบรารีแต่ละรูปจะต้องมีรายละเอียดของข้อมูลที่เป็นชื่อ จุดอ้างอิง ขนาดของไลบรารี และจำนวนส่วนประกอบต่าง ๆ ทั้งหมดของการสร้างรูปอุปกรณ์ไลบรารี โครงสร้างข้อมูลการสร้างรูปไลบรารีที่จะนำลงเก็บเป็นไฟล์ไลบรารี (.LIB) มีรายละเอียดในภาคผนวก 2

3.3.2 โครงสร้างข้อมูลผังภาพวงจร

รูปผังภาพวงจรโดยทั่วไปจะประกอบด้วย ข้อมูลอุปกรณ์ไลบรารี เส้นตรง จุดต่อของสัญญาณโหนดต่าง ๆ และข้อความ ในการสร้างผังภาพวงจรจึงเป็นการนำเอาส่วนต่าง ๆ เหล่านี้มาประกอบเข้าด้วยกันให้เป็นวงจรตามที่ต้องการ ข้อมูลประเภทต่าง ๆ จึงได้กำหนดให้เป็นรหัสของตัวเลขที่มีความหมายต่างกันดังนี้

รหัส 1 หมายถึง กำหนดให้เป็นรหัสข้อมูลเส้นตรง (LINE)

รหัส 2 หมายถึง กำหนดให้เป็นข้อมูลจุดต่อสัญญาณ (CONNECT)

รหัส 3 หมายถึง กำหนดให้เป็นข้อมูลแบบข้อความ (TEXT)

รหัส 4 หมายถึง กำหนดให้เป็นของข้อมูลโหนดต่าง ๆ (NODE)

และรหัสตั้งแต่ 5 ขึ้นไปจะกำหนดให้เป็นรหัสข้อมูลไลบรารีที่นำมาใช้สร้างผังภาพวงจรนอกจากการกำหนดรหัสให้กับข้อมูลเหล่านี้แล้ว แต่ละส่วนยังประกอบด้วยส่วนย่อยอื่น ๆ เพิ่มเติมอีก ซึ่งมีลักษณะของข้อมูลดังนี้

- ข้อมูลเส้นตรงของวงจร

ข้อมูลเส้นตรงของวงจรจะประกอบด้วยจุดอ้างอิง จุดเริ่มต้น จุดสุดท้าย ข้อความ สีของเส้นตรง ฉะนั้นโครงสร้างข้อมูลเส้นตรงจึงเป็นดังนี้

รหัสเส้นตรงของวงจร	เป็น	1
จุดอ้างอิงของเส้นตรง	เป็น	(REFX,REFY)
จุดเริ่มต้น จุดสุดท้าย	เป็น	(X0,Y0) และ (X1,Y1)
ข้อความ	เป็น	LINE
สีเส้นตรง	เป็น	COLOR

- ข้อมูลจุดต่อของสัญญาณ

ข้อมูลจุดต่อของสัญญาณจะประกอบด้วยจุดอ้างอิง จุดบอกขนาดของโหนด ข้อความ และสีของจุดต่อ ฉะนั้นโครงสร้างของจุดต่อจึงเป็นดังนี้

รหัสข้อมูลจุดต่อ	เป็น	2
จุดอ้างอิงของจุดต่อ	เป็น	(REFX,REFY)
จุดบอกขนาดของข้อมูลจุดต่อ	เป็น	(X0,Y0) และ (X1,Y1)
ข้อความ	เป็น	CONNECT
สีของจุดต่อ	เป็น	COLOR

- ข้อมูลแบบข้อความ

ข้อมูลแบบข้อความจะประกอบด้วยจุดอ้างอิงของข้อความ และรหัสสีของ

ข้อความ ฉะนั้นโครงสร้างข้อมูลแบบข้อความจึงเป็นดังนี้

รหัสข้อมูลแบบข้อความ	เป็น	3
จุดอ้างอิงของข้อความ	เป็น	(REFX,REFY)
จุดบอกตำแหน่งข้อความ	เป็น	(X0,Y0) และ (X1,Y1)
ข้อมูลข้อความ.	เป็น	TEXT
สีของข้อความ	เป็น	COLOR

- ข้อมูลโหนดของวงจร

ข้อมูลโหนดของวงจรจะคล้ายกับข้อมูลแบบข้อความ แต่จุดประสงค์การนำไปใช้งานต่างกัน ฉะนั้นโครงสร้างของข้อมูลโหนดจึงเป็นดังนี้

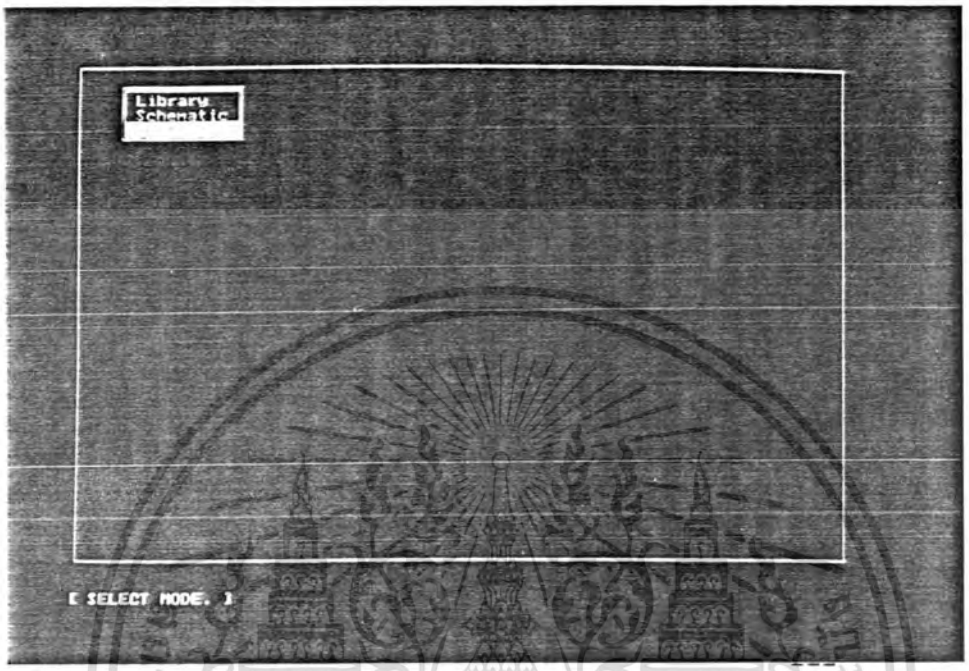
รหัสข้อมูลของโหนด	เป็น	4
จุดอ้างอิงของโหนด	เป็น	(REFX,REFY)
จุดบอกตำแหน่งของโหนด	เป็น	(X0,Y0) และ (X1,Y1)
ข้อมูลโหนด	เป็น	NODE
สีของข้อความ	เป็น	COLOR

- ข้อมูลอุปกรณ์ไลบรารี

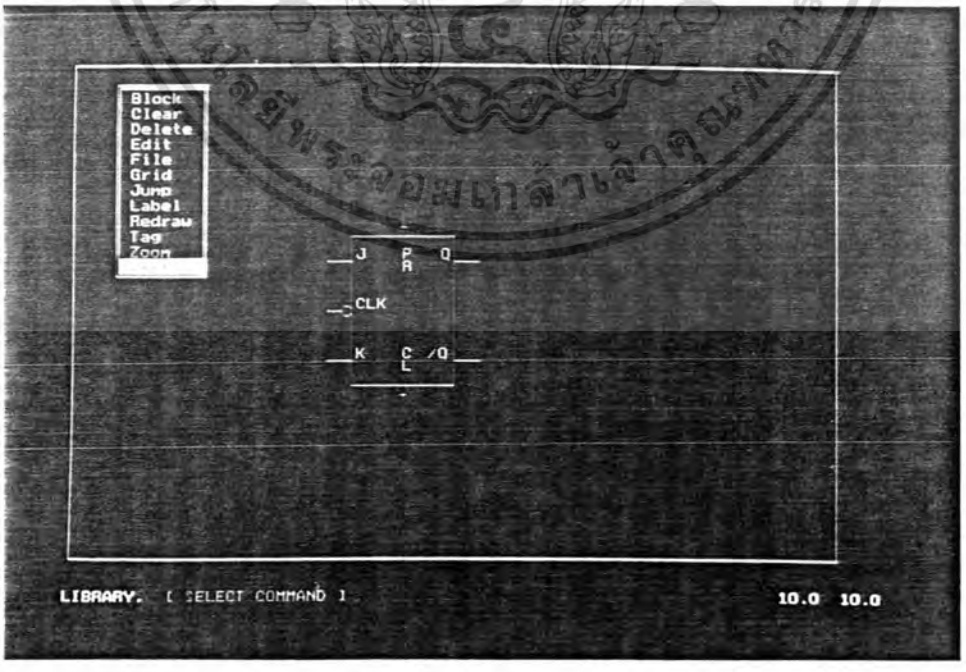
ข้อมูลไลบรารีนำมาทำวงจรจะมีรหัสข้อมูลตั้งแต่ 5 ขึ้นไป ฉะนั้นโครงสร้างของข้อมูลไลบรารีจึงเป็นดังนี้

รหัสข้อมูลไลบรารี	เป็น	เลขตั้งแต่ 5 ขึ้นไป (ขึ้นอยู่กับลำดับของไฟล์ไลบรารีที่อ่านลงสู่หน่วยความจำ)
จุดอ้างอิงของไลบรารี	เป็น	(REFX,REFY)
จุดบอกขอบเขตของไลบรารี	เป็น	(X0,Y0) และ (X1,Y1)
ข้อความ	เป็น	NAME
สีของรูปไลบรารี	เป็น	COLOR

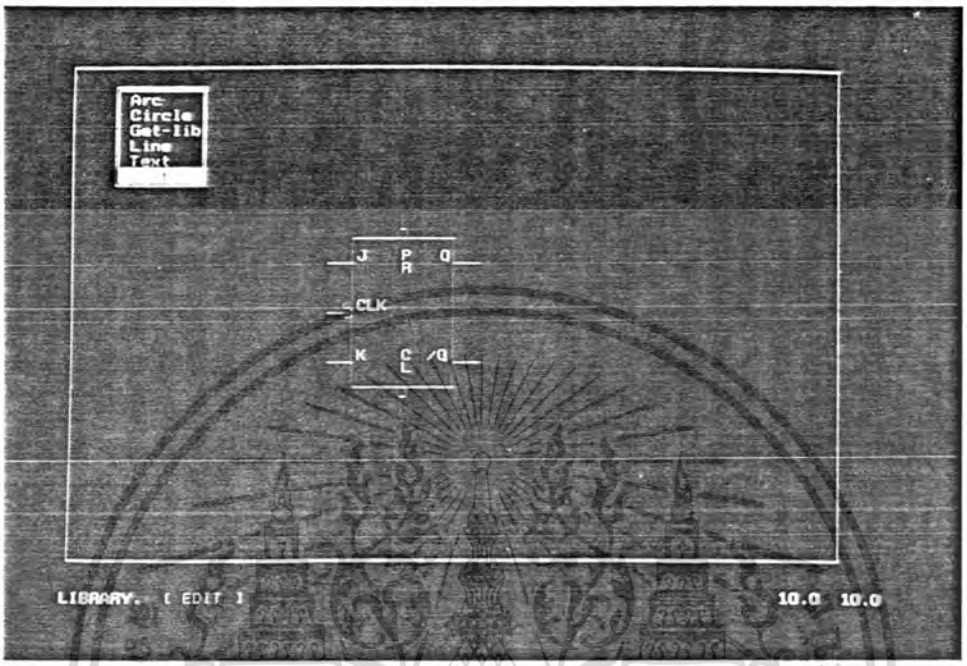
จากข้อมูลส่วนประกอบต่าง ๆ นี้แล้ว การสร้างผังภาพวงจรเก็บลงแผ่นหน่วยความจำนั้นต้องเพิ่มเติมรายละเอียดของชื่อจำนวนส่วนประกอบทั้งหมดในการสร้างวงจร ฉะนั้นโครงสร้างข้อมูลรูปวงจรที่สร้างเพื่อเก็บลงไฟล์ผังภาพวงจร (.SCH) มีรายละเอียดในภาคผนวก 2



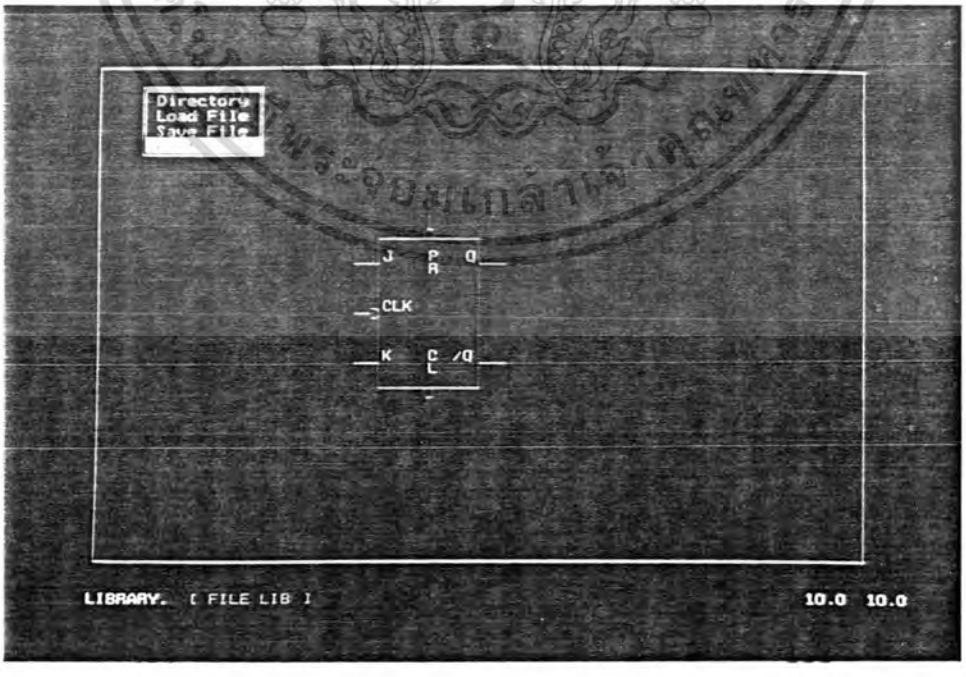
รูปที่ 3-19 แสดงรายการคำสั่งเมนูของโปรแกรม DRAFT 1



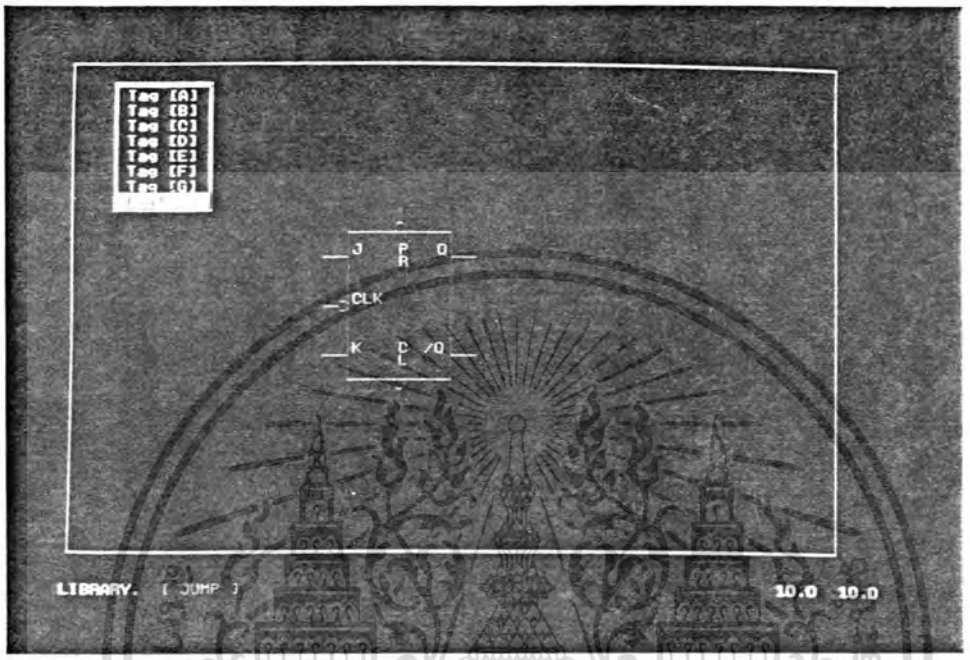
รูปที่ 3-20 คำสั่งหลักทั้ง 12 คำสั่งในส่วนของ LIBRARY EDITOR



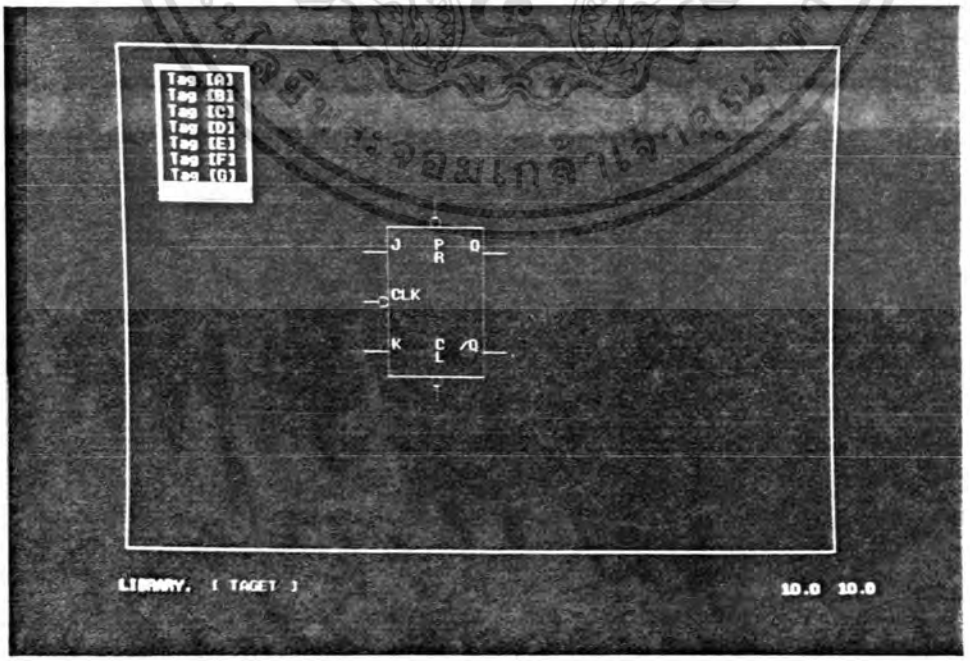
รูปที่ 3-21 แสดงรายการคำสั่งการสร้างรูปอุปกรณ์



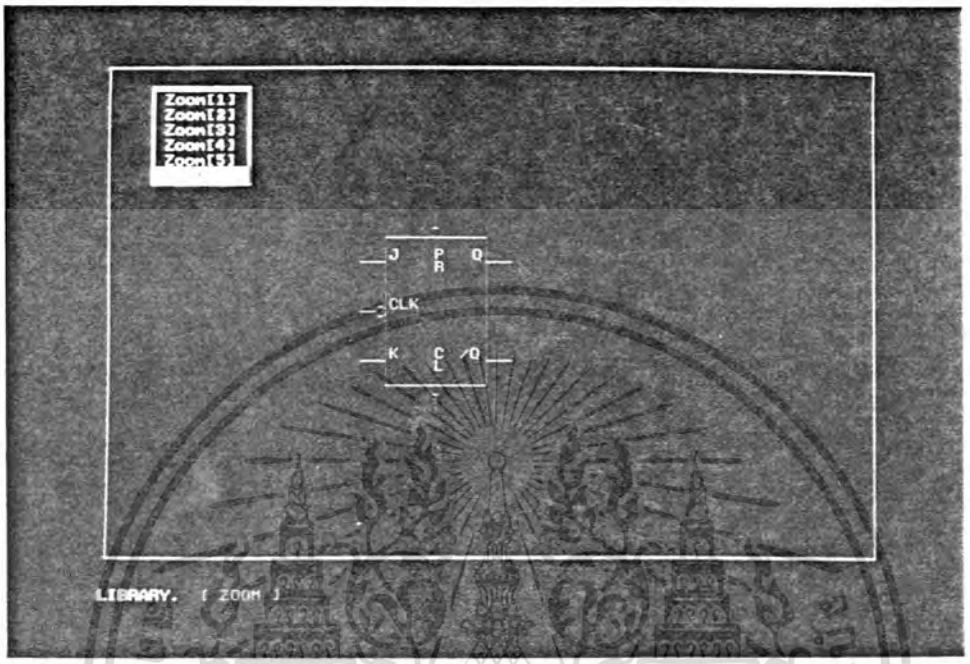
รูปที่ 3-22 แสดงรายการคำสั่งบริการไฟล์ไลบรารี



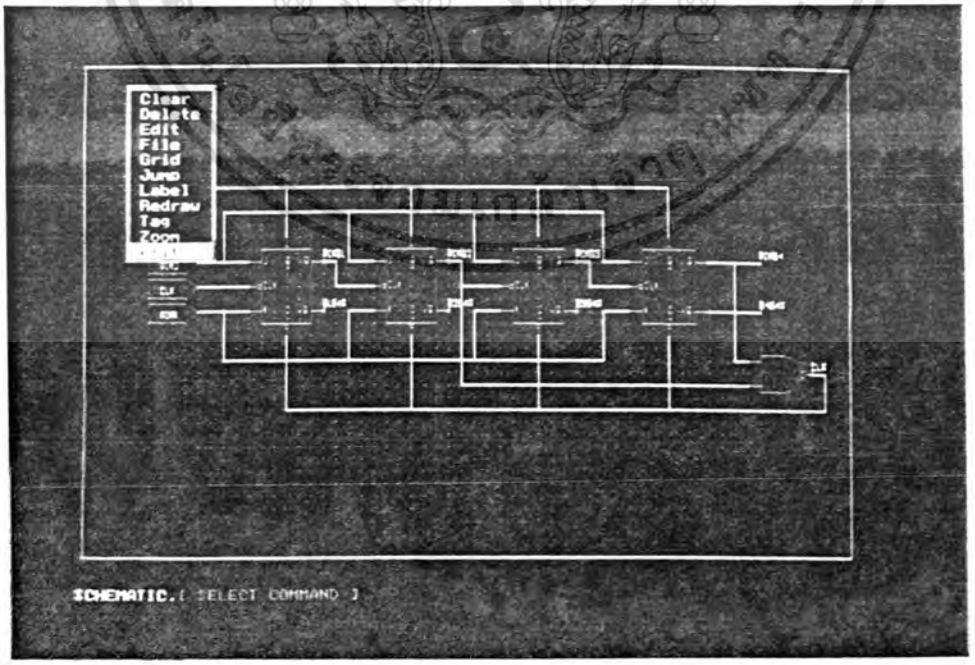
รูปที่ 3-23 แสดงรายการคำสั่งการเลือกจุดที่ต้องการย้ายตำแหน่งของตัวชี้



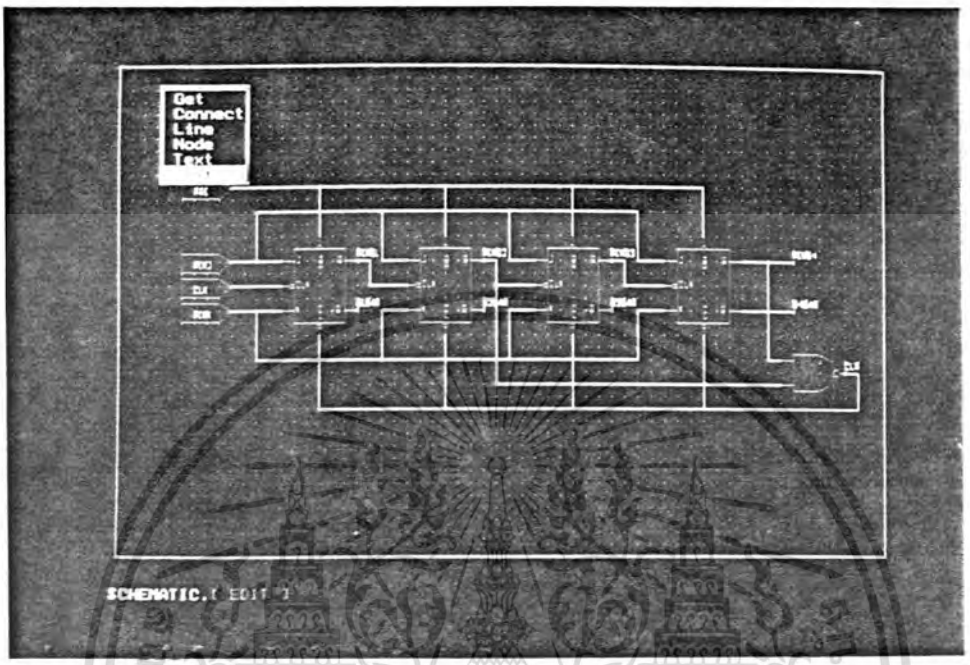
รูปที่ 3-24 แสดงรายการคำสั่งการกำหนดตำแหน่งจุดที่ต้องการย้ายของตัวชี้



รูปที่ 3-25 แสดงรายการเลือกขนาดแสดงผลหน้าจอ



รูปที่ 3-26 คำสั่งหลักทั้ง 11 คำสั่ง ในส่วนของ SCHEMATIC EDITOR



รูปที่ 3-27 แสดงรายการคำสั่งใช้ในการสร้างรูปผังภาพวงจร

สรุป ในบทที่ 3 จะเป็นการกล่าวถึงโปรแกรมการสร้างรูป หรือ LOGCAD EDITOR ในการสร้างจะแบ่งการสร้างเป็น 2 ลักษณะ

- การสร้างรูปอุปกรณ์ไลบรารี
- การสร้างรูปผังภาพวงจร

และในการสร้างแต่ละแบบนี้จะใช้คำสั่งต่าง ๆ มาช่วยในการสร้าง โดยที่คำสั่งเหล่านี้มีลักษณะการใช้งานที่แตกต่างกันออกไป นอกจากนี้ยังได้กล่าวถึงโครงสร้างของข้อมูลในส่วนของไลบรารี และในส่วนของผังภาพวงจรนั้นจะมีโครงสร้างที่แตกต่างกันตามประเภทหรือตามชนิดของข้อมูลที่ผู้สร้างโปรแกรมได้มีการกำหนดรูปแบบไว้ก่อนแล้ว

บทที่ 4

โปรแกรมการสร้างไฟล์ข้อมูลโหนด

จากที่ได้กล่าวมาแล้วในบทที่ 2 โปรแกรม LOGCAD เป็นโปรแกรมที่ประกอบด้วย 2 ส่วนใหญ่ ๆ ด้วยกัน คือ ส่วนของการสร้างผังภาพวงจรตรรกและส่วนของการจำลองการทำงานทั้งสองส่วนนี้จะติดต่อกับด้วย ไฟล์ข้อมูลโหนด โดยที่ส่วนของการสร้างผังภาพวงจรตรรกจะมีหน้าที่ในการสร้างโหนดต่าง ๆ ของวงจรให้ ในขณะที่ส่วนของการจำลองการทำงานจะรับเอาข้อมูลที่เป็นโหนดเหล่านี้ไปเป็นข้อมูลหลัก เพื่อใช้ในการประมวลผลสำหรับการจำลองการทำงาน

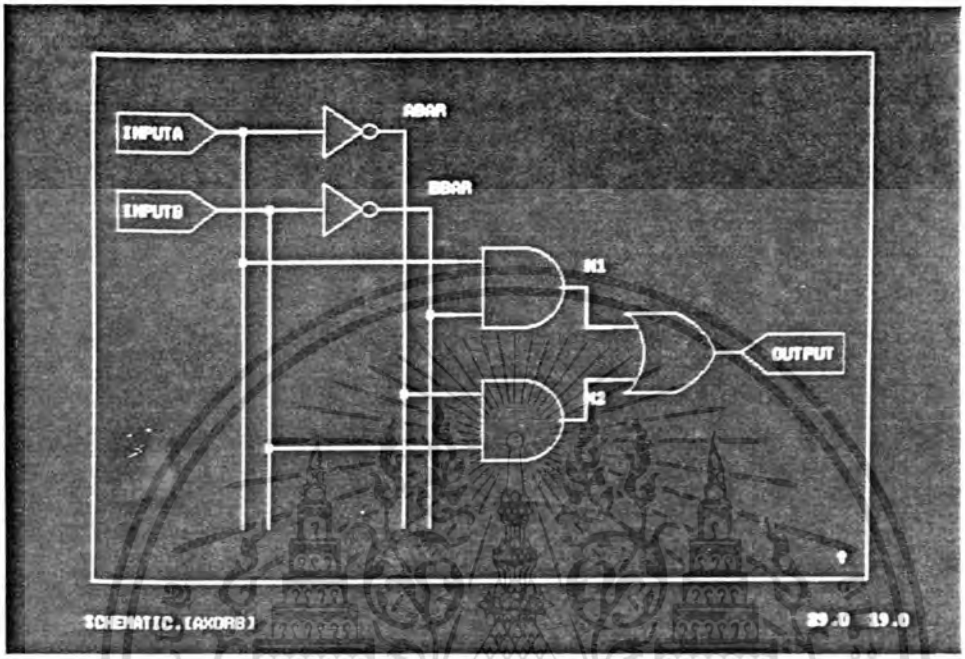
แต่เนื่องจากว่าในการสร้างผังภาพวงจรตรรกนั้นมีวิธีการสร้างได้ 2 ลักษณะคือ (จากรูป 2-1) ผังภาพวงจรที่สร้างจากโปรแกรม LOGCAD และผังภาพวงจรที่สร้างจากโปรแกรม ORCAD ซึ่งแนวทางที่ได้ดำเนินทั้งสองลักษณะนี้จะให้ผลลัพธ์สุดท้ายคือได้ไฟล์ข้อมูลโหนดที่เหมือนกัน แต่วิธีในการปฏิบัติเพื่อให้ได้ไฟล์ข้อมูลโหนดนั้นจะต่างกันออกไป ดังได้กล่าวมาแล้วในหัวข้อ 2.1.2

การสร้างไฟล์ข้อมูลโหนดสามารถสร้างได้โดยโปรแกรม 2 โปรแกรมดังนี้

- LOGNODE1
- LOGNODE

4.1 LOGNODE1 เป็นโปรแกรมที่ใช้ในการสร้างไฟล์ข้อมูลโหนดโดยการอ่านจากไฟล์ผังภาพวงจรที่สร้างโดยโปรแกรม LOGCAD EDITOR ซึ่งลักษณะของข้อมูลที่เป็นผังภาพวงจรนี้จะประกอบด้วยรหัสของเส้นต่าง ๆ และรหัสอื่น ๆ อีกหลายส่วนด้วยกันดังได้กล่าวมาแล้วในบทที่ 3 ฉะนั้นในการอ่านไฟล์ข้อมูลจะเลือกมาเฉพาะข้อมูลที่เป็นรหัสของโหนดวงจรเท่านั้น

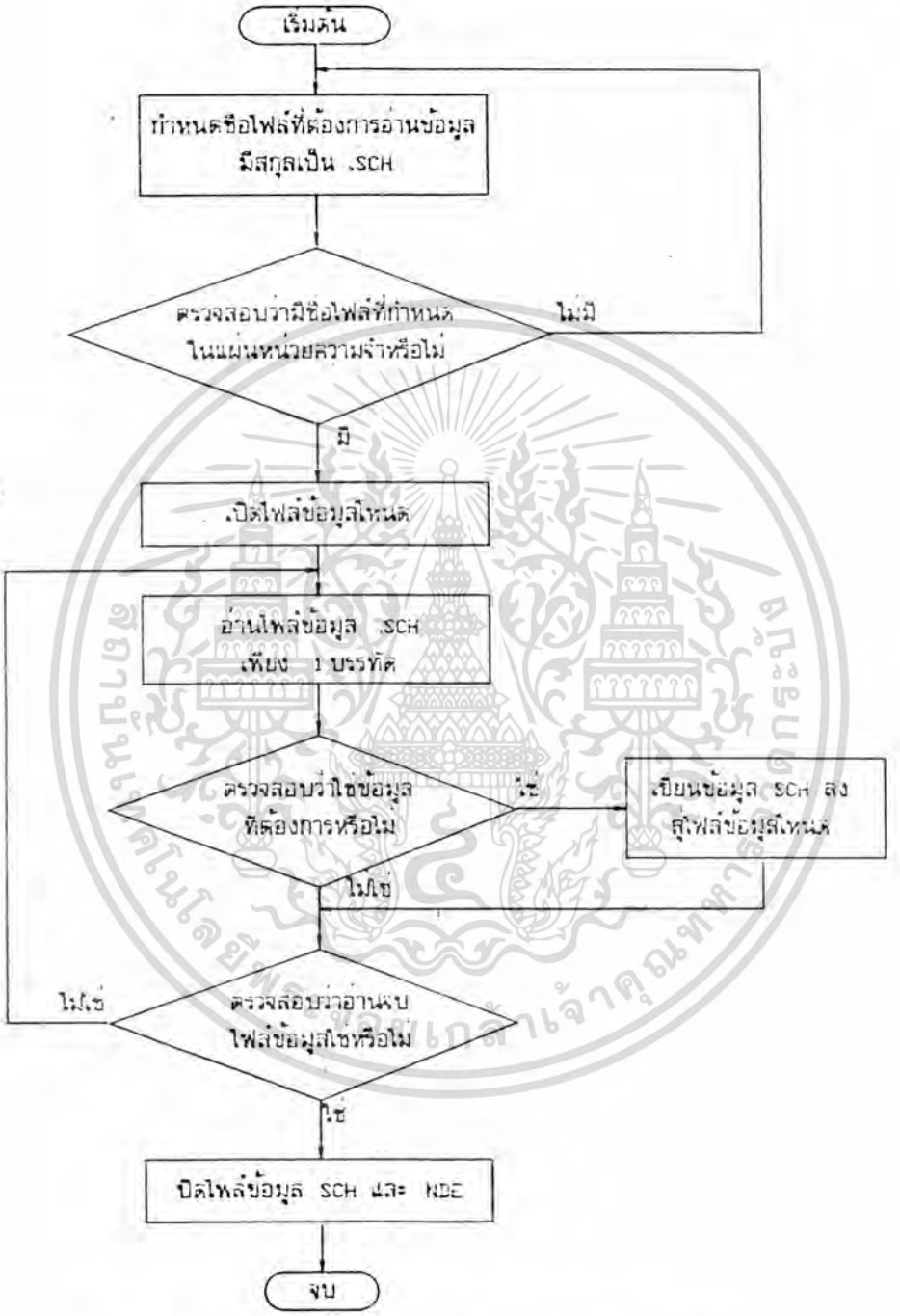
ตัวอย่าง การสร้างผังภาพวงจรจากสมการ $A \oplus B = A.\bar{B} + \bar{A}.B$



รูปที่ 4.1 ผังภาพวงจรที่ได้จากโปรแกรม LOGCAD EDITOR

เมื่อการสร้างผังภาพวงจรเป็นไปตามรูป 4.1 แล้วจะได้ไฟล์ข้อมูลโหนดดังนี้

```
INPUTB
INPUTA
OUTPUT
ABAR
BBAR
N1
N2
```



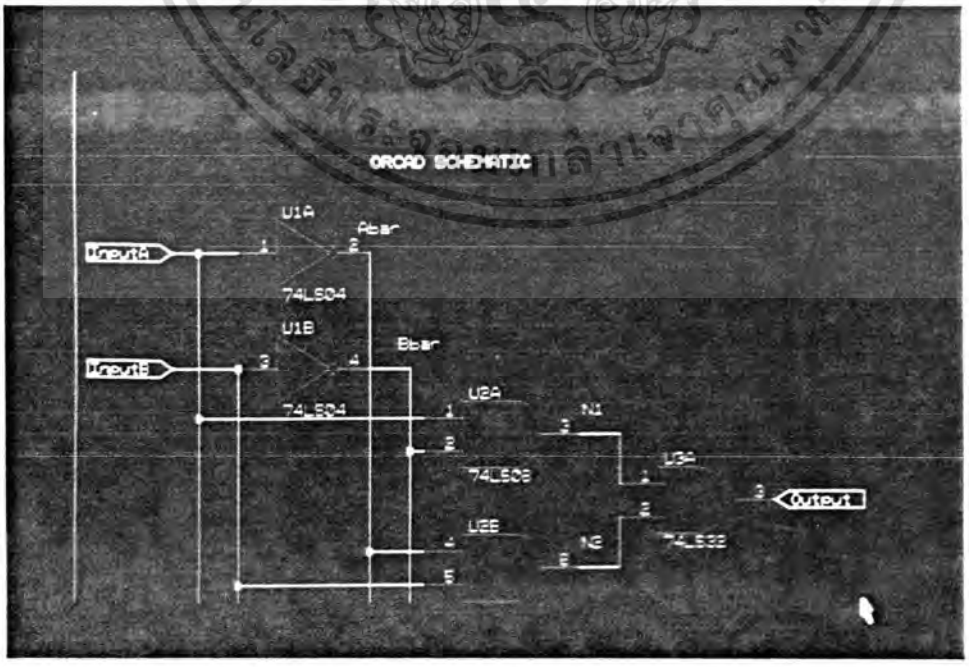
รูปที่ 4-2 แสดงแผนผังการทำงานของโปรแกรม LOGNODE1

อธิบายแผนผังการทำงานของโปรแกรม LOGNODE1

1. จะต้องมีการกำหนดชื่อไฟล์ข้อมูลผังภาพวงจร (FILENAME.SCH) ทางแป้นพิมพ์ เพื่อนำมาตรวจสอบ และผลการตรวจสอบปรากฏว่าไม่มีไฟล์ข้อมูลดังกล่าวอยู่ ในแผ่นหน่วยความจำโปรแกรมจะจัดการให้กำหนดชื่อไฟล์ข้อมูลใหม่อีกครั้ง แต่ ถ้าปรากฏว่ามีไฟล์ข้อมูลดังกล่าว โปรแกรมจะทำการเปิดไฟล์ข้อมูลโหนด (FILENAME.NDE) ทันที
2. ทำการอ่านไฟล์ข้อมูล เพื่อที่จะได้นำมาตรวจสอบว่าเป็นข้อมูลโหนดที่ต้องการ ผล การตรวจสอบปรากฏว่าเป็นข้อมูลที่เป็นโหนดของวงจรจริง โปรแกรมจะจัดการ บันทึกข้อมูลเหล่านี้ลงสู่ไฟล์ข้อมูลโหนด แต่ถ้าไม่เป็นข้อมูลโหนดของวงจร โปรแกรมจะไม่นำข้อมูลเหล่านี้ปฏิบัติการใด ๆ ทั้งสิ้น
3. โปรแกรมจะตรวจสอบไฟล์ข้อมูลที่อ่านอยู่นั้นว่าหมดไฟล์แล้วหรือยัง ถ้ายังไม่ หมดจะปฏิบัติในข้อ 2 แต่ถ้าหมดแล้วก็จะปิดไฟล์ข้อมูลต่าง ๆ และยกเลิกการทำ งานของโปรแกรม

4.2 LOGNODE เป็นโปรแกรมใช้สำหรับสร้างไฟล์ข้อมูลโหนด โดยการอ่านจาก ไฟล์ข้อมูลที่เกิดจากโปรแกรม NETLIST ของ ORCAD

ตัวอย่าง การสร้างผังภาพวงจรจากสมการ $A \oplus B = A.\overline{B} + \overline{A}.B$



รูปที่ 4-3 ผังภาพวงจรที่ได้จากโปรแกรม ORCAD

ข้อกำหนดในการสร้างจุดเชื่อมต่อระหว่างอุปกรณ์ของโปรแกรม ORCAD นี้คือจะต้องเลือกใช้เฉพาะคำสั่ง PLACE/LABEL/INTERNAL เท่านั้น ถ้าเป็นคำสั่งอื่นแล้วจะถือว่าไม่ใช่จุดเชื่อมต่อระหว่างอุปกรณ์ในวงจรตรรกทันที ดังนั้นสิ่งนี้จึงเป็นสิ่งที่สำคัญและมีความจำเป็นอย่างยิ่งในการสร้างผังภาพวงจร

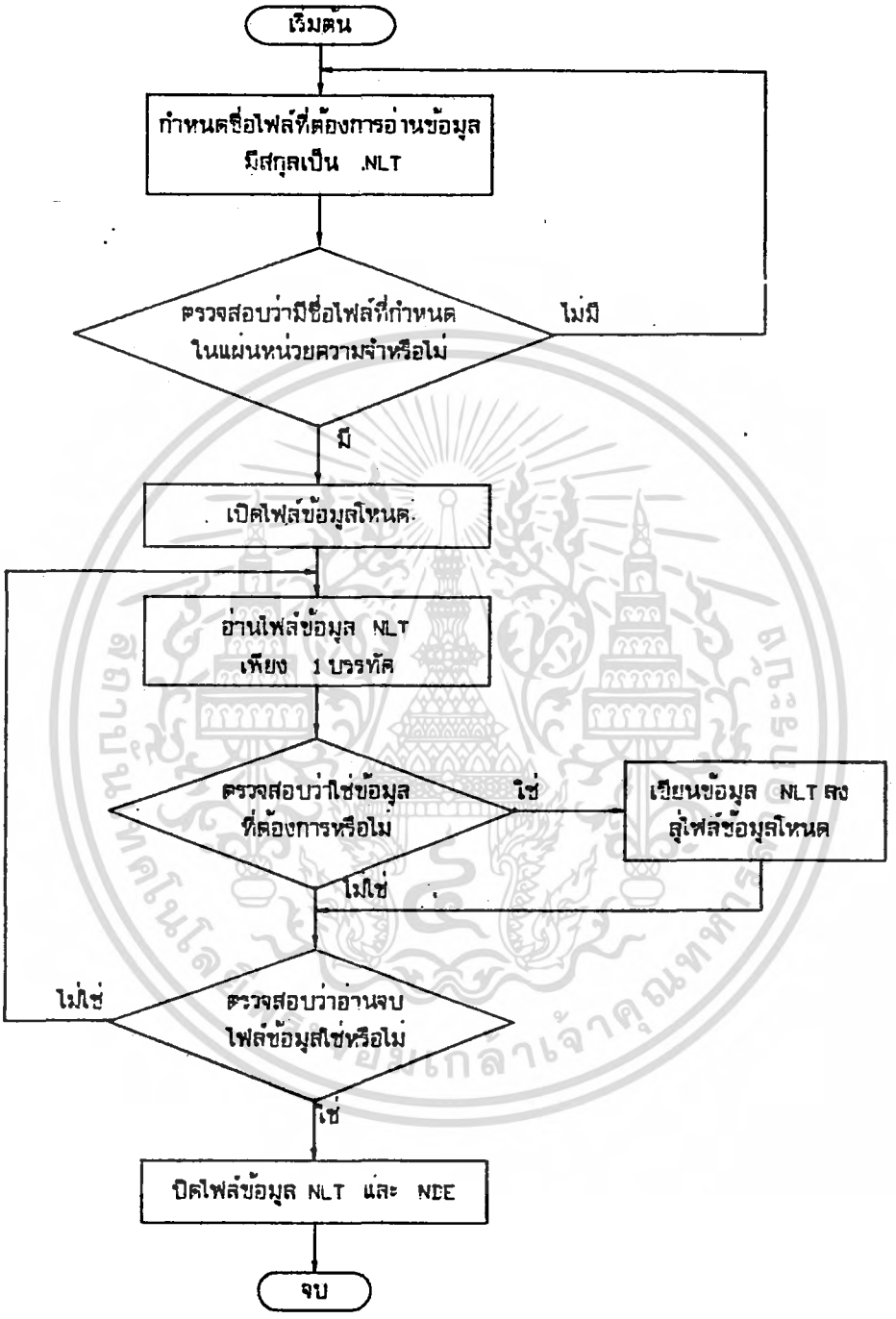
หลังจากที่ได้ผังภาพวงจรตามรูปที่ 4-3 จะได้ไฟล์ข้อมูลของผังภาพวงจรที่ผ่านการทำงานของโปรแกรม NETLIST ซึ่งจะบอกถึงรายละเอียดต่าง ๆ ของอุปกรณ์ที่ประกอบกันเป็นรูปตามผังภาพวงจรดังนี้ (ข้อมูลนี้ตัดเฉพาะบางส่วนของไฟล์ AxorB.NLT)

```
(view NETLIST root_NET
 (interface
  (define local signal ABAR)
  (define input port INPUTA))
  (define local signal BBAR)
  (define input port INPUTB))
  (define local signal N1)
  (define output port OUTPUT))
  (define local signal N2)
```

```
(contents
 (instance (qualify TTL_LIB X_74LS04) X_74LS04_NET U1A)
 (instance (qualify TTL_LIB X_74LS04) X_74LS04_NET U1B)
 (instance (qualify TTL_LIB X_74LS08) X_74LS08_NET U2A)
 (instance (qualify TTL_LIB X_74LS08) X_74LS08_NET U2B)
 (instance (qualify TTL_LIB X_74LS32) X_74LS32_NET U3A)
 (joined
```

ไฟล์ข้อมูลที่สร้างจากโปรแกรม NETLIST (FILENAME.NLT) โดยมีข้อมูลหลักมาจากการสร้างผังภาพวงจร ข้อมูลที่ได้จากโปรแกรม NETLIST นั้น เป็นข้อมูลที่มากเกินกว่าความต้องการ ดังนั้นจึงต้องมีการลดทอนข้อมูลเหล่านี้ลง เพื่อให้เหลือเฉพาะข้อมูลหลักที่เป็นอินพุท เอาท์พุท และจุดเชื่อมต่อระหว่างอุปกรณ์เท่านั้น ความต้องการลักษณะเช่นนี้ทำได้โดยการใช้โปรแกรม LOGNODE หลังจากนั้นจะได้ไฟล์ข้อมูลที่เป็นไฟล์ข้อมูลโหนด (FILENAME.NDE) ซึ่งถือว่าเป็นขั้นตอนสุดท้ายในส่วนแรกของโปรแกรม LOGCAD เมื่อสร้างผังภาพวงจรด้วย ORCAD เป็นไปตามรูปที่ 4-3 แล้วจะได้ไฟล์ข้อมูลโหนดดังนี้

```
ABAR
INPUTA
BBAR
INPUTB
N1
OUTPUT
N2
```



รูปที่ 4-4 แผนผังแสดงการทำงานของโปรแกรม LOGNODE

อธิบายแผนผังการทำงานของโปรแกรม LOGNODE

1. จะต้องมีการกำหนดชื่อไฟล์ข้อมูลที่ผ่านโปรแกรม NETLIST (FILENAME.NLT) นำมาตรวจสอบ และผลการตรวจสอบปรากฏว่าไม่มีไฟล์ข้อมูลดังกล่าวอยู่ในแผ่นหน่วยความจำโปรแกรมจะจัดการให้กำหนดชื่อไฟล์ข้อมูลใหม่อีกครั้ง แต่ถ้าปรากฏว่ามีไฟล์ข้อมูลดังกล่าว โปรแกรมจะทำการเปิดไฟล์ข้อมูลโหนด (FILENAME.NDE) ทันที
2. ทำการอ่านไฟล์ข้อมูล เพื่อที่จะได้นำมาตรวจสอบว่าเป็นข้อมูลโหนดที่ต้องการ ซึ่งข้อมูลเหล่านี้ประกอบด้วย
   - INPUT PORT
   - OUTPUT PORT
   - LOCAL SIGNAL

   ในจำนวน 3 ประเภท จะเป็นประเภทใดประเภทหนึ่งก็ตามโปรแกรมจะจัดการบันทึกข้อมูลเหล่านี้ลงสู่ไฟล์ข้อมูลโหนด แต่ถ้าไม่เป็นข้อมูลโหนดของวงจรโปรแกรมจะไม่นำข้อมูลเหล่านี้ปฏิบัติการใด ๆ ทั้งสิ้น
3. โปรแกรมจะตรวจสอบไฟล์ข้อมูลที่อ่านอยู่นั้นว่าหมดไฟล์แล้วหรือยัง ถ้ายังไม่หมดจะปฏิบัติในข้อ 2 แต่ถ้าหมดแล้วก็จะปิดไฟล์ข้อมูลต่าง ๆ และยกเลิกการทำงานของโปรแกรม

จากการสร้างผังภาพวงจรโดยโปรแกรม LOGCAD EDITOR และ ORCAD ดังรูป 4-1 และ 4-3 ตามลำดับ ทั้งสองโปรแกรมนี้จะให้ไฟล์ข้อมูลโหนดที่เหมือนกันซึ่งหมายถึง ชื่อของโหนด แต่จะแตกต่างกันเพียงตำแหน่งของข้อมูลวางสลับที่กันเท่านั้น สิ่งนี้ไม่เป็นอุปสรรคสำหรับการที่จะนำข้อมูลเหล่านี้ไปจำลองการทำงานในขั้นตอนต่อไป เพราะถึงอย่างไรก็ตามจะต้องมีการนำข้อมูลเหล่านี้ไปตรวจสอบ เพื่อที่จะกำหนดให้เป็น ข้อมูลอินพุท เอาท์พุท หรือจุดเชื่อมต่ออีกครั้ง ซึ่งจะได้กล่าวในบทที่ 6 ต่อไป

สรุป ในบทที่ 4 กล่าวถึงการสร้างไฟล์ข้อมูลโหนด (FILENAME.NDE) ซึ่งนับว่าเป็นสิ่งจำเป็นอย่างยิ่งในการที่จะนำไปเป็นข้อมูลหลักสำหรับการจำลองการทำงาน และวิธีการสร้างไฟล์ข้อมูลโหนด นอกจากจะเป็นวิธีการสร้างโดย LOGNODE 1 และ LOGNODE แล้วยังสามารถสร้างโดยวิธีลัดได้อีกวิธีหนึ่งในกรณีที่ผู้ใช้งานทราบชื่อโหนดต่าง ๆ ของวงจรตรรก และ ทราบถึงลักษณะรูปแบบของข้อมูลที่เป็นไฟล์ข้อมูลโหนด แล้วจึงทำการเขียนชื่อโหนดเหล่านี้ลงยังไฟล์ที่มีชื่อหลังจุดเป็น NDE ได้จากแป้นพิมพ์โดยตรงหรือเป็นการสร้างโดยโปรแกรมอิดิเตอร์อื่น ๆ จะทำให้ได้ไฟล์ข้อมูลโหนดเช่นกัน

บทที่ 5

การใช้ TURBO PASCAL เป็นตัวแปลภาษาให้กับแบบจำลองของวงจรตรรก

ในการที่จะทำให้โปรแกรมใด ๆ ก็ตามสามารถจำลองการทำงานของวงจรให้ได้ทุกเงื่อนไข และ ทุกฟังก์ชันการทำงานนั้นย่อมเป็นสิ่งที่ยากถ้าไม่ใช้วิธีการของ ตัวแปรภาษา (COMPILER) และโปรแกรม LOGCAD SIMULATOR ก็ได้ใช้วิธีการนี้เช่นเดียวกัน โดยการเขียนเงื่อนไขและฟังก์ชันต่าง ๆ ให้เป็นสมการบูลลีนที่จัดอยู่ในรูปของโปรแกรม TURBO PASCAL แล้วอาศัยตัวแปลภาษาของ TURBO PASCAL นี้ช่วยในการแปลฟังก์ชันการทำงานต่าง ๆ เหล่านี้ให้ ฉะนั้นโปรแกรม LOGCAD จึงสามารถจำลองได้ทุกเงื่อนไขและทุกฟังก์ชันการทำงาน ตัวอย่างการสร้างฟังก์ชัน เช่น ฟลิบฟลอบ (FLIP-FLOP) ประเภทต่าง ๆ ตัวถอดรหัส (DECODER) ตัวเข้ารหัส (ENCODER) และฟังก์ชันประเภทอื่น ๆ เป็นต้น

แต่เดิมโปรแกรม LOGCAD ถูกพัฒนาบน TURBO PASCAL VERSION 3.1 ซึ่งมีความสามารถของโปรแกรมอยู่ในขีดจำกัด จึงทำให้การพัฒนาโปรแกรม LOGCAD ไม่สามารถขยายระบบให้เป็นระบบที่สมบูรณ์แบบได้ เช่น ในการแปลภาษาจะทำในลักษณะ COMMAND LINE ไม่ได้ กล่าวคือการแปลภาษาแต่ละครั้งผู้ใช้จะต้องเรียกเข้าไปยังโปรแกรมของ TURBO PASCAL ทุกครั้ง ซึ่งนับว่าเป็นสิ่งที่ยุ่งยากและไม่เหมาะที่จะนำมาเป็นโปรแกรมใช้งาน

เพราะการที่ TURBO PASCAL VERSION 3.1 ไม่สามารถทำการแปลภาษาแบบ COMMAND LINE ดังนั้นวิธีการที่จะกำหนดฟังก์ชั่นให้เป็นลักษณะของโปรแกรมโดยผู้ใช้นั้นย่อมเป็นไปไม่ได้ แต่เพื่อให้โปรแกรม LOGCAD สามารถจำลองการทำงานได้จึงใช้วิธีบรรจุฟังก์ชันต่าง ๆ ลงในตัวซอร์สโปรแกรม (SOURCE PROGRAM) ของ LOGCAD เลย ดังนั้นถ้าผู้ใช้ต้องการใช้งานในฟังก์ชันใดก็สามารถเรียกได้จากโปรแกรม LOGCAD ได้โดยตรง วิธีการนี้จะสะดวกที่ผู้ใช้ไม่ต้องเขียนเงื่อนไขของฟังก์ชันขึ้นเอง เพราะผู้สร้างโปรแกรมได้บรรจุฟังก์ชันเหล่านี้เอาไว้เรียบร้อยแล้ว แต่จะมีข้อเสียอยู่ที่ขีดจำกัดของโปรแกรม เพราะจะไม่สามารถจำลองการทำงานฟังก์ชันที่นอกเหนือไปจากฟังก์ชันที่บรรจุเอาไว้ได้ หรือ ในกรณีที่ต้องการให้โปรแกรมสามารถจำลองการทำงานได้มากฟังก์ชันมากเงื่อนไขตัวซอร์สโปรแกรมก็จะต้องบรรจุฟังก์ชันต่าง ๆ เหล่านี้เพิ่มมากขึ้น ซึ่งมีผลทำให้โปรแกรมมีขนาดใหญ่เกินความจำเป็น วิธีการนี้จึงไม่ใช่วิธีปฏิบัติที่ถูกต้อง

ประการสำคัญย่อยเป็นสิ่งที่ยากมากและเป็นไปไม่ได้ ที่จะบรรจุเอาฟังก์ชันที่มีอยู่ทั้ง

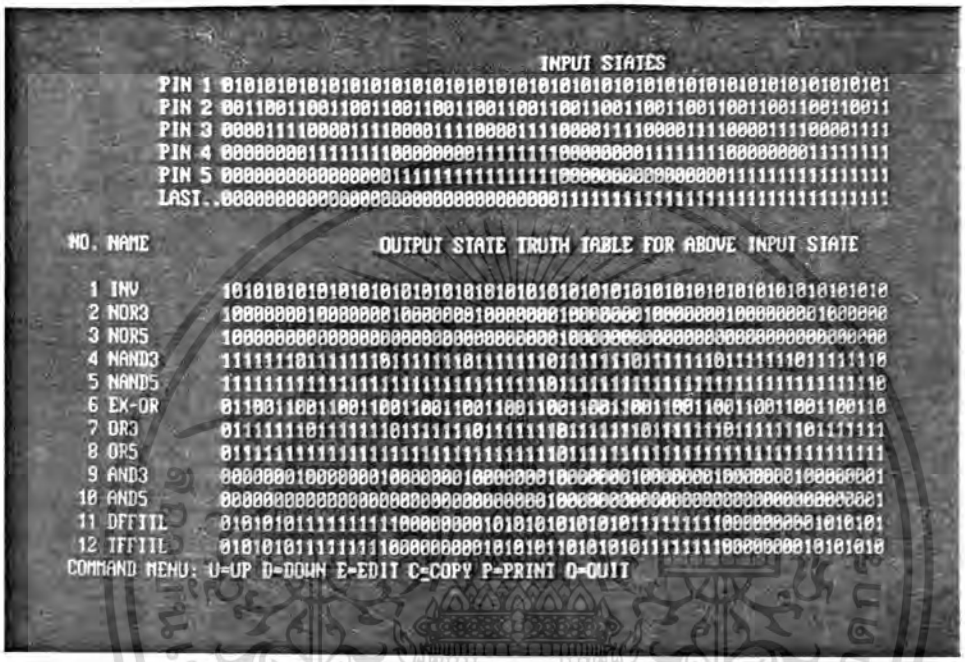
หมดในคู่มือไอซีทีทีแอล (TTL DATA BOOK) ลงในส่วนของซอร์สโปรแกรม LOGCAD ได้ จากเหตุผลที่กล่าวมาจะเห็นได้ว่าโปรแกรม LOGCAD ในอดีตที่ถูกพัฒนาบน TURBO PASCAL VERSION 3.1 ไม่เหมาะที่จะเป็นโปรแกรมใช้งาน แต่ถึงอย่างไรก็ตามการการพัฒนาโปรแกรม LOGCAD บน TURBO PASCAL VERSION 4.0 นี้ก็ได้อาศัยแนวทางที่ได้จากการพัฒนาโปรแกรม LOGCAD ในอดีตที่ผ่านมา

```
JKFFNL : begin
         { Negative edge-triggered JK flip-flop with optional }
         { active-low set and reset }
         if (SimData^[PresentState,Network^[GateCount,Clk]] = 1) and
            (SimData^[NextState,Network^[GateCount,Clk]] = 0) then

          begin { Clock Input JK-FF }
           if (SimData^[NextState,Network^[GateCount,PinJ]] = 1) and
              (SimData^[NextState,Network^[GateCount,PinK]] = 1) then
              Invert (SimData^[NextState,Network^[GateCount,PinQ]],
                      SimData^[NextState,Network^[GateCount,PinQ]]);
           if (SimData^[NextState,Network^[GateCount,PinJ]] = 0) and
              (SimData^[NextState,Network^[GateCount,PinK]] = 1) then
              SimData^[NextState,Network^[GateCount,PinQ]] := 0;
           if (SimData^[NextState,Network^[GateCount,PinJ]] = 1) and
              (SimData^[NextState,Network^[GateCount,PinK]] = 0) then
              SimData^[NextState,Network^[GateCount,PinQ]] := 1;
           if (SimData^[NextState,Network^[GateCount,PinJ]] = 0) and
              (SimData^[NextState,Network^[GateCount,PinK]] = 0) then
               SimData^[NextState,Network^[GateCount,PinQ]] :=
               SimData^[PresentState,Network^[GateCount,PinQ]] ;
          end; { Clock Input JK-FF }

          if SimData^[NextState,Network^[GateCount,PinS]] = 0 then
             SimData^[NextState,Network^[GateCount,PinQ]] := 1;
          if SimData^[NextState,Network^[GateCount,PinR]] = 0 then
             SimData^[NextState,Network^[GateCount,PinQ]] := 0;

           Invert (SimData^[NextState,Network^[GateCount,PinQ]],
                   SimData^[NextState,Network^[GateCount,PinQBar]]);
        end;
```

รูปที่ 5-1 ตัวอย่างฟังก์ชันที่บรรจุในซอร์สโปรแกรม LOGCAD ในอดีต

อีกวิธีการหนึ่ง ที่โปรแกรมบางโปรแกรมได้ใช้ในการสร้างฟังก์ชันการทำงานหลาย ๆ ฟังก์ที่แตกต่างกันออกไป โดยการสร้างตารางให้มีลักษณะคล้ายกับตารางความจริง ดังรูปที่ 5-2 ย่อมเป็นสิ่งที่ยากอีกเช่นกันสำหรับการที่จะบรรจุฟังก์ชันการทำงานทั้งหมดที่มีอยู่ลงในตารางความจริงนี้ได้ ประการสำคัญจำนวนอินพุท- เอาท์พุท

ของตารางที่มีไว้สำหรับกำหนดฟังก์ชันนั้นมีจำนวนน้อยเกินไป ที่จะกำหนดให้กับบางฟังก์ชัน ตัวอย่างเช่น ตัวถอดระหัส ตัวเข้าระหัส หน่วยความจำ เป็นต้น ดังนั้นฟังก์ชันเหล่านี้จะไม่สามารถจำลองการทำงานได้เลย

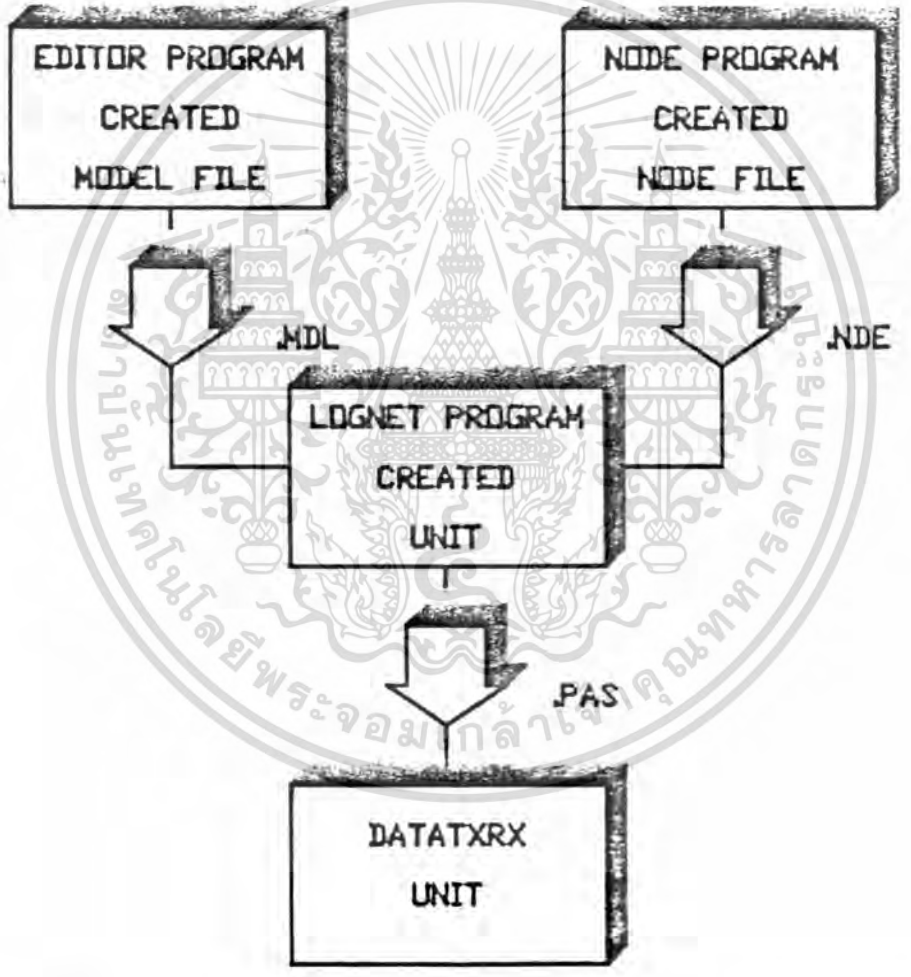


รูปที่ 5-2 การสร้างตารางความจริงสำหรับฟังก์ชัน ของโปรแกรม MICRO LOGIC

หลังจากที่ TURBO PASCAL VERSION 4.0 ได้มีใช้กันอยู่แพร่หลายจึงได้ศึกษาการใช้งานตลอดจนขีดความสามารถอื่น ๆ ปรากฏว่าสิ่งที่เป็นจุดบกพร่องของโปรแกรม LOGCAD ซึ่งได้พัฒนาในอดีตนั้นสามารถปรับปรุงแก้ไขให้มีสมรรถนะสูงขึ้นได้

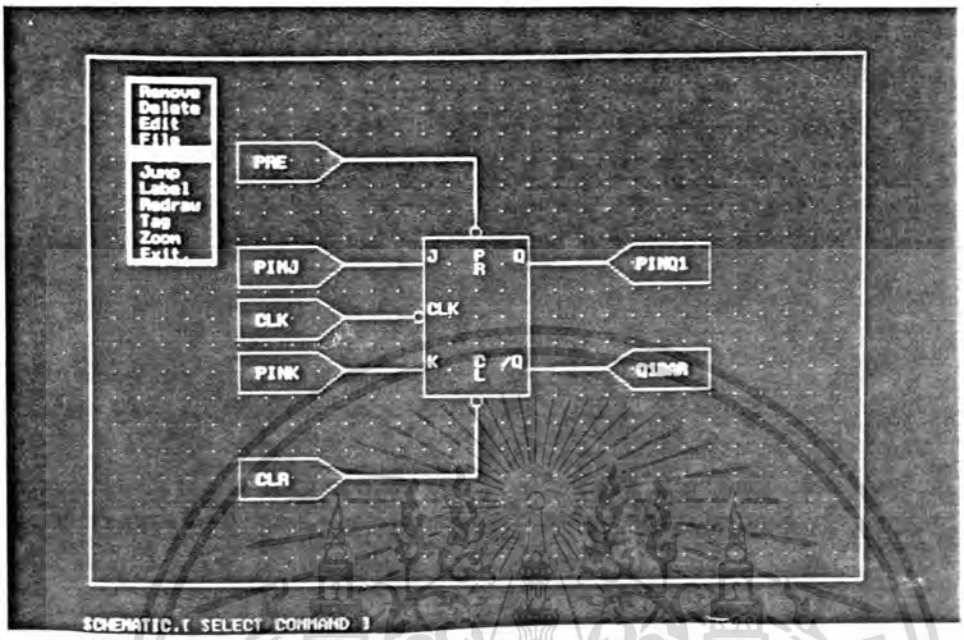
TURBO PASCAL VERSION 4.0 เป็นโปรแกรมที่มีความคล่องตัวในการใช้งานสูง และสามารถทำการแปลงภาษาแบบ COMMAND LINE ประกอบกับการแปลภาษาให้กับโปรแกรมย่อยในแต่ละครั้ง ก็สามารถเก็บโปรแกรมย่อยที่ได้แปลแล้วเหล่านี้ เอาไว้เป็นส่วน ๆ เพื่อที่จะได้ไม่ต้องแปลโปรแกรมย่อยนี้ซ้ำอีกครั้งกรณีที่มีการเปลี่ยนแปลง แก้ไขบางส่วนของโปรแกรมหลัก หรือโปรแกรมย่อยอื่น ๆ ซึ่งไม่เกี่ยวข้องกับส่วนของโปรแกรมย่อยที่ได้แปลเอาไว้แล้ว หรือกล่าวได้อีกลักษณะหนึ่งคือ สามารถทำให้เป็นยูนิต (UNIT) ได้นั้นเอง จากเหตุผลดังกล่าวนี้นับว่าเป็นสิ่งที่จำเป็นสำหรับโปรแกรมที่ต้องการแปลภาษาทุกครั้งเมื่อมีการเปลี่ยนแปลงบางส่วนของโปรแกรมอยู่เสมอ ๆ อย่างเช่นโปรแกรม LOGCAD ที่จะต้องบรรจุแบบจำลองลงในส่วนของโปรแกรม และแบบจำลองเหล่านี้มีการเปลี่ยนแปลง

อยู่เสมอตามลักษณะของผังภาพวงจรที่เปลี่ยนไป ดังนั้นจึงต้องมีการแปลภาษาใหม่ทุกครั้งที่มีการเปลี่ยนแปลงวงจรการทำงาน และการแปลภาษาภาษาผู้ใช้ไม่ต้องเรียกเข้ายังโปรแกรมอิดิเตอร์ของ TURBO PASCAL แต่อย่างใด เพียงแต่ทำเป็น BATCH FILE ก็เพียงพอสำหรับการแปลภาษา จึงสรุปได้ว่าในการจำลองการทำงานวิธีการของตัวแปลภาษานับได้ว่าเป็นวิธีที่ดีที่สุดเมื่อเทียบกับวิธีอื่น ๆ ตามที่ได้กล่าวมา



รูปที่ 5-3 แผนผังแสดงการทำงานของโปรแกรมการสร้างยูนิตให้เป็นโมดูลระดับไฟล์

ในการอ้างอิงถึงส่วนต่าง ๆ ที่ใช้เป็นข้อมูลสำหรับการสร้างยูนิตใหม่จะถือเอาผังภาพวงจร JKFFNL (NEGATIVE EDGE-TRIGGERED JK FLIP-FLOP WHIT OPTION ACTIVE-LOW SET AND RESET) เป็นวงจรตัวอย่างที่ใช้สำหรับการสร้างยูนิตใหม่นี้



รูปที่ 5-4 ตัวอย่างผังภาพวงจร JKFFNL (JKFFNL.SCH)

จากรูปที่ 5-3 เป็นการสร้างยูนิตใหม่จากไฟล์ข้อมูลโดยมีขั้นตอนการทำงานดังนี้

5.1 ไฟล์ข้อมูลโหนด และ ไฟล์ข้อมูลแบบจำลอง (NODE FILE & MODEL FILE)

5.1.1 ไฟล์ข้อมูลโหนด (NODE FILE)

ไฟล์ข้อมูลลักษณะนี้ได้กล่าวมาแล้วในบทที่ 4 คือเป็นโหนดของผังภาพวงจรที่สร้างมาจากโปรแกรม LOGCAD EDITOR หรือ โปรแกรม ORCAD เพื่อใช้เป็นข้อมูลในการสร้างยูนิตใหม่คือ DATATXRX UNIT จากรูปที่ 5-4 จะได้ไฟล์ข้อมูลโหนด JKFFNL.NDE ดังนี้

```
PRE
PINJ
PINQ1
CLK
PINK
Q1BAR
CLR
```

5.1.2 ไฟล์ข้อมูลแบบจำลอง (MODEL FILF)

เป็นแบบจำลองที่ผู้ใช้จะต้องสร้างขึ้นเองจากโปรแกรมอิดิเตอร์ใดก็ได้ เช่น SIDE-KICK WORDSTAR TURBO PASCAL หรืออื่น ๆ เป็นต้น แบบจำลองนี้จะต้องสร้างในลักษณะของความสัมพันธ์ระหว่างโหนด โดยที่โหนดแต่ละโหนดมีความสัมพันธ์กันอย่างต่อเนื่องและเขียนโหนดเหล่านี้ให้เป็นสมการบูลลีนในรูปของโปรแกรม ที่สอดคล้องกับผังภาพวงจร นอกจากนี้รูปแบบในการสร้างแบบจำลอง จะต้องเป็นไปตามข้อกำหนดของ TURBO PASCAL กล่าวคือคำสั่งปฏิบัติจะต้องอยู่ระหว่างคำว่า BEGIN กับ END สำหรับข้อกำหนดอื่น ๆ นั้นสามารถศึกษารายละเอียดได้จากหนังสือ TURBO PASCAL OWNER'S HANDBOOK หนังสือการเรียนรู้ภาษาปาสคาล หรือ หนังสืออื่น ๆ ที่เขียนเกี่ยวกับ TURBO PASCAL ที่สำคัญในส่วนของการเขียนแบบจำลองนี้สามารถสร้างตัวแปรชนิด บูลลีนเพิ่มเติมขึ้นใหม่ได้ ซึ่งเป็นตัวแปรที่นอกเหนือไปจากตัวแปรเหล่านี้

- ชื่อโหนดของผังภาพวงจร
- ชื่อในคำสงวน (RESERVED WORDS) ของ TURBO PASCAL
- ชื่อตัวแปรร่วม (GLOBAL) ของโปรแกรม LOGCAD SIMULATOR จะได้กล่าวในภาคผนวก 1

การที่สร้างตัวแปรในแบบจำลองเพิ่มเติมใหม่ จุดประสงค์เพื่อเก็บค่าสภาวะไว้ชั่วคราว ทั้งนี้เนื่องจากว่ามีบางวงจรต้องมีการเก็บค่าสภาวะลักษณะนี้ไว้ใช้เป็นเงื่อนไขในการตรวจสอบ ตัวอย่างเช่นวงจรลำดับ (SEQUENTIAL CIRCUIT) เป็นต้น ซึ่งจะต้องตรวจสอบค่าสภาวะในสเต็ปการทำงานที่ผ่านมา กับ สเต็ปการทำงานปัจจุบัน ในการที่จะกำหนดสภาวะให้กับตัวแปรที่เกี่ยวข้อง เป็นต้น แต่ถ้าเป็นวงจรซึ่งไม่ต้องมีการเก็บค่าสภาวะชั่วคราวนี้เอาไว้ ก็ไม่จำเป็นต้องสร้างตัวแปรเพิ่มเติม ตัวอย่างเช่นวงจรที่ทำพีชคณิตบูลลีน (BOOLEAN ALGEBRA)

รูปแบบมาตรฐานของการสร้างแบบจำลอง

```
VAR  VARIABLE 1 : BOOLEAN;
     VARIABLE 2 : BOOLEAN;
          .          .
          .          .
     VARIABLE N : BOOLEAN;
```

```
BEGIN
  STATEMENT 1 ;
  STATEMENT 2 ;
    .     .
    .     .
  STATEMENT N ;
END ;
```

ตัวอย่าง การสร้างแบบจำลองของฟังก์ชัน JKFFNL (JKFFNL.MDL) ตามรูปที่ 5-4

```
VAR LAST1 : BOOLEAN;
BEGIN
  IF(PRE=ON)AND(CLR=ON)AND(CLK=OFF)AND(LAST1=ON) THEN
   BEGIN
     IF(PINJ=ON)AND(PINK=ON) THEN
       PINQ1 := NOT(PINQ1);
     IF(PINJ=ON)AND(PINK=OFF) THEN
       PINQ1 := ON;
     IF(PINJ=OFF)AND(PINK=ON) THEN
       PINQ1 := OFF;
     IF(PINJ=OFF)AND(PINK=OFF) THEN
       PINQ1 := PINQ1;
   END;
  IF(CLR=OFF) THEN
    PINQ1 := OFF;
  IF(PRE=OFF)AND(CLR=ON) THEN
    PINQ1 := ON;
  LAST1 := CLK;
  Q1BAR := NOT(PINQ1);
END;
```

5.2 DATATXRX UNIT

จะเป็นยูนิตที่สร้างขึ้นมาใหม่แล้วเก็บลงสู่ไฟล์ที่มีชื่อ DATATXRX.PAS ในการสร้างนี้สามารถนำข้อมูลมาจากไฟล์ข้อมูลโหนด และ ไฟล์ข้อมูลแบบจำลองดังที่กล่าวมาแล้ว และวิธีการ ทำได้โดยเรียกใช้โปรแกรม LOGNET ซึ่งเป็นโปรแกรมเพื่อใช้ในการสร้าง UNIT นี้โดยเฉพาะ

รูปแบบมาตรฐานของการสร้างยูนิต

```
UNIT UNIT_NAME;
INTERFACE
```

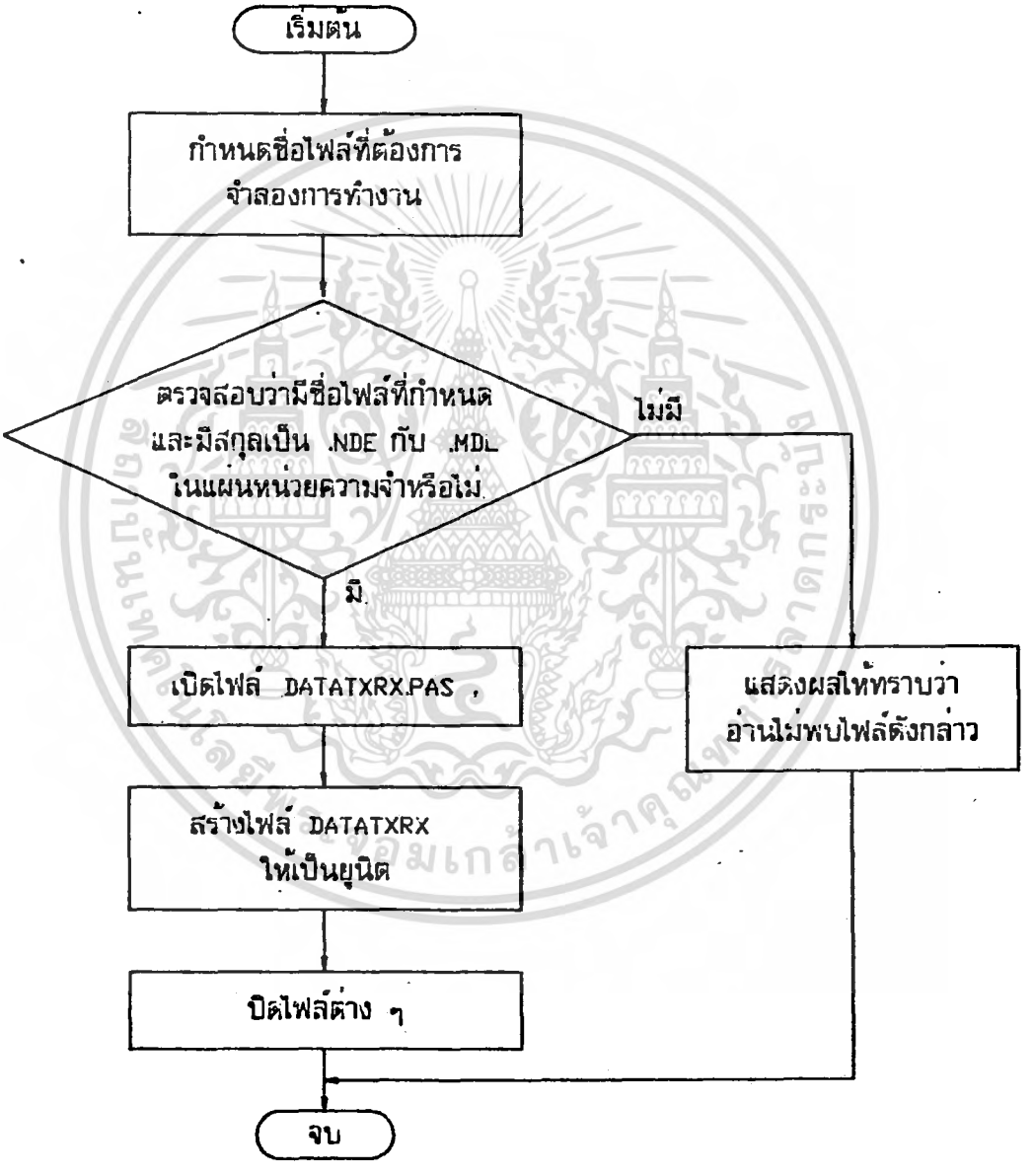
```
USES CRT,GRAPH,...;
VAR  VARIABLE 1 : BOOLEAN;
          .       .
          .       .
     VARIABLE N : BOOLEAN;
PROCEDURE 1;
     .
     .
PROCEDURE N;
INPREMENTATION;
PROCEDURE 1 ;
BEGIN
  STATEMENT 1 ;
END;
     .
     .
PROCEDURE N ;
BEGIN
  STATEMENT N ;
END;
BEGIN { Main Unit }
END. { Main Unit }
```

ตัวอย่าง การสร้าง DATATXRX UNIT ใหม่ของผังภาพวงจรตาม รูปที่ 5-4

```
UNIT DATATXRX;
INTERFACE
USES CRT,GRAPH,DECLARE,SIMUNIT1,SIMUNIT2;
VAR
   PRE   : BOOLEAN;
   PINJ  : BOOLEAN;
   PINQ1 : BOOLEAN;
   CLK   : BOOLEAN;
   PINK  : BOOLEAN;
   Q1BAR : BOOLEAN;
   CLR   : BOOLEAN;
   LAST1 : BOOLEAN;
PROCEDURE MoveNodeToNet;
PROCEDURE MoveNetToNode;
PROCEDURE SimulationModel;
IMPREMENTATION
PROCEDURE MoveNodeToNet;
  BEGIN
    NETWORK[1]^.STATUS := PRE;
    NETWORK[2]^.STATUS := PINJ;
    NETWORK[3]^.STATUS := PINQ1;
    NETWORK[4]^.STATUS := CLK;
    NETWORK[5]^.STATUS := PINK;
    NETWORK[6]^.STATUS := Q1BAR;
    NETWORK[7]^.STATUS := CLR;
END;
PROCEDURE MoveNetToNode;
  BEGIN
    PRE := NETWORK[1]^.STATUS;
    PINJ := NETWORK[2]^.STATUS;
    PINQ1 := NETWORK[3]^.STATUS;
    CLK := NETWORK[4]^.STATUS;
    PINK := NETWORK[5]^.STATUS;
    Q1BAR := NETWORK[6]^.STATUS;
    CLR := NETWORK[7]^.STATUS;
  END;
PROCEDURE SimulationModel;
  BEGIN
    IF(PRE=ON)AND(CLR=ON)AND(CLK=OFF)AND(LAST1=ON) THEN
      BEGIN
        IF(PINJ=ON)AND(PINK=ON) THEN
          PINQ1 := NOT(PINQ1);
        IF(PINJ=ON)AND(PINK=OFF) THEN
          PINQ1 := ON;
        IF(PINJ=OFF)AND(PINK=ON) THEN
          PINQ1 := OFF;
        IF(PINJ=OFF)AND(PINK=OFF) THEN
          PINQ1 := PINQ1;
      END;
    IF(CLR=OFF) THEN
      PINQ1 := OFF;
    IF(PRE=OFF)AND(CLR=ON) THEN
      PINQ1 := ON;
    LAST1 := CLK;
    Q1BAR := NOT(PINQ1);
  END;
BEGIN  { Main Unit }
  PRE   := OFF;
  PINJ  := OFF;
  PINQ1 := OFF;
  CLK   := OFF;
  PINK  := OFF;
  Q1BAR := OFF;
  CLR   := OFF;
END. { Main Unit }
```

5.3 LOGNET

เป็นส่วนของโปรแกรม LOGCAD ที่ใช้อ่านไฟล์ข้อมูลโหนด กับ ไฟล์ข้อมูลแบบจำลอง มาจัดเรียงใหม่ให้เป็นรูปแบบของยูนิต จากนั้นเก็บลงสู่ไฟล์อีกครั้งโดยที่ไฟล์ที่สร้างขึ้นมาใหม่นี้คือ DATATXRX.PAS ตามที่ได้กล่าวมาแล้วในหัวข้อ 5.2 ยูนิตประเภทนี้โปรแกรมหลักสามารถเรียกโปรแกรมย่อยภายในยูนิตใช้งานได้ เมื่ออยู่ในขั้นตอนการประมวลผล



รูปที่ 5-5 ผังแสดงการทำงานของโปรแกรม LOGNET

สรุป ในบทที่ 5 กล่าวถึงวิธีที่จะสร้างฟังก์ชันเพื่อจำลองการทำงานของวงจรสามารถทำได้หลายวิธี แต่วิธีที่ดีที่สุดคือการใช้ตัวแปลภาษา และ โปรแกรม LOGCAD ก็ได้ใช้วิธีการนี้ด้วยเช่นกัน นอกจากนี้ยังกล่าวถึงการสร้างแบบจำลองที่มีรูปแบบแน่นอนตามลักษณะของ TURBO PASCAL ตลอดจนการสร้างยูนิตใหม่ (DATATXRX UNIT) ซึ่งนับว่าเป็นส่วนที่สำคัญอย่างยิ่งในการนำสมการบูลลีนที่เขียนเป็นแบบจำลองนี้ให้ตัวแปลภาษาของ TURBO PASCAL ทำการแปลแล้วสามารถนำไปใช้งานได้

บทที่ 6

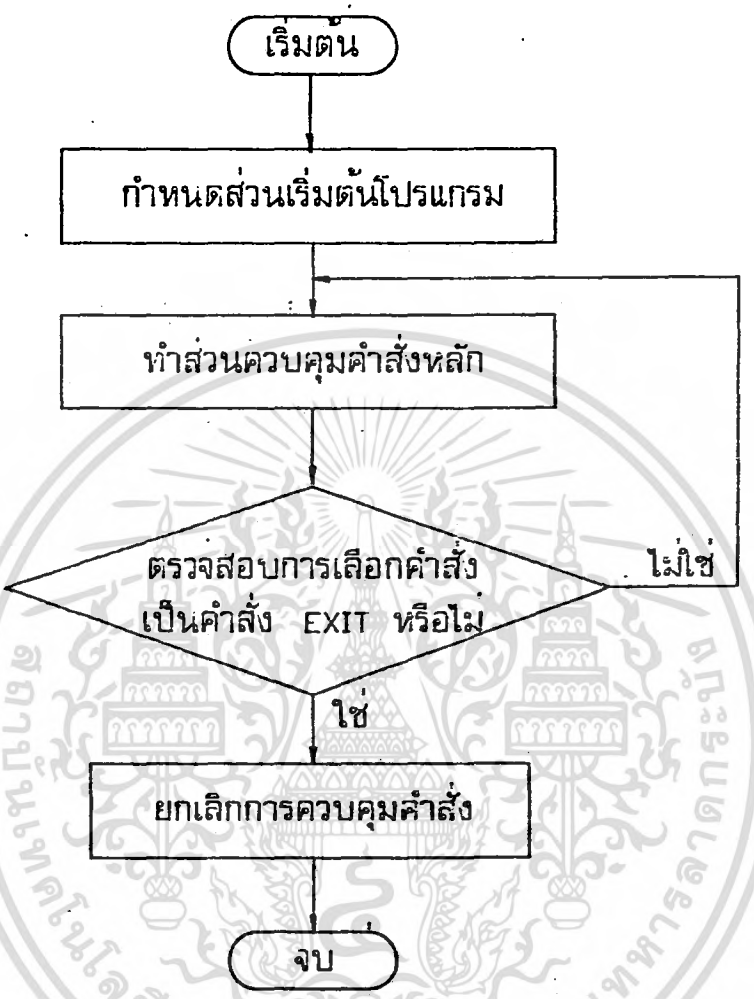
โปรแกรมจำลองการทำงานของวงจรตรรก

LOGCAD SIMULATOR เป็นโปรแกรมที่ใช้ในการจำลองการทำงานโดยมีหลักการพื้นฐานดังนี้ ก่อนอื่นจะต้องมีการกำหนดข้อมูล เข้าทางแป้นพิมพ์ ให้กับข้อมูลหลักที่เป็นตัวแปรอินพุทโหนด โดยที่ข้อมูล เข้า เหล่านี้จะกำหนดให้เป็นลักษณะของสัญญาณนาฬิกา (CLOCK) ตั้งแต่ส เต็ปการทำงานแรกจนกระทั่งส เต็ปการทำงานสุดท้ายลงสู่หน่วยความจำหลัก เพื่อเก็บค่าสภาวะของตัวแปรอินพุท เหล่านี้ไว้ จากนั้นจะมีการนำสภาวะของตัวแปรอินพุทผ่านฟังก์ชันการทำงานในหลาย ๆ ฟังก์ชันที่เขียนเป็นแบบจำลองเอาไว้ แล้วจึงนำค่าสภาวะที่ได้ เก็บยังตัวแปร เอาท์พุทโหนดของแต่ละฟังก์ชันอีกครั้ง การทำงานในลักษณะนี้ถือว่าเสร็จสิ้นการประมวลผลในหนึ่งส เต็ป สำหรับในส เต็ปอื่น ๆ ก็สามารถทำได้ในลักษณะเดียวกันจนกระทั่งถึงส เต็ปสุดท้าย เป็นอันสิ้นสุดการทำงานในส่วนของการประมวลผล และในการแสดงผลบนจอภาพนับได้ว่า เป็นกระบวนการสุดท้ายของการจำลองการทำงานวงจรตรรก การแสดงผลนั้นผู้ใช้สามารถกำหนดโหนดต่าง ๆ ของวงจรที่ต้องการแสดง หรือ เรียกอีกชื่อหนึ่งว่า โพรบ (PROBE) มาทำการแสดงได้โดยการนำโพรบ เหล่านั้น เรียงต่อกันตามลำดับ ก่อน-หลังได้อย่างอิสระบนจอภาพ เพื่อแสดง เป็นภาพแผนภูมิ เวลา ด้วยเหตุนี้จึงสามารถทำการ เปรียบ เทียบภาพแผนภูมิ เวลา ณ. โหนดต่าง ๆ ของวงจรได้ตามความต้องการ

โปรแกรม LOGCAD SIMULATOR สามารถแบ่งหัวข้อตามลักษณะการทำงานได้ดังนี้

- โปรแกรมหลัก
- โปรแกรมควบคุมการแสดงคำสั่ง
- โปรแกรมคำสั่งไฟล์ข้อมูล
- โปรแกรมการกำหนดข้อมูล เข้าของวงจรตรรก
- โปรแกรมการแสดงผล
- โปรแกรมการประมวลผล และการ เลื่อนภาพแผนภูมิ เวลา
- โปรแกรมการพิมพ์ภาพแผนภูมิ เวลา

6.1 โปรแกรมหลัก มีการทำงานของโปรแกรมตามแผนผังดังนี้



รูปที่ 6-1 แผนผังแสดงการทำงานของโปรแกรมหลัก

อธิบายแผนผังการทำงานของโปรแกรมหลัก

1. กำหนดส่วนเริ่มต้นของโปรแกรม

   - โดยการกำหนดการแสดงผลเป็นแบบกราฟฟิค สามารถแสดงได้กับจอภาพชนิดโมโนโครม (MONOCROME) และจอภาพสีความละเอียดสูง EGA (ENHANCE GRAPHIC ADAPTER) นอกจากนี้ยังมีการคำนวณเพื่อทำการปรับแต่งการแสดงผลของข้อความในโหมดกราฟฟิค (GRAPHIC MODE) ให้เป็นแบบ 80 แถวในแนวตั้ง และ 25 บรรทัดในแนวนอน เหมือนกับการแสดงผลของอักขระในโหมดข้อความ (TEXT MODE)

   - แสดงข้อความต่าง ๆ ดังรูปที่ 6-15 ในส่วนนี้ยังมีการใช้รหัสผ่าน (PASS WORD) เพื่อเข้าสู่โปรแกรมโดยที่รหัสผ่านคือ ชื่อของโปรแกรม "LOGCAD"

   - กำหนดค่าเริ่มต้นต่าง ๆ ให้กับตัวแปรในโปรแกรม

   - กำหนดการแสดงกริด ลักษณะของกริดที่แสดงเป็นแบบจุด 2 สี ทั้งนี้เพื่อใช้เป็นแนวอ้างอิงเมื่อเทียบกับแกนอ้างอิง

2. ทำงานส่วนควบคุมคำสั่งหลัก ในส่วนนี้จะทำหน้าที่ควบคุมการใช้งานของโปรแกรมที่เป็นคำสั่งหลักทั้งหมดอันได้แก่

   - โปรแกรมคำสั่งไฟล์ข้อมูล (FILE COMMAND)
   - โปรแกรมการกำหนดข้อมูลเข้าของวงจร (DATA COMMAND)
   - โปรแกรมการแสดงผล (DISP COMMAND)
   - โปรแกรมการประมวลผลและการเลื่อนภาพแผนภูมิเวลา (SIM COMMAND)
   - โปรแกรมเกี่ยวกับการพิมพ์ภาพแผนภูมิเวลา (PRINT COMMAND)
   - โปรแกรมออกจากการทำงาน (EXIT COMMAND)

3. ตรวจสอบการเลือกคำสั่ง ถ้าปรากฏว่าการเลือกคำสั่งไม่ใช่คำสั่ง EXIT แล้วให้กลับไปทำงานยังส่วนควบคุมคำสั่งหลักอีกครั้ง แต่ถ้าเป็นคำสั่ง EXIT จะต้องออกโปรแกรม LOGCAD SIMULATOR เพื่อกับเข้าสู่โปรแกรมการควบคุมระบบ (OS)

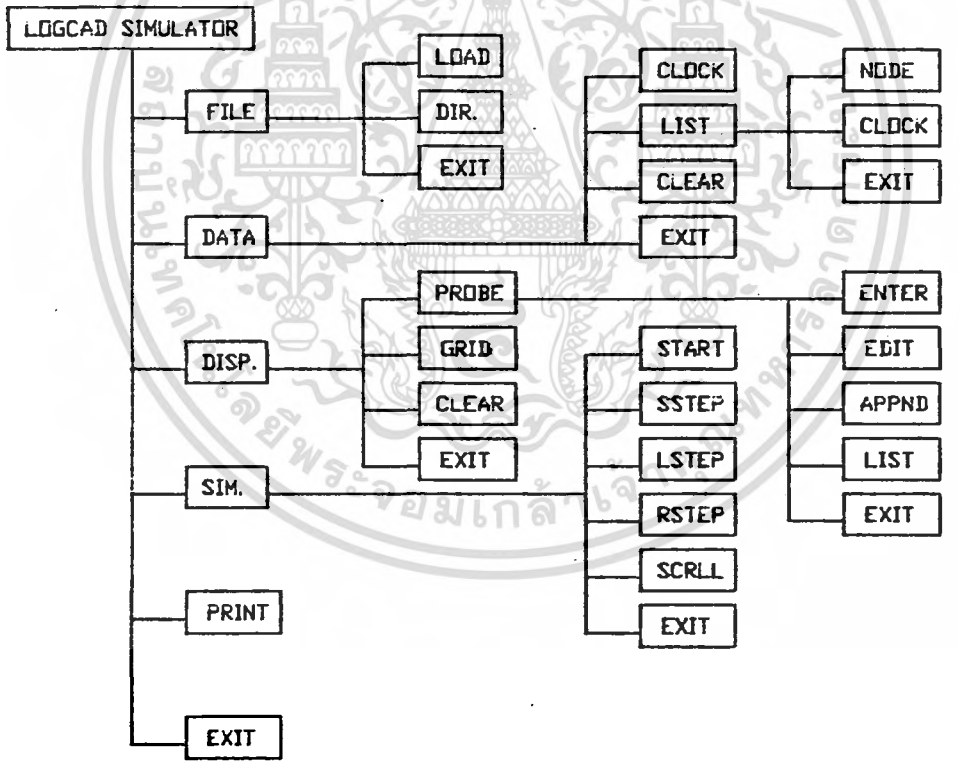
4. ยกเลิกการควบคุมคำสั่งต่าง ๆ ทั้งหมดและกำหนดการแสดงผลให้กลับเข้าสู่การแสดงผลแบบปรกติ (TEXT MODE)

6.2 โปรแกรมควบคุมการแสดงคำสั่ง

ในส่วนนี้มีหน้าที่ในการแสดงคำสั่งที่ใช้ในการทำงานทั้งหมดของโปรแกรมซึ่งผู้ใช้เป็นผู้เลือกตามขั้นตอนและมีการแสดงผลดังนี้

- ส่วนควบคุมคำสั่งหลัก FILE. DISP. SIM.. DATA. PRINT EXIT.
- ส่วนควบคุมเกี่ยวกับไฟล์ LOAD. DIREC ----- ----- ----- EXIT.
- ส่วนกำหนดข้อมูลเข้าให้กับวงจร CLOCK LIST. CLEAR ----- ----- EXIT.
- ส่วนแสดงผลที่ภาพแผนภูมิเวลา PROBE GRID. CLEAR ----- ----- EXIT.
- ส่วนแสดงจุดตรวจสอบ (PROBE) ENTER EDIT. APPND LIST. ----- EXIT.
- ส่วนการประมวลผลและการเลื่อนภาพ START SSTEP LSTEP RSTEP SCRLL EXIT.

การควบคุมการใช้คำสั่งในส่วนย่อยสามารถแสดงรายการตามแผนภาพรูปที่ 6-2

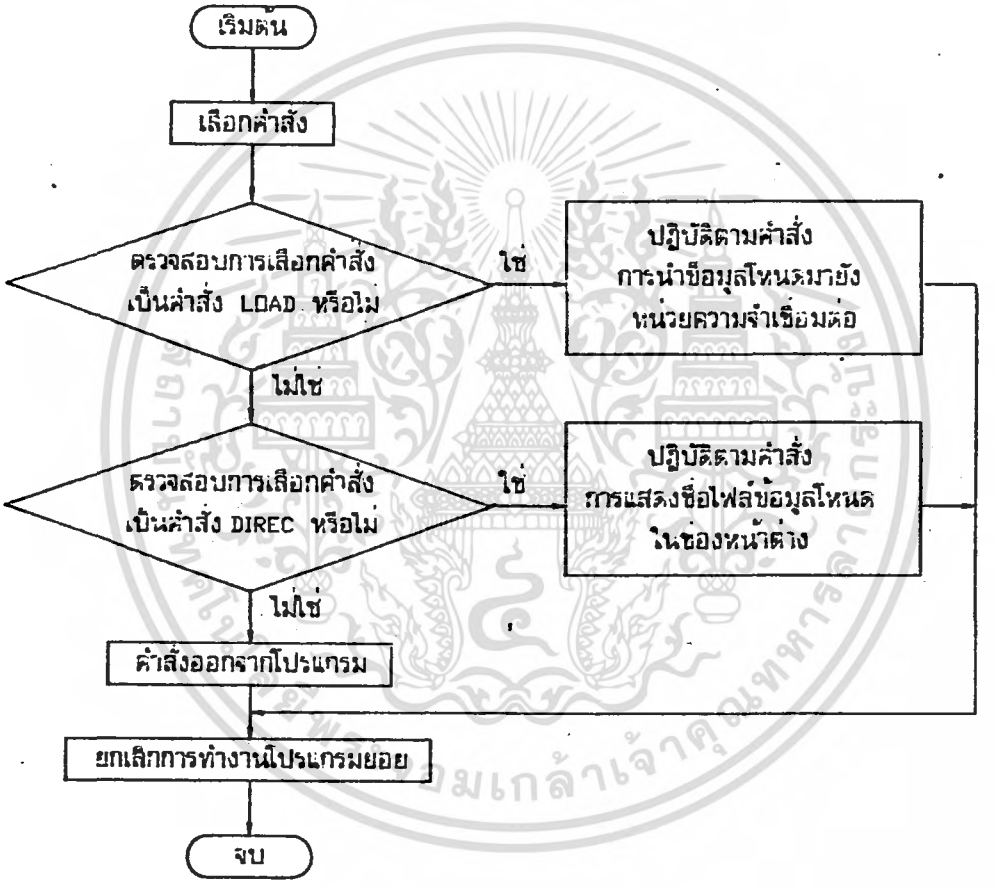


รูปที่ 6-2 แสดงผังการควบคุมคำสั่งเป็นแบบ ROOT COMMAND MENU

6.3 โปรแกรมคำสั่งไฟล์ข้อมูล (FILE COMMAND)

ในส่วนนี้จะมีหน้าที่ในการจัดการเกี่ยวกับไฟล์ข้อมูลโหนดเท่านั้น พอจะแยกประเภทตามลักษณะการทำงานดังนี้

- คำสั่งการอ่านไฟล์ข้อมูล (FILE/LOAD COMMAND)
- คำสั่งการแสดงไดเรคทอรี (FILE/DIREC COMMAND)
- คำสั่งออกจากโปรแกรมไฟล์ (FILE/EXIT COMMAND)



รูปที่ 6-3 แผนผังแสดงการทำงานของโปรแกรมจัดการไฟล์ข้อมูล

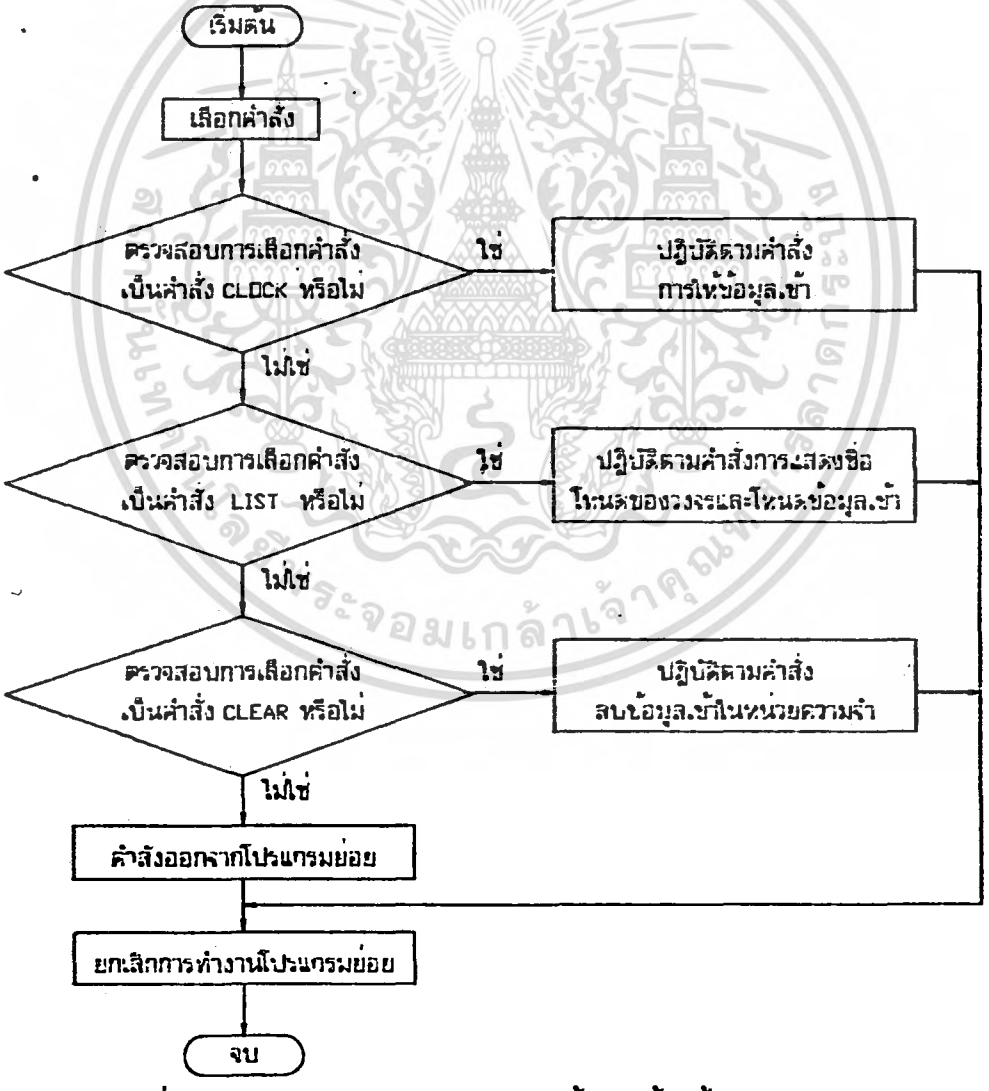
อธิบายแผนผังการทำงานของโปรแกรมคำสั่งเกี่ยวกับไฟล์ข้อมูล ดังรูปที่ 6-3

1. เลือกคำสั่งการทำงานในส่วนของเมนูย่อย และทำการตรวจสอบการเลือกคำสั่ง
2. ผลการตรวจสอบปรากฏว่าเป็นคำสั่ง FILE/LOAD ก็ให้ไปปฏิบัติตามคำสั่งคือ อ่านไฟล์ข้อมูลโหนดจากแผ่นหน่วยความจำลงสู่หน่วยความจำเชื่อมต่อ (NETWORK BUFFER) ที่เป็นข้อมูลแบบเรคอร์ดของชื่อโหนด ลำดับที่โหนด และ สภาวะของโหนดเพื่อใช้เป็นข้อมูลหลักสำหรับอ้างอิงในกรณีที่มีการเรียกใช้งาน
3. ผลการตรวจสอบปรากฏว่าเป็นคำสั่ง FILE/DIREC ให้ปฏิบัติตามคำสั่งการแสดงไดเรคทอรี โดยแสดงไฟล์ข้อมูลโหนดของวงจรในช่องหน้าต่าง โปรแกรมในส่วนนี้จำเป็นต้องเรียก ฟังก์ชันคอลล์ (CALL) ใน DOS โดยการใช้ INT 21 H
4. ยกเลิกการทำงานของโปรแกรมและกลับยังส่วนคำสั่ง FILE

6.4 โปรแกรมการกำหนดข้อมูลเข้าของวงจร (DATA COMMAND)

โดยข้อมูลเข้าเหล่านี้จะมีการกำหนดเป็นสัญญาณนาฬิกา ที่ผู้ใช้งานสามารถสร้างให้มีลักษณะตามความต้องการได้ทางแป้นพิมพ์ ในส่วนของคำสั่งนี้จะประกอบไปด้วยส่วนต่าง ๆ ดังนี้

- คำสั่งการให้ข้อมูลเข้า (DATA/CLOCK COMMAND)
- คำสั่งแสดงชื่อโหนดของวงจร (DATA/LIST/NODE COMMAND)
- คำสั่งแสดงชื่อโหนดข้อมูลเข้า (DATA/LIST/CLOCK COMMAND)
- คำสั่งออกจากคำสั่งการแสดงชื่อ (DATA/LIST/EXIT COMMAND)
- คำสั่งการลบข้อมูลที่เป็นข้อมูลเข้า (DATA/CLEAR COMMAND)
- คำสั่งออกจากการกำหนดข้อมูลเข้า (DATA/EXIT COMMAND)

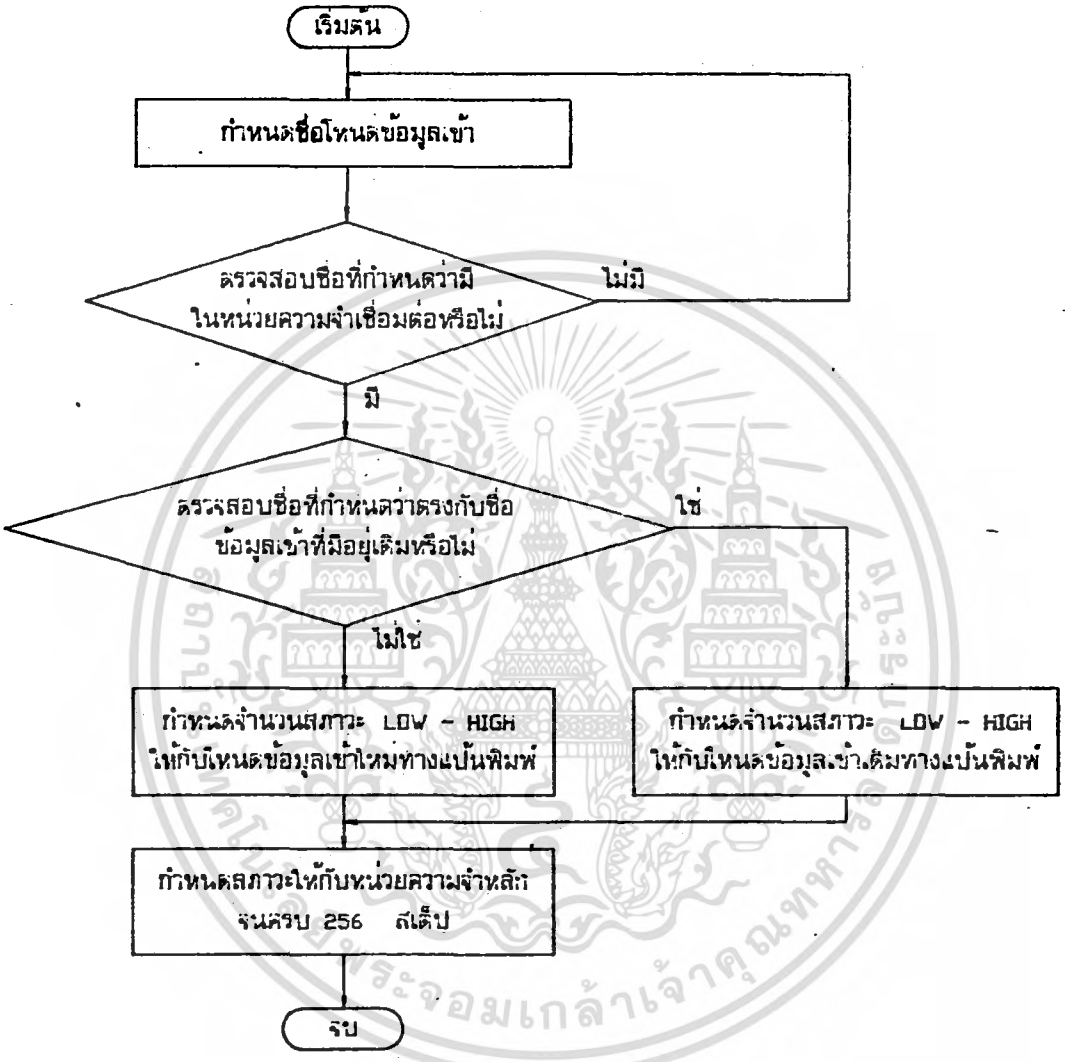


รูปที่ 6-4 แผนผังแสดงเมนูการกำหนดข้อมูลเข้าให้กับวงจร

อธิบายแผนผังโปรแกรม เกี่ยวกับการกำหนดข้อมูล เข้าของวงจรตรรก ดังรูปที่ 6-4 มีขั้นตอนการทำงานดังนี้

1. เลือกคำสั่งการทำงานในส่วนของ เมนูย่อย และ ทำการตรวจสอบคำสั่งนั้น ๆ
2. ผลการตรวจสอบว่า เป็นคำสั่ง DATA/CLOCK ให้ปฏิบัติตามคำสั่งการกำหนดข้อมูล เข้าสำหรับวงจรตรรก
3. ผลการตรวจสอบว่า เป็นคำสั่ง DATA/LIST ให้ปฏิบัติตามคำสั่งการแสดงชื่อของโหนดทั้งหมด หรือ ชื่อโหนดข้อมูล เข้าของวงจรบนส่วนแสดงผล เป็นแบบช่องหน้าต่าง
4. ผลการตรวจสอบว่า เป็นคำสั่ง DATA/CLEAR ให้ปฏิบัติตามคำสั่ง การลบชื่อของโพรบที่แสดงบนจอภาพ และ กำหนดข้อมูลในหน่วยความจำหลักที่ เป็น เฉพาะข้อมูล เข้าให้มีค่าสภาวะ เป็น "0" หรือ FALSE ทั้ง 256 สเต็ป
5. ยกเลิกการทำงานของโปรแกรมและกลับยังส่วนคำสั่ง DATA

6.4.1 โปรแกรมวิธีการให้ข้อมูลเข้าให้กับวงจร (DATA/CLOCK COMMAND) ดังรูปที่ 6-17 และ รูปที่ 6-18

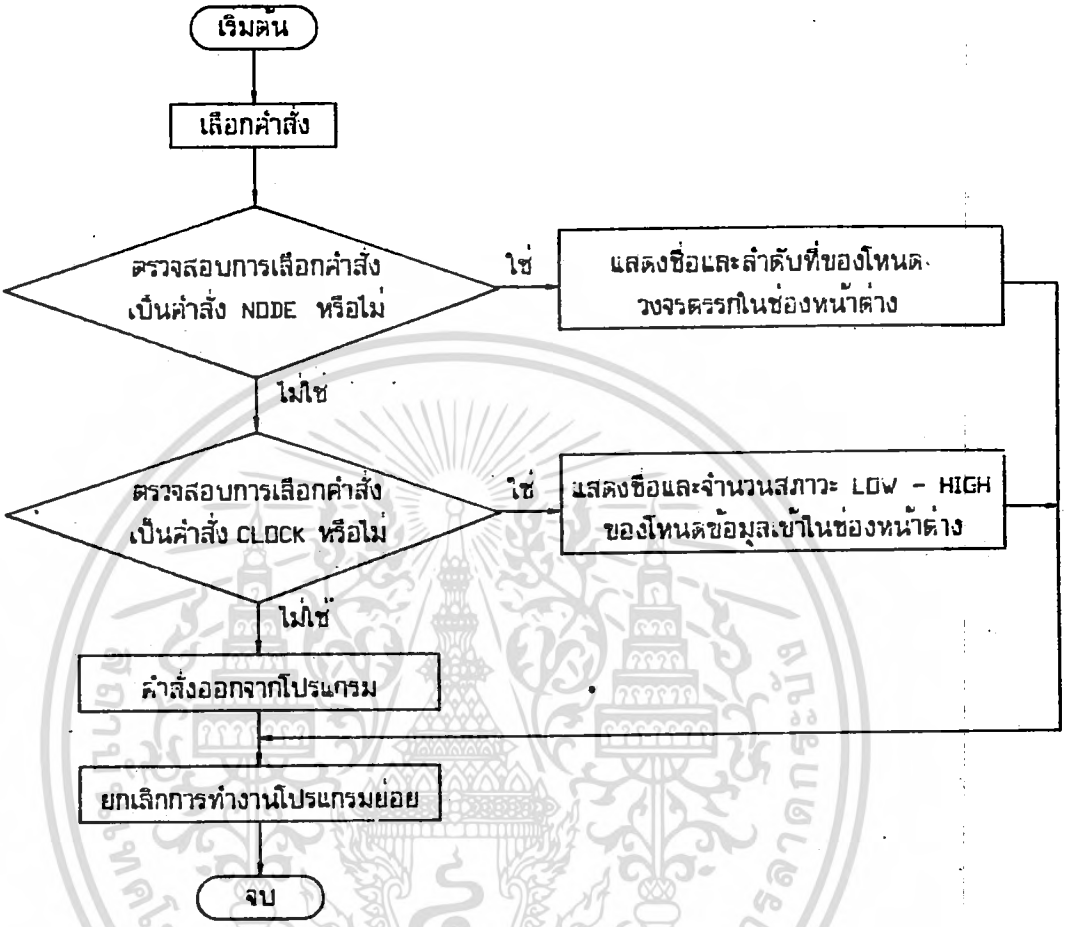


รูปที่ 6-5 แผนผังแสดงการทำงานของโปรแกรมกำหนดข้อมูลเข้า

อธิบายแผนผังการทำงานของโปรแกรมกำหนดข้อมูลเข้าให้กับวงจร ดังรูป 6-5

1. กำหนดชื่อโหนดที่จะให้เป็นข้อมูลเข้าของวงจรทางแป้นพิมพ์
2. ตรวจสอบชื่อที่กำหนดมีอยู่ใน NETWORK BUFFER หรือไม่ ถ้าผลการตรวจสอบแล้วปรากฏว่าไม่มีให้ปฏิบัติในข้อ 1 ใหม่ แต่ถ้ามีให้ปฏิบัติตามข้อ 3
3. ตรวจสอบชื่อที่กำหนด กับ ชื่อข้อมูลเข้าที่มีอยู่เดิมว่าตรงกันหรือไม่ ถ้าปรากฏว่าตรงให้ปฏิบัติตามข้อ 4 แต่ถ้าไม่ตรงให้ปฏิบัติตามข้อ 5
4. กำหนดจำนวน LOW STATE และ HIGH STATE ทางแป้นพิมพ์ให้กับโหนดข้อมูลเข้าเดิมโดยผู้ใช้งานสามารถกำหนดได้ดังนี้
   - N1 = LOW STATE
   - N2 = HIGH STATE
5. กำหนดจำนวน LOW STATE และ HIGH STATE ทางแป้นพิมพ์ให้กับโหนดข้อมูลเข้าใหม่ วิธีการเดียวกับข้อ 4.
6. กำหนดค่าสภาวะให้กับหน่วยความจำหลัก (SIMULATED BUFFER) ที่เป็นโหนดข้อมูลเข้าเท่านั้น โดยมีสภาวะ "0" เท่ากับ N1 สเต็ป และมีสภาวะ "1" เท่ากับ N2 สเต็ปสลับกันจนครบ 256 สเต็ป ในโหนดข้อมูลเข้านั้น ๆ

6.4.2 โปรแกรมแสดงชื่อโหนดทั้งหมดหรือโหนดข้อมูล เข้าของวงจร (DATA/LIST COMMAND) ดังรูปที่ 6-19 และรูปที่ 6-20



รูปที่ 6-6 แผนผังการแสดงผลชื่อโหนด เป็นแบบช่องหน้าต่าง

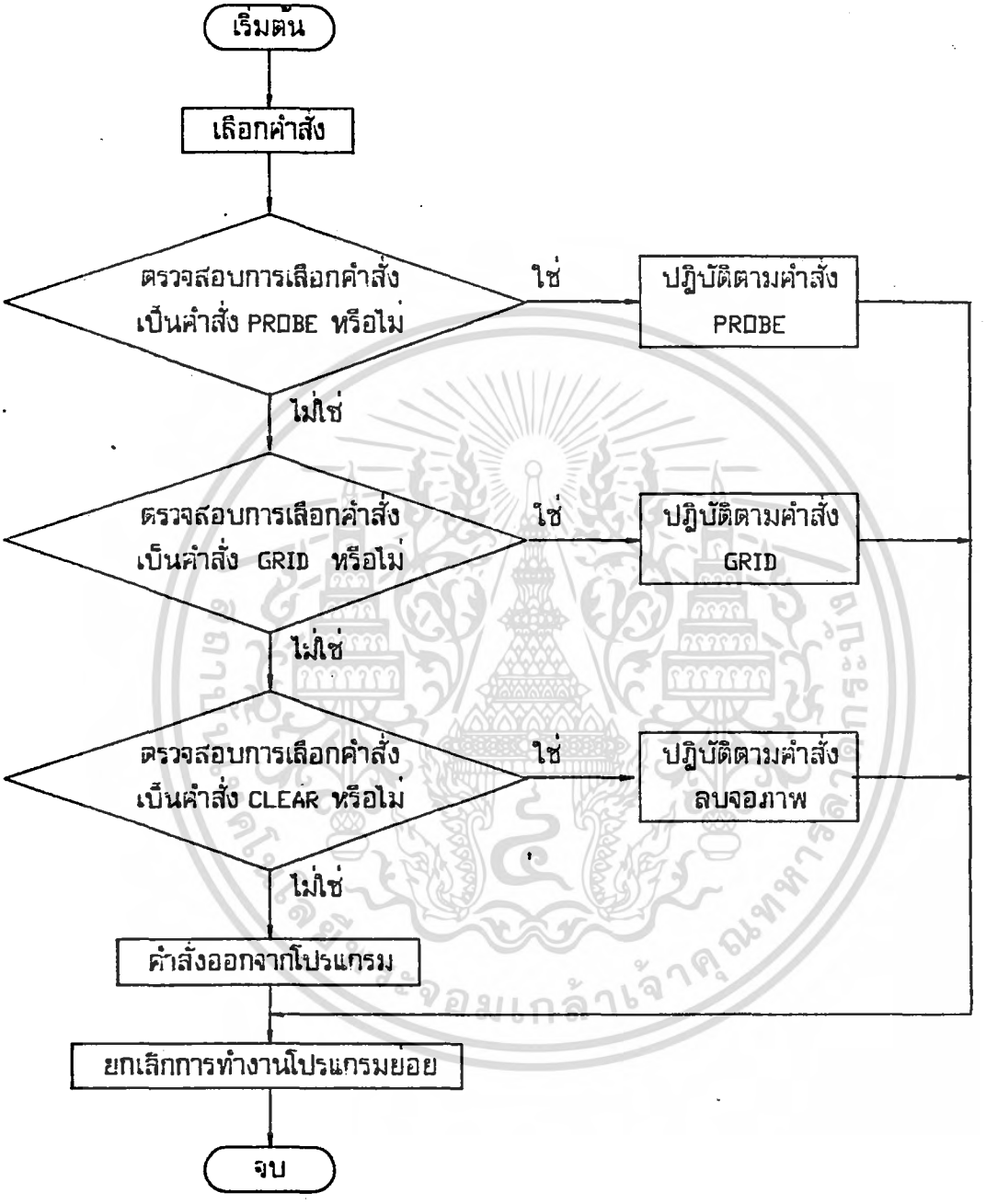
อธิบายแผนผังการทำงานของโปรแกรมแสดงชื่อโหนดทั้งหมดและโหนดข้อมูลเข้าของวงจร ดังรูปที่ 6-6

1. เลือกคำสั่งการทำงานในส่วนของเมนูย่อย และ ทำการตรวจสอบคำสั่งนั้น ๆ
2. ผลการตรวจสอบว่าเป็นคำสั่ง DATA/LIST/NODE จะมีการแสดงชื่อโหนดและลำดับที่ของโหนดในวงจรเป็นแบบช่องหน้าต่าง โดยที่ในหนึ่งช่องหน้าต่างแสดงได้มากสุด 20 โหนด
3. ผลการตรวจสอบว่าเป็นคำสั่ง DATA/LIST/CLOCK จะมีการแสดงชื่อโหนดข้อมูลเข้าทั้งหมดกับจำนวน N1 และ N2 ของโหนดข้อมูลเข้าเหล่านั้น ๆ เป็นแบบช่องหน้าต่าง โดยแสดงได้ 1 ช่องหน้าต่างหรือ 20 ชื่อของโหนดข้อมูลเข้า
4. ผลการตรวจสอบว่าเป็นคำสั่ง DATA/LIST/EXIT แล้วจะยกเลิกการทำงานของโปรแกรมและกลับยังส่วนคำสั่ง DATA

6.5 โปรแกรมเกี่ยวกับการแสดงผล (DISP COMMAND)

เป็นส่วนของโปรแกรมที่ใช้เกี่ยวกับการแสดงผลของโหนดต่าง ๆ ซึ่งอยู่ในรูปแผนภูมิเวลา ประกอบด้วยส่วนย่อย ๆ ดังนี้

- คำสั่งกำหนดตำแหน่งของวงจรเพื่อตรวจสอบ (DISP/PROBE COMMAND)
- คำสั่งการแสดงโพรบไม่ซ้ำชื่อเดิม (DISP/PROBE/ENTER COMMAND)
- คำสั่งการเปลี่ยนแปลงแก้ไขโพรบที่จะแสดง (DISP/PROBE/EDIT COMMAND)
- คำสั่งการแสดงโพรบแบบซ้ำชื่อเดิมได้ (DISP/PROBE/APPND COMMAND)
- คำสั่งการแสดงชื่อของโหนดในช่องหน้าต่าง (DISP/PROBE/LIST COMMAND)
- คำสั่งออกจากโปรแกรมการโพรบ (DISP/PROBE/EXIT COMMAND)
- คำสั่งกริด (DISP/GRID COMMAND)
- คำสั่งลบภาพแผนภูมิเวลา (DISP/CLEAR COMMAND)
- คำสั่งออกจากโปรแกรมการแสดงผล (DISP/EXIT COMMAND)



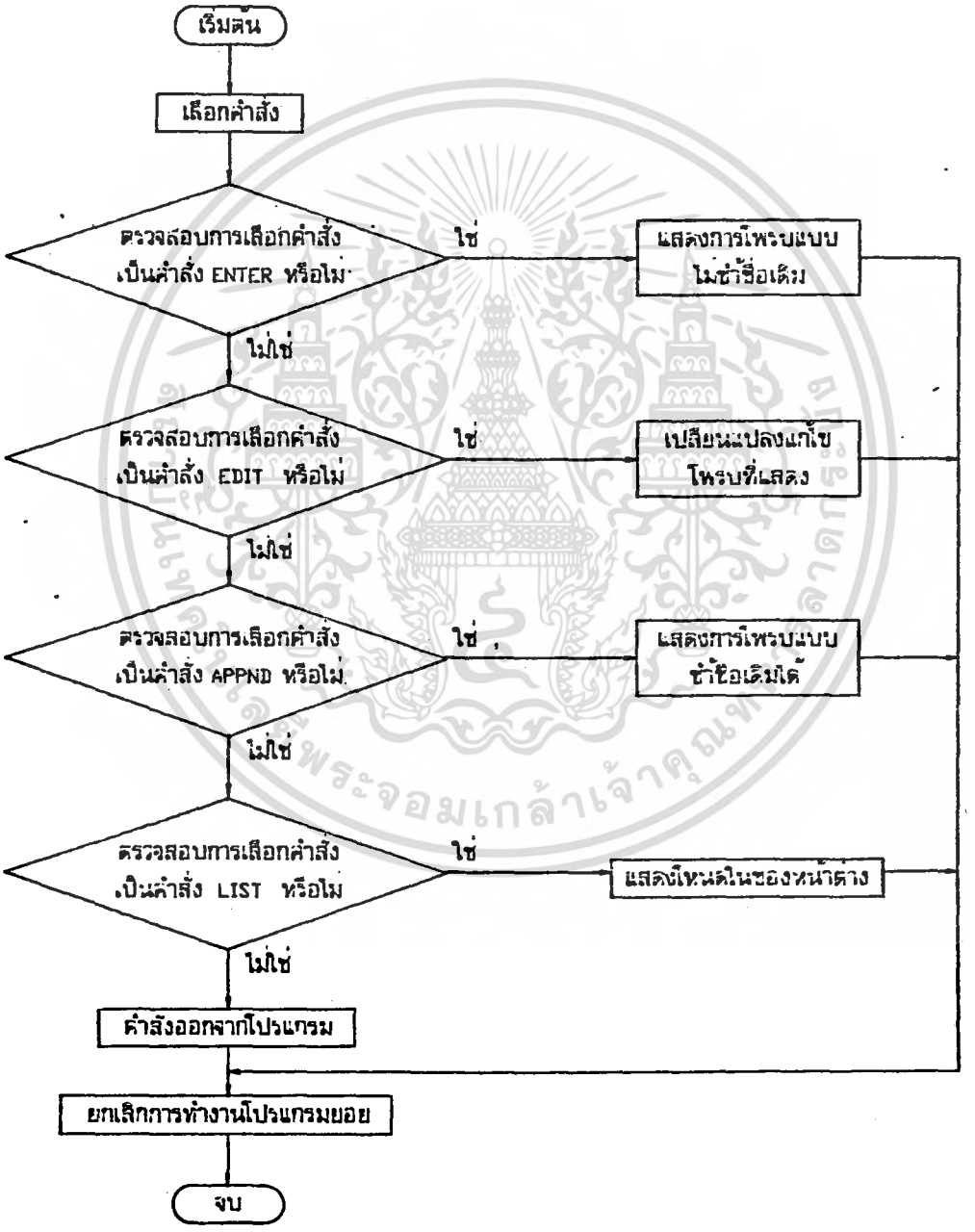
รูปที่ 6-7 แผนผังแสดงการทำงานของโปรแกรมการแสดงผล

อธิบายแผนผังการทำงานของโปรแกรมเกี่ยวกับการแสดงผล ดังรูปที่ 6-7

1. เลือกคำสั่งการใช้งานในส่วนของเมนูย่อย และ ทำการตรวจสอบคำสั่งนั้น ๆ
2. ผลของการตรวจสอบว่าเป็นคำสั่ง DISP/PROBE ให้ปฏิบัติตามคำสั่ง การโพรบตำแหน่งของโหนดในวงจรเพื่อการตรวจสอบ
3. ผลการตรวจสอบว่าเป็นคำสั่ง DISP/GRID จะปฏิบัติตามคำสั่งให้มีการแสดงกริดและไม่แสดงกริดกลับไม่กลับมาเป็นแบบ TOGGLE เมื่อมีการเลือกในคำสั่งนี้
4. ผลการตรวจสอบว่าเป็นคำสั่ง DISP/CLEAR จะปฏิบัติตามคำสั่งให้มีการลบเฉพาะส่วนที่เป็นภาพแผนภูมิเวลาเท่านั้น แต่ชื่อของโหนดที่ทำการโพรบจะไม่ถูกลบออกไป ทั้งนี้เพื่อวัตถุประสงค์ที่จะแสดงภาพแผนภูมิเวลาใหม่อีกครั้งในกรณีที่โปรแกรมมีแสดงผลผิดพลาดจะด้วยสาเหตุใดก็ตาม
5. ปฏิบัติตามคำสั่งออกจากโปรแกรม โดยการยกเลิกการทำงานของคำสั่งต่าง ๆ เหล่านี้แล้วกลับยังส่วนคำสั่ง DISP

6.5.1 โปรแกรมกำหนดตำแหน่งในวงจร เพื่อการตรวจสอบ (DISP/PROBE COMMAND)

ในการเลือกโหนดใดของวงจรมา เพื่อใช้ในการแสดงผล เป็นภาพแผนภูมิเวลา โหนดที่ถูกเลือกเหล่านั้นจะเรียกว่าโพรบ (PROBE) และในการโพรบแต่ละจุดผู้ใช้สามารถทำการเปลี่ยนแปลงแก้ไขได้อย่างสดวก เพราะมีคำสั่งต่าง ๆ ให้ใช้อย่างเพียงพอ การใช้งานในส่วนนี้สามารถแสดงการทำงานตามได้ดังรูปที่ 6-8



รูปที่ 6-8 แผนผังการทำงานของโปรแกรมการโพรบ

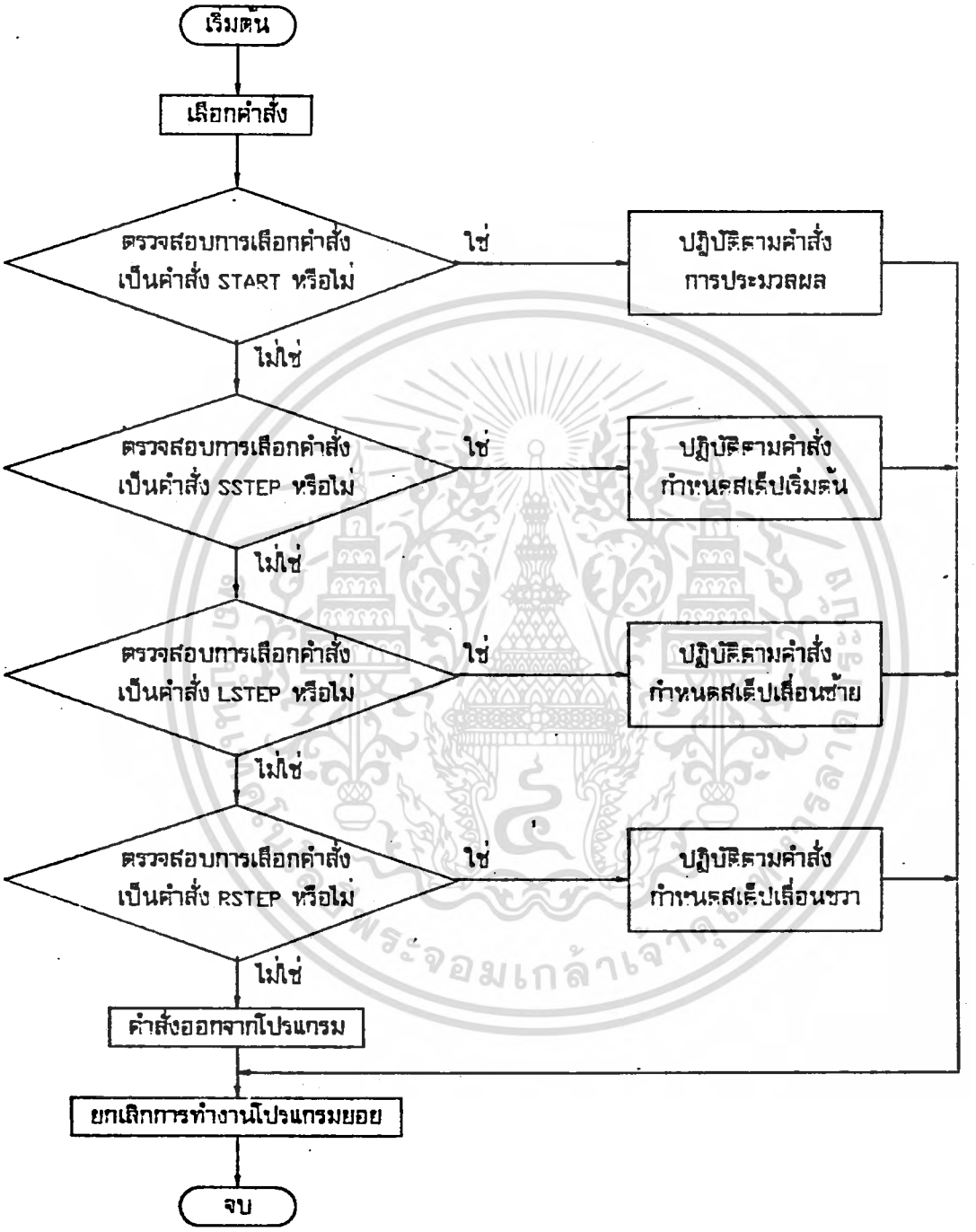
อธิบายแผนผังโปรแกรมกำหนดตำแหน่งของวงจร เพื่อการตรวจสอบ ดังรูปที่ 6-8

1. เลือกคำสั่งการใช้งานในส่วนของ เมนูย่อย และ ทำการตรวจสอบคำสั่งนั้น ๆ
2. ผลการตรวจสอบพบว่า เป็นคำสั่ง DISP/PROBE/ENTER ดังรูปที่ 6-21 ให้ปฏิบัติตามคำสั่งการแสดงผลอันหมายถึงโหนดใด ๆ ของวงจร ผู้ใช้สามารถกำหนดชื่อได้ทางแป้นพิมพ์ จากนั้นจะมีการตรวจสอบโดยโปรแกรมคือชื่อ เหล่านี้ต้องมีอยู่ใน NETWORK BUFFER และชื่อโพรบใหม่นี้จะต้องไม่ซ้ำกับชื่อโพรบ เดิมที่ได้กำหนดไว้ก่อนหน้านี้ แต่ถ้าซ้ำจะถือว่าโพรบใหม่ที่กำหนดนั้นคือโพรบ เดิม และแสดงในตำแหน่ง เดียวโพรบ เดิมทุกประการ การตรวจสอบของโปรแกรมลักษณะ เช่นนี้ คล้ายกับการตรวจสอบการกำหนดข้อมูล เข้าของวงจรที่ได้กล่าวมาแล้ว
3. ผลการตรวจสอบพบว่า เป็นคำสั่ง DISP/PROBE/EDIT ให้ปฏิบัติตามคำสั่งการ เปลี่ยนแปลงแก้ไขโพรบที่แสดงอยู่ ในส่วนนี้จะต้องมีการกำหนดชื่อโพรบไว้ จำนวนหนึ่งที่มากกว่าหนึ่งโพรบขึ้นไปอยู่ก่อนแล้ว และต้องการแก้ไขโพรบใดโพรบหนึ่งในจำนวน เหล่านี้สามารถทำได้โดยการ กำหนดชื่อโพรบที่ต้องการแก้ไข โปรแกรมจะจัดการค้นหาข้อมูลจากชื่อโพรบ เก่านี้ แล้วถามชื่อโพรบใหม่ที่จะใส่แทน ดังนั้นผู้ใช้งานจึงต้องใส่ชื่อโพรบใหม่นี้ เข้าไป โปรแกรมจะนำชื่อโพรบใหม่นี้ไปลงในตำแหน่ง เดิมของชื่อโพรบ เก่านั้น ๆ บนจอภาพ
4. ผลการตรวจสอบพบว่า เป็นคำสั่ง DISP/PROBE/APPND ดังรูปที่ 6-22 ให้ปฏิบัติตามคำสั่งการแสดงโพรบแบบซ้ำชื่อ เดิมได้ ลักษณะการใช้งานและวิธีการ เช่น เดียวกับคำสั่ง ENTER แต่ถ้ากำหนดชื่อโพรบที่ซ้ำกับโพรบ เดิมแล้วการแสดงผลจะ เลื่อนไปในตำแหน่งถัดไปจากโพรบล่าสุด ที่ได้กำหนดไว้ก่อนหน้านี้ ดังนั้นในคำสั่งนี้จึงสามารถแสดงโพรบที่มีชื่อ เหมือน เดิม ณ.ตำแหน่งใดก็ได้ เพื่อทำการ เปรียบ เทียบภาพแผนภูมิ เวลาระหว่างโพรบนั้น ๆ กับโพรบข้าง เคียง
5. ผลการตรวจสอบพบว่า เป็นคำสั่ง DISP/PROBE/LIST ให้ปฏิบัติตามคำสั่งการแสดงโหนดของวงจรในช่องหน้าต่าง เช่น เดียวกับคำสั่ง DATA/LIST/NODE
6. ผลการตรวจสอบพบว่า เป็นคำสั่ง DISP/PROBE/EXIT ให้ออกจากโปรแกรม และ ยกเลิกการทำงานของโปรแกรมกลับยังส่วนคำสั่ง DISP/PROBE

6.6 โปรแกรมการประมวลผลและการเลื่อนภาพแผนภูมิเวลา (SIM COMMAND)

เป็นส่วนของโปรแกรมที่ใช้ในการประมวลผล และเก็บค่าสภาวะที่ได้จากการคำนวณลงสู่หน่วยความจำ เพื่อนำค่าสภาวะเหล่านี้มาแสดงเป็นภาพแผนภูมิเวลา ในส่วนของคำสั่งจะประกอบด้วยคำสั่งย่อยดังนี้

- คำสั่งการประมวลผล (SIM/START COMMAND)
- คำสั่งการกำหนดสเต็ปเริ่มต้นการแสดงผล (SIM/SSTEP COMMAND)
- คำสั่งการกำหนดจำนวนสเต็ปเลื่อนภาพไปทางซ้าย (SIM/LSTEP COMMAND)
- คำสั่งการกำหนดจำนวนสเต็ปเลื่อนภาพไปทางขวา (SIM/RSTEP COMMAND)
- คำสั่งการเลื่อนภาพ (SIM/SCRLL COMMAND)
- คำสั่งออกจากโปรแกรมการทำงาน (SIM/EXIT COMMAND)

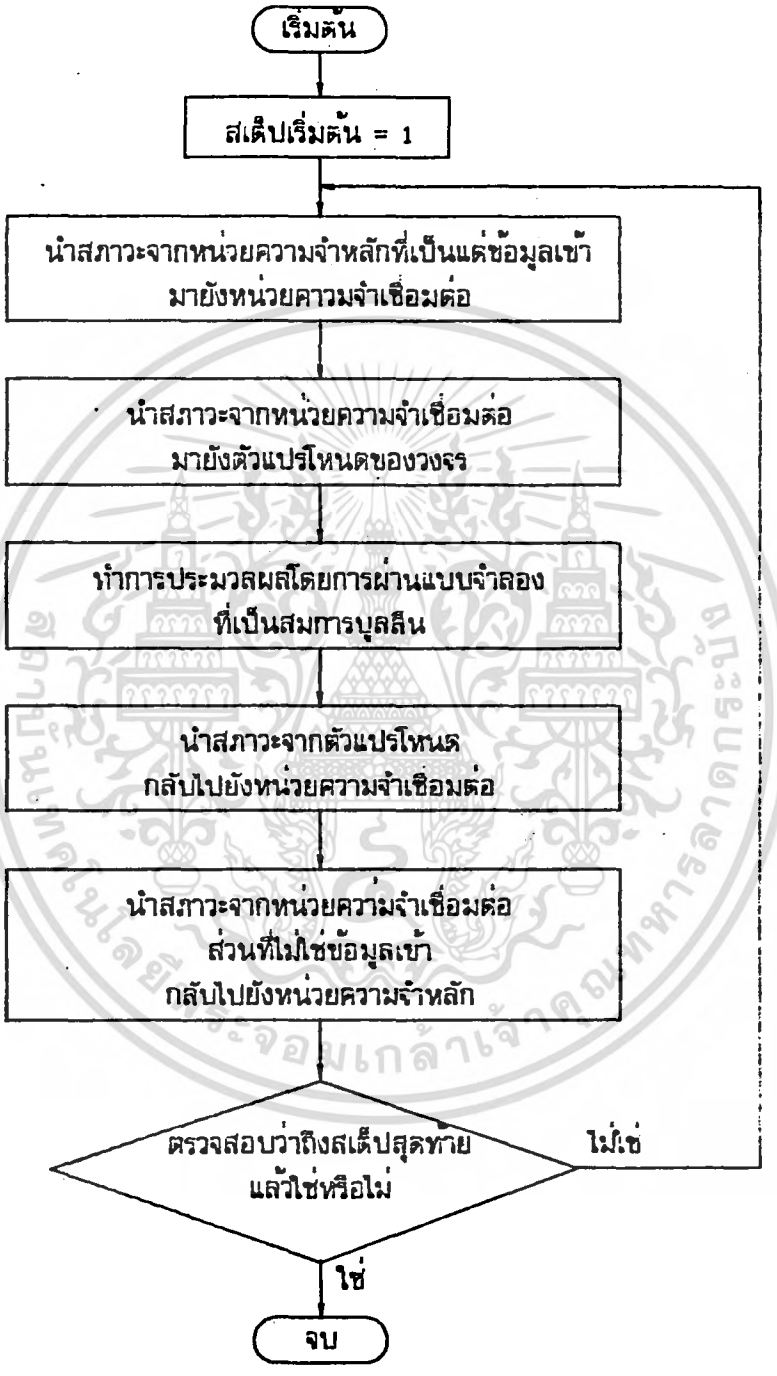


รูปที่ 6-9 แผนผังการทำงานโปรแกรมการประมวลผลและการเลื่อนภาพแผนภูมิเวลา

อธิบายแผนผังการทำงานของโปรแกรมการประมวลผลและการเลื่อนภาพแผนภูมิเวลา

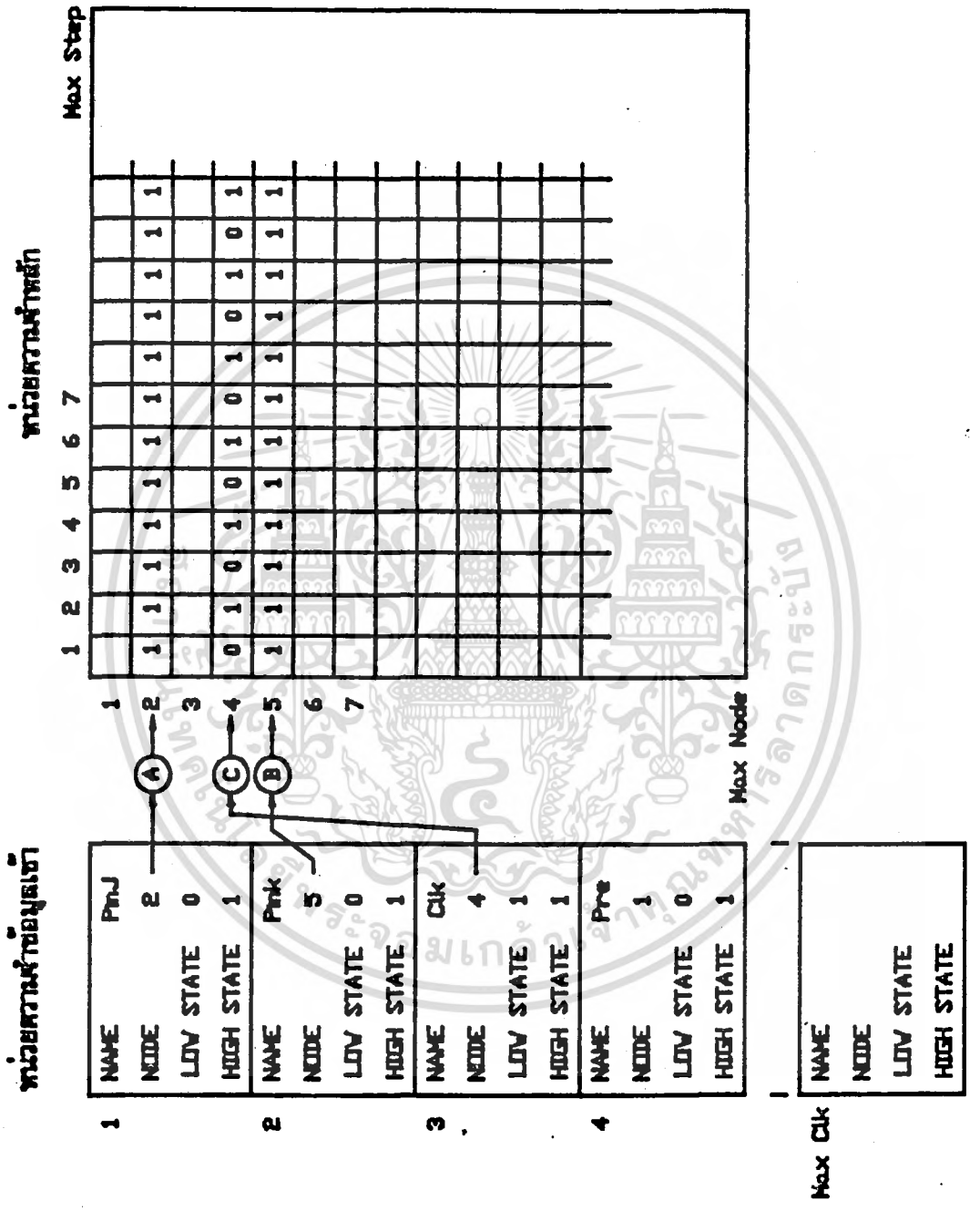
1. เลื่อนคำสั่งการใช้งานโปรแกรม และ ทำการตรวจสอบคำสั่งนั้น ๆ
2. ตรวจสอบแล้วปรากฏว่าเป็นคำสั่ง SIM/START ให้ปฏิบัติงานตามโปรแกรมการประมวลผล ได้อธิบายดังรูปที่ 6-11 และ 6-12
3. ตรวจสอบแล้วปรากฏว่าเป็นคำสั่ง SIM/SSTEP ดังรูปที่ 6-23 ให้ปฏิบัติงานตามโปรแกรมกำหนดสเต็ปเริ่มต้นสำหรับการแสดงผล
4. ตรวจสอบแล้วปรากฏว่าเป็นคำสั่ง SIM/LSTEP ดังรูปที่ 6-24 ให้ปฏิบัติงานตามโปรแกรมกำหนดจำนวนสเต็ปเลื่อนภาพไปทางซ้าย โดยการกำหนดทางแป้นพิมพ์
5. ตรวจสอบแล้วปรากฏว่าเป็นคำสั่ง SIM/RSTEP ดังรูปที่ 6-25 ให้ปฏิบัติงานตามโปรแกรมกำหนดจำนวนสเต็ปเลื่อนภาพไปทางขวา โดยการกำหนดทางแป้นพิมพ์เช่นเดียวกัน
6. ตรวจสอบแล้วปรากฏว่าเป็นคำสั่ง SIM/SCRLL ดังรูปที่ 6-26 ให้ปฏิบัติงานตามโปรแกรมการเลื่อนภาพแผนภูมิเวลา โดยกำหนดการเลื่อนภาพไปทางซ้ายหรือขวาได้จากแป้นพิมพ์ลูกศร "<---" "--->" และการเลื่อนภาพไปเป็นจำนวนที่เท่าใดนั้น จะขึ้นอยู่กับจำนวนสเต็ปที่ได้กำหนดจากคำสั่ง SIM/LSTEP และ SIM/RSTEP ซึ่งได้กล่าวมาแล้ว
7. ปฏิบัติตามคำสั่งออกจากโปรแกรม โดยการยกเลิกการทำงานของคำสั่งต่าง ๆ เหล่านี้แล้วกลับยังส่วนคำสั่ง SIM

6.6.1 โปรแกรมการประมวลผล (SIM/START COMMAND)



รูปที่ 6-10 แผนผังโปรแกรมเกี่ยวกับการประมวลผล

จากบทที่ 4 ตัวอย่างการสร้างผังภาพวงจร JKFFNL สามารถแสดงขั้นตอนการกำหนดข้อมูลเข้าให้พิจารณาจากรูปที่ 6-20 ประกอบ



รูปที่ 6-11 แสดงการกำหนดข้อมูลเข้าให้กับหน่วยความจำหลัก

อธิบายขั้นตอนการกำหนดข้อมูลเข้าดังรูปที่ 6-11

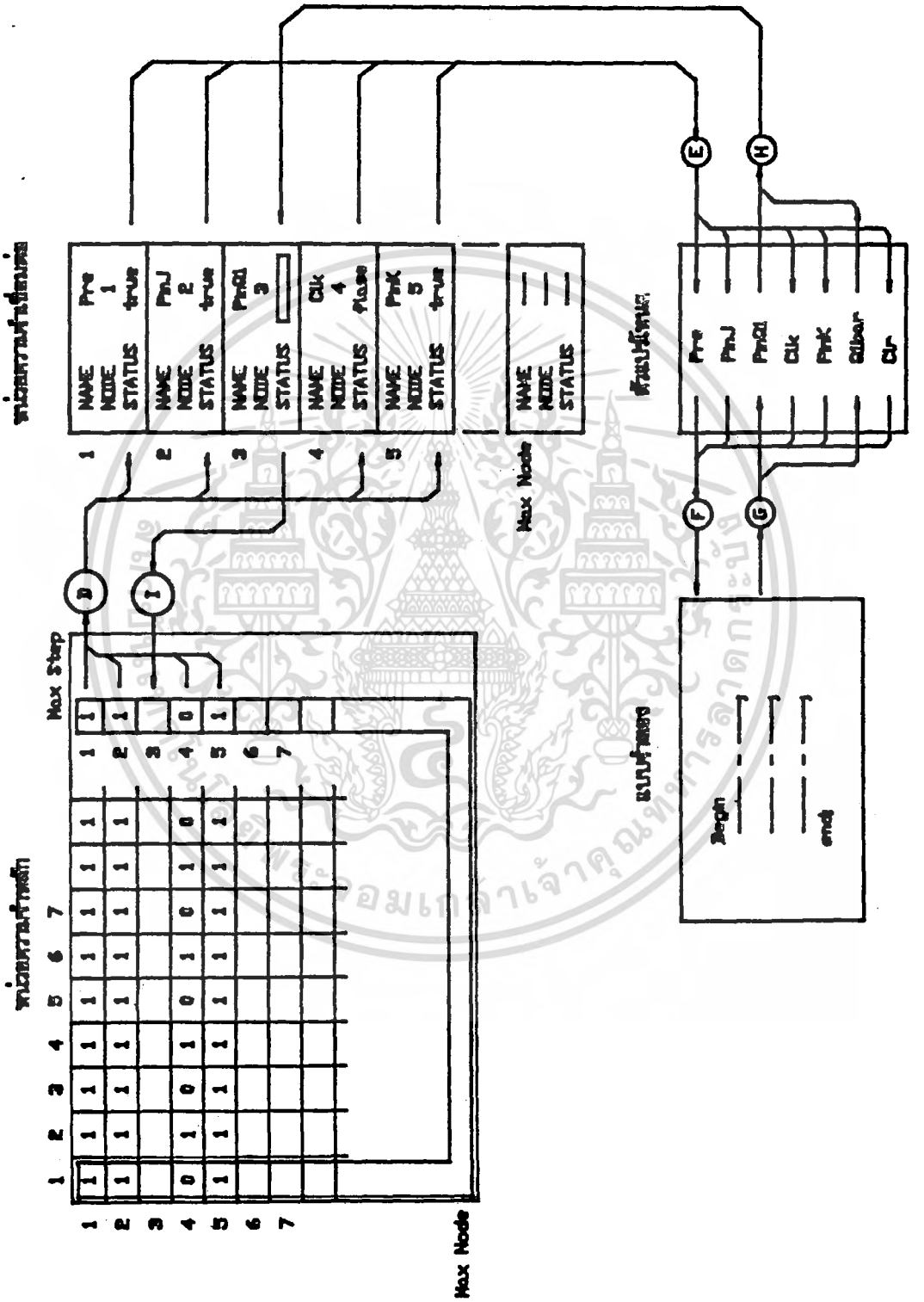
ขั้นตอน A เป็นการกำหนดข้อมูลเข้า ชื่อโหนด PINJ โหนดลำดับที่ 2
LOW STATE = 0 และ HIGH STATE = 1 ให้กับหน่วยความจำหลัก

ขั้นตอน B เป็นการกำหนดข้อมูลเข้า ชื่อโหนด PINK โหนดลำดับที่ 5
LOW STATE = 0 และ HIGH STATE = 1 ให้กับหน่วยความจำหลัก

ขั้นตอน C เป็นการกำหนดข้อมูลเข้า ชื่อโหนด CLK โหนดลำดับที่ 4
LOW STATE = 1 และ HIGH STATE = 1 ให้กับหน่วยความจำหลัก

กระทำในลักษณะเช่นนี้จนกระทั่งหมดทุกโหนดของข้อมูลเข้า

จากบทที่ 4 ตัวอย่างการสร้างผังภาพวงจร JKFFNL สามารถแสดงขั้นตอนการประมวลผลให้พิจารณาจากรูปที่ 6-19 ประกอบ



รูปที่ 6-12 แสดงขั้นตอนการประมวลผล

หลังจากที่ผ่านขั้นตอนการกำหนดข้อมูลเข้า A B และ C แล้วจะทำให้หน่วยความจำหลักมีข้อมูลพร้อมที่จะนำไปประมวลผลซึ่งมีขั้นตอนต่อไปดังนี้

ขั้นตอน D เป็นการนำสภาวะจากหน่วยความจำหลักที่เป็นเฉพาะข้อมูลเข้าเท่านั้นไปเก็บยังหน่วยความจำเชื่อมต่อ

ขั้นตอน E เป็นการนำสภาวะจากหน่วยความจำเชื่อมต่อไปเก็บที่ตัวแปรโหนด

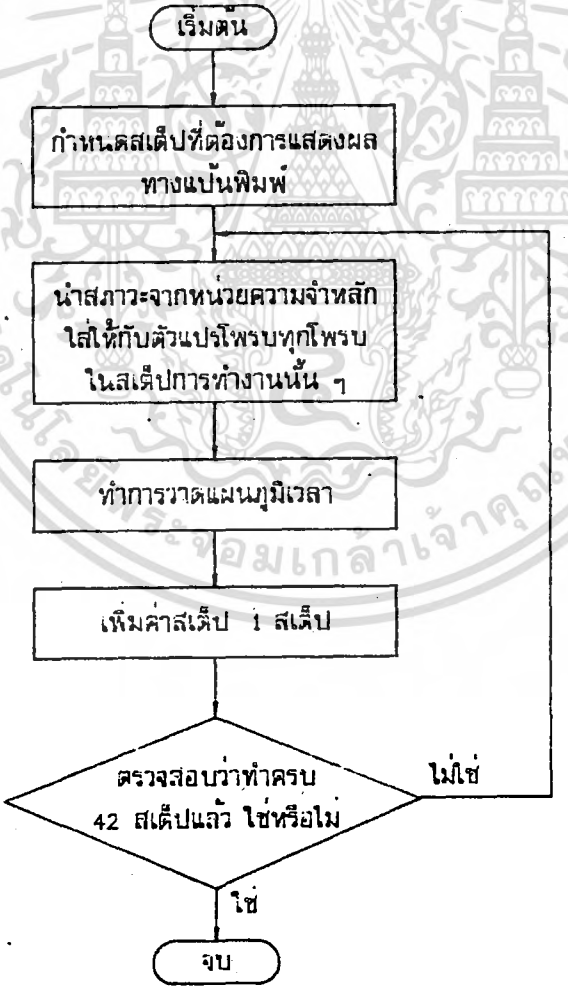
ขั้นตอน F เป็นการนำสภาวะจากตัวแปรโหนดผ่านการทำงานของแบบจำลอง

ขั้นตอน G เป็นการนำสภาวะที่ได้จากการผ่านแบบจำลองกลับมายังตัวแปรโหนด

ขั้นตอน H เป็นการนำสภาวะจากตัวแปรโหนดกลับมายังหน่วยความจำเชื่อมต่อ

ขั้นตอน I เป็นการนำสภาวะจากหน่วยความจำเชื่อมต่อ ส่วนที่นอกเหนือจากข้อมูลเข้ากลับมายังหน่วยความจำหลัก

6.6.2 ส่วนกำหนดสเต็ปเริ่มต้นการแสดงผล (SIM/SSTEP COMMAND)



รูปที่ 6-13 แผนผังการทำงานของโปรแกรมกำหนดสเต็ปเริ่มต้นการแสดงผล

อธิบายแผนผังการทำงานของโปรแกรมกำหนดสเต็ป เริ่มต้นการแสดงผล ดังรูปที่ 6-13

1. กำหนดสเต็ปแรกที่ต้องการแสดงภาพแผนภูมิเวลาทางแป้นพิมพ์
2. นำสภาวะจากหน่วยความจำหลักให้กับตัวแปรโพรบทุกโพรบ ที่ต้องการแสดง โดยเริ่มจากสเต็ปการทำงานที่ได้กำหนดในข้อ 1.
3. ทำการแสดงภาพแผนภูมิเวลาบนส่วนแสดงผล ลักษณะการแสดงจะวาดภาพแผนภูมิเวลาที่มีทิศทางจากบนลงล่าง (กวาดตามแนวโพรบ) และทิศทางจากซ้ายไปขวา (กวาดตามแนวสเต็ป) สำหรับการตรวจสอบเพื่อกำหนดแนวการวาดเส้น จะแบ่งการตรวจสอบออกเป็น 2 ลักษณะดังนี้

   3.1 ตรวจสอบสภาวะภายในสเต็ปการทำงานนั้น ๆ ว่าในสเต็ปนั้นของแต่ละตัวแปรโพรบมีค่าสภาวะเป็นอย่างไร ทั้งนี้เพื่อทำการกำหนดทิศทางการวาดในแนวระดับ ถ้าผลการตรวจสอบแล้วปรากฏว่าตัวแปรโพรบนั้น ๆ มีสภาวะเป็น

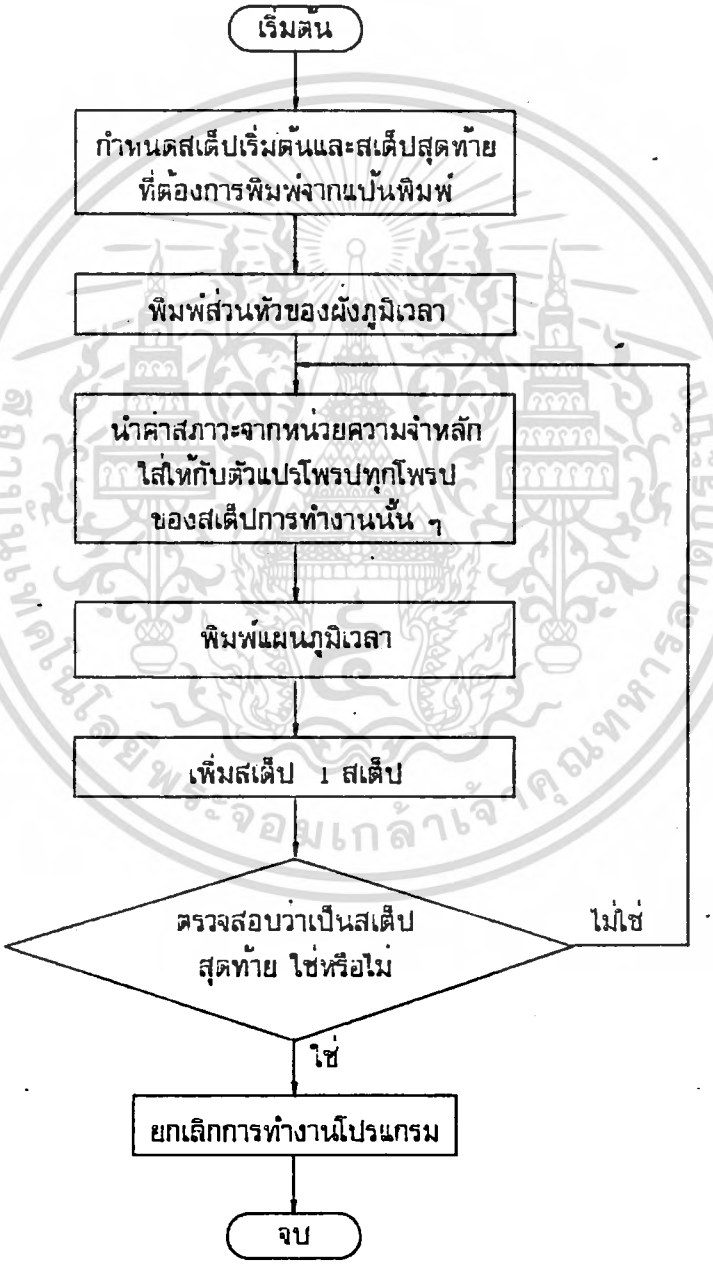
   - TRUE จะกำหนดแนวการวาดในลักษณะขีดเส้นด้านบน "‾"
   - FLASE จะกำหนดแนวการวาดในลักษณะขีดเส้นด้านล่าง "_"

   3.2 ตรวจสอบสภาวะระหว่างสเต็ปปัจจุบันกับสเต็ปที่ผ่านมาว่ามีสภาวะของตัวแปรโพรบแตกต่างกันหรือไม่ เพื่อที่จะกำหนดทิศทางการวาดในแนวดิ่งถ้าผลการตรวจสอบปรากฏว่า

   - มีค่าสภาวะต่างกัน ให้กำหนดทิศทางการวาดในลักษณะขีดเส้นแนวดิ่ง "|"
   - มีค่าสภาวะไม่ต่างกัน ก็ไม่ต้องมีการวาดเส้นในแนวดิ่งนี้
4. เพิ่มค่าสเต็ปการทำงานให้เป็นสเต็ปใหม่
5. ทำการตรวจสอบว่าครบ 42 สเต็ปการทำงานหรือยัง ถ้าผลการตรวจสอบปรากฏว่ายังไม่ครบ โปรแกรมจะไปทำในข้อ 2 แต่ถ้าครบแล้วก็จะยกเลิกการทำงานของโปรแกรม และในการแสดงผลภาพแผนภูมิเวลาบนจอภาพสามารถแสดงได้สูงสุด 42 สเต็ปการทำงานเท่านั้น

6.7 โปรแกรมการพิมพ์ภาพแผนภูมิเวลา (PRINT COMMAND)

เป็นส่วนของโปรแกรมที่ใช้ในการพิมพ์ภาพแผนภูมิเวลา ผู้ใช้งานสามารถจะกำหนดสเต็ปเริ่มต้นและสเต็ปสุดท้ายของการพิมพ์ได้ โดยไม่ต้องพิจารณาการแสดงผลของแผนภูมิเวลาบนจอภาพแต่อย่างใด นอกจากนี้ยังสามารถบอก วัน-เดือน-ปี-เวลา และชื่อไฟล์ของวงจรที่ทำการพิมพ์ ทั้งนี้จะเป็นการระบุรายละเอียดต่าง ๆ เพื่อประโยชน์ในการแบ่งแยกภาพแผนภูมิเวลาที่ได้พิมพ์ออกมาแล้ว



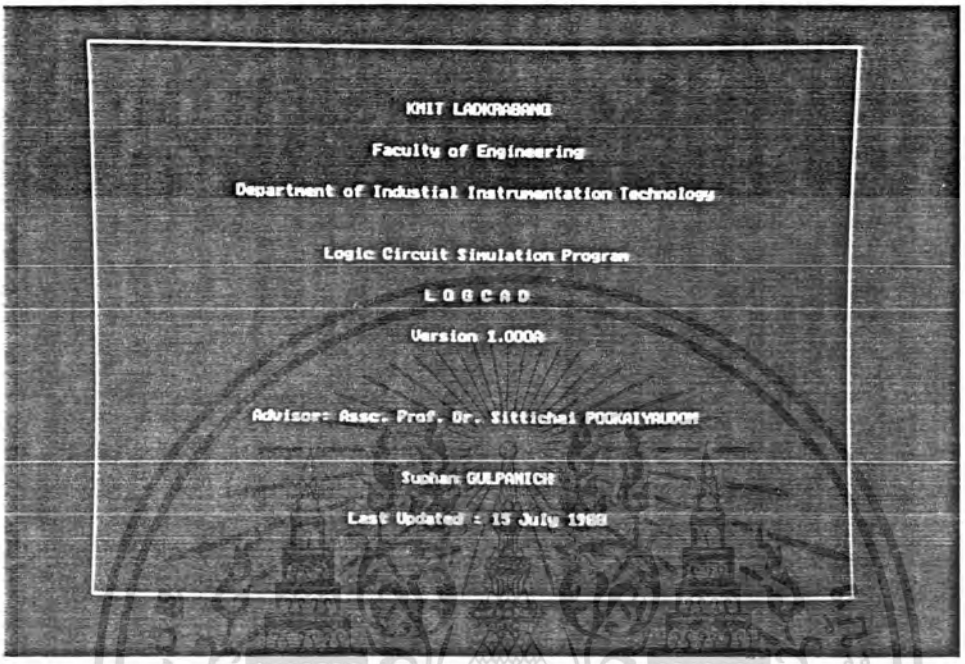
รูปที่ 6-14 แผนผังโปรแกรมการพิมพ์ภาพแผนภูมิเวลา

อธิบายการทำงานของโปรแกรมการพิมพ์ภาพแผนภูมิเวลา ดังรูปที่ 6-14

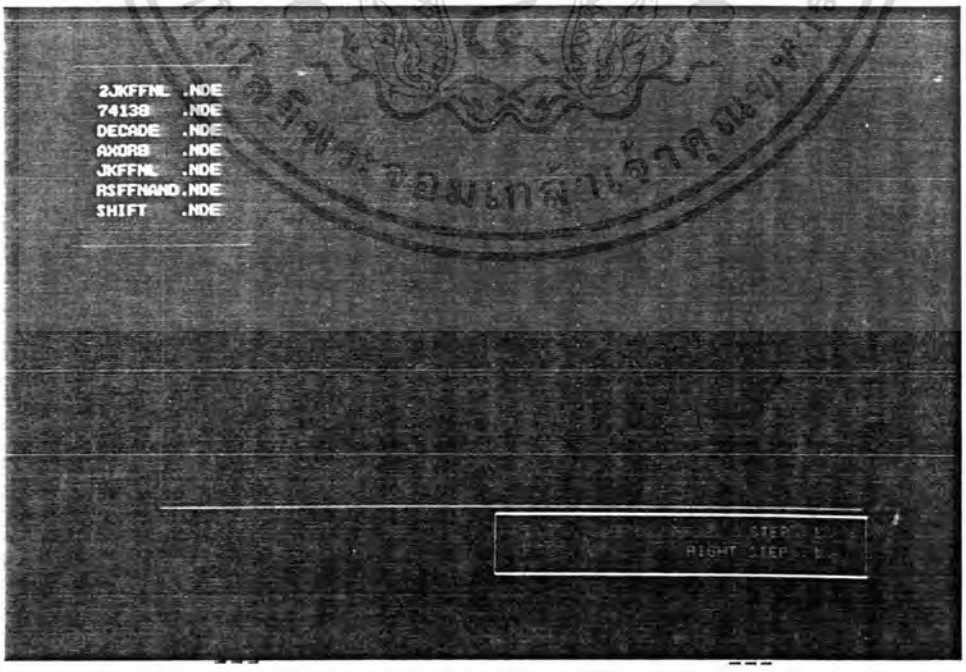
1. กำหนดสเต็ปเริ่มต้นและสเต็ปสุดท้ายของการพิมพ์
2. พิมพ์ส่วนหัวของแผนภูมิเวลาที่อยู่ใน TEXT MODE ของเครื่องพิมพ์โดยแบ่งการพิมพ์ดังนี้
   - วัน-เดือน-ปี ที่ทำการพิมพ์
   - เวลาเป็น ชั่วโมง-นาที-วินาที ในขณะนั้นที่ทำการพิมพ์
   - ชื่อวงจรที่พิมพ์ภาพแผนภูมิเวลา
   - ชื่อโพรบต่าง ๆ โดยมีการเลือกพิมพ์ทีละตัวอักษรของชื่อโพรบ
3. การนำค่าสภาวะจากหน่วยความจำหลัก ใส่ให้กับตัวแปรโพรบทุกโพรบ ในสเต็ปการทำงานนั้น ๆ เพื่อพิมพ์ภาพแผนภูมิเวลาซึ่งเป็นการพิมพ์ที่เปลี่ยนจาก TEXT MODE มาเป็น BIT-IMAGE MODE เพื่อให้ภาพแผนภูมิเวลาเป็นภาพที่แสดงได้ต่อเนื่องในแนวตั้ง การพิมพ์ในสเต็ปหนึ่ง ๆ จะต้องมีการตรวจสอบจากสภาวะของตัวแปรโพรบในสเต็ปที่ผ่านมา กับ สเต็ปปัจจุบัน ทั้งนี้เพื่อที่จะกำหนดลักษณะการพิมพ์ซึ่งสามารถแบบได้เป็น 4 รูปแบบ
   - ให้พิมพ์แบบ " ┐" หมายถึง การเปลี่ยนแปลงที่เป็นแบบขอบนำ
   - ให้พิมพ์แบบ "┌ " หมายถึง การเปลี่ยนแปลงที่เป็นแบบขอบตาม
   - ให้พิมพ์แบบ " |" หมายถึงไม่มีการเปลี่ยนแปลงและยังคงสภาวะ HIGH
   - ให้พิมพ์แบบ "| " หมายถึงไม่มีการเปลี่ยนแปลงและยังคงสภาวะ LOW

   <u>หมายเหตุ</u> รูปแบบการพิมพ์ภาพแผนภูมิเวลาจะถูกจัดให้อยู่ในแนวตั้ง

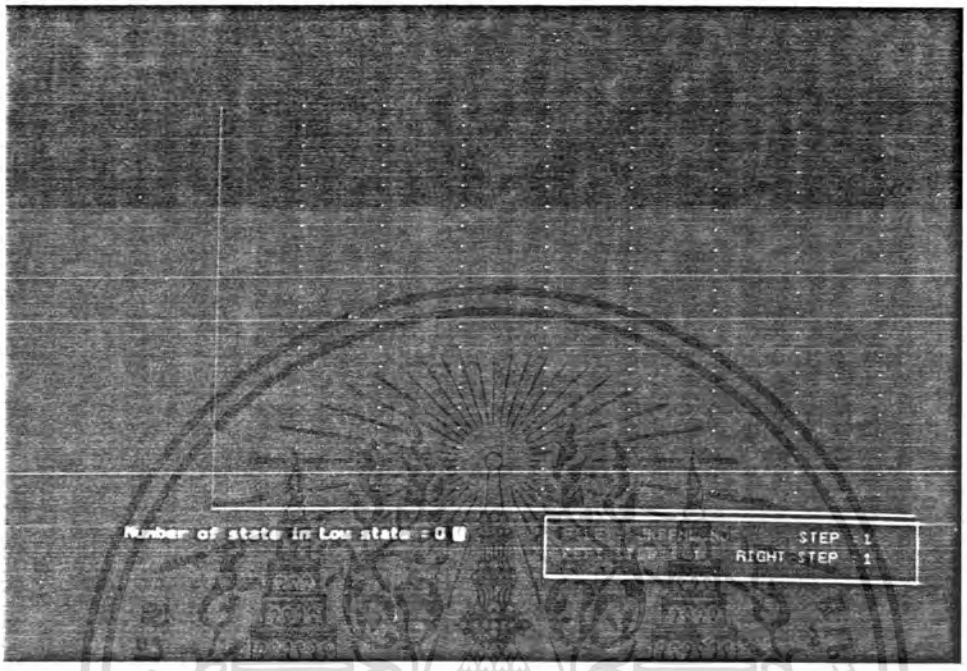
   นอกจากนี้ในการจัดลักษณะการพิมพ์ของแต่ละโพรบ จะต้องมีการกำหนดช่องว่างระหว่างโพรบเหล่านี้ไว้ด้วย และที่สำคัญยังมีการบอกถึงเส้นอ้างอิงในทุก ๆ 5 สเต็ปการทำงานเพื่อง่ายและสะดวกต่อการพิจารณา ดังแสดงใน รูปที่ 6-27
5. เพิ่มสเต็ปการทำงานและทำการตรวจสอบว่า ถึงสเต็ปสุดท้ายที่ได้กำหนดแล้วหรือยัง ถ้าผลการตรวจสอบปรากฏว่ายัง โปรแกรมจะปฏิบัติในข้อ 3 ใหม่ แต่ถ้าครบจนถึงสเต็บสุดท้ายแล้ว ก็จะยกเลิกการทำงานของโปรแกรมและกลับยังส่วนคำสั่ง PRINT



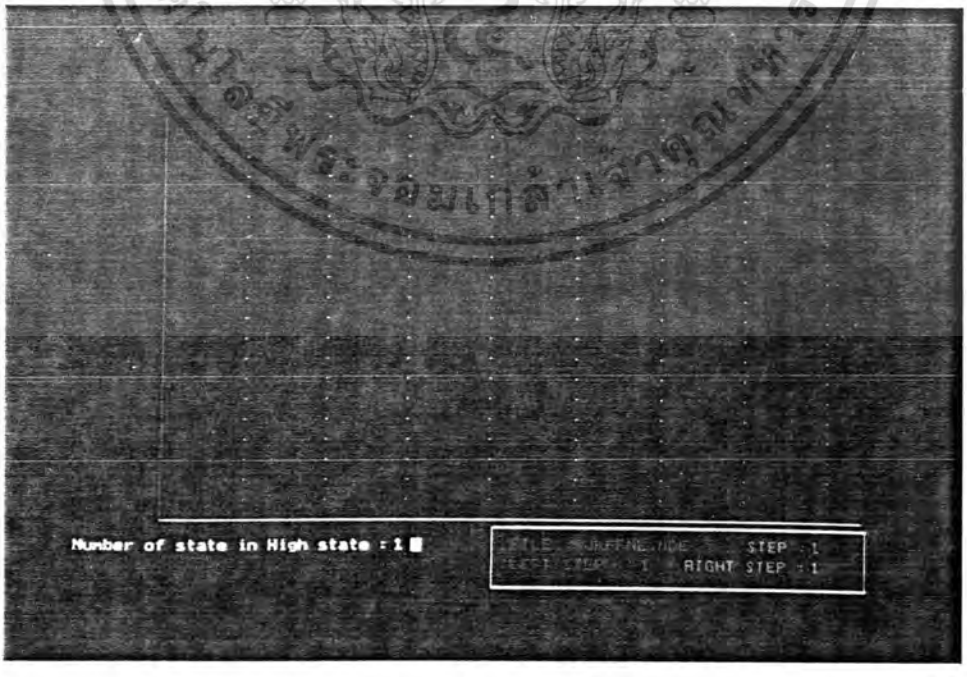
รูปที่ 6-15 แสดงข้อความต่าง ๆ



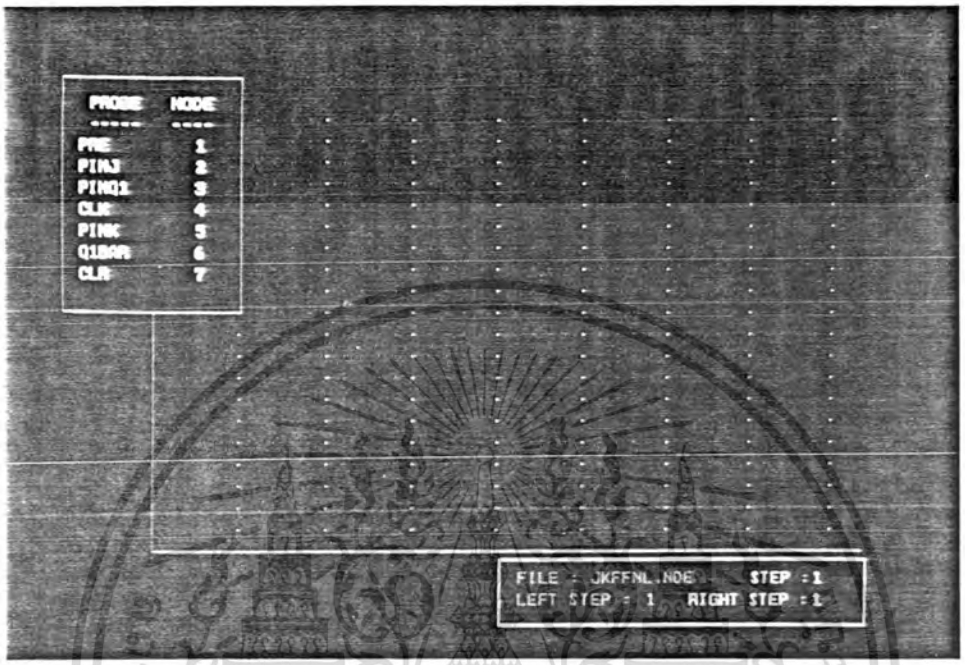
รูปที่ 6-16 แสดงไดเรคทอรีไฟล์ที่มีชื่อหลังจุดเป็น NDE



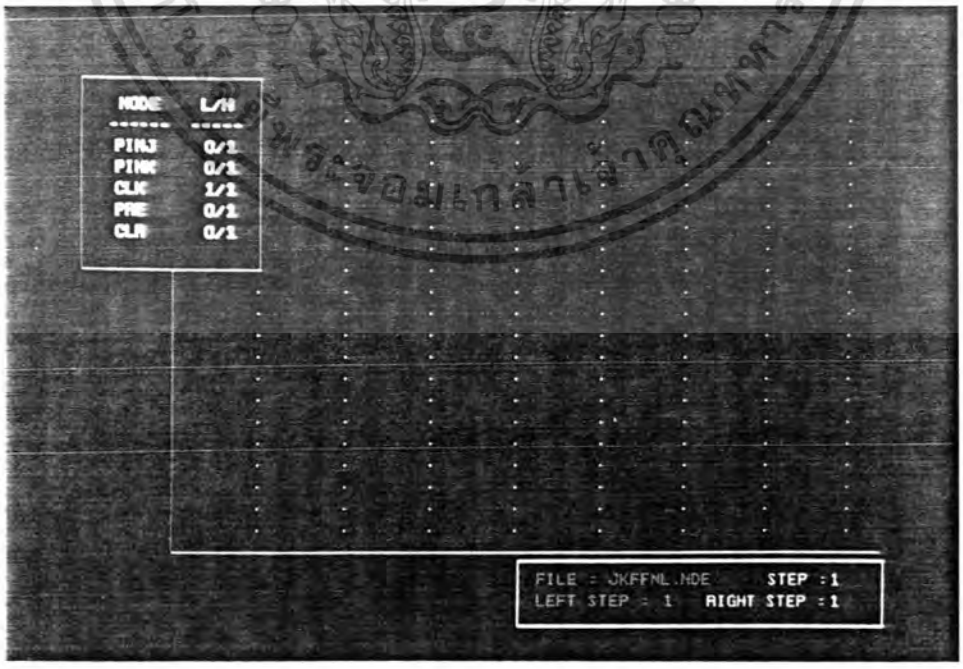
รูปที่ 6-17 แสดงการรับข้อมูลเข้าที่มีสภาวะ "0" จำนวน 0 สเต็ป



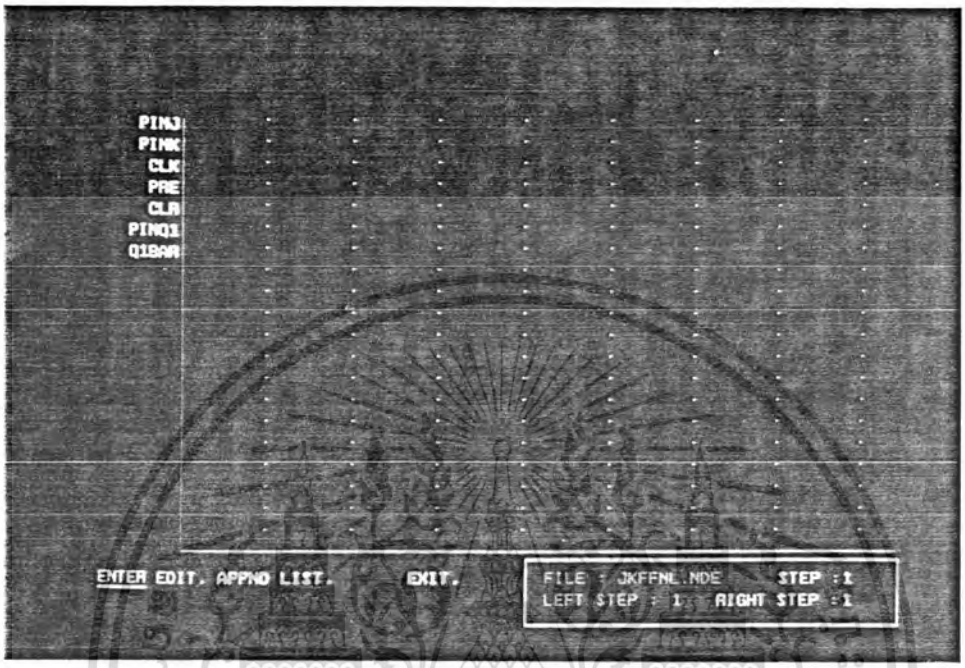
รูปที่ 6-18 แสดงการรับข้อมูลที่มีสภาวะ "1" จำนวน 1 สเต็ป



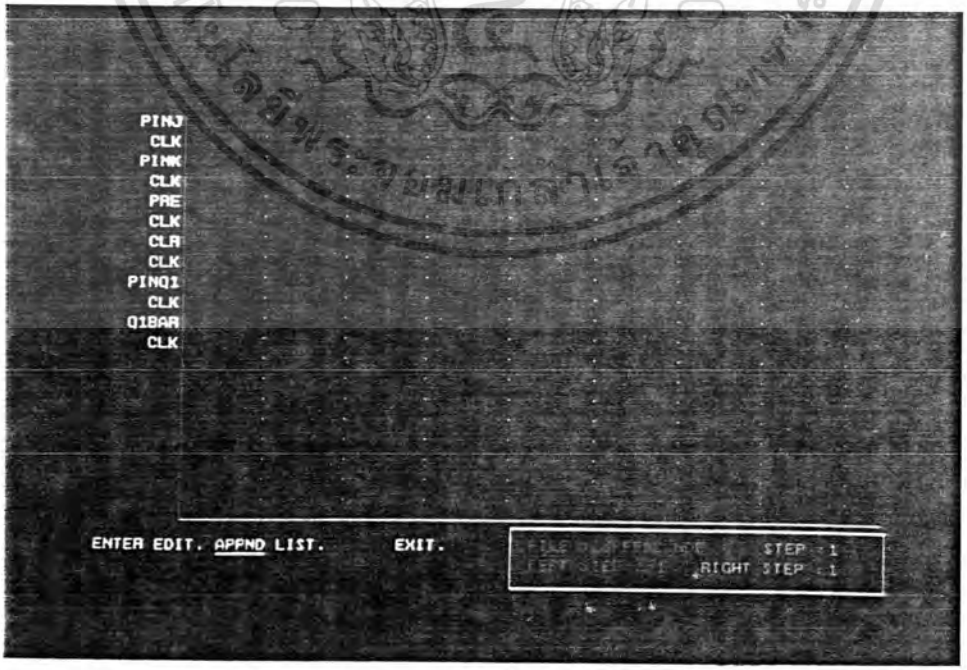
รูปที่ 6-19 แสดงช่องหน้าต่างที่เป็นชื่อโหนดของวงจร JKFFNL



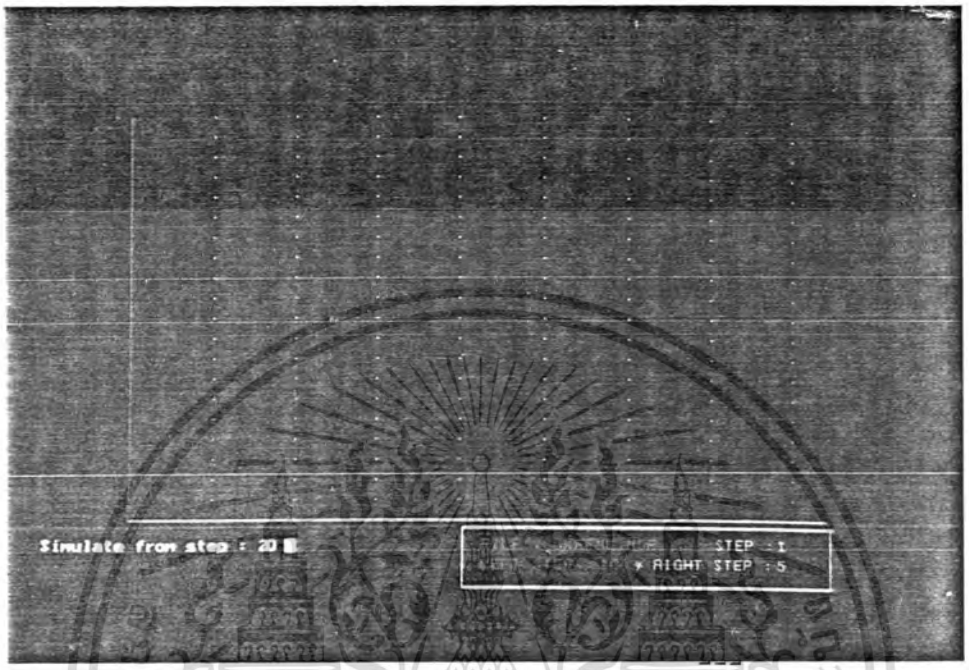
รูปที่ 6-20 แสดงช่องหน้าต่างที่เป็นชื่อข้อมูลเข้าของวงจร JKFFNL



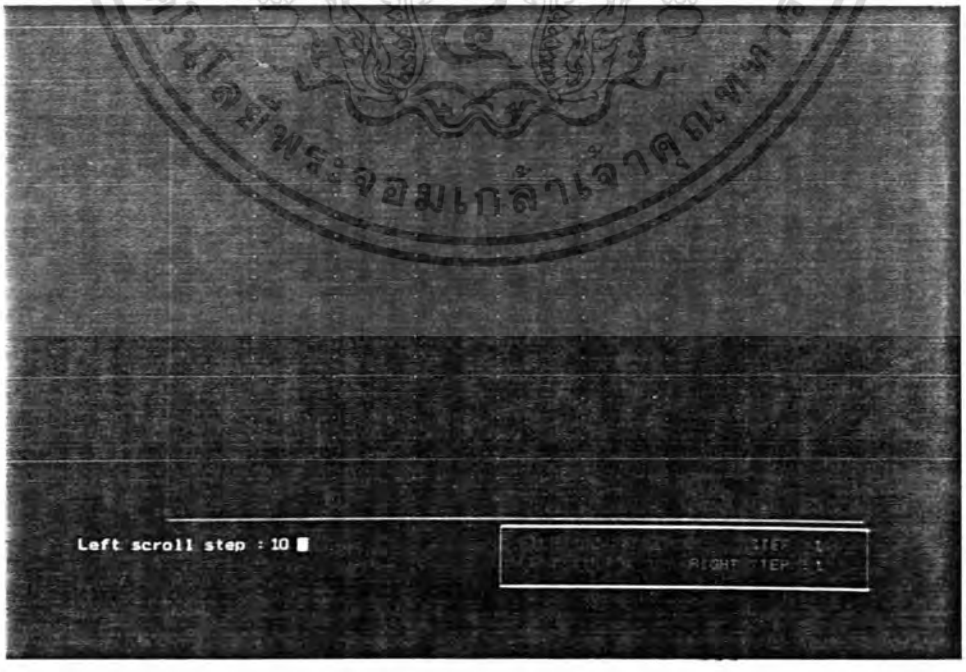
รูปที่ 6-21 แสดงการโพรบตำแหน่งต่าง ๆ ของวงจร JKFFNL



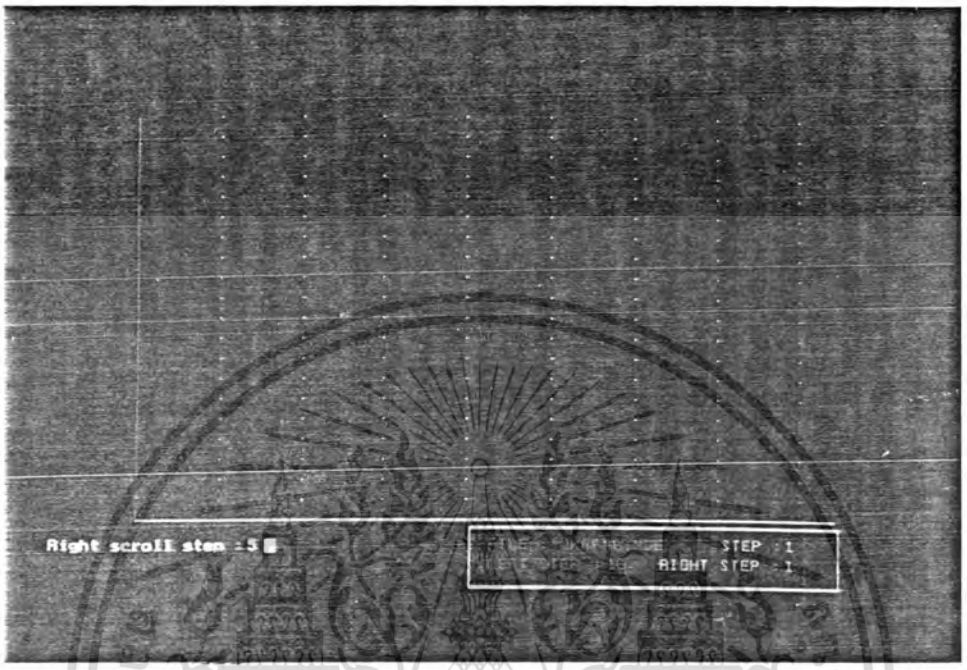
รูปที่ 6-22 แสดงการโพรบแบบซ้ำได้



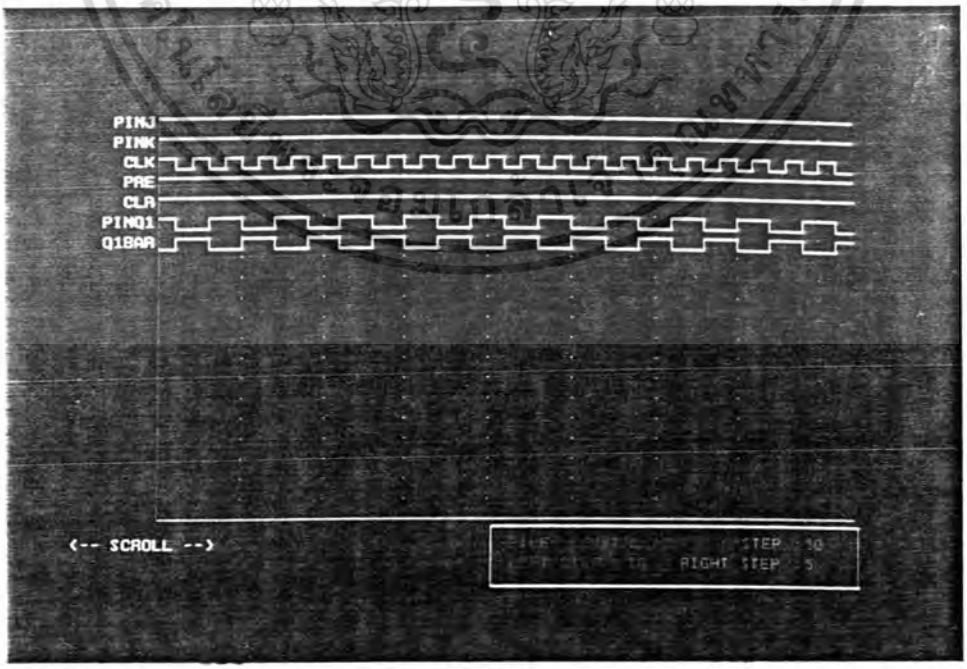
รูปที่ 6-23 แสดงการกำหนดสเต็ปเริ่มต้นการแสดงผล



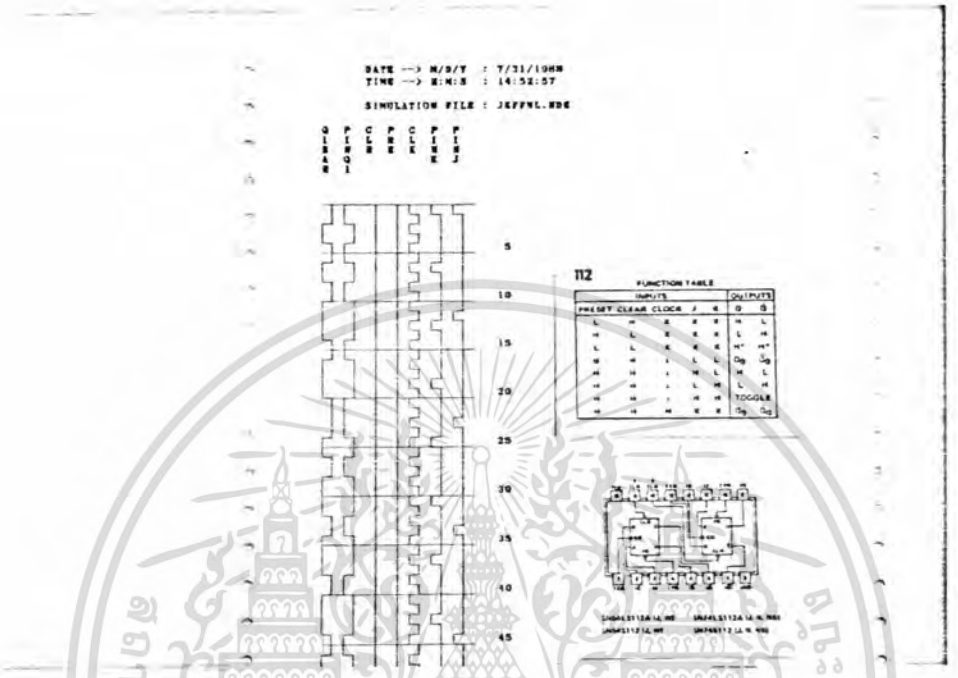
รูปที่ 6-24 แสดงการกำหนดสเต็ปเลื่อนภาพทางซ้าย



รูปที่ 6-25 แสดงการกำหนดสเต็ปเลื่อนภาพทางขวา



รูปที่ 6-26 แสดงการเลื่อนภาพแผนภูมิเวลา



รูปที่ 6-27 แสดงภาพแผนภูมิเวลาที่ได้จากการพิมพ์

สรุป ในบทที่ 6 นี้เป็นการกล่าวถึงขั้นตอนการทำงานซึ่งนับได้ว่าเป็นส่วนสำคัญอย่างยิ่งของโปรแกรม LOGCAD ที่ใช้ในการเลียนแบบการทำงานของผังภาพวงจรตรรก (LOGCAD SIMULATOR)

อันจะประกอบด้วยส่วนทำงานหลัก ๆ ดังนี้

1. โปรแกรมหลักของ LOGCAD SIMULATOR
2. โปรแกรมควบคุมการแสดงคำสั่ง
3. โปรแกรมคำสั่งไฟล์ข้อมูล
4. โปรแกรมการกำหนดข้อมูลเข้าวงจรตรรก
6. โปรแกรมการแสดงผล
6. โปรแกรมการประมวลผลและการเลื่อนภาพแผนภูมิเวลา
7. โปรแกรมการพิมพ์ภาพแผนภูมิเวลา

การทำงานในขั้นตอนท้ายสุดจะได้ภาพแผนภูมิเวลา ณ.ตำแหน่งต่าง ๆ ของวงจร ซึ่งแสดงยังส่วนแสดงผลของคอมพิวเตอร์ทั้งที่เป็นจอภาพ และ เครื่องพิมพ์

บทที่ 7

ตัวอย่างการใช้งานโปรแกรม LOGCAD

ในขณะเริ่มต้นใช้งานโปรแกรม บนแผ่นหน่วยความจำดิสค์ปัจจุบันจะต้องมีไฟล์ที่จำเป็นบางไฟล์อยู่ ทั้งนี้เพื่อให้โปรแกรมสามารถทำงานได้ ไฟล์ต่าง ๆ เหล่านี้ได้แก่

- ไฟล์ไดร์เวอร์กราฟฟิค จะต้องมีทุกครั้งก่อนเรียกใช้โปรแกรม DRAFT1 และไม่จำเป็นต้องมีครบทุกไฟล์ไดร์เวอร์ แต่มีเฉพาะไฟล์ไดร์เวอร์ที่สามารถทำงานร่วมกับฮาร์ดแวร์การแสดงผลบนจอภาพก็เพียงพอ ดังนั้นกรณีที่ผู้ใช้โปรแกรมใช้กับจอภาพ EGA ไฟล์ไดร์เวอร์กราฟฟิคก็จะเป็น EGAVGA.BGI และถ้าใช้กับจอภาพ MONOCHROME ไฟล์ไดร์เวอร์กราฟฟิคก็จะเป็น HERC.BGI ถ้าหากใช้จอภาพที่นอกเหนือไปจากนี้โปรแกรมก็จะรายงานความผิดพลาดที่เกิดขึ้นบนจอแสดงผลดังนี้

```
GRAPHIC ERROR : DEVICE DRIVER FIND NOT FOUND.
```

- ไฟล์จัดขนาดตัวอักษร เพื่อให้การแสดงผลของข้อความในโหมดกราฟฟิคเป็นไปอย่างถูกต้อง ไฟล์นี้ได้แก่ LITT.CHR

สำหรับการใช้งานโปรแกรม LOGCAD จะแบ่งหัวข้อได้ดังนี้

- การสร้างรูปจาก LOGCAD EDITOR และ การสร้างไฟล์ข้อมูลโหนด
- การสร้างรูปจาก ORCAD EDITOR และ การสร้างไฟล์ข้อมูลโหนด
- การสร้างแบบจำลอง และ การแปลภาษา
- การจำลองการทำงานด้วย LOGCAD SIMULATOR

7.1 การสร้างรูปจาก LOGCAD EDITOR และการสร้างไฟล์ข้อมูลโหนด

ในส่วนการสร้างรูปจาก LOGCAD EDITOR และการสร้างไฟล์ข้อมูลโหนด จะแบ่งการใช้งานของโปรแกรมได้ดังนี้

- สร้างอุปกรณ์ไลบรารี
- สร้างผังภาพวงจร
- การสร้างไฟล์ข้อมูลโหนด

ตัวอย่าง การสร้างผังภาพวงจร DECADE COUNTER ดังรูปที่ 7-13

ก่อนที่จะทำการสร้างผังภาพวงจร ผู้ใช้จะต้องสร้างอุปกรณ์ไลบรารีต่าง ๆ ที่จำเป็นขึ้นมาเสียก่อน เพื่อกำหนดในผังภาพวงจรนั้น ๆ ดังรูปตัวอย่างอุปกรณ์ไลบรารีต้องสร้างได้แก่ INPUT NAND2 และ JKFFNL สำหรับในกรณีที่อุปกรณ์เหล่านี้มีพร้อมอยู่แล้ว ผู้ใช้ก็สามารถสร้างผังวงจรได้ทันที

7.1.1 การใช้งานโปรแกรมการสร้างไฟล์อุปกรณ์ไลบรารี

จากรายการเมนูดังรูปที่ 3-19 ผู้ใช้เลือกรายการคำสั่ง LIBRARY สำหรับสร้างรูปอุปกรณ์ไลบรารี ซึ่งจะต้องมีการกำหนดแบบและขนาดของรูปเสียก่อน เพื่อความเหมาะสม เพราะการสร้างรูปอุปกรณ์นั้นสามารถสร้างให้มีขนาด เล็ก-ใหญ่ ได้ตามความต้องการของผู้ใช้

ต่อจากนี้จะเป็นขั้นตอนของการสร้างไฟล์รูปอุปกรณ์ไลบรารี JKFFNL รูปอุปกรณ์ JKFFNL ประกอบด้วยส่วนย่อย 3 ส่วนได้แก่ เส้นตรง วงกลม และข้อความ ดังนั้นลำดับขั้นการสร้างจึงกำหนดได้ดังนี้

- สร้างรูปภายนอกด้วยเส้นตรง
- สร้างวงกลม
- สร้างเส้นตรงส่วนที่ขาดเพิ่มเติม
- กำหนดข้อความประกอบในรูป
- กำหนดขอบเขตการเก็บข้อมูลลงไฟล์อุปกรณ์
- กำหนดชื่ออุปกรณ์
- นำรูปอุปกรณ์เก็บเป็นไฟล์ไลบรารี

1. สร้างรูปภายนอกด้วยเส้นตรง

1.1 เลือกรายการเมนูคำสั่ง LIBRARY/EDIT/LINE COMMAND

1.2 ทำการสร้างเส้นตรงรูปภาพนอกของอุปกรณ์ โดยเปลี่ยนตำแหน่งตัวชี้ CURSOR ไปยังจุดแรกที่ต้องการสร้างแล้วกดแป้นพิมพ์ RETURN จะปรากฏจุดเครื่องหมายแสดงให้ผู้ใช้ทราบ จากนั้นเปลี่ยนตำแหน่งตัวชี้ไปยังจุดที่สองแล้วกดแป้นพิมพ์ RETURN อีกครั้งก็จะได้เส้นตรงเส้นแรกขึ้นมา สำหรับเส้นตรงอื่น ๆ ก็สามารถสร้างได้ในลักษณะเดียวกัน

1.3 หลังจากสร้างเส้นตรงครบแล้วจะได้ดังรูปที่ 7-2

2. สร้างวงกลม

2.1 เลือกรายการเมนูคำสั่ง LIBRARY/EDIT/CIRCLE COMMAND

2.2 สร้างวงกลม เริ่มจากการกำหนดจุดศูนย์กลาง โดยการเปลี่ยนตำแหน่งตัวชี้ไปยังจุดที่ต้องการแล้วจึงกดแป้นพิมพ์ RETURN จากนั้นให้กำหนดขนาดของวงกลมโดยเลื่อนตำแหน่งของตัวชี้ไปในทิศทางใด ๆ จนได้ขนาดที่ต้องการ จึงคอยกดแป้นพิมพ์ RETURN อีกครั้ง ก็จะได้วงกลมตามต้องการโดยมีรัศมีของวงกลมเท่ากับ ระยะการเคลื่อนที่ของตัวชี้ที่มีการกำหนดจากจุดศูนย์กลางของวงกลม

ในบางกรณีมีความจำเป็นที่ต้องสร้างวงกลมขนาดเล็ก ๆ ก็สามารถทำได้โดยการควบคุมระยะของ STEP CURSOR ให้มีความละเอียดมากขึ้นได้จากการกดแป้นพิมพ์ SHIFT เข้าช่วย ดังนั้นระยะการเคลื่อนที่ของ CURSOR ก็จะเปลี่ยนจากครั้งละ 10 สเต็ป ไปเป็น ครั้งละ 1 สเต็ปทันที

2.3 หลังจากสร้างวงกลมครบแล้วจะได้ดังรูปที่ 7-3

3. สร้างเส้นตรงส่วนที่ขาดเพิ่มเติม

3.1 เลือกรายการเมนูคำสั่ง LIBRARY/EDIT/LINE COMMAND

3.2 สร้างเส้นตรงส่วนที่ยังขาดอยู่ (เส้นตรงส่วนที่จะต่อกับรูปวงกลม) ทำเหมือนกับในหัวข้อ 1.2

3.3 หลังจากสร้างเส้นตรงครบแล้วจะได้ดังรูปที่ 7-4

4. กำหนดข้อความประกอบในรูป

4.1 เลือกรายการเมนูคำสั่ง LIBRARY/EDIT/TEXT COMMAND

4.2 กำหนดข้อความที่ต้องการ หลังจากที่โปรแกรมแสดงข้อความถามกลับมายังผู้ใช้

4.3 เลือกตำแหน่งของข้อความเพื่อกำหนดในรูป โดยการเปลี่ยนตำแหน่งของตัวชี้ไป จนกระทั่งได้ข้อความอยู่ในตำแหน่งที่เหมาะสมแล้วให้กดแป้นพิมพ์ RETURN

4.4 หลังจากกำหนดข้อความประกอบในรูปครบแล้ว จะได้ดังรูปที่ 7-5

5. การกำหนดขอบเขตการนำข้อมูลลงไฟล์อุปกรณ์ไลบรารี

5.1 เลื่อนตำแหน่งตัวชี้ไปยังจุดแรกของพื้นที่ขอบเขตที่ต้องการ

5.2 เลือกรายการเมนูคำสั่ง LIBRARY/BLOCK COMMAND

5.3 เลื่อนตำแหน่งตัวชี้ไปยังจุดสุดท้ายของพื้นที่ขอบเขตที่ต้องการ ในขณะนี้จะเห็นการเปลี่ยนสีของรูปเป็นอาณาบริเวณเหมือนกับรูปที่ 7-6 แล้วกดแป้นพิมพ์ RETURN

5.4 จากนั้นโปรแกรมจะแสดงข้อความให้ทำการกำหนดจุดอ้างอิง โดยการเลื่อนตำแหน่งตัวชี้ไปยังจุดอ้างอิงที่ต้องการและกดแป้นพิมพ์ RETURN

6. กำหนดชื่ออุปกรณ์

6.1 เลือกรายการคำสั่งไปที่ LIBRARY/LABEL COMMAND

6.2 โปรแกรมจะแสดงข้อความให้กำหนดชื่ออุปกรณ์ ซึ่งชื่อนี้เป็นชื่อเดียวกับชื่อไฟล์ของอุปกรณ์คือชื่อ JKFFNL ตามด้วยการกดแป้นพิมพ์ RETURN

7. การนำข้อมูลลงเก็บเป็นไฟล์ไลบรารี

7.1 เลือกรายการเมนูคำสั่ง LIBRARY/FILE/SAVE COMMAND

7.2 กำหนดชื่อไฟล์อุปกรณ์ไลบรารีในที่นี้หมายถึง JKFFNL แล้วกดแป้นพิมพ์ RETURN

7.3 ตรวจสอบข้อมูลที่เก็บโดยการเลือก LIBRARY/FILE/LOAD COMMAND อ่านข้อมูลจากไฟล์ JKFFNL ซึ่งจะต้องได้เหมือนดังรูปที่ 7-5

สำหรับ INPUT และ NAND2 ก็มีลำดับขั้นตอนการสร้างเช่นเดียวกันกับ JKFFNL และเมื่อสร้างอุปกรณ์ NAND2 และ INPUT เสร็จแล้วจะได้ดังรูปที่ 7-7 และ 7-8 ตามลำดับ

7.1.2 การใช้งานโปรแกรมการสร้างไฟล์ผังภาพวงจร

จากรายการเมนูดังรูปที่ 3-19 ผู้ใช้เลือกรายการ SCHEMATIC แล้วจะเป็นการใช้งานในส่วนของผังภาพวงจรตรรกก่อนเข้าสู่การทำงานส่วนนี้ จะต้องมีไฟล์อุปกรณ์เหล่านี้บนแผ่นหน่วยความจำได้แก่ไฟล์ INPUT.LIB NADN2.LIB และ JKFFNL.LIB

จากตัวอย่างวงจร DECADE COUNTER ประกอบด้วยส่วนย่อย 4 ส่วนคือ อุปกรณ์ไลบรารี เส้นตรง จุดต่อ และข้อความบอกโหนดของวงจร ดังนั้นลำดับขั้นการสร้างรูปวงจร DECADE COUNTER กำหนดได้ดังนี้

- กำหนดอุปกรณ์ไลบรารีประเภทต่าง ๆ
- ลากเส้นตรงทางเดินของสัญญาณ
- กำหนดจุดต่อสัญญาณและข้อความของโหนด
- กำหนดชื่อวงจร
- นำลงเก็บเป็นไฟล์ผังภาพวงจร

1. กำหนดอุปกรณ์ไลบรารีต่าง ๆ จะต้องมีการจัดตำแหน่งการลงเพื่อให้ง่ายต่อการพิจารณาวงจรมีลำดับขั้นการสร้างดังนี้

1.1 เลือกรายการเมนูคำสั่ง SCHEMATIC/EDIT/GET COMMAND

1.2 เลือกอุปกรณ์ JKFFNL ในเมนูรายการของอุปกรณ์ไลบรารี

1.3 นำรูปอุปกรณ์ JKFFNL ลงในตำแหน่งที่ต้องการโดยการเลื่อนตำแหน่งเหมือนกับการเลื่อนตัวชี้ และเมื่อได้ตำแหน่งของอุปกรณ์ตามความต้องการแล้วจึงค่อยกดแป้นพิมพ์ RETURN

1.4 เมื่อกำหนดตำแหน่งของอุปกรณ์ JKFFNL ครบแล้วจะได้ดังรูปที่ 7-9

1.5 กำหนดอุปกรณ์ INPUT และ NAND2 โดยมีขั้นตอนเหมือนกับการกำหนดอุปกรณ์ JKFFNL และเมื่อกำหนดเสร็จแล้วจะได้ดังรูปที่ 7-10

2. ลากเส้นตรงทางเดินของสัญญาณ

2.1 เลือกรายการเมนูคำสั่ง SCHEMATIC/EDIT/LINE COMMAND

2.2 ทำการลากเส้นตรงที่เป็นทางเดินของสัญญาณ มีขั้นตอนเหมือนกับการสร้างเส้นตรงในส่วนการสร้างอุปกรณ์ไลบรารี

2.3 เมื่อทำการลากเส้นตรงไปได้บางส่วนจะได้ดังรูปที่ 7-11

2.4 ทำการลากเส้นตรงภายในรูป เมื่อครบแล้วจะได้ดังรูปที่ 7-12

3. การลงจุดต่อสัญญาณและข้อมูลแบบข้อความของโหนด

3.1 เลือกรายการเมนูคำสั่ง SCHEMATIC/EDIT/CONNECT COMMAND

3.2 เลื่อนตำแหน่งจุดต่อที่ปรากฏตรงตัวชี้ไปยังจุดที่ต้องการแล้วกดแป้นพิมพ์ RETURN จากนั้นโปรแกรมจะถามให้ใส่ข้อความที่เป็น โหนดของวงจร (ถ้าไม่ต้องการใส่ข้อความดังกล่าว ให้กดแป้นพิมพ์ ESC หรือ RETURN)

3.3 เมื่อกำหนดจุดต่อสัญญาณ และข้อความของโหนดครบแล้วจะได้ดังรูปที่ 7-13

4. การกำหนดชื่อผังภาพวงจร

4.1 เลือกรายการเมนูคำสั่ง SCHEMATIC/LABEL COMMAND

4.2 กำหนดชื่อของผังภาพวงจร เมื่อโปรแกรมแสดงข้อความถามกลับมาการกำหนดนี้จะบอกเพียง ชื่อให้ทราบว่าเป็นวงจรอะไรแต่ไม่ได้หมายถึงชื่อไฟล์ของผังภาพวงจร

5. การนำเก็บเป็นไฟล์ผังภาพวงจร

5.1 เลือกรายการเมนูคำสั่ง SCHEMATIC/FILE/SAVE COMMAND

5.2 กำหนดชื่อไฟล์ที่จะนำข้อมูลของรูปลงเก็บ

เมื่อผ่านขั้นตอนเหล่านี้แล้วจะปรากฏไฟล์ DECADE.SCH บนแผ่นหน่วยความจำ การตรวจสอบทำโดยการใช้ SCHEMATIC/FILE/LOAD COMMAND

ซึ่งจะต้องได้รูปกลับออกมาแสดงเหมือนกับรูปที่ 7.13

7.1.3 การใช้งานโปรแกรมการสร้างไฟล์ข้อมูลโหนด

การสร้างไฟล์ข้อมูลโหนดเริ่มจากการเรียกโปรแกรม LOGNODE1 จากแผ่นหน่วยความจำที่อยู่ในไดร์ไหนก็ได้จากนั้นจะปรากฏข้อความ

```
ENTER FILE NAME [.SCH] :
```

ผู้ใช้กำหนดชื่อไฟล์ผังภาพวงจรเพื่อที่จะให้โปรแกรมอ่านข้อมูล นอกจากนี้ยังสามารถกำหนดไดร์ลงได้อีกด้วย และหลังจากนี้โปรแกรมจะอ่านไฟล์ผังภาพวงจรเพื่อที่จะลดทอนข้อมูลให้เหลือเฉพาะไฟล์ข้อมูลโหนด (FILENAME.NDE) ในกรณีชื่อไฟล์ที่กำหนดดังกล่าวไม่มีในแผ่นหน่วยความจำแล้ว โปรแกรมจะแสดงข้อความเตือนดังนี้

```
CANNOT OPEN FILE (xxxxxxxx.SCH) PROGRAM TERMINATED.
```

เมื่อเสร็จสิ้นการใช้โปรแกรม LOGNODE1 แล้วจะปรากฏไฟล์ข้อมูลโหนดที่มีชื่อเดียวกันกับไฟล์วงจรแต่มีนามสกุลเป็น NDE ขึ้นมาอีกหนึ่งไฟล์

```
PRE
PINJ
CLK
PINK
PINQ1
Q1BAR
PINQ2
Q2BAR
PINQ3
Q3BAR
PINQ4
Q4BAR
CLR
```

ไฟล์ข้อมูลโหนด (DECACE.NDE) จากโปรแกรม LOGCAD EDITOR

7.2 การสร้างรูปจากโปรแกรม ORCAD EDITOR และการสร้างไฟล์ข้อมูลโหนด

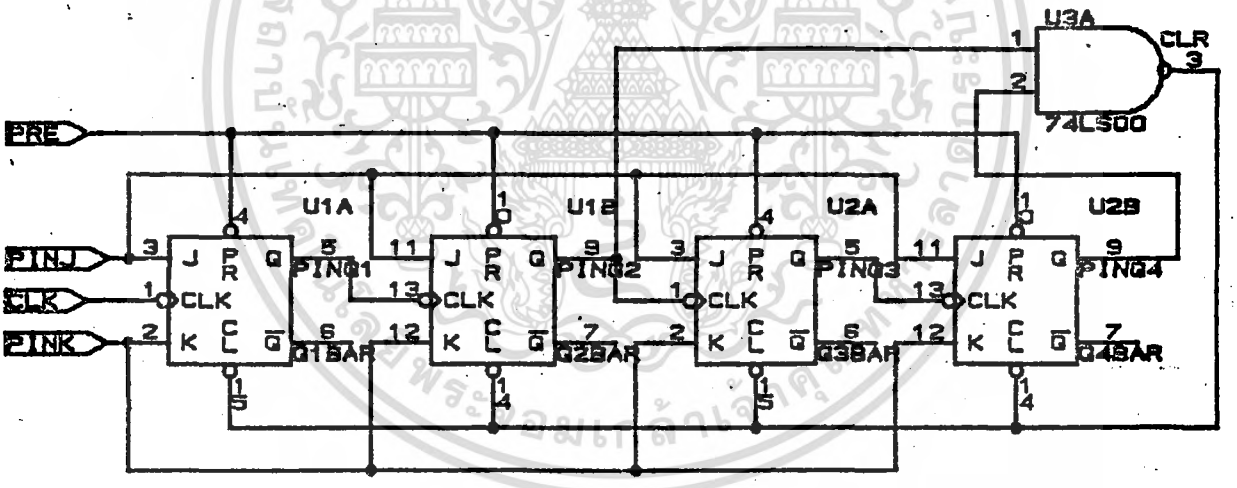
การใช้งานในส่วนนี้จะประกอบด้วยส่วนต่าง ๆ คือ

- การสร้างผังภาพวงจรจากโปรแกรม ORCAD EDITOR
- การสร้างไฟล์ข้อมูลโหนด

7.2.1 การใช้งานโปรแกรมการสร้างผังภาพวงจรจาก ORCAD EDITOR

โปรแกรม ORCAD เป็นโปรแกรมที่นิยมกันมากในวงกว้าง ดังนั้นขั้นตอนในการสร้างผังภาพวงจร DECADE COUNTER จะไม่ขอกล่าวถึง แต่มีสิ่งสำคัญอย่างหนึ่งที่ผู้ใช้จะต้องมีการกำหนดในระหว่างที่สร้างผังภาพวงจรอยู่นั้นก็คือ โหนดต่าง ๆ ของวงจรจะต้องสร้างด้วยคำสั่ง PLACE/MODULE PORT/INPUT และ PLACE/MODULE PORT/OUTPUT และ PLACE/LABLE/INTERNAL ในส่วนที่เป็น INPUT OUTPUT และจุดเชื่อมต่อตามลำดับ

เมื่อสร้างวงจรเสร็จเป็นที่เรียบร้อยแล้วจะได้ดังรูปที่ 7-14 นำข้อมูลของผังภาพวงจรที่สร้างขึ้นนี้ไปทำการ PLOT กับเครื่อง PLOTTER จะได้ดังรูป 7-1



รูปที่ 7-1 ผังภาพวงจร DECADE COUNTER ที่ได้จากเครื่อง PLOTTER

7.2.2 การใช้งานโปรแกรมสร้างไฟล์ข้อมูลโหนด

การใช้งานวิธีสร้างไฟล์ข้อมูลโหนดมีขั้นตอนดังนี้

- การทำงานของโปรแกรม NETLIST
- การทำงานของโปรแกรม LOGNODE

สมมุติว่าผู้ใช้โปรแกรมใช้งานอยู่ที่ LOCK DRIVE A ดังนั้นในแต่ละขั้นตอนจึงมีวิธีการดังนี้

1. การใช้งานโปรแกรม NETLIST ซึ่งเป็นส่วนหนึ่งของโปรแกรม ORCAD ที่ต้อง

ใช้ร่วมกับโปรแกรมการสร้างรูปผังภาพวงจร (DRAFT PROGRAM) การใช้งานให้กำหนดตามรูปมาตรฐานดังนี้

A > NETLIST SOURCE FILE DESTINATION FILE

- SOURCE FILE จะเป็นไฟล์ของผังภาพวงจรที่สร้างจากโปรแกรม DRAFT ของ ORCAD ที่มีสกุลเป็น SCH

- DESTINATION FILE จะเป็นไฟล์ผลลัพธ์ที่ได้จากการทำงาน ของโปรแกรม NETLIST ไฟล์ผลลัพธ์นี้จะเป็นข้อมูลที่บอกถึงรายละเอียดของอุปกรณ์ต่าง ๆ สำหรับประกอบเป็นผังภาพวงจรนั้น ๆ ซึ่งข้อมูลเหล่านี้จะมีมากเกินความต้องการ ดังนั้นจึงต้องมีการลดจำนวนของข้อมูลลง เพื่อให้เหลือเฉพาะข้อมูลที่เป็นโหนดของวงจรเท่านั้น จะได้กล่าวถึงในขั้นตอนต่อไป

ดังนั้นในขั้นตอนนี้กรณีที่ผังภาพวงจรเป็นวงจร DECADE COUNTER ผู้ใช้จะต้องกำหนดดังนี้

A > NETLIST DECADE.SCH DECADE.NLT

2. การใช้งานโปรแกรม LOGNODE

ผู้ใช้สามารถใช้งานโปรแกรม LOGNODE ดังนี้

A > LOGNODE แล้วกดแป้นพิมพ์ RETURN

โปรแกรมจะแสดงข้อความถามต่อดังนี้

ENTER FILE NAME[.NLT]· BUT RETURN OT EXIT :

จากนั้นผู้ใช้จะต้องกำหนดชื่อไฟล์ที่ต้องการลดทอนข้อมูล ในขั้นตอนนี้ (ใส่เฉพาะชื่อไฟล์เท่านั้น เพราะว่าโปรแกรมจะกำหนดสกุลเป็น NLT ไว้ให้แล้ว) เมื่อผู้ใช้กำหนดชื่อไฟล์ไปเรียบร้อยแล้ว โปรแกรมจะทำการตรวจสอบชื่อไฟล์กับแผ่นหน่วยความจำว่ามีหรือไม่ ถ้าผลการตรวจสอบแล้วปรากฏว่า

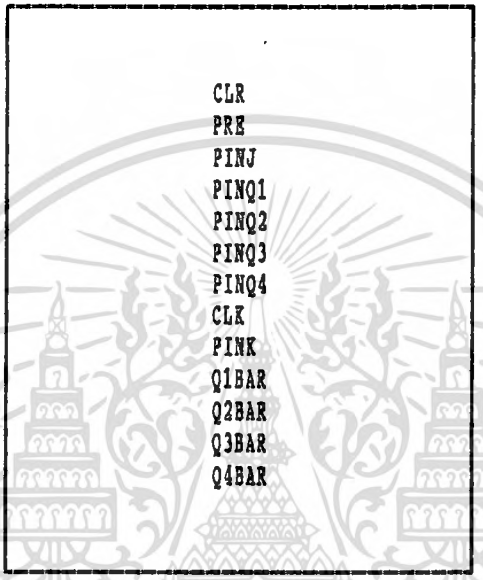
- มี จะแสดงข้อความ

NODE FILE CREATED.

- ไม่มี จะแสดงข้อความ

CANNOT OPEN FILE! PROGRAM TERMINATED.

ในกรณีที่ไฟล์ข้อมูลโหนดสามารถสร้างขึ้นมาได้ ดังนั้นในแผ่นหน่วยความจำจะมีชื่อไฟล์ DECADE.NDE อยู่



ไฟล์ข้อมูลโหนด (DECADE.NDE) จากโปรแกรม ORCAD

7.3 การสร้างแบบจำลอง และ การแปลภาษา

การใช้งานในส่วนนี้จะประกอบด้วยส่วนต่าง ๆ คือ (ให้ดูจากรูป 2-2 ประกอบจะเข้าใจการทำงานมากขึ้น)

- การเขียนแบบจำลอง
- การใช้งานโปรแกรม LOGNET เพื่อสร้างไฟล์ DATATXRX.PAS
- การใช้งานโปรแกรม PRESIM เพื่อทำการแปลภาษา

7.3.1 การเขียนแบบจำลอง

การเขียนแบบจำลองได้กล่าวมาแล้วในหัวข้อ 5.1.2 ดังนั้นแบบจำลองของวงจร DECADE COUNTER จะถูกสร้างเก็บลงสู่แผ่นหน่วยความจำที่มีชื่อไฟล์เป็น DECADE.MDL ซึ่งมีรายละเอียดในการเขียนแบบจำลองดังนี้

```
Var Last1 : Boolean;
    Last2 : Boolean;
    Last3 : Boolean;
    Last4 : Boolean;
begin
  IF (PRE = ON) AND (CLR = ON) AND (CLK = OFF) AND (LAST1 = ON)  THEN
    BEGIN     { FIRST JK / FF }
      IF (PINJ = ON) AND (PINK = ON) THEN
        PINQ1 := NOT(PINQ1);
      IF (PINJ = ON) AND (PINK = OFF) THEN
        PINQ1 := ON;
      IF (PINJ = OFF) AND (PINK = ON) THEN
        PINQ1 := OFF;
      IF (PINJ = OFF) AND (PINK = OFF) THEN
        PINQ1 := PINQ1;
    END;
  IF (PRE = ON) AND (CLR = ON) AND (PINQ1 = OFF) AND (LAST2 = ON)  THEN
    BEGIN      { SECOUND JK/FFN }
      IF (PINJ = ON) AND (PINK = ON) THEN
        PINQ2 := NOT(PINQ2);
      IF (PINJ = ON) AND (PINK = OFF) THEN
        PINQ2 := ON;
      IF (PINJ = OFF) AND (PINK = ON) THEN
        PINQ2 := OFF;
      IF (PINJ = OFF) AND (PINK = OFF) THEN
        PINQ2 := PINQ2;
    END;
  IF (PRE = ON) AND (CLR = ON) AND (PINQ2 = OFF) AND (LAST3 = ON)  THEN
    BEGIN      { THIRD JK/FFN }
      IF (PINJ = ON) AND (PINK = ON) THEN
        PINQ3 := NOT(PINQ3);
      IF (PINJ = ON) AND (PINK = OFF) THEN
        PINQ3 := ON;
      IF (PINJ = OFF) AND (PINK = ON) THEN
        PINQ3 := OFF;
      IF (PINJ = OFF) AND (PINK = OFF) THEN
        PINQ3 := PINQ3;
    END;
  IF (PRE = ON) AND (CLR = ON) AND (PINQ3 = OFF) AND (LAST4 = ON)  THEN
    BEGIN      { FOURTH JK/FFN }
      IF (PINJ = ON) AND (PINK = ON) THEN
        PINQ4 := NOT(PINQ4);
      IF (PINJ = ON) AND (PINK = OFF) THEN
        PINQ4 := ON;
      IF (PINJ = OFF) AND (PINK = ON) THEN
        PINQ4 := OFF;
      IF (PINJ = OFF) AND (PINK = OFF) THEN
        PINQ4 := PINQ4;
    END;
    CLR := Not(PINQ2 AND PINQ4);     { Clear Flip Flop }
    IF (CLR = OFF) THEN
      BEGIN
        PINQ1 := OFF;
        PINQ2 := OFF;
        PINQ3 := OFF;
        PINQ4 := OFF;
      END;
    IF (PRE = OFF) AND (CLR = ON) THEN
      BEGIN
        PINQ1 := ON;
        PINQ2 := ON;
        PINQ3 := ON;
        PINQ4 := ON;
      END;
    LAST1 := CLK;
    LAST2 := PINQ1;
    LAST3 := PINQ2;
    LAST4 := PINQ3;
    Q1BAR := NOT(PINQ1);
    Q2BAR := NOT(PINQ2);
    Q3BAR := NOT(PINQ3);
    Q4BAR := NOT(PINQ4);
END;
```

7.3.2 การใช้งานโปรแกรม LOGNET

เพื่อสร้างไฟล์ DATATXRX.PAS ซึ่งเป็นยูนิต ขั้นตอนการสร้างไฟล์ DATATXRX นี้จะต้องผ่านการสร้างไฟล์ข้อมูลโหนด และ ไฟล์ข้อมูลแบบจำลอง จากนั้นจึงจะสร้างไฟล์ DATATXRX ได้ ถ้าขาดไฟล์ใดไฟล์หนึ่งแล้วโปรแกรมจะไม่สามารถสร้างไฟล์ DATATXRX.PAS ขึ้นมาได้

การใช้งานโปรแกรม LOGNET ให้ผู้ใช้ทำการกำหนดดังนี้

A > LOGNET แล้วกดแป้นพิมพ์ RETURN

โปรแกรมจะแสดงข้อความถามกลับมาดังนี้

```
ENTER INCLUDE FILE NAME [.NDE AND .MDL]  BUT RETURN TO EXIT :
```

จากนั้นผู้ใช้จะต้องกำหนดชื่อไฟล์ที่ต้องการสร้าง ในที่นี้คือ DECADE แล้วโปรแกรมจะจัดการอ่านไฟล์ทั้งสอง (.NDE และ .MDL) มาสร้างเป็นไฟล์ DATATXRX.PAS ที่เป็นยูนิต มาทำการตรวจสอบว่าไฟล์ทั้งสองดังกล่าวมีอยู่ในแผ่นหน่วยความจำ หรือ ไม่ ผลการตรวจสอบปรากฏว่า

- มี จะแสดงข้อความว่า DATATXRX.PAS FILE CREATED.
- ไม่มี จะแสดงข้อความว่า CANNOT OPEN FILE! PROGRAM TERMINATED.

ขั้นตอนดังกล่าวนี้ก็จะได้ไฟล์ DATATXRX.PAS ที่เป็นยูนิต

7.3.3 การใช้งานโปรแกรม PRESIM เพื่อทำการแปลภาษา

ในขั้นตอนนี้จะเป็นเพียง BATCH FILE ดังนั้นผู้ใช้งานเพียงแต่กำหนดชื่อไฟล์ โปรแกรมจะจัดการแปลภาษาให้ เช่น

A > PRESIM แล้วกดแป้นพิมพ์ RETURN

โปรแกรมจะทำการแปลภาษาทันที และในกรณีที่ผู้ใช้เขียนแบบจำลองผิดไปจากวิธีเขียนโปรแกรมที่เป็นมาตรฐานของ TURBO PASCAL แล้ว ตัวแปลภาษาจะไม่สามารถทำการแปลได้ ในขณะเดียวกันจะแสดงความผิดพลาดต่าง ๆ เหล่านี้ที่เกิดขึ้นให้ผู้ใช้ทราบ ดังนั้นผู้ใช้งานจะต้องกลับไปแก้ไขแบบจำลองกันใหม่ แล้วจึงดำเนินการตามขั้นตอนต่าง ๆ อย่างเดิม สำหรับกรณีที่โปรแกรมสามารถแปลภาษาได้ก็จะสร้างไฟล์ LOGSIM.EXE ให้ซึ่งเป็นไฟล์ที่จะใช้ในการจำลองการทำงานของวงจร DECADE CONTER ต่อไป

7.4 การจำลองการทำงานของวงจร LOGCAD SIMULATOR

ส่วนในการใช้งานของโปรแกรมจะแบ่งออกเป็นส่วน ๆ ตามลำดับขั้นการใช้งานดังนี้

- การกำหนดชื่อไฟล์เพื่อจำลองการทำงาน
- การกำหนดข้อมูลเข้าของวงจร
- การโพรบตำแหน่งต่าง ๆ เพื่อแสดงภาพแผนภูมิเวลา
- การประมวลผลและการกำหนดส่วนต่าง ๆ เพื่อใช้ในการเลื่อนภาพแผนภูมิเวลา
- การพิมพ์ภาพแผนภูมิเวลา

ในการใช้งานโปรแกรมในส่วนนี้เริ่มต้นโดยการเรียกไฟล์ LOGSIM.EXE

A > LOGSIM แล้วกดแป้นพิมพ์ RETURN

จากนั้นโปรแกรมจะแสดง TITLE ดังรูปที่ 5-15 ให้ผู้ใช้กดแป้นพิมพ์ข้อความ "LOGCAD" ก็จะเป็นการเข้าสู่การทำงานโปรแกรม LOGCAD SIMULATOR และเมนูคำสั่งหลักของโปรแกรมจะแสดงดังรูปที่ 7-15

7.4.1 การกำหนดชื่อไฟล์เพื่อจำลองการทำงาน

ในขั้นตอนการกำหนดชื่อไฟล์เพื่อจำลองการทำงานแบ่งออกได้ดังนี้

1. ในกรณีที่ผู้ใช้ไม่สามารถจำชื่อไฟล์ที่จะทำการจำลองการทำงานได้โปรแกรมจะมีวิธีการแสดง DIRECTORY ของไฟล์ข้อมูลโหนดให้เห็นได้ด้วยการเลือกคำสั่ง FILE/DIREC ก็จะสามารถแสดงชื่อไฟล์ข้อมูลโหนดต่าง ๆ ได้ดังรูป 7-16
2. ในกรณีที่ผู้ใช้จำชื่อไฟล์ได้ก็ไม่จำเป็นที่ต้องปฏิบัติในข้อ 1 ให้ปฏิบัติในข้อ 2 ได้เลย ซึ่งเป็นขั้นตอนการอ่านไฟล์ข้อมูลโหนดให้ผู้ใช้เลือกคำสั่ง FILE/LOAD จากนั้นกำหนดเฉพาะชื่อในที่นี้คือ DECADE
3. ออกจากโปรแกรย่อย FILE ด้วยคำสั่ง FILE/EXIT

7.4.2 การกำหนดข้อมูลเข้าของวงจร

โดยการระบุชื่อบางชื่อของไฟล์ข้อมูลโหนดให้เป็นข้อมูลเข้าของวงจร ส่วนชื่อที่ไม่ได้ถูกระบุจะถือว่าเป็น โหนดเชื่อมต่อ และ โหนดเอาท์พุท ณ.จุดต่าง ๆ ของวงจรนั้น คำสั่งที่ใช้ในการกำหนดข้อมูลเข้านี้คือ DATA/CLOCK ดังรูปที่ 7-17

ขั้นตอนการกำหนดข้อมูลเข้ามีดังนี้

1. กำหนดชื่อของโหนดข้อมูลเข้าทางแป้นพิมพ์ ดังรูปที่ 7-18 จะให้ PINJ เป็น

โหนดข้อมูลเข้าของวงจร

2. กำหนดจำนวน LOW STATE ของโหนด PINJ = 0 สเต็ป ดังรูป 7-19
3. กำหนดจำนวน HIGH STATE ของโหนด PINJ = 1 สเต็ป ดังรูป 7-20
4. ในโหนดอื่น ๆ ที่จะกำหนดให้เป็นข้อมูลเข้าสามารถทำได้เช่นเดียวกับโหนด PINJ สำหรับวงจร DECADE COUNTER นี้โหนดที่เป็นข้อมูลเข้า นอกจาก PINJ แล้วยังมีโหนด PINK CLK PRE อีกซึ่งสามารถแสดงการกำหนดข้อมูลเข้าเหล่านี้ได้ดังรูปที่ 7-21 ด้วยคำสั่ง DATA/LIST./CLOCK

7.4.3 การโพรบตำแหน่งต่าง ๆ เพื่อแสดงภาพแผนภูมิเวลา

เป็นการโพรบชื่อของโหนดที่ต้องการแสดงภาพแผนภูมิเวลามีขั้นตอนดังนี้

1. เมื่อผู้ใช้โปรแกรมไม่สามารถจำชื่อโหนดของวงจรได้สามารถใช้คำสั่ง DISP/PROBE/LIST เพื่อดูโหนดต่าง ๆ ของวงจร DECADE COUNTER ได้ดังรูปที่ 7-22
2. กรณีที่ผู้ใช้จำชื่อโหนดต่าง ๆ ของวงจรได้ก็ให้กำหนดชื่อโหนดเหล่านั้นภายในคำสั่ง DISP/PROBE/ENTER ดังรูปที่ 7-23
3. ออกจากโปรแกรมย่อย PROBE ด้วยคำสั่ง DISP/PROBE/EXIT
4. ออกจากโปรแกรมย่อย DISP ด้วยคำสั่ง DISP/EXIT

7.4.4 การประมวลผลและการกำหนดส่วนต่าง ๆ เพื่อใช้ในการเลื่อนภาพแผนภูมิเวลา การประมวลผลและการกำหนดสเต็ปเพื่อใช้ในการแสดงและเลื่อนภาพแผนภูมิเวลามีขั้นตอนดังนี้

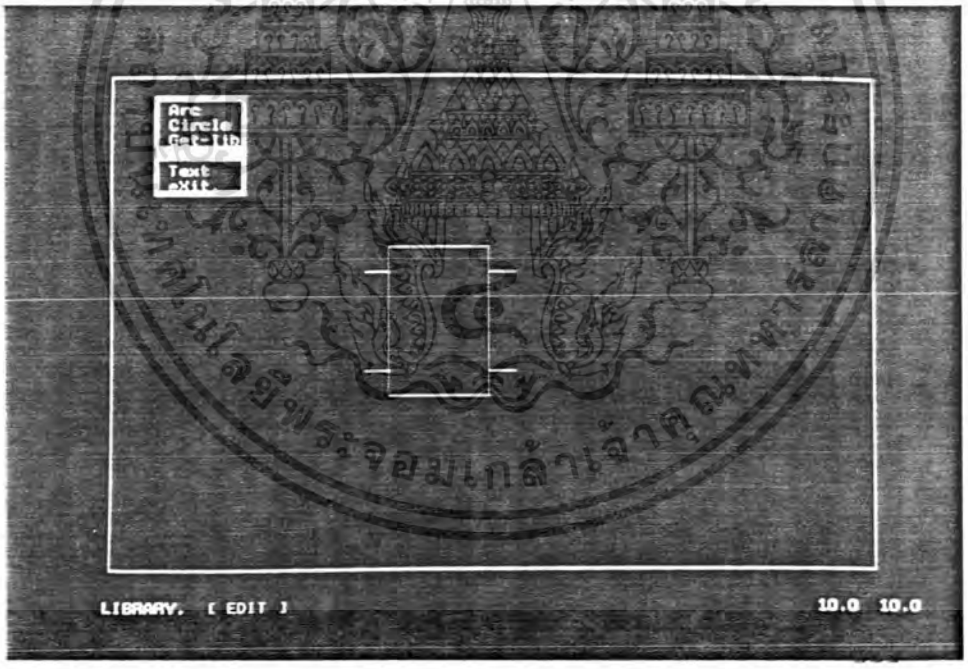
1. การประมวลผลจะมีได้ก็ต่อเมื่อมีการกำหนดส่วนต่าง ๆ ทั้งหมดดังที่ได้กล่าวมาแล้วข้างต้นและการประมวลผลนี้จะทำได้ด้วยคำสั่ง SIM/START ดังรูปที่ 7-24
2. การกำหนดจำนวนสเต็ปเพื่อใช้ในการเลื่อนภาพแผนภูมิไปทางซ้าย สามารถกำหนดได้ด้วยคำสั่ง SIM/LSTEP ดังรูปที่ 7-25 สำหรับการกำหนดจำนวนสเต็ปเพื่อให้ในการเลื่อนภาพแผนภูมิไปทางขวาก็สามารถทำได้เช่นเดียวกันด้วยคำสั่ง SIM/RSTEP
3. การกำหนดสเต็ปเริ่มต้นเพื่อแสดงภาพแผนภูมิเวลา สามารถทำได้โดยการใช้คำสั่ง SIM/SSTEP ดังรูป 7-26

7.4.5 การพิมพ์ภาพแผนภูมิเวลา

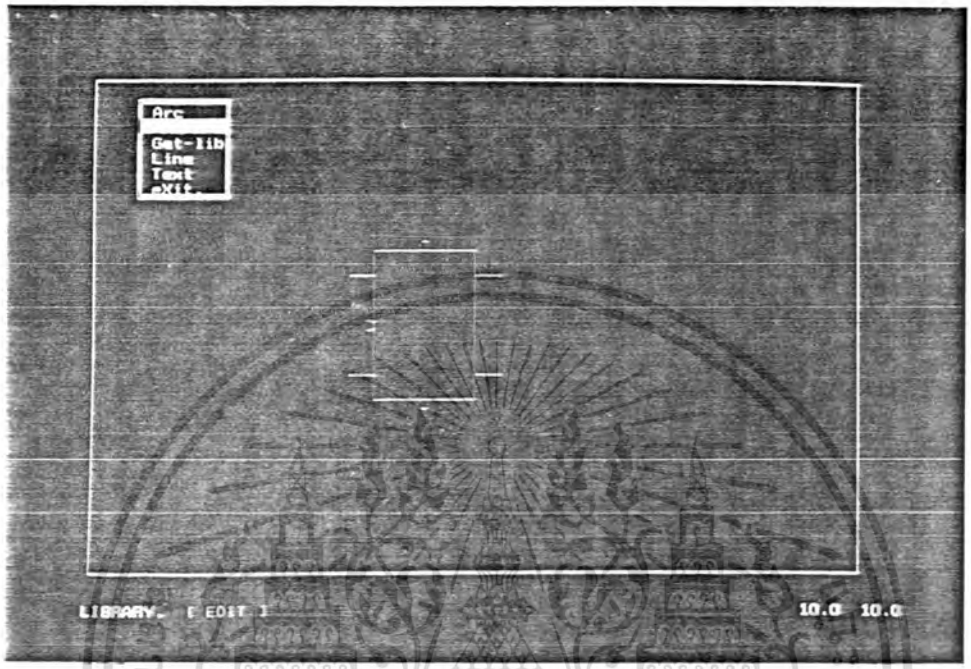
ในการพิมพ์ภาพแผนภูมิเวลา ผู้ใช้สามารถจะพิมพ์ที่สเต็ปการทำงานเท่าใดของวงจรก็ได้ซึ่งมีขั้นตอนการใช้งานดังนี้

1. กำหนดการพิมพ์จากคำสั่ง PRINT COMMAND
2. กำหนดสเต็ปเริ่มต้นที่ต้องการพิมพ์ทางแป้นพิมพ์
3. กำหนดสเต็ปสุดท้ายที่จะให้พิมพ์ถึงทางแป้นพิมพ์
4. ผลการพิมพ์จะได้ดังรูป 7-27

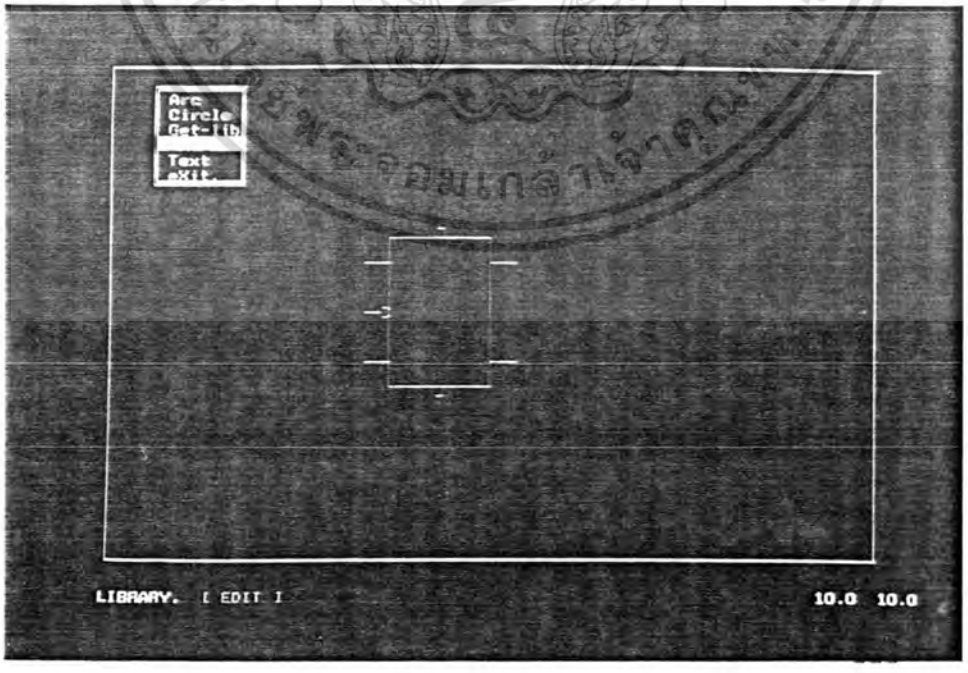
เมื่อเสร็จจากขั้นตอนการทำงานตั้งแต่หัวข้อ 7.1 จนกระทั่งถึงหัวข้อ 7.4.5 จะเป็นการใช้งานของโปรแกรม LOGCAD ที่ทำการจำลองการทำงานของผังภาพวงจร DECADE COUNTER สำหรับผังภาพวงจรอื่น ๆ นั้นก็สามารถทำได้ในลักษณะเช่นเดียวกัน



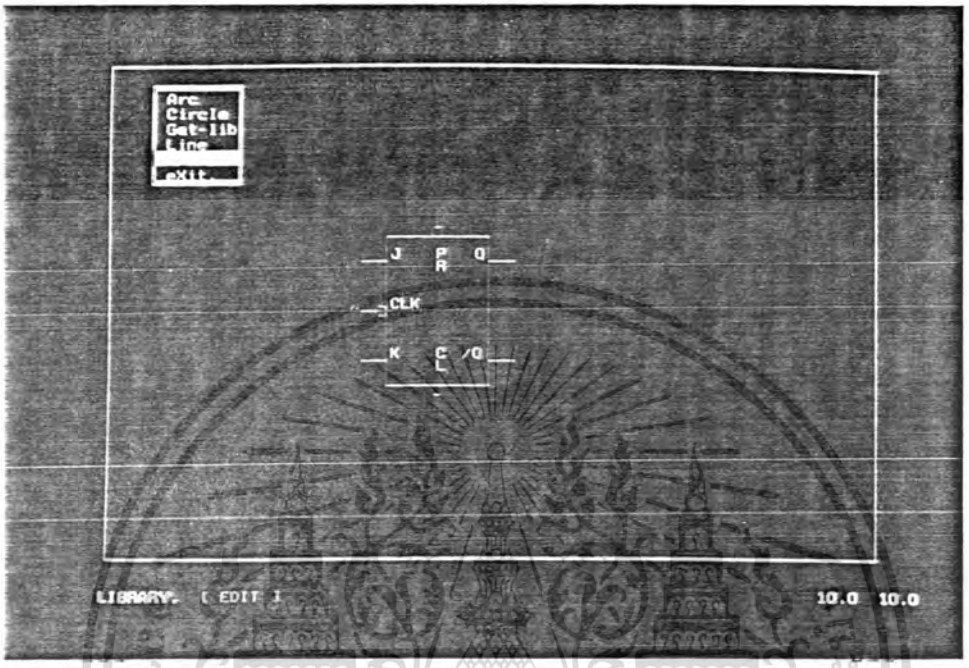
รูปที่ 7-2 รูปอุปกรณ์ไลบรารีในระหว่างที่ทำการสร้างด้วยคำสั่ง LINE



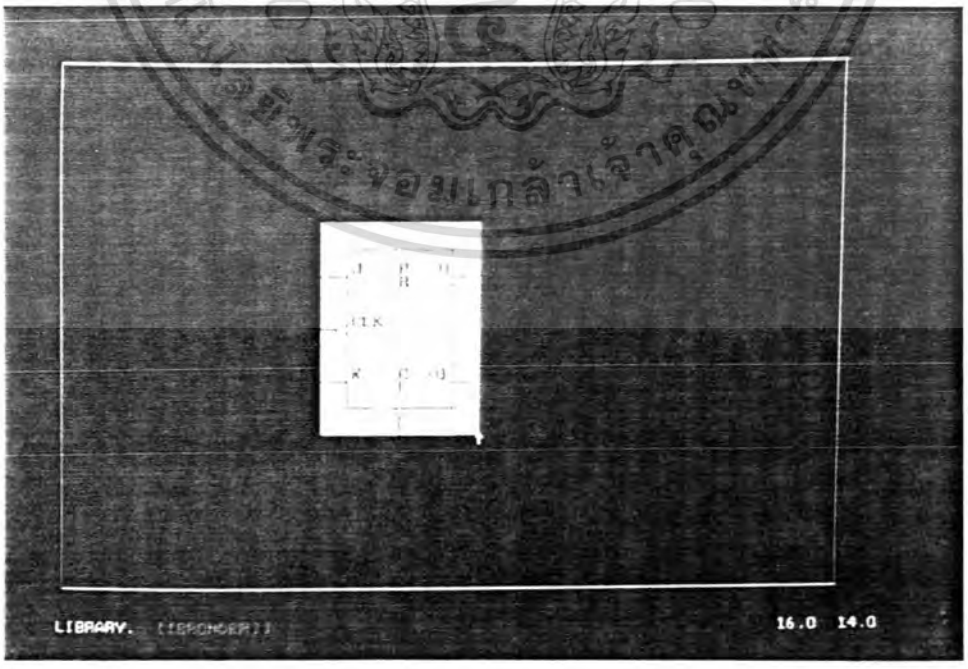
รูปที่ 7-3 รูปอุปกรณ์ไลบรารีในระหว่างที่ทำการสร้างด้วยคำสั่ง CIRCLE



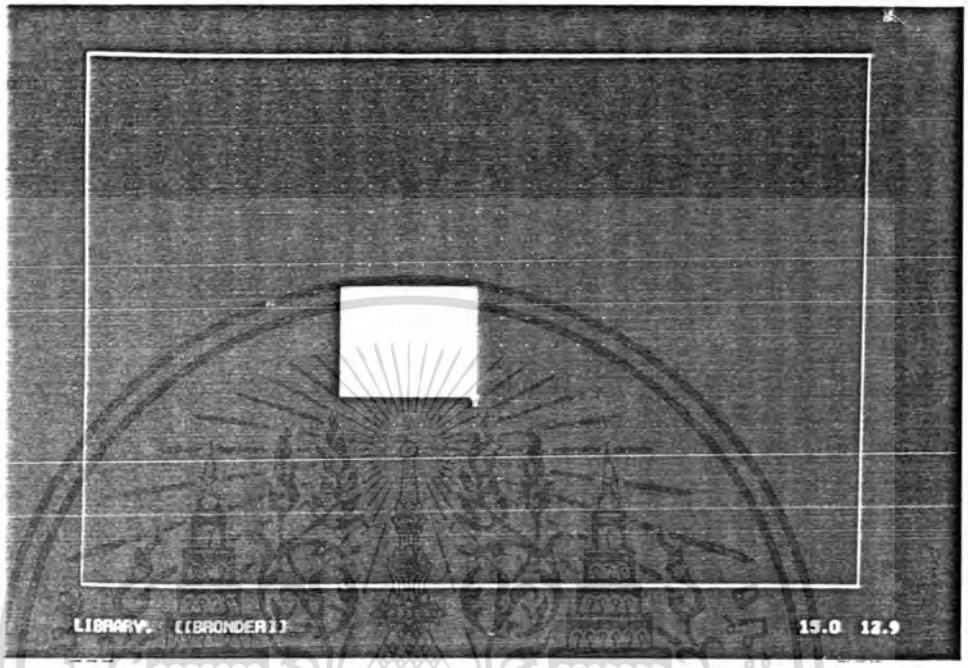
รูปที่ 7-4 รูปอุปกรณ์ไลบรารีเพิ่มเติมในระหว่างที่ทำการสร้างด้วยคำสั่ง LINE



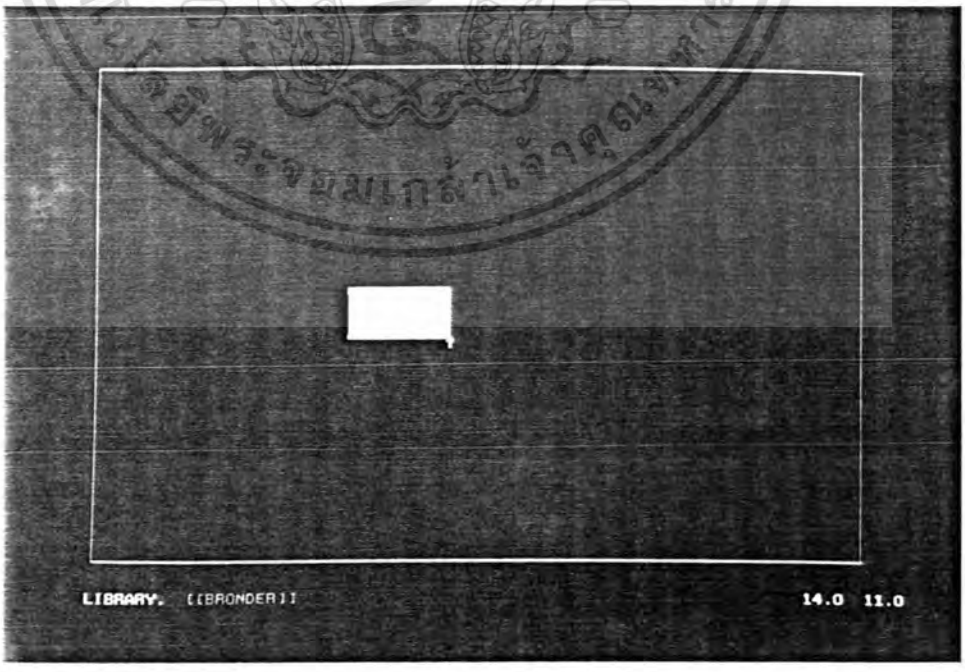
รูปที่ 7-5 รูปอุปกรณ์ไลบรารีในระหว่างที่ทำการสร้างด้วยคำสั่ง TEXT



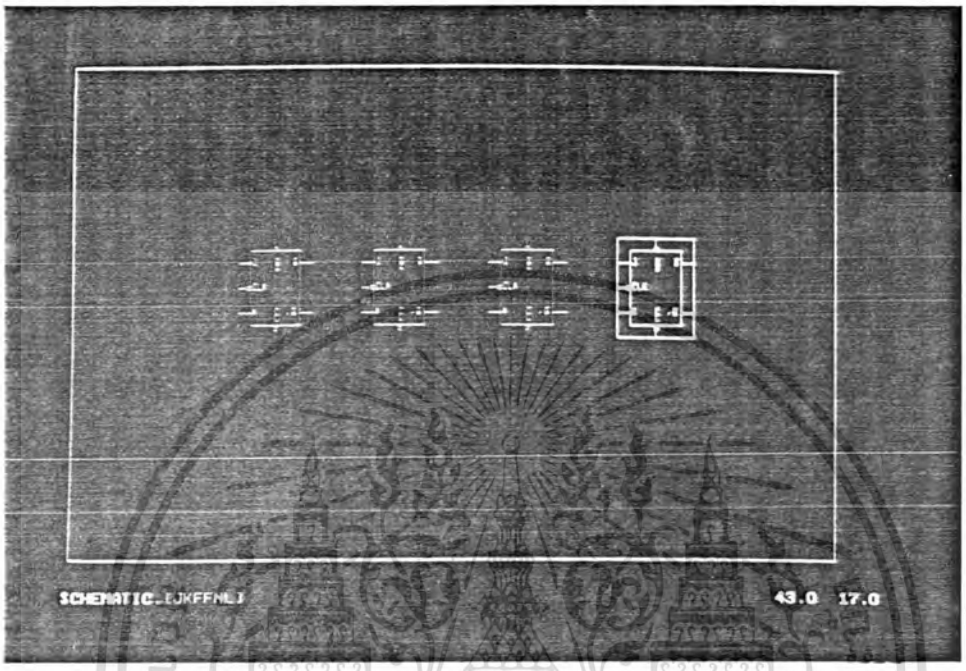
รูปที่ 7-6 รูปอุปกรณ์ JKFFNL ที่มีการกำหนดขอบเขตด้วยคำสั่ง BLOCK



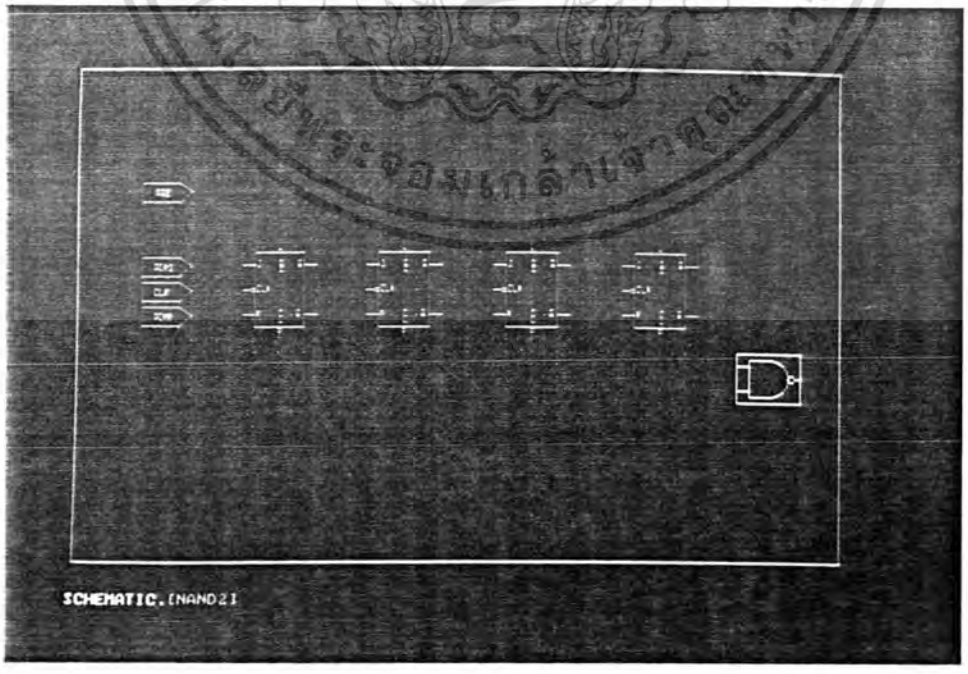
รูปที่ 7-7 รูปอุปกรณ์ NAND2 ที่มีการกำหนดขอบเขตด้วยคำสั่ง BLOCK



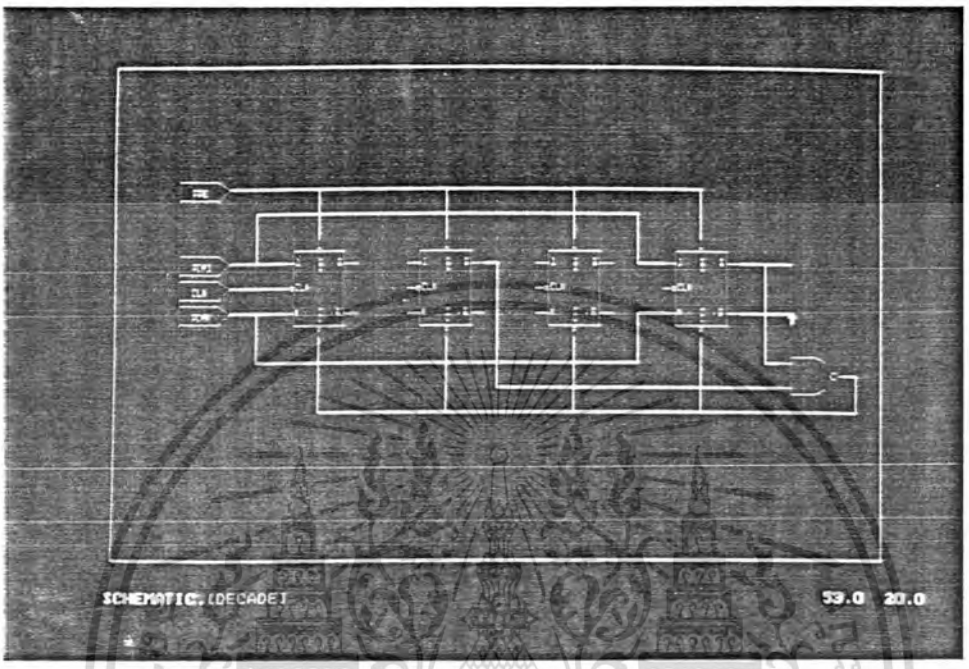
รูปที่ 7-8 รูปอุปกรณ์ INPUT ที่มีการกำหนดขอบเขตด้วยคำสั่ง BLOCK



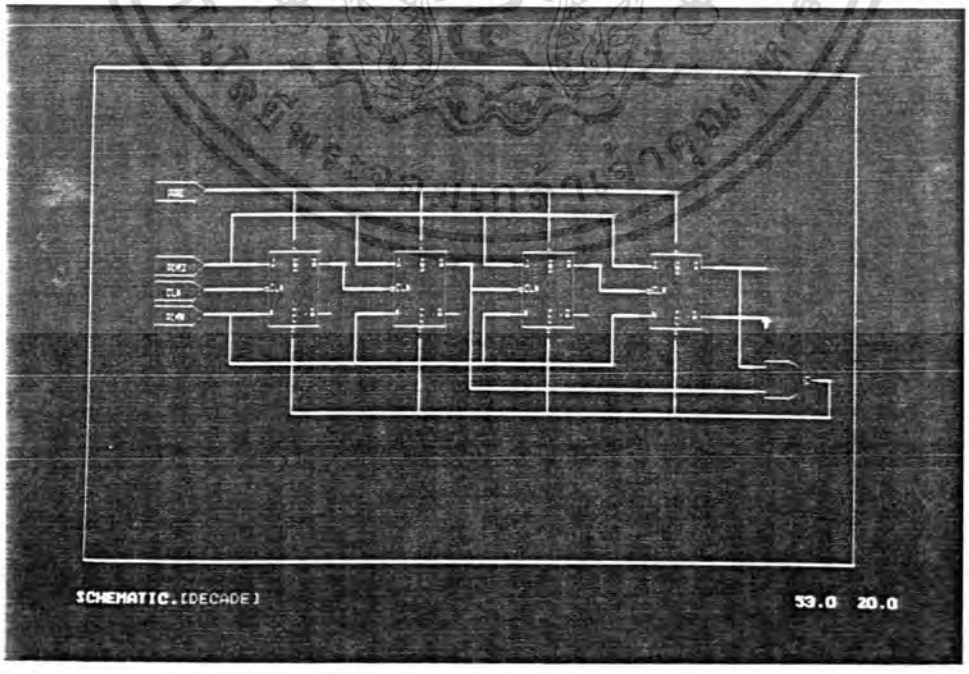
รูปที่ 7-9 รูปผังภาพวงจรที่มีการกำหนดด้วยอุปกรณ์ไลบรารี JKFFNL



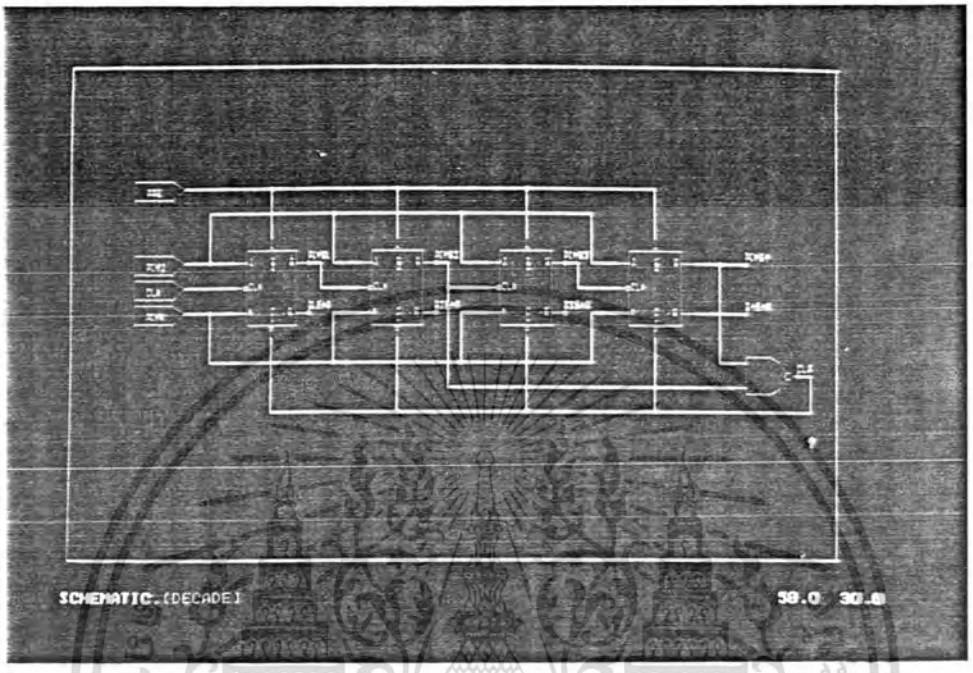
รูปที่ 7-10 รูปผังภาพวงจรที่มีการกำหนดด้วยอุปกรณ์ไลบรารี NAND2



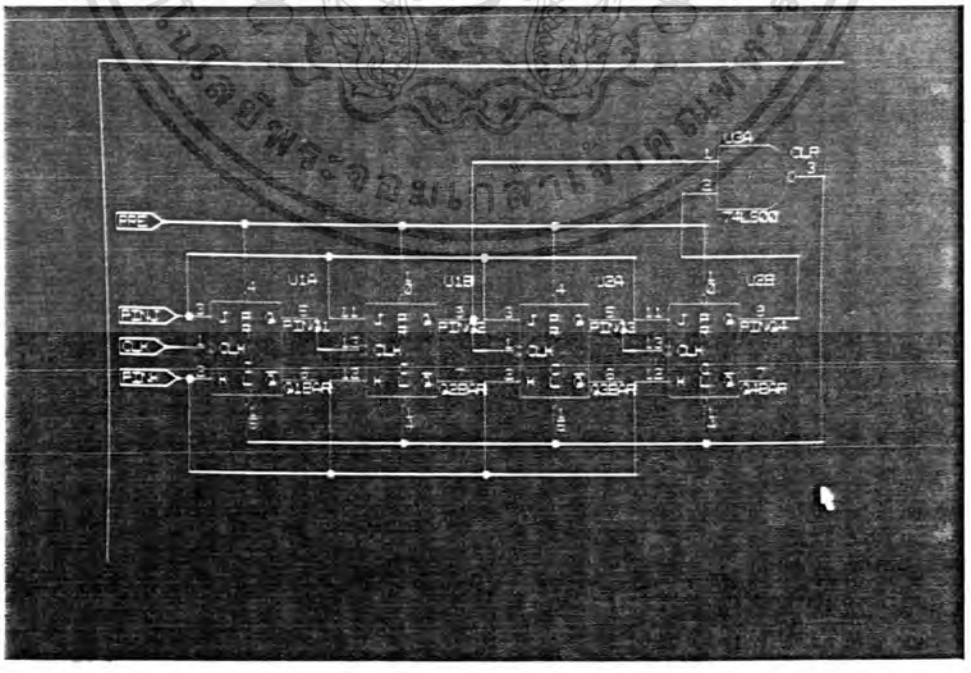
รูปที่ 7-11 รูปผังภาพวงจรในระหว่างที่ทำการสร้างด้วยคำสั่ง LINE



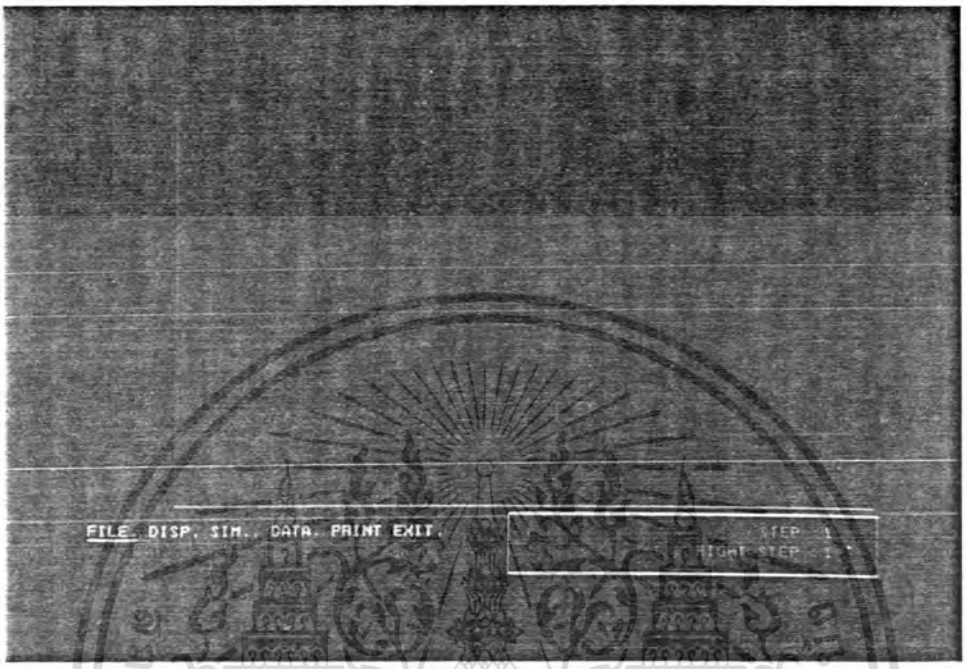
รูปที่ 7-12 รูปผังภาพวงจรในระหว่างที่ทำการสร้างด้วยคำสั่ง LINE



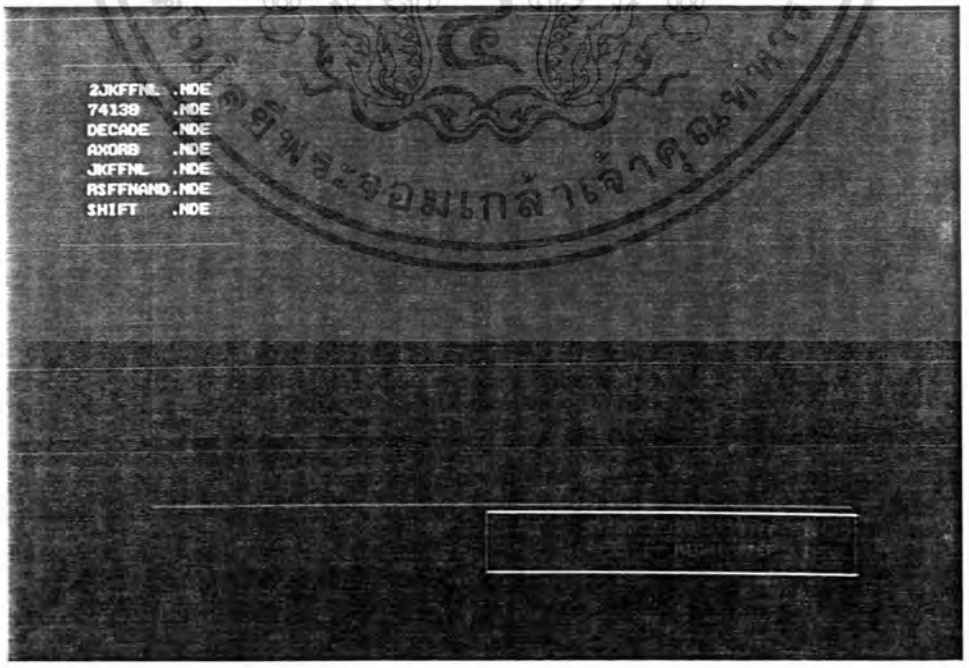
รูปที่ 7-13 รูปผังภาพวงจร DECADE COUNTER ที่สร้างจาก LOCAD EDITOR



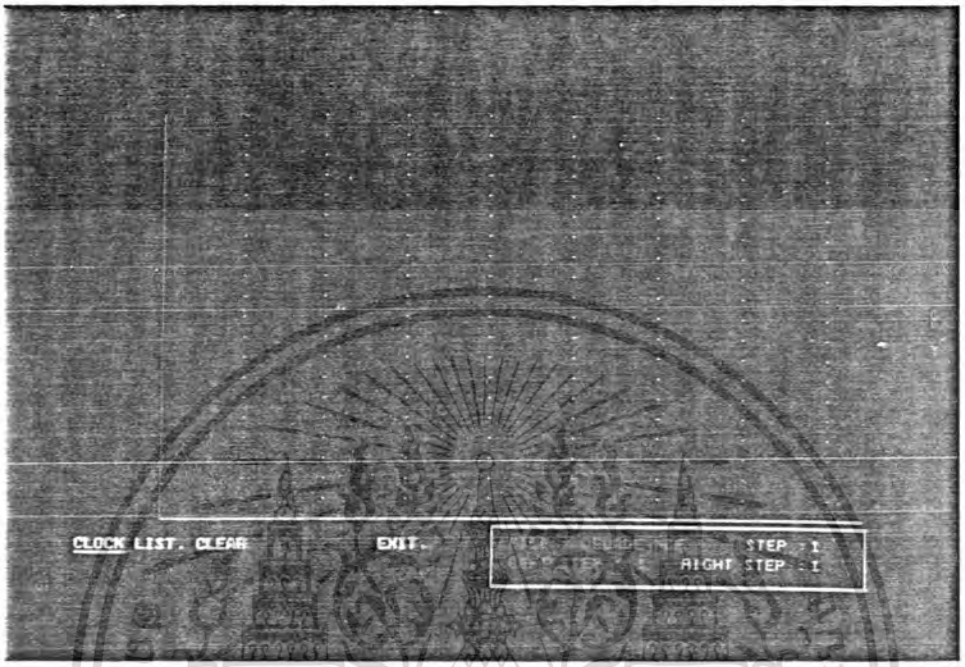
รูปที่ 7-14 รูปผังภาพวงจร DECADE COUNTER ที่สร้างจาก ORCAD



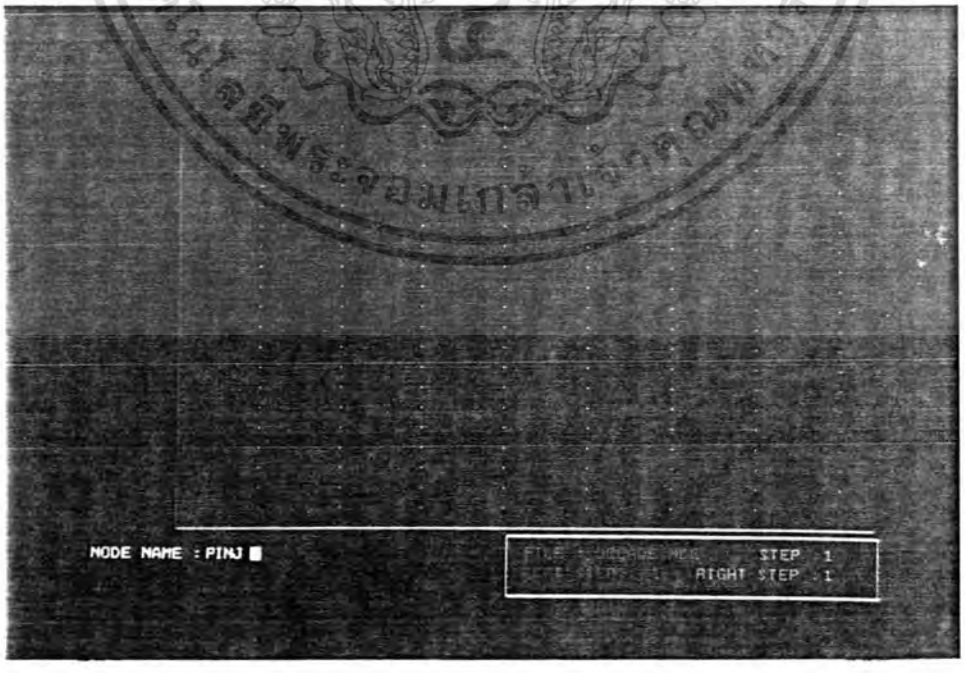
รูปที่ 7-15 แสดงเมนูหลักที่มีของของโปรแกรม LOGCAD SIMULATOR



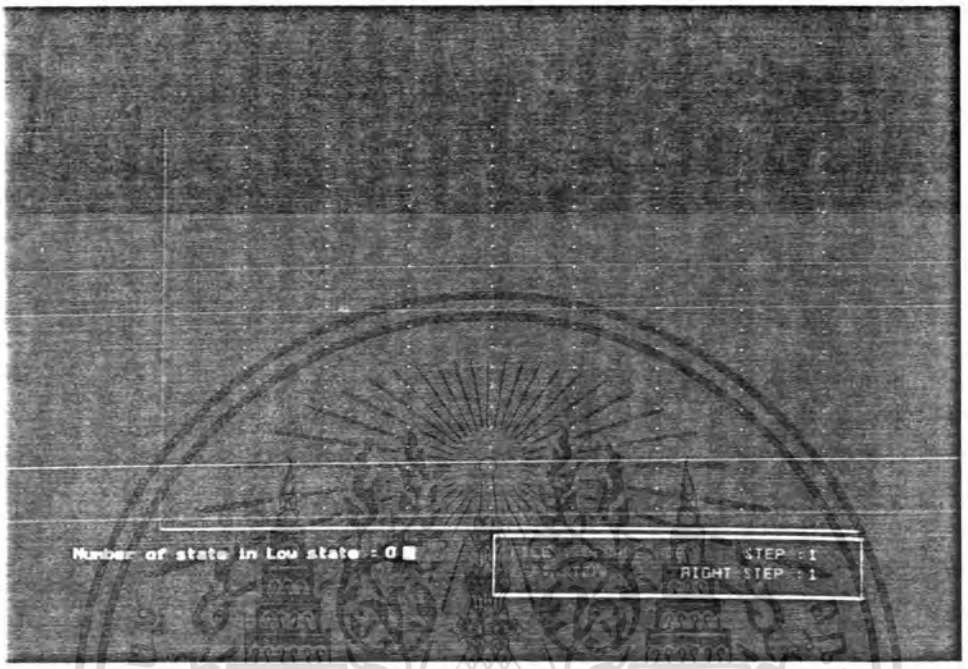
รูปที่ 7-16 แสดงชื่อไฟล์ข้อมูลโหนดของวงจรต่าง ๆ



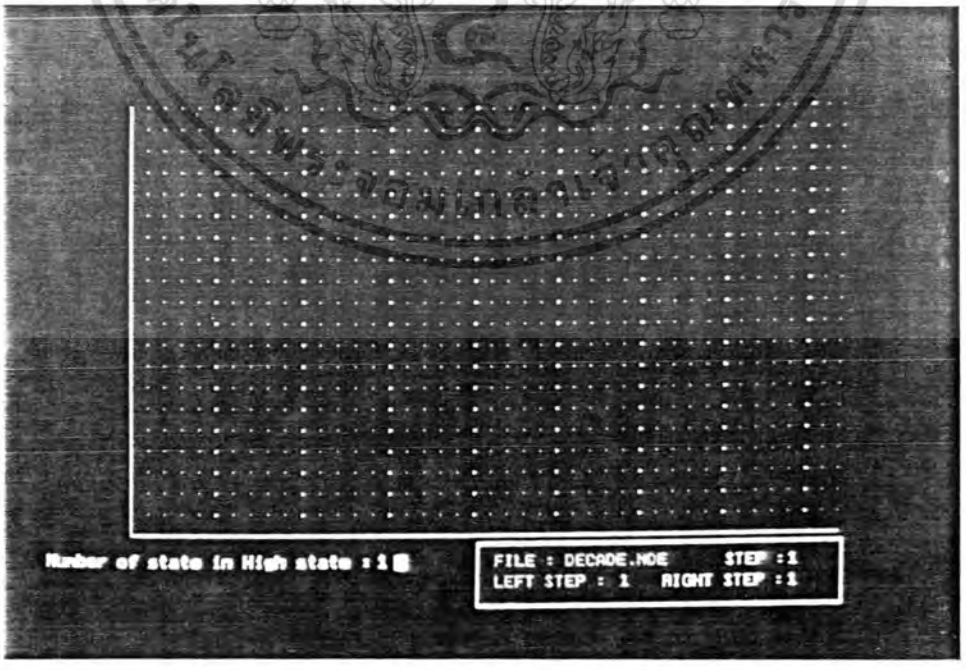
รูปที่ 7-17 แสดงเมนูคำสั่งย่อยจากเมนูคำสั่งหลัก DATA



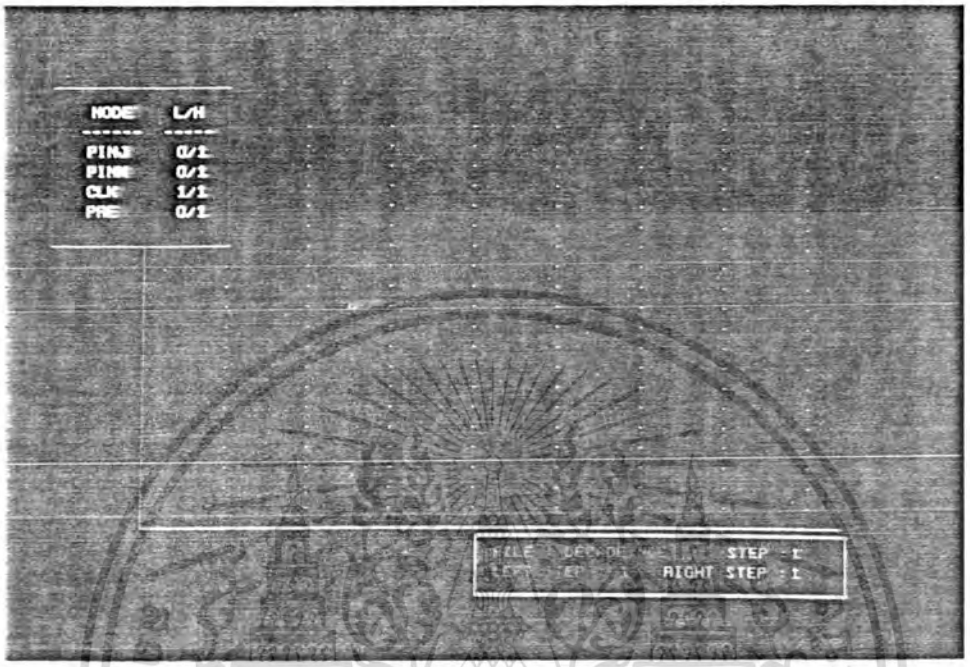
รูปที่ 7-18 แสดงการกำหนดโหนด PINJ ให้เป็นข้อมูลเข้าของวงจร



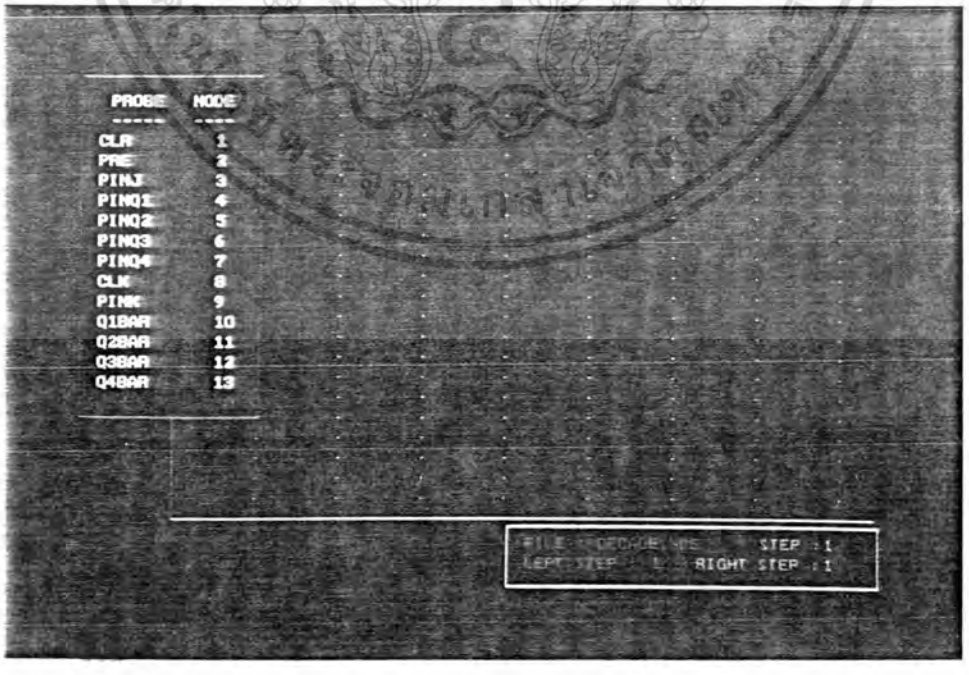
รูปที่ 7-19 การกำหนดจำนวน LOW STATE ของโหนด PINJ



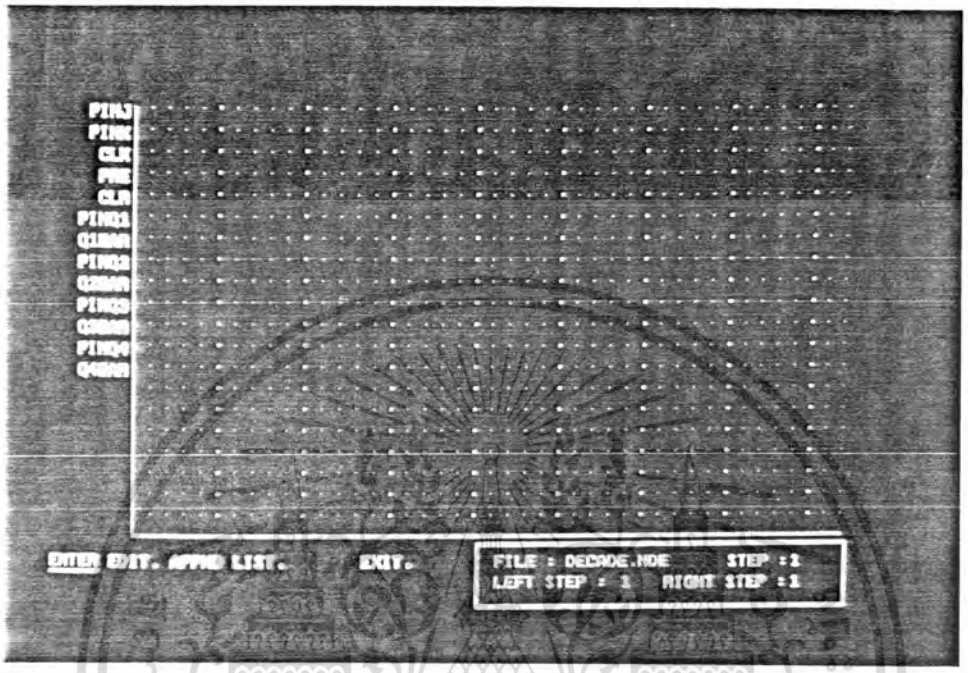
รูปที่ 7-20 การกำหนดจำนวน HIGH STATE ของโหนด PINJ



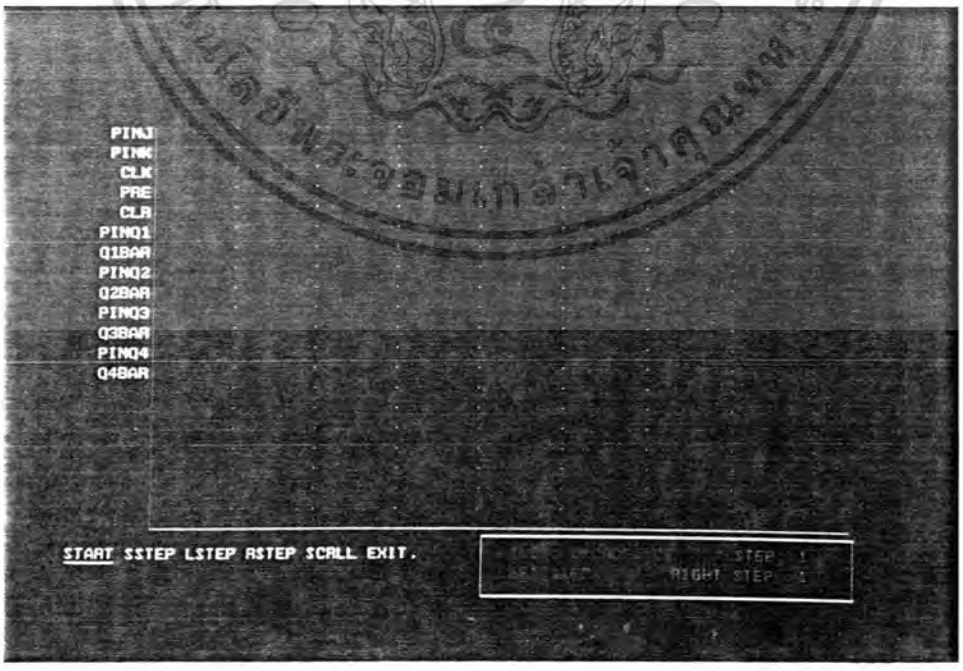
รูปที่ 7-21 แสดงการกำหนดข้อมูลเข้าทั้งหมดของวงจร DECADE COUNTER



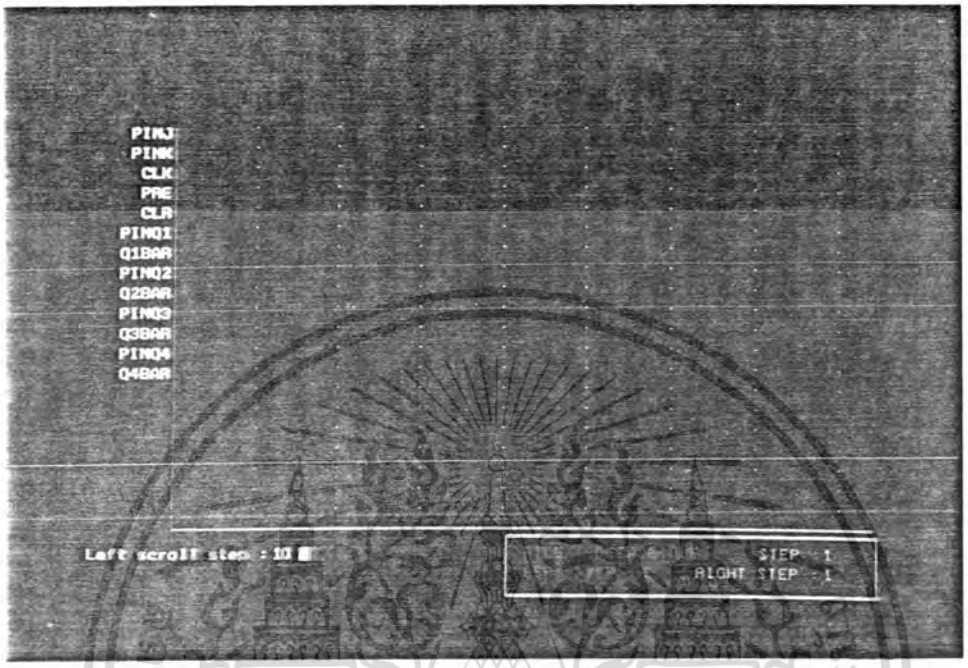
รูปที่ 7-22 แสดงชื่อโหนดต่าง ๆ ของวงจร DECADE COUNTER



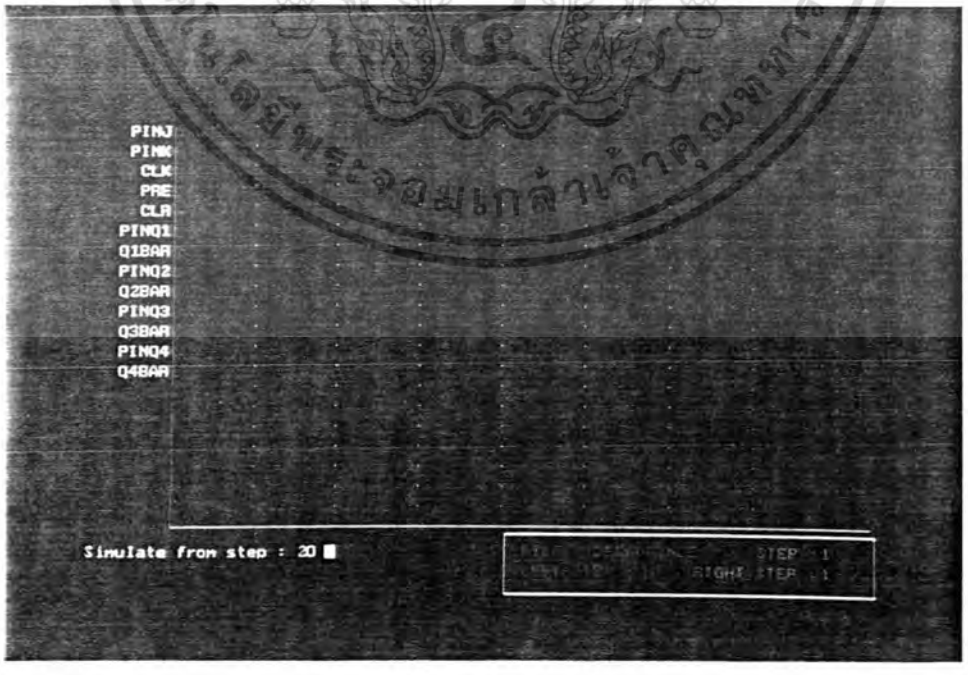
รูปที่ 7-23 แสดงการทำงานของคำสั่ง DISP./PROBE/ENTER



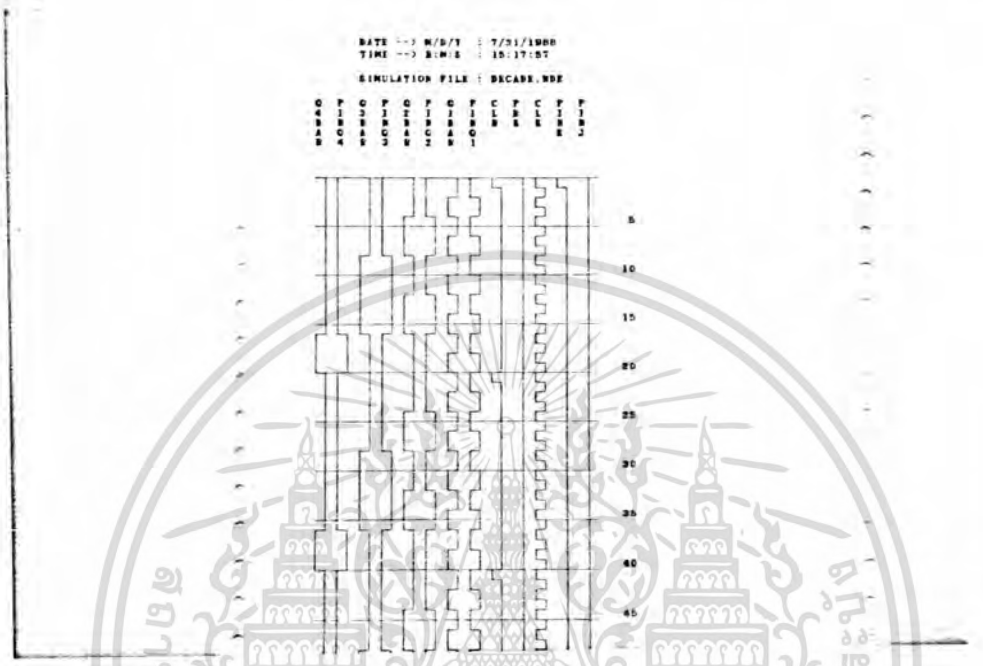
รูปที่ 7-24 แสดงการเลือกคำสั่งการประมวลผล SIM/START



รูปที่ 7-25 กำหนดจำนวนสเต็ปเพื่อใช้ในการเลื่อนภาพไปทางซ้าย



รูปที่ 7-26 กำหนดสเต็ปเริ่มต้นเพื่อแสดงภาพแผนภูมิเวลา



รูปที่ 7-27 แสดงการพิมพ์ภาพแผนภูมิเวลาออกทางเครื่องพิมพ์

บทที่ 8

บทสรุป

ในการออกแบบวงจรตรรกนั้น การทดสอบการทำงานของวงจรที่ออกแบบโดยใช้วิธีนำอุปกรณ์ต่างๆมาเชื่อมต่อกันเป็นวิธีที่ยุ่งยาก และ สิ้นเปลืองค่าใช้จ่ายสูง นอกจากนั้นยังทำให้เสียเวลามาก การประยุกต์ใช้ไมโครคอมพิวเตอร์ในการจำลองการทำงานของวงจรตรรกนั้นนับว่ามีประโยชน์และมีประสิทธิภาพมาก เนื่องจากผู้ที่ออกแบบสามารถตรวจสอบการทำงานของวงจรได้สะดวก และ ไม่ต้องใช้อุปกรณ์หรือเครื่องมือวัดใด ๆ เมื่อพบว่าวงจรทำงานไม่ถูกต้อง หรือ ต้องการเปลี่ยนแปลงแก้ไขการทำงานของวงจร ก็สามารถกระทำได้ทันที โปรแกรมที่ใช้ในการจำลองการทำงานของวงจรตรรกดังกล่าวนั้นในปัจจุบันมีอยู่หลายโปรแกรม แต่ละโปรแกรมก็มีข้อดีและข้อเสียแตกต่างกันออกไปแต่ส่วนใหญ่แล้วมักจะมีฟังก์ชั่นของอุปกรณ์จำกัดและไม่สามารถเปลี่ยนแปลงแก้ไขได้ สำหรับโปรแกรม LOGCAD ที่ได้พัฒนาขึ้นนี้ได้ออกแบบให้แก้ข้อเสียดังกล่าวไว้แล้ว โดยผู้ใช้จะเป็นผู้กำหนดฟังก์ชั่นการทำงานต่างๆได้เอง ซึ่งอาจจะไม่สะดวกและเสียเวลาในการศึกษาการใช้งานในระยะเริ่มต้นบ้าง แต่วิธีนี้เป็นวิธีที่มีประสิทธิภาพมาก เนื่องจากไม่จำกัดฟังก์ชั่นและการใช้งาน

ในวิทยานิพนธ์ฉบับนี้ได้ออกแบบโปรแกรมดังกล่าวไว้โดยแบ่งออกเป็นส่วนใหญ่ ๆ 2 ส่วนคือ โปรแกรมสำหรับการสร้างและแก้ไขผังภาพของวงจรซึ่งใช้สำหรับสร้างหรือวาดวงจรที่ต้องการ และ โปรแกรมการจำลองการทำงานของวงจรตรรก โปรแกรมในส่วนแรกนั้นได้ออกแบบและพัฒนาให้ใช้งานได้ง่าย โดยได้พัฒนาคำสั่งและฟังก์ชั่นต่างๆไว้เพื่อให้ใช้งานได้สะดวก แต่จากการทดลองนำไปใช้งานกับจอแสดงผลแบบ CGA พบว่าความละเอียดในการแสดงผลไม่สูงพอทำให้ไม่สะดวกในการใช้งาน ดังนั้นในวิทยานิพนธ์ฉบับนี้จึงได้ตัดการแสดงผลในโหมด CGA ออก สำหรับโปรแกรมการจำลองการทำงานของวงจรตรรกนั้นก็เช่นเดียวกัน ซึ่งในการใช้งานจริงแล้วความละเอียดของส่วนแสดงผลเป็นส่วนสำคัญมากในการทำงาน ความละเอียดของส่วนแสดงผลสูงจะทำให้ผู้ใช้ปฏิบัติงานได้สะดวก เนื่องจากพื้นที่ในการแสดงผลจะกว้าง ส่วนการทำงานอื่นๆของโปรแกรมนั้นได้ออกแบบให้ผู้ใช้สามารถตรวจสอบการทำงานของวงจรได้สะดวก โดยสามารถเลื่อนภาพไปทางซ้ายและขวาได้ทันที นอกจากนั้นยังสามารถพิมพ์ผลลัพธ์ที่เป็นแผนภูมิเวลา (TIMING DIAGRAM) ของวงจรดังกล่าวออกทางเครื่องพิมพ์ชนิดดอทแมททริกซ์ (DOT MATRIX PRINTER) โดยมีการจัดให้แผนภูมิเวลาดังกล่าวอยู่ในแนวตั้ง เพื่อให้แผนภูมิเวลาที่มี

คาบเวลาจำลองการทำงานที่ยาวมาก ๆ ได้โดยต่อเนื่อง

ในการพัฒนาในช่วงแรกนั้นโปรแกรม LOGCAD ได้ถูกพัฒนาบน TURBO PASCAL VERSION 3.1 ซึ่งมีขีดจำกัดทางด้านตัวแปลภาษา และ การแสดงผลทางด้านกราฟฟิค

ขีดความสามารถทางด้านตัวแปลภาษานั้นในเวอร์ชั่น 3.1 จะไม่สามารถแปลภาษาแบบ COMMAND LINE ซึ่งนับได้ว่าเป็นสิ่งที่จำเป็น ในกรณีที่ต้องการจะให้ตัวแปลภาษาของ TURBO PASCAL ช่วยในการแปลฟังก์ชั่นต่าง ๆ ของวงจรตรรกที่ผู้ใช้กำหนดขึ้น

ขีดความสามารถทางด้านการแสดงผลแบบกราฟฟิค ไม่สามารถแสดงผลในโหมดอื่น ๆ ได้นอกจาก CGA ดังนั้นในการพัฒนาระยะแรกจึงได้ทำการออกแบบโปรแกรมย่อยสำหรับการแสดงผลในโหมด EGA ขึ้นเอง ซึ่งก็ได้รับผลเป็นที่น่าพอใจแต่ก็ประสบปัญหาทางด้านความเร็วในการแสดงผล ต่อมา TURBO PASCAL ได้พัฒนาเป็นเวอร์ชั่น 4.0 ที่มีขีดความสามารถทางด้านกราฟฟิคสูงขึ้น จึงได้แก้ไขและพัฒนาโปรแกรม LOGCAD เพื่อให้สามารถใช้ตัวแปลภาษาเวอร์ชั่นใหม่นี้ได้ จากการพัฒนาโปรแกรมมาใช้ตัวแปลภาษาเวอร์ชั่นใหม่นี้ก็พบว่า โปรแกรมที่มีหลักการทำงานลักษณะเดิมนั้นสามารถทำงานได้ด้วยความเร็วสูงขึ้นและมีประสิทธิภาพสูงขึ้นตามไปด้วย ดังนั้นการพัฒนาโปรแกรม LOGCAD จึงทำได้รวดเร็วและสะดวก แต่อย่างไรก็ดีโปรแกรม LOGCAD ในบางส่วนยังมีข้อเสียอยู่บ้าง เช่นไม่สามารถจำลองการทำงานฟังก์ชั่นที่มีการหน่วงเวลาเข้ามาเกี่ยวข้อง ทั้งนี้เพราะการสร้างฟังก์ชั่นของโปรแกรม LOGCAD นั้น อาศัยฟังก์ชั่นที่กำหนดได้จากโปรแกรม TURBO PASCAL เพียงอย่างเดียว ดังนั้นจะต้องมีการสร้างฟังก์ชั่นเพิ่มเติมขึ้นมาสำหรับฟังก์ชั่นประเภทนี้โดยเฉพาะ เพื่อทำการตรวจสอบการเปลี่ยนแปลงสภาวะในส่วนที่เป็นขอบขาขึ้น และ ขอบขาลง ของโหนดที่กำลังพิจารณาอยู่ สิ่งนี้จะเป็นแนวทางสำหรับผู้ที่จะทำการศึกษาหรือพัฒนาโปรแกรมให้มีประสิทธิภาพมาขึ้นกว่าเดิมต่อไป

กิตติกรรมประกาศ

(ACKNOWLEDGMENT)

ในการทำวิทยานิพนธ์ครั้งนี้ ข้าพเจ้าขอกราบขอบพระคุณ รศ.ดร. สิทธิชัย โภไคยอุดม และ รศ. กิตติ ตีรเศรษฐ เป็นอย่างสูงที่ได้กรุณาให้คำปรึกษา และ แนะนำ แนวทาง วิธีการแก้ไข ตลอดจนอำนวยความสะดวกในเรื่องอุปกรณ์การทำวิทยานิพนธ์ และขอขอบคุณ คุณวรวิทย์ เตชะพันธ์ อาจารย์ภากร หุตะสังกาศ และอาจารย์ท่านอื่น ๆ ในภาควิชาวิศวกรรมการวัดคุมฯ ที่มีส่วนช่วยให้วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี

หนังสืออ้างอิง
(REFERENCES)

1. สุพรรณ กุลพาณิชย์, สิทธิชัย โภไคยอุดม, กิตติ ตรีเสรษฐ " การใช้ไมโครคอมพิวเตอร์ช่วยในการออกแบบวงจรตรรก " การประชุมทางวิชาการวิศวกรรมไฟฟ้า สถาบันอุดมศึกษาแห่งประเทศไทย ครั้งที่ 9 มหาวิทยาลัยขอนแก่น หน้าที่ 2-10-1 ถึง 2-10-9, เล่มที่ 1, 3-4 ธันวาคม 2529

2. สุพรรณ กุลพาณิชย์ กิตติ ตรีเสรษฐ " การใช้ไมโครคอมพิวเตอร์ช่วยในการออกแบบวงจรตรรก 2 " การประชุมทางวิชาการวิศวกรรมไฟฟ้า สถาบันอุดมศึกษาแห่งประเทศไทย ครั้งที่ 10 จุฬาลงกรณ์มหาวิทยาลัย,หน้าที่ 1-140 ถึง 1-151, เล่มที่ 1, 24-25 พฤศจิกายน 2530

3. Lee, C.Y. " An Algorithm for Path Connection and its Application ", IEE Trans., EC 10, p. 346-365, 1961.

4. " Disk Operating System ", IBM Personal Computer, Computer Language Series, Microsoft Corp

5. " AUTOCAD (TM) ", User Reference, Version 2.0 , 1984.

6. " Turbo pascal ",Version 3.1, Reference Manual, BORLAND INTERNATIONAL Inc.,1985.

7. " Turbo pascal ",Version 4.0, Owner's Handbook, BORLAND INTERNATIONAL Inc.,1987

8. P-CAD User's Manuals 1986 by PERSONAL CAD SYSTEM, Inc.

9. " MICRO - LOGIC " Logic Design Simulation by Unipress Software, Inc.

ภาคผนวก 1

คำสงวนที่ห้ามใช้กำหนดเป็นชื่อตัวแปรของโหนดในการสร้างผังภาพวงจรและแบบจำลอง

ในการเขียนแบบจำลอง หรือ การกำหนดโหนดในส่วนของการสร้างผังภาพวงจรจะต้องมีการกำหนดชื่อของตัวแปร หรือชื่อโหนดต่าง ๆ เพื่อป้องกันมิให้เกิดการซ้ำกันของชื่อตัวแปร จึงขอสงวนชื่อตัวแปรเหล่านี้ไว้

- คำสงวนและคำมาตรฐานของ TURBO PASCAL
- ตัวแปรร่วมที่กำหนดในโปรแกรม LOGCAD SIMULATOR

1. คำสงวนและคำมาตรฐานของ TURBO PASCAL

RESERVED WORDS

* ABSOLUTE	* EXTERNAL	NIL	* SHR
AND	FILE	NOT	* STRING
ARRAY	FOR	OF	THEN
BEGIN	FORWARD	OR	TO
CASE	FUNCTION	PACKED	TYPE
CONST	GOTO	PROCEDURE	UNTIL
DIV	IF	PROGRAM	VAR
DO	IN	RECORD	WHILE
DOWNTO	* INLINE	REPEAT	WITH
ELSE	LABEL	SET	* XOR
END	MOD	* SHL	

The asterisks indicate reserved words not defined in standard Pascal

คำสงวนในภาษาของ TURBO PASCAL

TURBO Pascal defines a number standard indentifiers of predefined types, constants, variables, procedures, and functions. Any of these identifiers may be redefined but it will mean the loss of the facility offered by that particular identifier and may lead to confusion. The following standard identifiers are therefore best left to their special purposes:

ArcTan	False	Odd	Usr
Assign	FilePos	Ord	UsrInPtr
Aux	FileSize	Output	UsrOutPtr
AuxInPtr	FillChar		
AuxOutPtr	Flush	Pi	Val
	Frac	Port	
BlockRead		Pos	Write
BlockWrite	GetMem	Pred	Writeln
Boolean	GotoXY	Ptr	
BufLen			
Byte	HeapPtr	Ramdom	
	Hi	Randomize	
Chain		Read	
Char	IOresult	ReadIn	
Chr	Input	Real	
Close	InsLine	Release	
ClrEOL	Insert	Rename	
ClrScr	Int	Reset	
Con	Integer	Rewrite	
ConInPtr		Round	
ConOutPtr	Kbd		
Concat	KeyPressed	Seek	
ConstPtr	Length	Sin	
Copy	Ln	SizeOf	
Cos	Lo	Sqr	
CrtExit	LowVideo	Sqrt	
CrtInit	Lst	Str	
	LstOutPtr	Succ	
DelLine		Swap	
Delay	Mark		
Delete	MaxInt	Text	
	Mem	Trm	
EOF	MemAval 1	True	
EOLN	Move	Trunc	
Erase			
Execute	New		
Exp	NormVideo	UpCase	

คำมาตรฐานในภาษาของ TURBO PASCAL

2. ตัวแปรร่วมที่กำหนดในโปรแกรม LOGCAD SIMULATOR แล้ว

Ch	ClkData	ClkDataNumber	DataExist
DataFile	Err	_First	FileName
FirstClkData	High	IntNo	Left
LeftStep	Low	MaxClkData	MaxDisplayProbe
MaxGridOnScreen	MaxNode	MaxProbe	MaxStep
MaxStepOnScreen	Network	NewDta	Null

No	Num	Number	Ok
OldColor	On	Off	Pass
Probe	ProbeNumber	Reg	Right
RightStep	Simulate	SimStep	StartStep
Step			

คำเหล่านี้ทั้งหมดห้ามนำมาใช้ในการตั้งชื่อให้กับตัวแปรสำหรับเขียนเป็นแบบจำลอง และตัวแปรที่เป็นชื่อโหนดของผังภาพวงจร

ภาคผนวก 2

ข้อมูลอุปกรณ์ไลบรารี และ ข้อมูลผังภาพวงจรของโปรแกรม LOGCAD EDITOR

1. ข้อมูลอุปกรณ์ไลบรารี JKFFNL.LIB ที่เลือกมาเฉพาะบางส่วน

```
LIBRARY'S  NAME   (JKFFNL)
REFERENCE (X,Y)
100       80
XMIN     YMIN
100       60
XMAX     YMAX
160      140
NUMBER OF COMMAND
   23
    1
 110       70
 110      130
    2
    1
 110      130
 150      130
    2
    1
 150      130
 150       70
    2
    1
 150       70
 110       70
    2
    2
```

```
130      68
  2
  2
  2
108     100
  2
  2
  2
130     132
  2
  2
  1
130      66
130      60
  2
  4
112      75
  J
  2
  4
112      95
CLK
  2
  4
112     115
  K
  2
  1
130     140
130     134
```

2
4
130 75
P
2
4
130 80
R
2
4
130 115
C
2
4
130 120
L
2
4
145 75
Q
2
4
140 115
/Q
2

2. ข้อมูลผังภาพวงจร DECADE.SCH ที่เลือกมาเฉพาะบางส่วน

```
SCHEMATIC'S NAME  (DECADE)
NUMBER OF DATA
  90
  13  (INPUT)
  50 100
  50  90  90 110
  INPUT
   2
  13  (INPUT)
  50 160
  50 150  90 170
 INPUT
   2
  13  (INPUT)
  50 180
  50 170  90 190
  INPUT
   2
  13  (INPUT)
  50 200
  50 190  90 210
  INPUT
   2
  15  (JKFFNL)
 130 160
 130 140 190 220
 JKFFNL
   2
```

```
 15  (JKFFNL)
230 160
230 140 290 220
JKFFNL
  2
 15  (JKFFNL)
330 160
330 140 390 220
JKFFNL
  2
 15  (JKFFNL)
430 160
430 140 490 220
JKFFNL
  2
 16  (NAND2)
520 240
520 230 570 270
NAND2
  2
  1  (LINE)
110 120
110 120 110 160
LINE
  3
  1  (LINE)
110 200
110 200 110 240
LINE
  2  (CONNECT)
```

```
309 239
309 239 311 241
CONNECT
  2
  2  (CONNECT)
259 279
259 279 261 281
CONNECT
  2
  2  (CONNECT)
359 279
359 279 361 281
CONNECT
  2
  2  (CONNECT)
459 279
459 279 461 281
CONNECT
  2
  4  (NODE)
 60  98
 60  98  60  98
PRE
  1

  4  (NODE)
 60 158
 60 158  60 158
PINJ
  1
```

```
  4  (NODE)
 60 178
 60 178  60 178
CLK
  1
  4  (NODE)
 60 198
 60 198  60 198
PINK
  1
  4  (NODE)
190 150
190 150 190 150
PINQ1
  1
  4  (NODE)
190 190
190 190 190 190
Q1BAR
  1
  4  (NODE)
290 150
290 150 290 150
PINQ2
  1
  4  (NODE)
290 190
290 190 290 190
Q2BAR
 1
```

```
  4  (NODE)
390 150
390 150 390 150
PINQ3
  1
  4  (NODE)
390 190
390 190 390 190
Q3BAR
  1
  4  (NODE)
530 150
530 150 530 150
PINQ4
  1
  4  (NODE)
530 190
530 190 530 190
Q4BAR
  1
  4  (NODE)
570 240
570 240 570 240
CLR
  1
 2  (CONNECT)
189 159
189 159 191 161
CONNECT
 2
```

```
  2  (CONNECT)
 189 199
 189 199 191 201
 CONNECT
   2
   2  (CONNECT)
 289 199
 289 199 291 201
 CONNECT
   2
   2  (CONNECT)
 289 159
 289 159 291 161
CONNECT
   2
   2  (CONNECT)
 389 159
 389 159 391 161
 CONNECT
   2
   2  (CONNECT)
 389 199
 389 199 391 201
 CONNECT
   2
   2  (CONNECT)
 569 249
 569 249 571 251
CONNECT
   2
```

ภาคผนวก 3

```
Text File List Processing Program V4.00C Page : 1
File : B:FILEVAR1.PAS
Current Date : Sunday September 4, 1988
Current Time : 11:20 AM

Program : Logcad Editor Variable
Programmer : Suphan Gulpanich.

{
              +-----------------------------------+
              |                                   |
              | Unit  : Simulation I              |
              | Programmer : Suphun Gulpanich.    |
              | Version    : 1.00A                |
              | Last Updated :  24 June 1988      |
              |                                   |
              +-----------------------------------+
}
Unit SimUnit1;

interface

uses Crt,Graph;

const
  Hidden : FillPatternType = ($00,$00,$00,$00,$00,$00,$00,$00);
  XCharOffset = 10; { x,y lines offset from upper left corner }
  YCharOffset = 2;

var
  GraphDriver : integer;  { The Graphics device driver }
  GraphMode   : integer;  { The Graphics mode value }
  MaxX, MaxY  : word;     { The maximum resolution of the screen }
  MaxColor    : word;     { The maximum color value available }
  ViewPort : ViewPortType;
  GraphTextWidth : byte;
  GraphTextHeight : byte;
  AxisX0,AxisY0,AxisX1,AxisY1 : word;
  WindowX0 : word;
  WindowY0 : word;
  WindowX1 : word;
  WindowY1 : word;
  TextWindow : pointer;
  MaxStepOnScreen : Word;

Procedure SetCrtColor (Color : byte);

procedure WriteTextXY (x,y : byte; Str : string);

procedure SaveScreen;

procedure LoadScreen;
```

```
procedure OpenWindow (X0,Y0,X1,Y1 : byte);

procedure CloseWindow;

procedure ClearWindow;

implementation

var
  ErrorCode   : integer;  { Reports any graphics errors }
  Buffer : pointer;
  Size : word;
  Number : word;

Const
  ScrollStatusCol = 20;

Procedure SetCrtColor (Color : byte);
Begin
  Case GraphDriver of
    EGA : SetColor(Color);
    CGA : SetColor(Color);
    HercMono : Begin
                 if Color = Black then
                   Color := Color
                 else
                   Color := White;
                 SetColor (Color);
               end;
  end; { Case }
end; { SetCrtColor }

procedure InitialGraphics;
begin { InitialGraphics }
{ Initialize graphics and report any errors that may occur }
{ when using Crt and graphics, turn off Crt's memory-mapped writes }
  DetectGraph (GraphDriver,GraphMode);
  case GraphDriver of
     EGA : begin
             DirectVideo := True;
             GraphMode := EGAHi;
             MaxStepOnScreen := 42;
           end;
     CGA : begin
             DirectVideo := False;
             GraphMode := CGAHi;
             MaxStepOnScreen := 29;
           end;
HercMono : begin
             DirectVideo := True;
             GraphMode := HercMonoHi;
             MaxStepOnScreen := 42;
           end;
  end { case };
```

```
  InitGraph(GraphDriver,GraphMode, '');  { activate graphics }
  SetGraphMode(GraphMode);
  ErrorCode := GraphResult;              { error? }
  if ErrorCode <> grOk then
  begin
    Writeln('Graphics error: ', GraphErrorMsg(ErrorCode));
    Halt(1);
  end;
  MaxColor := GetMaxColor;  { Get the maximum allowable drawing color }
  MaxX := GetMaxX;          { Get screen resolution values }
  MaxY := GetMaxY;
  SetCrtColor(MaxColor);
  SetLineStyle(SolidLn, 0, NormWidth);
  SetViewPort(0,0,MaxX,MaxY,ClipOn);
  GetViewSettings(ViewPort);
  case GraphDriver of
    EGA : begin
            GraphTextHeight := MaxY div 25;
          end;
    CGA : begin
            GraphTextHeight := TextHeight ('M');
          end;
    HercMono : begin
                 GraphTextHeight := MaxY div 25;
               end;
  end { case };
  GraphTextWidth := MaxX div 80;
end { InitialGraphics };

procedure WriteTextXY (x,y : byte; Str : string);
begin { WriteTextXY }
  Dec (X);
  Dec (Y);
  OutTextXY(X * GraphTextWidth,Y * GraphTextHeight,Str);
end { WriteTextXY };

procedure SaveScreen;
begin { SaveScreen }
  GetImage(AxisX0,AxisY0,AxisX1,AxisY1,Buffer^);
end { SaveScreen };

procedure LoadScreen;
begin { LoadScreen }
  PutImage(AxisX0,AxisY0,Buffer^,NormalPut);
end { LoadScreen };

procedure OpenWindow (X0,Y0,X1,Y1 : byte);
begin { OpenWindow }
  Dec(X0);
  Dec(Y0);
  WindowX0 := X0 * GraphTextWidth;
  WindowY0 := Y0 * GraphTextHeight;
  WindowX1 := X1 * GraphTextWidth;
  WindowY1 := Y1 * GraphTextHeight;
```

```
  GetImage(WindowX0,WindowY0,WindowX1,WindowY1,TextWindow^);
  SetFillPattern(Hidden,Blue);
  Bar(WindowX0,WindowY0,WindowX1,WindowY1);
  SetCrtColor(LightRed);
  Rectangle(WindowX0,WindowY0,WindowX1,WindowY1);
end { OpenWindow };

procedure CloseWindow;
begin { CloseWindow }
  PutImage(WindowX0,WindowY0,TextWindow^,NormalPut);
end { CloseWindow };

procedure ClearWindow;
begin { ClearWindow }
  SetFillPattern(Hidden,Blue);
  FloodFill(WindowX0+1,WindowY0+1,LightRed);
end { ClearWindow };

begin { unit Simunit1 }
  InitialGraphics;
  if GraphDriver = CGA then
    GraphMode := CGAC2;
  SetGraphMode(GraphMode);
  GetMem(Buffer,65535);
  GetMem(TextWindow,65535);
 end { unit Simunit1 }.
```

File : B:INITIAL1.PAS
Current Date : Sunday September 4, 1988
Current Time : 11:46 AM

Program : Logcad Editor Initialize
Programmer : Suphan Gulpanich.

```
{
        ***********************************************
        *                                             *
        *      PROGRAM UNIT : INITIAL  GRAPHIC        *
        *      PROGRAMER    : SUPHAN  GULPANICH       *
        *      VERSION      : 1.000A                  *
        *      LAST UPDATED : 15 JULY 1988            *
        *                                             *
        ***********************************************
}
UNIT INITIAL;

INTERFACE

USES CRT,GRAPH,DECLARE1;

VAR
   YOff_text      :    Byte       ;
   StepX,StepY    :    Byte       ;
   GraphTextWidth :    Byte       ;
   GraphTextHeight:    Byte       ;
   WindowX0       :    Word       ;
   WindowY0       :    Word       ;
   WindowX1       :    Word       ;
   WindowY1       :    Word       ;
   TEXT_X10,TEXT_MaxY:    Word    ;
   CX,CY          :    Word       ;{ CurSor (X,Y) }
   MaxX, MaxY     :    Word       ;{ The maximum resolution of the screen }
   GraphDriver    :    Integer    ;{ The Graphics device driver  }
   GraphMode      :    Integer    ;{ The Graphics mode value     }
   ErrorCode      :    Integer    ;{ Reports any graphics errors }
   TextWindow     :    Pointer    ;
   Text_BK        :    Pointer    ;
   StringXY       :    String     ;
   StringName     :    String     ;
   ViewPort       :    ViewPortType;
   Color          :    Array[0..4] Of Byte ;

FUNCTION SelectNo(SelStr:PassStr;No:Byte):Byte;
FUNCTION IntToStr(Value:Integer):String;
FUNCTION Adjust_X(X:Integer):Integer ;
FUNCTION Adjust_Y(Y:Integer):Integer ;
PROCEDURE Text_XY(X,Y:Integer;PassStr:STRING;COLOR:Byte);
PROCEDURE _Swap(VAR X0,X1:Integer);
PROCEDURE OpenWindow (X0,Y0,X1,Y1 : Word);
```

```
PROCEDURE CloseWindow ;
PROCEDURE ClearScreen ;
PROCEDURE SetScreen_1 ;
PROCEDURE SetScreen_2 ;
PROCEDURE DrawLine(X0,Y0,X1,Y1:Integer;Color:Byte);
PROCEDURE DrawCircle(X0,Y0,Radius:Integer;Color:Byte)  ;
PROCEDURE DrawArc(X0,Y0,Angle1,Angle2,Radius:Integer;Color:Byte) ;
PROCEDURE DrawText(X0,Y0:Integer; PassStr:String;Color:Byte);
PROCEDURE DrawConnect(X0,Y0,X1,Y1:Integer;Color:Byte);
PROCEDURE DrawLib(X,Y:Integer;PassLib:Pointer_Func;Colors:Byte);
PROCEDURE REDRAW_FUNCTION_NAME ;
PROCEDURE Plot_Pixel ;
PROCEDURE _Grid  ;
PROCEDURE ReDraw ;
PROCEDURE Left   ;
PROCEDURE Right  ;
PROCEDURE Top    ;
PROCEDURE Bottom ;
PROCEDURE GetKB  ;
PROCEDURE Jump   ;
PROCEDURE Tag    ;
PROCEDURE _Zoom  ;
PROCEDURE Find_Name(VAR NAME:STRING);
PROCEDURE ReadText(CommandStr:String;Var PassStr:String;Count:Byte);
PROCEDURE MenuFile  ;

IMPLEMENTATION

PROCEDURE InitialGraphics;
BEGIN { InitialGraphics }
{ Initialize graphics and report any errors that may occur }
{ when using Crt and graphics, turn off Crt's memory-mapped writes }
  GraphDriver := Detect;          { use autodetection }
  DetectGraph(GraphDriver,GraphMode);
  Case GraphDriver Of
    EGA : BEGIN
            GraphMode  :=  EGAHi   ;
            Color[0]   :=  Black   ;
            Color[1]   :=  Cyan    ;
            Color[2]   :=  Magenta ;
            Color[3]   :=  White   ;
            YOff_Text  :=  2       ;
            StepX      :=  20      ; StepY  := 16 ;
          END;
    HercMono : BEGIN
            GraphMode  := HercMonoHi ;
            Color[0]   :=  Black   ;
            Color[1]   :=  1       ; { WHITE = 1 }
            Color[2]   :=  1       ;
            Color[3]   :=  1       ;
            YOff_Text  :=  0       ;
            StepX      :=  21      ; StepY  := 16 ;
          END;
    CGA : BEGIN
```

```
            GraphMode := CGAC1;
            Color[0]   :=  0       ;  { Black }
            Color[1]   :=  1       ;  { Cyan  }
            Color[2]   :=  2       ;  { Magenta }
            Color[3]   :=  3       ;  { White }
            YOff_Text  :=  0       ;
            StepX      := 10       ;  StepY :=  8 ;
          END;
  END { case };
  InitGraph(GraphDriver, GraphMode, '');  { activate graphics }
  ErrorCode := GraphResult;               { error? }
  if ErrorCode <> GrOk then
  BEGIN
    Writeln('Graphics error: ', GraphErrorMsg(ErrorCode));
    Halt(1);
  END;
  MaxX := GetMaxX;          { Get screen resolution values }
  MaxY := GetMaxY;
  SetLineStyle(SolidLn,HorizDir,NormWidth);
  SetViewPort(0,0,MaxX,MaxY,ClipOn);
  GetViewSettings(ViewPort);
  GraphTextHeight := TextHeight('M');
  GraphTextWidth  := TextWidth('M');
  SetTextStyle(SmallFont,HorizDir,2);
  SetTextStyle(DefaultFont,HorizDir,1);
  GetMem(TextWindow,65535);
  GetMem(Text_BK,640);
  TEXT_X10  := 10*GraphTextWidth ;
  TEXT_MaxY :=  MaxY-GraphTextHeight;
  SetFillStyle(EmptyFill,GetBkColor);
END { InitialGraphics };

FUNCTION IntToStr(Value : Integer):String;
{ Converts an integer to a string for use with OutText, OutTextXY }
VAR
  S : string;
BEGIN
  Str(Value,S);
  IntToStr := S;
END; { IntToStr }

PROCEDURE TEXT_XY( X,Y : Integer ; PassStr : STRING ; Color : Byte);
BEGIN
  SETCOLOR(Color);
  OutTextXY(X,Y,passStr);
END;

PROCEDURE OpenWindow (X0,Y0,X1,Y1 : Word);
BEGIN { OpenWindow  }
  WindowX0 := X0 - 3;
  WindowY0 := Y0 - 3;
  WindowX1 := X1 + 3;
  WindowY1 := Y1 + 3;
  GetImage(WindowX0,WindowY0,WindowX1,WindowY1,TextWindow^);
```

```
  Bar(WindowX0,WindowY0,WindowX1,WindowY1);
  SetColor(Color[3]);
  Rectangle(WindowX0,WindowY0,WindowX1,WindowY1);
  Rectangle(WindowX0+2,WindowY0+2,WindowX1-2,WindowY1-2);
END;  { OpenWindow  }

PROCEDURE CloseWindow;
BEGIN { CloseWindow }
  PutImage(WindowX0,WindowY0,TextWindow^,NormalPut);
END;  { CloseWindow }

PROCEDURE _Swap(VAR X0,X1:Integer );
BEGIN
  CX  := X0 ;
  X0  := X1 ;
  X1  := CX ;
END;

FUNCTION Adjust_X(X:Integer):Integer ;
BEGIN
  X := (X DIV 10)*StepX+Round((X MOD 10)*StepX/10);
  Adjust_X := Round(X/Zoom) + GraphTextWidth DIV 2;
END;

FUNCTION Adjust_Y(Y:Integer):Integer ;
BEGIN
  Y := (Y DIV 10)*StepY + Round((Y MOD 10)*StepY/10);
  Adjust_Y := Round(Y/Zoom);
END;

PROCEDURE Plot_Pixel ;
begin
  X0 := (XRef + Col)Div 2 ;
  X1 := X0 ;
  Repeat
    Y0 := (YRef + Rol)Div 2 ;
    Y1 := Y0 ;
    Repeat
      PutPixel(ADJUST_X(X0),ADJUST_Y(Y0),3);
      PutPixel(ADJUST_X(X0),ADJUST_Y(Y1),3);
      PutPixel(ADJUST_X(X1),ADJUST_Y(Y0),3);
      PutPixel(ADJUST_X(X1),ADJUST_Y(Y1),3);
      Inc(Y1,5);Dec(Y0,5);
    Until Y0 <= YRef ;
    Inc(X1,5);Dec(X0,5);
  Until (X0 <= XRef) ;
  REDRAW ;
end;

PROCEDURE ClearScreen;
BEGIN
  Bar(ADJUST_X(0)+1,ADJUST_Y(0)+1,ADJUST_X(Col)-1,ADJUST_Y(Rol)-1);
END;
```

```
PROCEDURE SetScreen_1 ;
BEGIN
  SetViewPort(0,0,MaxX,MaxY,ClipON);
  GetViewSettings(ViewPort);
  SetFillStyle(EmptyFill,GetBkColor);
END;

PROCEDURE SetScreen_2 ;
BEGIN
  SetViewPort(ADJUST_X(0),ADJUST_Y(0),ADJUST_X(Col),ADJUST_Y(Rol),ClipON);
  GetViewSettings(ViewPort);
  SetFillStyle(EmptyFill,GetBkColor);
END;

PROCEDURE DrawLine(X0,Y0,X1,Y1:Integer ;Color:Byte);
BEGIN
  SETSCREEN_2 ;
  SETCOLOR(Color);
  X0 := ADJUST_X(X0-XRef)-(GraphTextWidth DIV 2) ;
  X1 := ADJUST_X(X1-XRef)-(GraphTextWidth DIV 2) ;
  Y0 := ADJUST_Y(Y0-YRef) ;
  Y1 := ADJUST_Y(Y1-YRef) ;
  LINE(X0,Y0,X1,Y1);
  SETSCREEN_1 ;
END;

PROCEDURE DrawCirCle(X0,Y0,Radius:Integer;Color:Byte);
BEGIN
  SETSCREEN_2 ;
  X0 := ADJUST_X(X0-XRef)-(GraphTextWidth DIV 2);
  Y0 := ADJUST_Y(Y0-YRef) ;
  Radius := ADJUST_X(Radius)-(GraphTextWidth DIV 2);
  SETCOLOR(Color);
  CIRCLE(X0,Y0,Radius);
  SETSCREEN_1 ;
END;

PROCEDURE DrawArc(X0,Y0,Angle1,Angle2,Radius:Integer;Color:Byte);
BEGIN
  SETSCREEN_2 ;
  X0 := ADJUST_X(X0-XRef)-(GraphTextWidth DIV 2);
  Y0 := ADJUST_Y(Y0-YRef) ;
  SETCOLOR(Color);
  Radius := ADJUST_X(Radius)-(GraphTextWidth DIV 2)+(YOff_Text DIV Zoom);
  ARC(X0,Y0,ANGLE1,ANGLE2,Radius);
  SETSCREEN_1 ;
END;

PROCEDURE  DrawText(X0,Y0:Integer; PassStr:String;Color:Byte);
BEGIN
  If Zoom in[1,2] then
  BEGIN
    SETSCREEN_2 ;
    X0 := ADJUST_X(X0-XRef)-(GraphTextWidth DIV 2) ;
```

```
    Y0 := ADJUST_Y(Y0-YRef) ;
    If Zoom = 2 then SetTextStyle(SmallFont,HorizDir,2);
    Text_XY(X0,Y0,PassStr,Color);
    SetTextStyle(DefaultFont,HorizDir,1);
    SETSCREEN_1 ;
  END;
END;

PROCEDURE DrawConnect(X0,Y0,X1,Y1:Integer;Color:Byte);
BEGIN
  SETSCREEN_2 ;
  X0 := ADJUST_X(X0-XRef)-(GraphTextWidth DIV 2) ;
  X1 := ADJUST_X(X1-XRef)-(GraphTextWidth DIV 2) ;
  Y0 := ADJUST_Y(Y0-YRef) ;
  Y1 := ADJUST_Y(Y1-YRef) ;
  SetFillStyle(SolidFill,Color);
  Bar(X0,Y0,X1,Y1);
  SETSCREEN_1 ;
END;

PROCEDURE DrawLib( X,Y : Integer ; PassLib : Pointer_Func;Colors:Byte);
BEGIN
  Pass := False ;
  With PassLib^.Head Do
  For  Num := 1 to PassLib^.NumOfCom Do
  With PassLib^.Data[Num] Do
  BEGIN
    X0 := X - X_Ref + PX0 ; X1 := X - X_Ref + PX1 ;
    Y0 := Y - Y_Ref + PY0 ; Y1 := Y - Y_Ref + PY1 ;
    Case Class Of
    LibArc    : DrawArc(X0,Y0,Angle1,Angle2,Radius,Color[Colors]);
    LibCircle : DrawCircle(X0,Y0,Radius,Color[Colors]);
    LibLine   : DrawLine(X0,Y0,X1,Y1,Color[Colors]);
    LibText   : DrawText(X0,Y0,Lib_Text,Color[Colors]);
    END;
  END;
END;

PROCEDURE RedrawLib ;
BEGIN
  PASS := TRUE ;
  FuncCode := FirstCode ;
  While (FuncCode <> LastCode) Do
  With FuncCode^.Lib Do
  BEGIN
    Case Class Of
    LibArc    :
    IF ((PX0-Radius > XRef)AND(PY0-Radius > YRef))OR
       ((PX0-Radius > XRef)AND(PY0+Radius > YRef))OR
       ((PX0+Radius < XRef+Col)AND(PY0+Radius < YRef+Rol))OR
       ((PX0+Radius < XRef+Col)AND(PY0-Radius < YRef+Rol))THEN
      DrawArc(PX0,PY0,ANGLE1,ANGLE2,Radius,Color[Layer]);
    LibCircle :
    IF ((PX0-Radius > XRef)AND(PY0-Radius > YRef)AND
```

```
        (PX0-Radius < XRef+Col)AND(PY0-Radius < YRef+Rol))OR
       ((PX0-Radius > XRef)AND(PY0+Radius > YRef)AND
        (PX0-Radius < XRef+Col)AND(PY0+Radius < YRef+Rol))OR
       ((PX0+Radius > XRef)AND(PY0+Radius > YRef)AND
        (PX0+Radius < XRef+Col)AND(PY0+Radius < YRef+Rol))OR
       ((PX0+Radius < XRef)AND(PY0-Radius > YRef)AND
        (PX0+Radius < XRef+Col)AND(PY0-Radius < YRef+Rol))THEN
      DrawCircle(PX0,PY0,Radius,Color[Layer]);
   LibLine   :
   IF ((XRef<=PX0)AND(PX0<=XRef+Col)AND(YRef<=PY0)AND(PY0<=YRef+Rol))OR
      ((XRef<=PX1)AND(PX1<=XRef+Col)AND(YRef<=PY1)AND(PY1<=YRef+Rol))OR
      ((PX0 = PX0)AND(PY0 < YRef)AND(YRef + Rol < PY1))OR
      ((PY0 = PY1)AND(PX0 < XRef)AND(XRef + Col < PX1))then
      DrawLine(PX0,PY0,PX1,PY1,Color[Layer]);
   LIBTEXT   :
   IF ((XRef < PX0)AND(PX0 < XRef+Col)AND(YRef < PY0)AND(PY0 < YRef+Rol))then
      DrawText(PX0,PY0,LIB_TEXT,Color[Layer]);
   END;
   FuncCode := FuncCode^.Next ;
 END;
 SetColor(Color[3]);
 Rectangle(ADJUST_X(0),ADJUST_Y(0),ADJUST_X(Col),ADJUST_Y(Rol));
END;

PROCEDURE RedrawSch;
BEGIN
 SetColor(Color[3]);
 Rectangle(ADJUST_X(0),ADJUST_Y(0),ADJUST_X(Col),ADJUST_Y(Rol));
 FuncSch := FirstSch ;
 While (FuncSch <> LastSch) Do
 With FuncSch^ Do
 BEGIN
   Case Func Of
   LibLine:
   IF ((XRef<=SX0)AND(SX0<=XRef+Col)AND(YRef<=SY0)AND(SY0<=YRef+Rol))OR
      ((XRef<=SX1)AND(SX1<=XRef+Col)AND(YRef<=SY1)AND(SY1<=YRef+Rol))OR
      ((SX0 = SX0)AND(SY0 < YRef)AND(YRef + Rol < SY1))OR
      ((SY0 = SY1)AND(SX0 < XRef)AND(XRef + Col < SX1))then
      DrawLine(SX0,SY0,SX1,SY1,Color[Layer]);
   Connect:
   IF ((XRef < SX0)AND(SX0 < XRef+Col)AND(YRef < SY0)AND(SY0 < YRef+Rol))OR
      ((XRef < SX1)AND(SX1 < XRef+Col)AND(YRef < SY1)AND(SY1 < YRef+Rol))then
      DrawConnect(SX0,SY0,SX1,SY1,Color[Layer]);
   TEXT_,NODE :
   IF ((XRef < SX0)AND(SX0 < XRef+Col)AND(YRef < SY0)AND(SY0 < YRef+Rol))then
      DrawText(RefX,RefY,SchStr,Color[Layer]);
   END;
   If Func in[5..NumOfFunc] then
   IF ((XRef < SX0)AND(SX0 < XRef+Col)AND(YRef < SY0)AND(SY0 < YRef+Rol))OR
      ((XRef < SX1)AND(SX1 < XRef+Col)AND(YRef < SY1)AND(SY1 < YRef+Rol))then
      BEGIN
        FuncLib := FirstLib ; PASS := True ;
        While (FuncLib <> Nil)AND PASS Do
        With  FuncLib^ Do
```

```
        If SchName[Func] = Head.Name then
           DRAWLIB(RefX,RefY,FuncLib,Layer) Else
           FuncLib := FuncLib^.Next ;
      END;
    FuncSch := FuncSch^.Next ;
  END;
  SetColor(Color[3]); PASS := True ;
  Rectangle(ADJUST_X(0),ADJUST_Y(0),ADJUST_X(Col),ADJUST_Y(Rol));
END;

PROCEDURE DrawGrid ;
BEGIN
  ClearScreen  ;
  Y0 := 10 ;
  If Grid AND (Zoom < 4) then
  Repeat
    X0 := 10 ;
    Repeat
      PutPixel(ADJUST_X(X0),ADJUST_Y(Y0),Color[1]);
      Inc(X0,10);
    Until X0 > Col-10 ;
    Inc(Y0,10);
  Until Y0 > Rol-10 ;
END;

PROCEDURE Redraw ;
BEGIN
  ClearScreen ;
  If Grid Then DrawGrid
          Else ClearScreen ;
  If LIBRARIES_NAME <> '' then REDRAWLIB
                          else REDRAWSCH ;
END;

PROCEDURE _Grid ;
BEGIN
  Grid := Not(Grid) ;
  Redraw ;
END;

PROCEDURE Left ;
BEGIN
  If (XRef = 0) then
  BEGIN
    X := CX ; Y := CY ;
  END
  ELSE
  BEGIN
    XRef := Xref - 150 ;
    X    := X    + 150 ;
    REDRAW ;
  END;
END;
```

```
PROCEDURE Right ;
BEGIN
  If (X+XRef > 900) then
  BEGIN
    X := CX ; Y := CY ;
  END
  ELSE
  BEGIN
    XRef := Xref + 150 ;
    X    := X    - 150 ;
    REDRAW ;
  END;
END;

PROCEDURE Top  ;
BEGIN
  If (YRef = 0) then
  BEGIN
    X := CX ; Y := CY ;
  END
  ELSE
  BEGIN
    YRef := Yref - 100 ;
    Y    := Y    + 100 ;
    REDRAW ;
  END;
END;

PROCEDURE Bottom ;
BEGIN
  If(Y+YRef > 600) then
  BEGIN
    X := CX ; Y := CY ;
  END
  ELSE
  BEGIN
    YRef := Yref + 100 ;
    Y    := Y    - 100 ;
    REDRAW ;
  END;
END;

PROCEDURE GetKB ;
VAR
   X0,Y0,X1,Y1 : Word    ;

BEGIN
  PASS :=  FALSE ; FunctionKey := False ;
  Bar(MaxX-TEXT_X10,TEXT_MaxY,MaxX,MaxY);
  StringXY := ConCat(IntToStr((Y+YRef)DIV 10),'.',IntToStr((Y+YRef)MOD 10)) ;
  X0   :=  MaxX - Length(StringXY)*GraphTextWidth ;
  TEXT_XY(X0,TEXT_MaxY,StringXY,Color[3]);
  StringXY := ConCat(IntToStr((X+XRef)DIV 10),'.',IntToStr((X+XRef)MOD 10)) ;
  X0   :=  MaxX - (6 + Length(StringXY))*GraphTextWidth;
```

```
  TEXT_XY(X0,TEXT_MaxY,StringXY,Color[3]);
  X0   :=  ADJUST_X(X)-(GraphTextWidth DIV 2) ;
  X1   :=  X0+GraphTextWidth ;
  Y0   :=  ADJUST_Y(Y)       ;
  Y1   :=  Y0+GraphTextHeight;
  GetImage(X0,Y0,X1,Y1,Text_BK^);
  TEXT_XY(X0,Y0,Chr(24),Color[3]);
  Ch := ReadKey ;
  PutImage(X0,Y0,Text_BK^,NormalPut);
  CX := X ; CY := Y ;
  Case Ch Of
       #0: BEGIN
             FunctionKey := True ;
             Ch := ReadKey ;
             Case Ch Of
             Up_Key   : Y := Y - 10 ;
             Left_Key : X := X - 10 ;
             Right_Key: X := X + 10 ;
             Down_Key : Y := Y + 10 ;
             END;
           END;
       #56 : Dec(Y) ;
       #52 : Dec(X) ;
       #54 : Inc(X) ;
       #50 : Inc(Y) ;
       Enter: ;
       END;
   If (X < 0)   then  Left   ;
   If (X > Col) then  Right  ;
   If (Y < 0)   then  Top    ;
   If (Y > Rol) then  Bottom ;
END;

PROCEDURE _Error ;
BEGIN
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ TAGET UNDEFIND ]',Color[2]);
  Write(#$07) ; Ch := ReadKey;
  Bar(TEXT_X10,TEXT_MaxY,MaxX,MaxY);
  Ch := #00 ;
END;

PROCEDURE Menu_Taget ;
BEGIN
  MenuStr[1] := ' Tag [A] ' ;
  MenuStr[2] := ' Tag [B] ' ;
  MenuStr[3] := ' Tag [C] ' ;
  MenuStr[4] := ' Tag [D] ' ;
  MenuStr[5] := ' Tag [E] ' ;
  MenuStr[6] := ' Tag [F] ' ;
  MenuStr[7] := ' Tag [G] ' ;
  MenuStr[8] := ' Exit.  ' ;
  Num := 1 ;
  Num := SelectNo(MenuStr,8);
END;
```

```
PROCEDURE Jump ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,MaxX-10*GraphTextWidth,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ JUMP ]',Color[2]);
  MENU_TAGET ;
  Bar(TEXT_X10,TEXT_MaxY,TEXT_X10+10*GraphTextWidth,MaxY);
  Case Num Of
  1 : IF XA <> 65535 then BEGIN X := XA ; Y := YA ; END Else _Error ;
  2 : IF XB <> 65535 then BEGIN X := XB ; Y := YB ; END Else _Error ;
  3 : IF XC <> 65535 then BEGIN X := XC ; Y := YC ; END Else _Error ;
  4 : IF XD <> 65535 then BEGIN X := XD ; Y := YD ; END Else _Error ;
  5 : IF XE <> 65535 then BEGIN X := XE ; Y := YE ; END Else _Error ;
  6 : IF XF <> 65535 then BEGIN X := XF ; Y := YF ; END Else _Error ;
  7 : IF XG <> 65535 then BEGIN X := XG ; Y := YG ; END Else _Error ;
  8 : Quit := True ;
  END;
  Pass := False ;
  If ((XRef+Col) <= X)OR(X <= XRef)OR((YRef+Rol) <= Y)OR(Y <= YRef) then
  BEGIN
    REPEAT
      If  (X < XRef) then If Zoom = 1 then XRef := XRef - 150
                                      Else XRef := 0 ;
      If  (Y < YRef) then If Zoom = 1 then YRef := YRef - 100
                                      Else YRef := 0 ;
      If  (X > (XRef+Col)) then If Zoom = 1 then XRef := XRef + 150
                                            Else XRef := 300 ;
      If  (Y > (YRef+Rol)) then If Zoom = 1 then YRef := YRef + 100
                                            Else YRef := 200 ;
    UNTIL (X >= XRef)AND(Y >= YRef)AND(X <= (XRef+Col))AND(Y <= (YRef+Rol));
    Pass := True ;
  END;
  X := X - XRef ;
  Y := Y - YRef ;
  If Pass then REDRAW ;
  Quit := False ;
  REDRAW_FUNCTION_NAME ;
END;

PROCEDURE Tag  ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,MaxX-10*GraphTextWidth,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ TAGET ]',Color[2]);
  MENU_TAGET ;
  Bar(TEXT_X10,TEXT_MaxY,TEXT_X10+10*GraphTextWidth,MaxY);
  Case Num Of
  1 : BEGIN XA := X + XRef; YA := Y + YRef; END;
  2 : BEGIN XB := X + XRef; YB := Y + YRef; END;
  3 : BEGIN XC := X + XRef; YC := Y + YRef; END;
  4 : BEGIN XD := X + XRef; YD := Y + YRef; END;
  5 : BEGIN XE := X + XRef; YE := Y + YRef; END;
  6 : BEGIN XF := X + XRef; YF := Y + YRef; END;
  7 : BEGIN XG := X + XRef; YG := Y + YRef; END;
  8 : Quit := True ;
```

```
  END;
  Quit := False ;
  REDRAW_FUNCTION_NAME ;
END;

PROCEDURE Zoom_Menu ;
BEGIN
  MenuStr[1] := ' Zoom[1] ' ;
  MenuStr[2] := ' Zoom[2] ' ;
  MenuStr[3] := ' Zoom[3] ' ;
  MenuStr[4] := ' Zoom[4] ' ;
  MenuStr[5] := ' Zoom[5] ' ;
  MenuStr[6] := ' Exit.   ' ;
  Num := SelectNo(MenuStr,6);
END;

PROCEDURE _Zoom ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,MaxX,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ ZOOM ]',Color[2]);
  Case Zoom Of
  10 : Num := 1 ;
   8 : Num := 2 ;
   4 : Num := 3 ;
   2 : Num := 4 ;
   1 : Num := 5 ;
  END;
  Zoom_Menu ;
  Bar(TEXT_X10,TEXT_MaxY,MaxX,MaxY);
  X := X + XRef ; Y := Y + YRef ;
  No := Zoom ;
  Case Num Of
  1 : BEGIN
        Zoom :=  10  ;
        Rol  :=  600 ; YRef := 0 ;
        Col  :=  900 ; XRef := 0 ;
      END;
  2 : BEGIN
        Zoom :=  8   ;
        Rol  :=  600 ; YRef := 0 ;
        Col  :=  900 ; XRef := 0 ;
      END;
  3 : BEGIN
        Zoom :=  4   ;
        Rol  :=  600 ; YRef := 0 ;
        Col  :=  900 ; XRef := 0 ;
      END;
  4 : BEGIN
        Zoom :=  2   ;
        If X > 600 Then XRef := 300 Else XRef := 0 ;
        If Y > 400 Then YRef := 200 Else YRef := 0 ;
        Rol  :=  400 ;  Col  := 600 ;
      END;
  5 : BEGIN
```

```
        Zoom :=  1   ;
        If Zoom <> No Then
        BEGIN
          Case X of
            0..150: XRef := 0 ;
          151..300: XRef := 150 ;
          301..450: XRef := 300 ;
          451..600: XRef := 450 ;
          601..900: XRef := 600 ;
          End;
          Case Y of
            0..100: YRef := 0 ;
          101..200: YRef := 100 ;
          201..300: YRef := 200 ;
          301..400: YRef := 300 ;
          401..600: YRef := 400 ;
          End;
          Rol  :=  200 ;Col  :=  300 ;
        END;
      END;
  6 : Quit := True ;
  END;
  X := X - XRef ; Y := Y - YRef ;
  If (No <> Zoom)AND(Ch <> ESC) then REDRAW ;
  REDRAW_FUNCTION_NAME ;
  Quit := False ;
END;

FUNCTION SelectNo( SelStr : PassStr; No : Byte ): Byte ;
BEGIN
  SelectNo := Num ;
  CX  := Length(SelStr[1]) ;
  For Loop := 2 to No do
      If CX < Length(SelStr[Loop]) then CX := Length(SelStr[Loop]) ;
  X0 := 5*GraphTextWidth ;
  X1 := X0 + CX*GraphTextWidth;
  Y0 := 2*GraphTextHeight - 2  ;
  OpenWindow(X0,Y0+2,X1,Y0+No*(GraphTextHeight+2));
  SetTextJustify(LeftText,TopText);
  For loop := 1 to No do
  BEGIN
    If loop = 1 then Y0 := 2*GraphTextHeight  + YOff_Text else
                     Y0 := Y0+GraphTextHeight + 2 ;
    Text_XY(X0,Y0,SelStr[loop],Color[1]);
  END;
  Repeat
    For loop := 1 to Num do
    If  loop = 1  then Y0 := 2*GraphTextHeight  else
                       Y0 := Y0+GraphTextHeight + 2 ;
    Y1 := Y0 + GraphTextHeight ;
    GetImage(X0,Y0,X1,Y1,Text_BK^);
    PutImage(X0,Y0,Text_BK^,NotPut);
    If GraphDriver <> HercMono  then
       Text_XY(X0,Y0 + YOff_Text,SelStr[Num],Color[2]);
```

```
    Ch := ReadKey ;
    PutImage(X0,Y0,Text_BK^,NormalPut);
    If Ch = #0 then
    BEGIN
      Ch := ReadKey ;
      Case Ch Of
         Up_Key   : Dec(Num);
         Down_Key: Inc(Num);
         END;
      If OK then
      BEGIN
         If Num > No then Num := 1  ;
         If Num < 1  then Num := No ;
      END
      ELSE
      BEGIN
         If Num > No then Pass := True ;
         If Num < 1  then Pass := False ;
         If (Num > No)OR(Num < 1) then OK := True ;
      END;
    END
    else
    For loop := 1 to No do
    If (UpCase(Ch) = Copy(SelStr[loop],2,1))OR(Ord(Ch)-48 = Loop) then
    BEGIN
      Num := loop ;loop   := No ;
    END;
    If UpCase(Ch) = 'X' then Num := No ;
  Until (Ch IN[Enter,Esc])OR (OK AND((Num > No)OR(Num<1)));
  If Ch = ESC then SelectNo := 0
              else SelectNo := Num;
  CloseWindow; OK := True ;
  SetTextJustify(LeftText,TopText);
END;

PROCEDURE ReadText( CommandStr:String;Var PassStr:String;Count:Byte);
BEGIN
  GetImage(0,TEXT_MaxY,MaxX,MaxY,TextWindow^);
  Bar(0,TEXT_MaxY,MaxX,MaxY);
  Text_XY(0,TEXT_MaxY,CommandStr,Color[3]);
  X0 := Length(CommandStr)*GraphTextWidth ;
  CommandStr := PassStr ;
  PassStr := '' ; Loop := 1 ;
  Repeat
    Text_XY(X0,TEXT_MaxY,'_',Color[2]);
    Ch := ReadKey ; Ch := UpCase(Ch) ;
    If (Loop <= Count)AND Not(Ch in[^H,Enter,ESC]) then
    BEGIN
      PassStr := PassStr + Ch ;
      Bar(X0,TEXT_MaxY,X0+GraphTextWidth,MaxY);
      Text_XY(X0,TEXT_MaxY,Ch,Color[2]);
      Inc(X0,GraphTextWidth);Inc(Loop);
    END;
    If (Ch = ^H)AND(Length(PassStr) <> 0) then
```

```
    BEGIN
      Delete(PassStr,Length(PassStr),1) ;
      Dec(X0,GraphTextWidth);Dec(Loop);
      Bar(X0,TEXT_MaxY,X0+2*GraphTextWidth,MaxY);
    END;
  Until Ch In[Enter,ESC];
  If PassStr = '' then PassStr := CommandStr ;
  If (Ch = ESC)OR(Copy(PassStr,1,1) = #00) then PassStr := '' ;
  PutImage(0,TEXT_MaxY,TextWindow^,NormalPut);
END;

PROCEDURE REDRAW_FUNCTION_NAME ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,MaxX,MaxY);
  If LIBRARIES_NAME <> '' then
     TEXT_XY(TEXT_X10,TEXT_MaxY,'['+Libraries_Name+']',Color[2])
  Else
     TEXT_XY(TEXT_X10,TEXT_MaxY,'['+Schematic_Name+']',Color[2]);
END;

PROCEDURE FIND_NAME(VAR NAME:STRING);
BEGIN
  If LIBRARIES_NAME <> '' then  COMMAND := 'LIBRARY'+Chr(39)+'S NAME  '
                        else  COMMAND := 'SCHEMATIC'+Chr(39)+'S NAME  ';
  StringName := '' ;
  ReadText(COMMAND,StringName,8);
  If StringName <> '' then Name := StringName ;
  REDRAW_FUNCTION_NAME
END;

PROCEDURE MenuFile;
BEGIN
  MenuStr[1]  := ' Directory ' ;
  MenuStr[2]  := ' Load File ' ;
  MenuStr[3]  := ' Save File ' ;
  MenuStr[4]  := ' eXit.     ' ;
  Num := 1 ;
  Num := SelectNo(MenuStr,4);
END;

BEGIN
  InitialGraphics
END.
```

Program : Logcad Editor Declaration
Programmer : Suphan Gulpanich.

```
{
        ***********************************************
        *                                             *
        *     PROGRAM UNIT : VARIABLE DECLARATION     *
        *     PROGRAMER    : SUPHAN  GULPANICH        *
        *     VERSION      : 1.000A                   *
        *     LAST UPDATED : 15 JULY 1988             *
        *                                             *
        ***********************************************
}
UNIT DECLARE1 ;

INTERFACE

CONST
   Enter          =   #13    ;
   Esc            =   #27    ;
   Up_Key         =   #72    ;
   Left_Key       =   #75    ;
   Right_Key      =   #77    ;
   Down_Key       =   #80    ;
   Libline        =    01    ;    { LINE       }
   LibCircle      =    02    ;    { CIRCLE     }
   LibArc         =    03    ;    { ARC        }
   LibText        =    04    ;    { TEXT       }
   Connect        =    02    ;    { CONNECTOR  }
   Text_          =    03    ;    { TEXT(.SCH) }
   Node           =    04    ;    { NODE       }

TYPE
   FileStr        =   Array[1..50] Of String;
   PassStr        =   Array[1..15] Of String;
   LibClass       =   Libline..libText      ;
   FileType       =   Text      ;

   Gate_Lib       =   Record
       Name       :   String    ;
       X_Ref,Y_Ref:   Integer   ;
       X_Min,Y_Min:   Integer   ;
       X_Max,Y_Max:   Integer   ;
       END;

   Com_Lib        =   Record
       Class      :   LibClass ;
       PX0,PY0    :   Integer   ;
```

```
        PX1,PY1     :   Integer  ;
        Angle1,Angle2:  Integer  ;
        Radius      :   Integer  ;
        Lib_Text    :   String   ;
        Layer       :   Byte     ;
        END;

    Pointer_lib     =   ^Lib_LIB ;
    Lib_Lib         =   Record
        Lib         :   Com_LIB  ;
        Next        :   Pointer_lib;
        END;

    Pointer_Sch     =   ^Sch_LIB;
    Sch_Lib         =   Record
        Func        :   Byte     ;
        RefX,RefY   :   Integer  ;
        SX0,SY0     :   Integer  ;
        SX1,SY1     :   Integer  ;
        SchStr      :   String   ;
        Layer       :   Byte     ;
        Next        :   Pointer_Sch;
        END;

    Pointer_Func    =   ^Func_LIB;
    Func_Lib        =   Record
        Head        :   Gate_Lib ;
        NumOfCom    :   Byte     ;
        Data        :   Array[1..50] Of Com_Lib ;
        Next        :   Pointer_Func;
        END;

VAR
    Ok,Pass         :   Boolean  ;
    FunctionKey     :   Boolean  ;
    Quit,Grid       :   Boolean  ;
    Num,No          :   Byte     ;
    Radius          :   Byte     ;
    NumOfFunc       :   Byte     ;
    loop,Zoom       :   Byte     ;
    MenuSelect      :   Byte     ;
    Ch              :   Char     ;
    X,Y             :   Integer  ;
    X0,Y0,X1,Y1     :   Integer  ;
    FileName        :   String   ;
    Command         :   String   ;
    LIBRARIES_NAME  :   String   ;
    SCHEMATIC_NAME  :   String   ;
    String10        :   String[10] ;
    YRef,XRef       :   Word     ;
    Rol,Col,KeyCount    :   Word ;
    NumOfSch,NumOfLib   :   Word ;
    XA,XB,XC,XD,XE,XF,XG :  Word ;
    YA,YB,YC,YD,YE,YF,YG :  Word ;
```

```
   FileData       :   FileType   ;
   FileSch        :   FileType   ;
   DirName,SchName:   FileStr    ;
   Libraries      :   Gate_Lib   ;
   MenuStr        :   PassStr    ;
   hTop           :   ^Integer   ;
   FirstCode,FuncCode,LastCode: Pointer_lib  ;
   FirstSch,FuncSch,LastSch   : Pointer_Sch  ;
   FirstLib,FuncLib,LastLib   : Pointer_Func ;

IMPLEMENTATION

BEGIN
   Quit := False ; Grid := False ;
   XRef :=     0 ; YRef :=     0 ;
   Rol  :=   200 ; Col  :=   300 ;
   X    :=   100 ; Y    :=   100 ; Zoom :=   1 ;
   XA   := 65535 ; XB   :=    XA ; XC   :=  XA ;
   XD   :=    XA ; XE   :=    XA ; XF   :=  XA ; XG := XA ;
END.
```

Program : Logcad Library Editor Program
Programmer : Suphan Gulpanich.

```
{
        ************************************************
        *                                              *
        *      PROGRAM UNIT : LIBRARIES EDITOR         *
        *      PROGRAMER    : SUPHAN  GULPANICH        *
        *      VERSION      : 1.000A                   *
        *      LAST UPDATED : 15 JULY 1988             *
        *                                              *
        ************************************************
}
UNIT LIB_EDIT;

INTERFACE

USES CRT,GRAPH,DECLARE1,INITIAL,DIRECTOR,OPENFILE;

PROCEDURE LIBRARY ;

IMPLEMENTATION

PROCEDURE MainMenu ;
BEGIN
  MenuStr[1]  := ' Block  ' ;
  MenuStr[2]  := ' Clear  ' ;
  MenuStr[3]  := ' Delete ' ;
  MenuStr[4]  := ' Edit   ' ;
  MenuStr[5]  := ' File   ' ;
  MenuStr[6]  := ' Grid   ' ;
  MenuStr[7]  := ' Jump   ' ;
  MenuStr[8]  := ' Label  ' ;
  MenuStr[9]  := ' Redraw ' ;
  MenuStr[10] := ' Tag    ' ;
  MenuStr[11] := ' Zoom   ' ;
  MenuStr[12] := ' eXit.  ' ;
  Num := SelectNo(MenuStr,12);
END;

PROCEDURE HelpMenu ;
BEGIN
  MenuStr[1] := ' F1:Edit  ' ;
  MenuStr[2] := ' F2:Save  ' ;
  MenuStr[3] := ' F3:Lode  ' ;
  MenuStr[4] := ' F4:Get   ' ;
  MenuStr[5] := ' F5:Zoom  ' ;
  MenuStr[6] := ' F6:Redraw' ;
```

```
  MenuStr[7] := ' F7:Menu  ' ;
  MenuStr[8] := ' F8:Help  ' ;
  MenuStr[9] := ' F9:Exit  ' ;
  Num := SelectNo(MenuStr,9);
END;

PROCEDURE MenuEdit ;
BEGIN
  MenuStr[1]  := ' Arc    ' ;
  MenuStr[2]  := ' Circle ' ;
  MenuStr[3]  := ' Get-lib' ;
  MenuStr[4]  := ' Line   ' ;
  MenuStr[5]  := ' Text   ' ;
  MenuStr[6]  := ' eXit.  ' ;
  Num := SelectNo(MenuStr,6);
END;

PROCEDURE INITIAL_LIBRARY_HEAD ;
BEGIN
  Mark(hTop);
  New(FuncCode);
  FirstCode := FuncCode ;
  LastCode  := FuncCode ;
  NumOfLib  :=    0     ;
  Schematic_Name :=  '' ;
  Libraries_Name := '?' ;
  With Libraries DO
  BEGIN
    X_Ref := 0 ; Y_Ref := 0 ;
    X_Min := 0 ; Y_Min := 0 ;
    X_Max := 0 ; Y_Max := 0 ;
  END;
  Grid := NOT Grid ;
 _Grid ;
  TEXT_XY(0,TEXT_MaxY,'LIBRARY.',Color[3]);
  REDRAW_FUNCTION_NAME
END;

PROCEDURE SAVE_CIRCLE(CX1,CY1,RadiusCircle:Integer);
BEGIN
  Writeln(FileData,LibCircle:4);
  Writeln(FileData,CX1:4,CY1:8);
  Writeln(FileData,Radiuscircle:4);
END;

PROCEDURE SAVE_ARC(AX1,AY1,Angle1,Angle2,RadiusArc:Integer);
BEGIN
  Writeln(FileData,LibArc:4);
  Writeln(FileData,AX1:4,AY1:8);
  Writeln(FileData,Angle1:4,Angle2:8);
  Writeln(FileData,Radiusarc:4);
END;

PROCEDURE SAVE_LINE(LX1,LY1,LX2,LY2:Integer);
```

```
BEGIN
  Writeln(FileData,LibLine:4);
  Writeln(FileData,LX1:4,LY1:8);
  Writeln(FileData,LX2:4,LY2:8);
END;

PROCEDURE SAVE_TEXT(TX1,TY1:Integer;SText:String);
BEGIN
  Writeln(FileData,LibText:4);
  Writeln(FileData,TX1:4,TY1:8);
  Writeln(FileData,' ',SText);
END;

PROCEDURE SAVE_FILE; { FILE LIBRARY }
BEGIN
  If Libraries_Name = '?' Then FIND_NAME(Libraries_Name);
  If Libraries_Name <>'?' Then
  BEGIN
    Assign(FileData,Libraries_Name+'.LIB');
    ReWrite(FileData);
    With Libraries DO
    BEGIN
      GetImage(0,TEXT_MaxY,MaxX,MaxY,TextWinDow^);
      Bar(0,TEXT_MaxY,MaxX,MaxY);
      TEXT_XY(0,TEXT_MaxY,'SAVE FILE['+FileName+']',Color[2]);
      Writeln(FileData,'LIBRARY',Chr(39),'S  NAME   (',Libraries_Name,')');
      Writeln(FileData);
      Writeln(FileData,'REFERENCE (X,Y)');
      Writeln(FileData,X_Ref:4,Y_Ref:8,Enter);
      Writeln(FileData,'XMIN    YMIN');
      Writeln(FileData,X_Min:4,Y_Min:8,Enter);
      Writeln(FileData,'XMAX    YMAX');
      Writeln(FileData,X_Max:4,Y_Max:8,Enter);
      FuncCode := FirstCode ; NumOfLib := 0 ;
      While (FuncCode <> Nil) DO
      With FuncCode^.Lib do
      BEGIN
        IF (X_Min<=PX0)AND(PX0<=X_Max)AND(Y_Min<=PY0)AND(PY0<=Y_Max) then
           Inc(NumOfLib);
        FuncCode := FuncCode^.Next ;
      END;
      Writeln(FileData,'NUMBER OF COMMAND');
      Writeln(FileData,NumOfLib:4,Enter);
      FuncCode := FirstCode ;
      While (FuncCode <> Nil) DO
      With FuncCode^.Lib do
      BEGIN
        IF (X_Min<=PX0)AND(PX0<=X_Max)AND(Y_Min<=PY0)AND(PY0<=Y_Max) then
        BEGIN
          Case Class Of
             Libline  : SAVE_LINE(PX0,PY0,PX1,PY1);
             Libcircle: SAVE_CIRCLE(PX0,PY0,Radius);
             Libarc   : SAVE_ARC(PX0,PY0,Angle1,Angle2,Radius);
             LibText  : SAVE_Text(PX0,PY0,Lib_Text);
```

```
          END;
          Writeln(FileData,Layer:4,Enter);
        END;
        FuncCode := FuncCode^.Next ;
      END;
      PutImage(0,TEXT_MaxY,TextWinDow^,NormalPut);
      Close(FileData);
    END;
  END;
END;

PROCEDURE LOAD_LIB ;
BEGIN
  GetImage(0,TEXT_MaxY,MaxX,MaxY,TextWinDow^);
  Bar(0,TEXT_MaxY,MaxX,MaxY);
  TEXT_XY(0,TEXT_MaxY,'LOAD FILE['+FileName+']',Color[2]);
  With Libraries DO
  BEGIN
    Release(hTop);
    Mark(hTop);
    Readln(FileData,Command);
    Libraries_Name := Copy(Command,20,Length(Command)-20);
    Readln(FileData);Readln(FileData);
    Readln(FileData,X_Ref,Y_Ref);
    Readln(FileData);Readln(FileData);
    Readln(FileData,X_Min,Y_Min);
    Readln(FileData);Readln(FileData,Command);
    Readln(FileData,X_Max,Y_Max);
    Readln(FileData);
    Readln(FileData,Command);
    Read(FileData,NumOfLib);
    Release(hTop); Mark(hTop);
    New(FuncCode); Num := 0 ;
    FirstCode := FuncCode ;
    LastCode  := FuncCode ;
    While (Num < NumOfLib) Do
    With FuncCode^.Lib do
    BEGIN
      Read(FileData,Class);
      Case Class Of
           Libline  : LOAD_LINE(PX0,PY0,PX1,PY1);
           Libcircle: LOAD_CIRCLE(PX0,PY0,Radius);
           Libarc   : LOAD_ARC(PX0,PY0,Angle1,Angle2,Radius);
           LibText  : LOAD_Text(PX0,PY0,Lib_Text);
      END;
      Read(FileData,Layer);
      New(FuncCode) ;  Inc(Num) ;
      LastCode^.Next := FuncCode ;
      LastCode       := FuncCode ;
    END;
    LastCode^.Next := Nil ;
    Close(FileData);
    REDRAW ;
  END;
```

```
  PutImage(0,TEXT_MaxY,TextWinDow^,NormalPut);
  REDRAW_FUNCTION_NAME ;
END;

PROCEDURE LOAD_FILE; { FILE LIBRARY }
BEGIN
  IF OpenReadFile(FileData,'.LIB') THEN LOAD_LIB ;
END;

PROCEDURE BLOCK ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[[BROWDER]]',Color[2]);
  With LIBRARIES do
  BEGIN
    X_Min := X + XRef ; Y_Min := Y + YRef ;
    REPEAT
      X0 := ADJUST_X(X_Min - XRef) ; X1 := ADJUST_X(X) ;
      Y0 := ADJUST_Y(Y_Min - YRef) ; Y1 := ADJUST_Y(Y) ;
      IF (Y0 > Y1) then _Swap(Y0,Y1);
      IF (X0 > X1) then _Swap(X0,X1);
      GetImage(X0,Y0,X1,Y1,TextWinDow^) ;
      PutImage(X0,Y0,TextWinDow^,NotPut);
      GETKB ;
      PutImage(X0,Y0,TextWinDow^,NormalPut);
    UNTIL Ch in[Enter,ESC] ;
    IF Ch = Enter Then
    BEGIN
      X_Max := X + XRef ; Y_Max := Y + YRef ;
      IF (X_Min > X_Max) then _Swap(X_Min,X_Max);
      IF (Y_Min > Y_Max) then _Swap(Y_Min,Y_Max);
      Bar(TEXT_X10,TEXT_MaxY,25*GraphTextWidth,MaxY);
      TEXT_XY(TEXT_X10,TEXT_MaxY,'[ REFFERENCE X,Y ]',Color[2]);
      REPEAT
        GETKB ;
      UNTIL Ch in[Enter,ESC];
      IF Ch = Enter Then
      BEGIN
        X_Ref := X + XRef ; Y_Ref := Y + YRef ;
      END
      ELSE
      BEGIN
        X_Ref := X_Min    ; Y_Ref := Y_Min    ;
      END;
    END
    ELSE
    BEGIN
      X_Min := 0 ; X_Max := 0 ;
      Y_Min := 0 ; Y_Max := 0 ;
      X_Ref := 0 ; Y_Ref := 0 ;
    END;
  END;
  REDRAW_FUNCTION_NAME ;
END;
```

```
PROCEDURE ClearWorkSheet ;
BEGIN
  GetImage(0,TEXT_MaxY,MaxX,MaxY,TextWinDow^);
  Bar(0,TEXT_MaxY,MaxX,MaxY);
  TEXT_XY(0,TEXT_MaxY,'CLEAR WORK SHEET [Y/N]',Color[2]);
  Ch := ReadKey ;
  PutImage(0,TEXT_MaxY,TextWinDow^,NormalPut);
  If Ch in[Enter,'Y','y'] then
  BEGIN
    Release(hTop);
    INITIAL_LIBRARY_HEAD
  END
END;

PROCEDURE DELETE_LIB( X,Y : Word);
VAR   DelCode : Pointer_lib ;

PROCEDURE DEL_LIB;
BEGIN
  TEXT_XY(TEXT_X10,TEXT_MaxY,'DELETE FUNCTION[Y/N]',Color[1]);
  Ch := readKey ;
  With DelCode^.Lib DO
  If Ch in[Enter,'y','Y'] then
  BEGIN
    Case Class Of
    LibLine:  DrawLine(PX0,PY0,PX1,PY1,Color[0]);
    LibArc :  DrawArc(PX0,PY0,Angle1,Angle2,Radius,Color[0]);
    LibCirCle:DrawCirCle(PX0,PY0,Radius,Color[0]);
    LibText:  DRAWTEXT(PX0,PY0,LIB_TEXT,Color[0]);
    END;
    If DelCode = FirstCode then
    BEGIN
      DelCode   :=  DelCode^.Next ;
      FirstCode :=  DelCode       ;
    END
    ELSE
    BEGIN
      DelCode   :=  DelCode^.Next ;
      FuncCode^.Next := DelCode   ;
    END;
    DEC(NumOfLib);
  END
  ELSE
    Case Class Of
    LibLine:  DrawLine(PX0,PY0,PX1,PY1,Color[Layer]);
    LibArc :  DrawArc(PX0,PY0,Angle1,Angle2,Radius,Color[Layer]);
    LibCirCle:DrawCirCle(PX0,PY0,Radius,Color[Layer]);
    LibText:  DRAWTEXT(PX0,PY0,LIB_TEXT,Color[Layer]);
    END;
  Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
END;

BEGIN
```

```
  FuncCode :=  FirstCode  ;
  DelCode  :=  FirstCode  ;
  While (DelCode <> Nil) DO
  With DelCode^.Lib DO
  BEGIN
    Case Class Of
    LibLine:  If ((PX0 = X)AND(PY0 = Y))OR((PX1 = X)AND(PY1 = Y)) THEN
              BEGIN
                DrawLine(PX0,PY0,PX1,PY1,Color[1]);
                DEL_LIB ;
              END;
    LibArc :  BEGIN
                X0 := PX0 + Round(Radius*Cos(Pi*Angle1/180));
                Y0 := PY0 - Round(Radius*Sin(Pi*Angle1/180));
                X1 := PX0 + Round(Radius*Cos(Pi*Angle2/180));
                Y1 := PY0 - Round(Radius*Sin(Pi*Angle2/180));
                If(((X0-5 <= X)AND(Y0-5 <= Y)  AND
                    (X0+5 >= X)AND(Y0+5 >= Y)))OR
                  (((X1-5 <= X)AND(Y1-5 <= Y)  AND
                    (X1+5 >= X)AND(Y1+5 >= Y)))then
                BEGIN
                  DrawArc(PX0,PY0,Angle1,Angle2,Radius,Color[1]);
                  DEL_LIB ;
                END;
              END;
    LibCirCle:If (PX0-Radius <= X)AND(PY0-Radius <= Y) AND
                (PX0+Radius >= X)AND(PY0+Radius >= Y) THEN
              BEGIN
                DrawCirCle(PX0,PY0,Radius,Color[1]);
                DEL_LIB ;
              END;
    LibText: BEGIN
               CX := ADJUST_X(X-XRef)   ; CY := ADJUST_Y(Y-YRef)    ;
               X0 := ADJUST_X(PX0-XRef) ; Y0 := ADJUST_Y(PY0-YRef) ;
               X1 := X0 + Length(Lib_Text)*GraphTextWidth ;
               Y1 := Y0 + GraphTextHeight;
               If (X0 <= CX)AND(CX <= X1)AND(Y0 <= CY)AND(CY <= Y1) THEN
               BEGIN
                  DRAWTEXT(PX0,PY0,LIB_TEXT,Color[1]);
                  DEL_LIB ;
               END;
             END;
    END;
    FuncCode := DelCode       ;
    DelCode  := DelCode^.next ;
  END;
  FuncCode := LastCode ;
END;

PROCEDURE _DELETE;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ DELETE ]',Color[2]);
  Repeat
```

```
    GETKB ;
  Until Ch in[Enter,ESC];
  Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
  If Ch = Enter then DELETE_LIB(X+XRef,Y+YRef);
  REDRAW_FUNCTION_NAME ;
END;

FUNCTION FIND_RADIUS(XF,YF : Integer):Integer ;
BEGIN
  Repeat
    X0 := X+XRef ; Y0 := Y+YRef ;
    Radius := Round(Sqrt(Sqr(XF-X)+Sqr(YF-Y)));
    Bar(TEXT_X10,TEXT_MaxY,25*GraphTextWidth,MaxY);
    Command := 'RADIUS = '+IntToStr(Radius) ;
    TEXT_XY(TEXT_X10,TEXT_MaxY,Command,Color[1]);
    X0 := ADJUST_X(XF-Radius); X1 := ADJUST_X(XF+Radius);
    Y0 := ADJUST_Y(YF-Radius); Y1 := ADJUST_Y(YF+Radius);
    IF (Y1 = 0) then Y1 := 1 ;
    GetImage(X0,Y0-1,X1,Y1+1,TextWinDow^);
    DrawLine(XF+XRef,YF+YRef,X+XRef,Y+YRef,Color[1]);
    GETKB ;
    PutImage(X0,Y0-1,TextWinDow^,NormalPut);
  Until Ch in[Enter,ESC] ;
  IF Ch = Enter Then FIND_RADIUS := Radius ;
END;

FUNCTION FIND_ANGLE(X0,Y0,X1,Y1:WORD):WORD;
BEGIN
  IF (X0 < X1)AND(Y0 > Y1) then
     FIND_ANGLE := Round(180/Pi*Arctan((Y0-Y1)/(X1-X0)));
  IF (X0 > X1)AND(Y0 > Y1) then
     FIND_ANGLE := 180 - Round(180/Pi*Arctan((Y0-Y1)/(X0-X1)));
  IF (X0 > X1)AND(Y0 < Y1) then
     FIND_ANGLE := 180 + Round(180/Pi*Arctan((Y1-Y0)/(X0-X1)));
  IF (X0 < X1)AND(Y0 < Y1) then
     FIND_ANGLE := 360 - Round(180/Pi*Arctan((Y1-Y0)/(X1-X0)));
  IF (Y0 = Y1) then
     IF (X0 < X1) then FIND_ANGLE := 0
                  else FIND_ANGLE := 180 ;
  IF (X0 = X1) then
     IF (Y0 > Y1) then FIND_ANGLE := 90
                  else FIND_ANGLE := 270 ;
END;

PROCEDURE _ARC ;
BEGIN
  With  FuncCode^.Lib do
  BEGIN
    TEXT_XY(TEXT_X10,TEXT_MaxY,' CENTER[X,Y] ',Color[2]);
    Repeat
      GETKB ;
    Until Ch in[Enter,ESC] ;
    Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
    IF Ch = Enter Then
```

```
    BEGIN
      Class  := LIBARC   ; Layer := 2        ;
      PX0    := X + XRef ; PY0   := Y + YRef ;
      Radius := FIND_RADIUS(X,Y);
      Angle1 := FIND_ANGLE(PX0-XRef,PY0-YRef,X,Y);
      Repeat
        X0 := ADJUST_X(PX0-XRef); X1 := ADJUST_X(X);
        Y0 := ADJUST_Y(PY0-YRef); Y1 := ADJUST_Y(Y);
        IF (Y1 = 0) then Y1 := 1 ;
        IF (Y0 > Y1) then _Swap(Y0,Y1);
        IF (X0 > X1) then _Swap(X0,X1);
        GetImage(X0,Y0-1,X1,Y1+1,TextWinDow^);
        PX1  := X + XRef ;
        PY1  := Y + YRef ;
        DrawLine(PX0,PY0,PX1,PY1,Color[1]);
        Angle2 := FIND_ANGLE(PX0-XRef,PY0-YRef,X,Y);
        DrawArc(PX0,PY0,Angle1,Angle2,Radius,Color[3]);
        Bar(TEXT_X10,TEXT_MaxY,28*GraphTextWidth,MaxY);
        Command := 'START '+IntToStr(Angle1)+' STOP '+IntToStr(Angle2) ;
        TEXT_XY(TEXT_X10,TEXT_MaxY,Command,Color[2]);
        GETKB ;
        DrawArc(PX0,PY0,Angle1,Angle2,Radius,Color[0]);
        If NOT PASS then PutImage(X0,Y0-1,TextWinDow^,NormalPut);
      Until Ch in[Enter,ESC] ;
      Bar(TEXT_X10,TEXT_MaxY,28*GraphTextWidth,MaxY);
      IF Ch = Enter then
      BEGIN
        DrawArc(PX0,PY0,Angle1,Angle2,Radius,Color[Layer]);
        Inc(NumOfLib);
        New(FuncCode);
        LastCode^.Next := FuncCode ;
        LastCode       := FuncCode ;
        LastCode^.Next := Nil      ;
      END
      ELSE PutImage(X0,Y0-1,TextWinDow^,NormalPut);
    END;
  END;
END;

PROCEDURE _CIRCLE ;
BEGIN
  With FuncCode^.Lib do
  BEGIN
    TEXT_XY(TEXT_X10,TEXT_MaxY,' CENTER[X,Y] ',Color[2]);
    Repeat
      GETKB ;
    Until Ch in[Enter,ESC] ;
    Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
    IF Ch = Enter Then
    BEGIN
      Class := LIBCIRCLE ; Layer := 2        ;
      PX0   := X + XRef  ; PY0   := Y + YRef;
      Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
      TEXT_XY(TEXT_X10,TEXT_MaxY,' MAXNITUDE ',Color[2]);
```

```
      Repeat
        X0 := X+XRef ; Y0 := Y+YRef ;
        Radius := Round(Sqrt(Sqr(PX0-X0)+Sqr(PY0-Y0)));
        X0 := ADJUST_X(PX0-Radius-XRef); X1 := ADJUST_X(PX0+Radius-XRef);
        Y0 := ADJUST_Y(PY0-Radius-YRef); Y1 := ADJUST_Y(PY0+Radius-YRef);
        IF (Y1 = 0) then Y1 := 1 ;
        GetImage(X0,Y0-1,X1,Y1+1,TextWinDow^);
        IF Radius <> 0 then DrawCirCle(PX0,PY0,Radius,Color[Layer]);
        GETKB ;
        PutImage(X0,Y0-1,TextWinDow^,NormalPut);
      Until Ch in[Enter,ESC] ;
      Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
      If Ch = Enter then
      BEGIN
        DrawCirCle(PX0,PY0,Radius,Color[Layer]);
        New(FuncCode); Inc(NumOfLib);
        LastCode^.Next := FuncCode ;
        LastCode       := FuncCode ;
        LastCode^.Next := Nil      ;
      END;
    END;
  END;
END;

PROCEDURE _LINE ;
BEGIN
  Ch := #0 ;
  Repeat
    TEXT_XY(TEXT_X10,TEXT_MaxY,'LINE[X1,Y1]',Color[2]);
    If Ch <> Enter Then
       Repeat GETKB ;
       Until Ch in[Enter,ESC,#67] ;
    Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
    IF Ch = Enter Then
    BEGIN
      TEXT_XY(TEXT_X10,TEXT_MaxY,'LINE[X2,Y2] C:Cancle',Color[2]);
      With FuncCode^.Lib do
      BEGIN
        Class := LIBLINE ; Layer := 2      ;
        PX0:= X + XRef ; PY0 := Y + YRef ;
        X0 := Adjust_X(PX0-XRef-1) ;Y0 := Adjust_Y(PY0-YRef-1) ;
        X1 := Adjust_X(PX0-XRef+1) ;Y1 := Adjust_Y(PY0-YRef+1) ;
        GetImage(X0,Y0,X1,Y1,TextWinDow^);
        Drawline(PX0,PY0-1,PX0,PY0+1,Color[3]);
        Drawline(PX0-1,PY0,PX0+1,PY0,Color[3]);
        REPEAT
          GETKB ;
          If Pass AND(XRef < PX0)AND(YRef < PY0)AND
            (PX0 < XRef+Col)AND(PY0 < YRef+Rol) Then
          BEGIN
             X0 := Adjust_X(PX0-XRef-1) ;Y0 := Adjust_Y(PY0-YRef-1) ;
             X1 := Adjust_X(PX0-XRef+1) ;Y1 := Adjust_Y(PY0-YRef+1) ;
             GetImage(X0,Y0,X1,Y1,TextWinDow^);
             Drawline(PX0,PY0-1,PX0,PY0+1,Color[3]);
```

```
            Drawline(PX0-1,PY0,PX0+1,PY0,Color[3]);
          END;
        UNTIL Ch in[Enter,ESC,#67,'C','c'] ;
      END;
      IF Ch = Enter THEN
      With FuncCode^.Lib Do
      BEGIN
        If(XRef < PX0)AND(PX0 < XRef+Col) AND
          (YRef < PY0)AND(PY0 < YRef+Rol) Then
           PutImage(X0,Y0,TextWinDow^,NormalPut);
        PX1 :=  X + XRef;  PY1 := Y + YRef ;
        DrawLine(PX0,PY0,PX1,PY1,Color[Layer]);
        Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
        TEXT_XY(TEXT_X10,TEXT_MaxY,'LINE[X1,Y1] C:Cancle',Color[2]);
        GETKB ;
        If Ch in['C','c'] then
         DrawLine(PX0,PY0,PX1,PY1,Color[0]) ELse
        BEGIN
          New(FuncCode)   ; Inc(NumOfLib);
          LastCode^.Next := FuncCode ;
          LastCode       := FuncCode ;
          LastCode^.Next := Nil      ;
        END;
      END
      Else PutImage(X0,Y0,TextWinDow^,NormalPut);
      Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
    END;
  Until Ch in[ESC,#67] ;
  Num := 4 ;
END;

PROCEDURE TEXT ;
BEGIN
  Command := '' ;
  ReadText('ENTER TEXT --> ',Command,8);
  If Command <> '' then
  BEGIN
    Bar(TEXT_X10,TEXT_MaxY,MaxX,MaxY);
    TEXT_XY(TEXT_X10,TEXT_MaxY,'TEXT <'+Command+'>',Color[2]);
    X0 := ADJUST_X(X); X1 := X0 + Length(Command)*GraphTextWidth ;
    Y0 := ADJUST_Y(Y); Y1 := Y0 + GraphTextHeight;
    If ((X0 < ADJUST_X(0))AND(XRef = 0))OR
       ((Y0 < ADJUST_Y(0))AND(YRef = 0))OR
       (X1 > ADJUST_X(900))OR(Y1 > ADJUST_Y(600))then
    BEGIN
      Write(#$007);
      TEXT_XY(TEXT_X10,TEXT_MaxY,'< RANGE OVER >',Color[2]);
      Ch := ReadKey;
      Bar(TEXT_X10,TEXT_MaxY,28*GraphTextWidth,MaxY);
      Ch := #00 ;
    END
    ELSE
      REPEAT
        X0 := ADJUST_X(X);Y0 := ADJUST_Y(Y);
```

```
        X1 := X0 + Length(Command)*GraphTextWidth ;
        Y1 := Y0 + GraphTextHeight;
        If X0 < ADJUST_X(0)   then Left  ;
        If X1 > ADJUST_X(Col)then Right ;
        If Y0 < ADJUST_Y(0)   then Top   ;
        If Y1 > ADJUST_Y(Rol)then Bottom;
        X0 := ADJUST_X(X);Y0 := ADJUST_Y(Y);
        X1 := X0 + Length(Command)*GraphTextWidth ;
        Y1 := Y0 + GraphTextHeight;
        GetImage(X0,Y0,X1,Y1,TextWindow^);
        DRAWTEXT(X+XRef,Y+YRef,Command,Color[3]);
        GetKB ;
        If Not Pass then PutImage(X0,Y0,TextWinDow^,NormalPut);
      Until Ch in[Enter,ESC];
    IF Ch = Enter Then
    With FuncCode^.Lib do
    BEGIN
      Class := LIBTEXT   ;
      PX0   := X + XRef  ; PY0 := Y + YRef ;
      LIB_TEXT := Command; Layer := 2      ;
      DRAWTEXT(PX0,PY0,LIB_TEXT,Color[Layer]);
      New(FuncCode)  ; Inc(NumOfLib);
      LastCode^.Next := FuncCode ;
      LastCode       := FuncCode ;
      LastCode^.Next := Nil      ;
    END;
    Bar(TEXT_X10,TEXT_MaxY,MaxX,MaxY);
  END;
  Num := 5
END;

PROCEDURE Get_Lib ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ GET LIBRARY ]',Color[2]);
  DIRPRO('.LIB');
  IF Ch = Enter Then
  BEGIN
    FileName := Copy(PassName[Num],2,Length(PassName[Num])-2);
    Assign(Filedata,FileName+'.LIB');
    Ok  := Create(FileData);
    If Ok then LOAD_LIB ;
  END;
  Num := 3 ;
  REDRAW_FUNCTION_NAME ;
END;

PROCEDURE EDIT;
BEGIN
  Num := 1 ;
  Repeat
    Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
    TEXT_XY(TEXT_X10,TEXT_MaxY,'[ EDIT ]',Color[2]);
    MenuEdit;
```

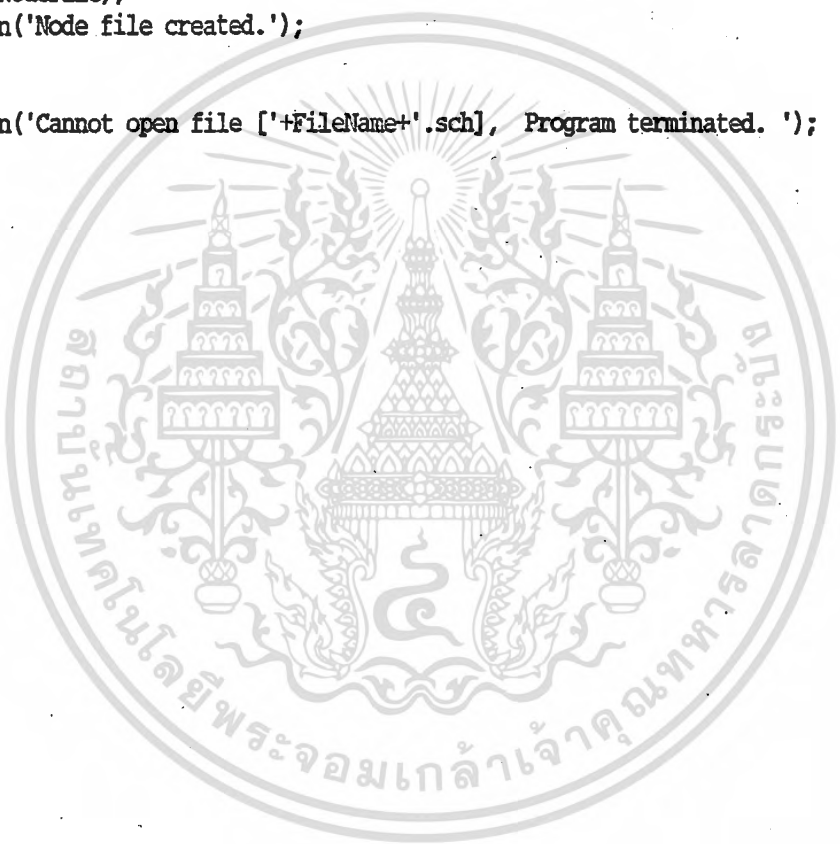
```
    Bar(TEXT_X10,TEXT_MaxY,18*GraphTextWidth,MaxY);
    IF Ch = ESC then Quit := True ;
    Case Num Of
         1 : _ARC     ;
         2 : _CIRCLE  ;
         3 :  Get_Lib ;
         4 : _LINE    ;
         5 :  TEXT    ;
         6 :  Quit := True ;
         END;
  Until Quit ;
  Quit := False ;
  REDRAW_FUNCTION_NAME ;
END;

PROCEDURE FILE_LIB ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ FILE LIB ]',Color[2]);
  MeNuFile;
  Bar(TEXT_X10,TEXT_MaxY,22*GraphTextWidth,MaxY);
  Case Num Of
       1 : DIRPRO('.LIB');
       2 : LOAD_FILE ;
       3 : SAVE_FILE ;
       4 : Quit := True ;
       END;
  Quit := False ;
  REDRAW_FUNCTION_NAME ;
END;

PROCEDURE MENU ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,30*GraphTextWidth,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ SELECT COMMAND ]',Color[2]);
  Num := 1 ;
  MainMenu;
  Bar(TEXT_X10,TEXT_MaxY,28*GraphTextWidth,MaxY);
  Case Num Of
       1:  Block    ;
       2:  ClearWorkSheet ;
       3: _DELETE   ;
       4:  EDIT     ;
       5:  FILE_LIB ;
       6: _Grid     ;
       7:  JUMP     ;
       8:  FIND_NAME(Libraries_Name);
       9:  REDRAW   ;
      10:  TAG      ;
      11: _ZOOM     ;
      12:  Quit := true ;
      END;
  REDRAW_FUNCTION_NAME ;
END;
```

```
PROCEDURE Help ;
BEGIN
  Bar(TEXT_X10,TEXT_MaxY,MaxX,MaxY);
  TEXT_XY(TEXT_X10,TEXT_MaxY,'[ SELECT FUNCTION KEY ]',Color[2]);
  Num := 1 ;
  HelpMenu;
  Bar(TEXT_X10,TEXT_MaxY,MaxX,MaxY);
  Case Num Of
       1: Edit     ;
       2: Save_File;
       3: Load_File;
       4: Get_Lib  ;
       5: _Zoom    ;
       6: Redraw   ;
       7: Menu     ;
       8: Help     ;
       9: Quit := true ;
       END;
  REDRAW_FUNCTION_NAME ;
END;

PROCEDURE LIBRARY ;
BEGIN
  INITIAL_LIBRARY_HEAD ;
  Repeat
    GETKB ;
    Num := 1 ;
    If FunctionKey then
    CASE Ch OF
    { F1 } #59: EDIT    ;
    { F2 } #60: SAVE_FILE;
    { F3 } #61: LOAD_FILE;
    { F4 } #62: Get_lib  ;
    { F5 } #63: _ZOOM    ;
    { F6 } #64: REDRAW   ;
    { F7 } #65: Menu     ;
    { F8 } #66: Help     ;
    { F9 } #67: Quit := True ;
          End
    Else
    CASE Ch Of
         Enter  : MENU     ;
         'B','b': BLOCK    ;
         'C','c': ClearWorkSheet ;
         'D','d': _DELETE  ;
         'E','e': EDIT     ;
         'F','f': FILE_LIB ;
         'G','g': _GRID    ;
         'h','H': HelpMenu ;
         'J','j': JUMP     ;
         'L','l': FIND_NAME(Libraries_Name);
         'R','r': REDRAW   ;
         'T','t': TAG      ;
```

```
          'Z','z':_ZOOM     ;
          'X','x': Quit := True ;
          END ;
  Until Quit ;
  Quit := False;
  Release(hTop);
END;

BEGIN
END.
```

```
Program : Logcad Logic Node Program
Programmer : Suphan Gulpanich.

{
              +-----------------------------------+
              |                                   |
              | Program  : Logic Node1.           |
              | Programmer : Suphan  Gulpanich.   |
              | Version    : 1.000A               |
              | Last Updated :  4 JANUARY  1988   |
              |                                   |
              +-----------------------------------+
}
PROGRAM Lognode1;

USES Dos;

VAR
  Ok       : Boolean;
  FileName : String ;
  SchFile  : Text   ;
  NodeFile : Text   ;
  NodeName : String ;

FUNCTION OpenFileOK : Boolean ;
BEGIN  { Open File }
  Assign(SchFile,FileName+'.SCH');
  {$I-} Reset(SchFile); {$I+}
  If IOResult = 0 then OpenFileOK := True
                  ELse OpenFileOK := False ;
END;   { Open File }

PROCEDURE MakeNodeFile ;
BEGIN
  Assign(NodeFile,FileName+'.NDE');
  Rewrite(NodeFile);
  Readln(SchFile,NodeName);
  Readln(SchFile,NodeName);
  Readln(SchFile,NodeName);
  While NOT EOF(SchFile) Do
  BEGIN
    Readln(SchFile,NodeName);
    Readln(SchFile,NodeName);
    If Copy(NodeName,7,6) = '(NODE)' then Ok := True
                                     else Ok := False ;
    Readln(SchFile,NodeName);
    Readln(SchFile,NodeName);
    Readln(SchFile,NodeName);
```

```
    NodeName := Copy(NodeName,2,Length(NodeName));
    If Ok then  Writeln(NodeFile,NodeName);
    Readln(SchFile,NodeName);
  END;
END;

BEGIN
  Writeln('LOGNODE  Version 1.00A by SUPHAN GULPANIC. 1 APRIL 1988 ');
  Writeln;
  Write('Enter file name [.SCH] :  ');
  Readln(FileName);
  If OpenFileOK then
  begin
    MakeNodeFile;
    Close(SchFile);
    Close(NodeFile);
    Writeln('Node file created.');
  end
  Else
    Writeln('Cannot open file ['+FileName+'.sch],  Program terminated. ');
END.
```



```
Program : Logcad Simulator Declaration
Programmer : Suphan Gulpanich.

{
              +-----------------------------------+
              |                                   |
              | Unit  : Variable Declaration      |
              | Programmer : Suphan  Gulpanich.   |
              | Version    : 1.000A               |
              | Last Updated :  24 June  1988     |
              |                                   |
              +-----------------------------------+
}
unit Declare; { Declaration Unit }

interface

const { Computing Section }
  MaxNode     = 255;
  MaxStep     = 255;
  MaxProbe    = 20;
  MaxClkData  = 20;
  On     = True;
  Off    = False;
  Left   = #75;
  Right = #77;
  High = True;
  Low  = False;

type
  FileType       = Text;
  SimStepType    = ^SimStepPointer;
  SimStepPointer = record
                     StepNo : array [1..MaxStep] of boolean;
                   end;
  NetworkType    = ^NetworkPointer;
  NetworkPointer = record
                     Name : string[8];
                     Node : 0..MaxNode;
                     Status : boolean;
                   end;
 ProbeType       = record
                     Name : string[8];
                     Node : 0..MaxNode;
                     Status : boolean;
                   end;
 ClkDataType     = record
                     Name : string[8];
```

```
                    Node      : 0..MaxNode;
                    LowState  : word;
                    HighState : word;
                  end;
var
  DataFile : FileType;
  Simulate : array [1..MaxNode] of SimStepType;
  Network  : array [1..MaxNode] of NetworkType;
  Probe    : array [1..MaxProbe] of ProbeType;
  ClkData  : array [1..MaxClkData] of ClkDataType;
  SimStep  : word;
  Step     : word;
  MaxDisplayProbe : byte;
  MaxStepOnScreen : Word;
  MaxGridOnScreen : Word;
  ClkDataNumber   : Word;
  ProbeNumber     : byte;
  FileName        : string;
  _First          : Boolean;
  DataExist       : Boolean;
  FirstClkData    : Boolean;
  LeftStep        : Word;
  StartStep       : Word;
  RightStep       : Word;

procedure Initialize;

implementation

procedure Initialize;
var
  Node : word;
  Count : Word;
begin { Initialize }
  for Node := 1 to MaxNode do
  begin
    New(Simulate[Node]);
    New(Network[Node]);
  end;
  for Count := 1 to MaxClkData do
  begin
    ClkData[Count].Name := '';
    ClkData[Count].Node := 0;
  end;
  for Count := 1 to MaxProbe do
  begin
    Probe[Count].Name := '';
    Probe[Count].Node := 0;
  end;
  _First := On;
  MaxDisplayProbe := 1;
  ClkDataNumber := 0;
  LeftStep  := 1;
  StartStep := 1;
```

```
  RightStep := 1;
  DataExist := False;
  FirstClkData := true;
end;

begin { Unit Declaration }
  Initialize;
end { Unit Declaration }.
```

```
Program : Orcad Logic Node Program
Programmer : Suphan Gulpanich.

{
              +-----------------------------------+
              |                                   |
              | Unit  : Logic Node.               |
              | Programmer : Suphan  Gulpanich.   |
              | Version    : 1.000A               |
              | Last Updated :  24 June  1988     |
              |                                   |
              +-----------------------------------+
}

Program LOGNODE;

uses Dos,Crt;

var
  FileName : string;
  EDIFFile : Text;
  NodeFile : Text;
  NodeName : string;
  ModelCommand : string;
  Number   : word;
  FileReady : Boolean;

function OpenFileOk : boolean;
begin { OpenFile }
  Assign(EDIFFile,FileName + '.NLT');
  {$I-} Reset(EDIFFile); {$I+}
  if IOResult = 0 then
  begin
    OpenFileOk := True;
    Assign(NodeFile,FileName + '.NDE');
    Rewrite(NodeFile);
  end
  else
    OpenFileOk := False;
end { OpenFile };

procedure MakeNodeList;
var
  A,B,C : string;
begin { MakeNodeList }
  Reset(EDIFFile);
  while not EOF(EDIFFile) do
  begin
```

```
    Readln(EDIFFile,A);
    B := Copy(A,1,23);
    if B = '      (define input port' then
    begin
      C := Copy(A,25,Length(A)-26);
      Writeln(NodeFile,C);
    end;
    B := Copy(A,1,24);
    if B = '      (define output port' then
    begin
      C := Copy(A,26,Length(A)-27);
      Writeln(NodeFile,C);
    end;
    B := Copy(A,1,25);
    if B = '      (define local signal' then
    begin
      C := Copy(A,27,Length(A)-27);
      Writeln(NodeFile,C);
    end;
  end;
end { MakeNodeList };

begin
  Clrscr;
  FileReady := False;
  Writeln('LOGNODE Version 1.000A by Suphan Gulpanich. 24 June 1988');
  repeat
    Writeln;
    Write('Enter file name [.NLT]. But Return to EXIT : ');
    Readln(FileName);
    Clrscr;
    if FileName <> '' then
    if OpenFileOk then
    begin
      MakeNodeList;
      Close(EDIFFile);
      Close(NodeFile);
      Writeln('Node File created.');
      FileReady := true;
    end
    else
      Writeln('Cannot open file! program terminated.');
  until  (FileName = '')  or (FileReady = true);
end.
```

Program : Logic Network Program
Programmer : Suphan Gulpanich.

```
{
              +-----------------------------------+
              |                                   |
              | Unit  : Logic Net.                |
              | Programmer : Suphun Gulpanich.    |
              | Version    : 1.00A                |
              | Last Updated :  30 June 1988      |
              |                                   |
              +-----------------------------------+
}

Program LOGNET;

uses Dos,Crt;

var
  FileName : string;
  NodeFile : Text;
  MdlFile  : Text;
  NetFile  : Text;
  NodeName : string;
  Number   : word;
  FileReady : Boolean;

function OpenFileOk : boolean;
begin { OpenFile }
  Assign(NodeFile,FileName + '.NDE');
  {$I-} Reset(NodeFile); {$I+}
  if IOResult = 0 then
  begin
    Assign(MdlFile,FileName + '.MDL');
    {$I-} Reset(MdlFile); {$I+}
    if IOResult = 0 then
    begin
      OpenFileOk := true;
      Assign(NetFile,'DATATXRX.PAS');
      Rewrite(NetFile);
    end
    else
      OpenFileOk := False;
  end
  else
    OpenFileOk := False;
end { OpenFile };
```

```
procedure UpCaseStr(Var Message : String);
Var Buffer : String;
    Name   : String[1];
    Ch     : Char;
    Count  : Word;
begin
  Buffer := '';
  for Count := 1 to Length(Message) do
  begin
    Name := Copy(Message,Count,1);
    Ch := Upcase(Name[1]);
    Buffer := Buffer + Ch;
  end;
  Message := Buffer;
end;

procedure MakeNetList;
Var A,B,C : string;
    Found1 : Boolean;
    Found2 : Boolean;
begin { MakeNetList }
  Number := 1;
  Found1 := False;
  Found2 := False;
  Writeln(NetFile,'unit DataTxRx;');
  Writeln(NetFile);
  Writeln(NetFile,'interface');
  Writeln(NetFile);
  Writeln(NetFile,'uses Crt,Graph,Declare,SimUnit1,SimUnit2;');
  Writeln(NetFile);
  Writeln(NetFile,'var');
  Reset(NodeFile);
  while not EOF(NodeFile) do
  begin
    Readln(NodeFile,NodeName);
    Writeln(NetFile,'   ',NodeName,' : boolean;');
  end;
  Reset(MdlFile);
  while not EOF(MdlFile) do
  begin
    Readln(MdlFile,A);
    B := Copy(A,1,4);
    UpcaseStr(B);
    if Found1 and (B <> 'BEGI') then     { Copy Message After Var }
    begin
      C := Copy(A,5,Length(A)-4);
      Writeln(NetFile,'   ',C);
    end;
    if B = 'VAR 'then
    begin
      C := Copy(A,5,Length(A)-4);
      Writeln(NetFile,'   ',C);
      Found1 := True;
    end;
```

```
    if B = 'BEGI' then
      Found1 := False;
  end;
  Writeln(NetFile,'procedure MoveNodeToNet;');
  Writeln(NetFile,'procedure MoveNetToNode;');
  Writeln(NetFile,'procedure SimulationModel;');
  Writeln(NetFile,'implementation');
  Writeln(NetFile);
  Writeln(NetFile,'procedure MoveNodeToNet;');
  Writeln(NetFile,'begin');
  Reset(NodeFile);
  while not EOF(NodeFile) do
  begin
    Readln(NodeFile,NodeName);
    Writeln(NetFile,'  Network[',Number,']^.Status := ',NodeName,';');
    Inc(Number);
  end;
  Writeln(NetFile,'end;');
  Writeln(NetFile);
  Number := 1;
  Writeln(NetFile,'procedure MoveNetToNode;');
  Writeln(NetFile,'begin');
  reset(NodeFile);
  while not EOF(NodeFile) do
  begin
    Readln(NodeFile,NodeName);
    Writeln(NetFile,'  ',NodeName,' := Network[',Number,']^.Status;');
    Inc(Number);
  end;
  Writeln(NetFile,'end;');
  Writeln(NetFile);
  Writeln(NetFile,'procedure SimulationModel;');
  Reset(MdlFile);
  while not EOF(MdlFile) do    { Model Command begin is Start Write File. }
  begin
    Readln(MdlFile,A);
    B := Copy(A,1,5);
    UpCaseStr(B);
    if Found2 then
      Writeln(NetFile,A)
    else
      if B = 'BEGIN' then
      begin
        Writeln(NetFile,A);
        Found2 := True;
      end;
  end;
  Writeln(NetFile);
  Writeln(NetFile,'begin');
  Number := 1;
  Reset(NodeFile);
  while not EOF(NodeFile) do
  begin
    Readln(NodeFile,NodeName);
```

```
    Writeln(NetFile,'  ',NodeName,' := OFF;');
    Inc(Number);
  end;
  Writeln(NetFile,'end.');
end { MakeNetList };

begin
  Clrscr;
  Writeln('LOGNET Version 1.000A by Suphan Gulphanich. 30 June 1988');
  FileReady := False;
  repeat
    Writeln;
    Write('Enter include file name [.NDE and .MDL]. But Return to EXIT : ');
    Readln(FileName);
    Clrscr;
    if (FileName <> '') and (Not FileReady) then
    if OpenFileOk then
    begin
      MakeNetList;
      Writeln(' DATATXRX.PAS  File  Created.');
      Close(NodeFile);
      Close(NetFile);
      Close(MdlFile);
      FileReady := true;
    end
    else
    Writeln('Cannot open file! program terminated.');
  until (FileName = '') or FileReady;
end.
```

File : B:PRNTIMIN.PAS
Current Date : Sunday September 4, 1988
Current Time : 12:04 PM

Program : Print Timing Diagram Program
Programmer : Suphan Gulpanich.

```
unit PrnTiming;

interface

uses Printer,Declare;

var
  Device : text;

procedure InitialPrinting;

procedure PrintTiming(SimStep,DrawStep : word);

procedure ProbeName;

implementation

const
  Esc  = #27;
  CR   = #$0D;
  LF   = #$0A;
  Period  = 5;
  Adjust  = 9;
var
  n1,n2 : byte;
  Data  : array [1..Adjust,1..MaxProbe] of byte;

procedure InitialPrinting;
begin {InitialPrinting}
  n1 := (MaxDisplayProbe * Adjust * 4) mod 256;
  n2 := (MaxDisplayProbe * Adjust * 4) div 256;
  Assign(Device,'PRN');
  Rewrite(Device);
end {InitialPrinting};

procedure ProbeName;
type  String8  = String[8];
Var   Probe_Name: String8;
      Count     : Byte;
      LengthStr : Byte;

procedure FindProbe;
var  j : Byte;
begin { FindProbe }
  Count := 0;
```

```
  repeat
    Inc(count);
    if Count <= MaxDisplayProbe then
    begin
      LengthStr := Length(Probe[Count].Name);
      Probe_Name := Probe[Count].Name;
      if LengthStr < 8 then
      begin
        for j := LengthStr+1 to 8 do
          Probe_Name := Probe_Name + ' ';
      end;
      Probe[Count].Name := Probe_Name;
    end;
  until (Count >= MaxDisplayProbe);
end; { FindProbe }

procedure PrintLabel;
var CountStr,CountProbe : Byte;
    Buffer : String;
begin
  for CountStr := 1 to 8 do
  begin
    for CountProbe := MaxDisplayProbe downto 1 do
    begin
      Buffer := '';
      if Probe[CountProbe].Name <> ''  then
      begin
        Buffer := Copy(Probe[CountProbe].Name,CountStr,1);
        Write(Lst,Buffer,'  ');
      end;
    end;
    Write(Lst,Esc,'3',Chr(24));  { Set Line Spacing ( Not line Space ) }
    writeln(Lst);
  end;
  Write(Lst,Esc,'2');            { Set to Defalt Line Spacing }
end;

begin { ProbeName }
  FindProbe;
  PrintLabel;
end; { ProbeName }

procedure PrintTiming(SimStep,DrawStep : word);
var
  Node : word;
  ByteCount : byte;
  Direction : (Tail,Lead,RemainLow,RemainHigh);
begin { PrintTiming }
  Write(Device,Esc,'L',Chr(n1),Chr(n2)); { Dual-Density Bit-Image }
  for Node := MaxDisplayProbe downto 1 do
  begin
    if (Simulate[Probe[Node].Node]^.StepNo[SimStep+DrawStep-2] = High)
       and (Probe[Node].Status = Low) then
      Direction := Tail;
```

```
    if (Simulate[Probe[Node].Node]^.StepNo[SimStep+DrawStep-2] = Low)
       and (Probe[Node].Status = High) then
      Direction := Lead;
    if (Simulate[Probe[Node].Node]^.StepNo[SimStep+DrawStep-2] = Low)
       and (Probe[Node].Status = Low) then
      Direction := RemainLow;
    if (Simulate[Probe[Node].Node]^.StepNo[SimStep+DrawStep-2] = High)
       and (Probe[Node].Status = High) then
      Direction := RemainHigh;
    case Direction of
      Lead : begin
               Data[1,Node] := $80;
               Data[2,Node] := $80;
               Data[3,Node] := $80;
               Data[4,Node] := $80;
               Data[5,Node] := $80;
               Data[6,Node] := $80;
               Data[7,Node] := $80;
               Data[8,Node] := $80;
               Data[9,Node] := $FF;
             end;
      Tail : begin
               Data[1,Node] := $FF;
               Data[2,Node] := $80;
               Data[3,Node] := $80;
               Data[4,Node] := $80;
               Data[5,Node] := $80;
               Data[6,Node] := $80;
               Data[7,Node] := $80;
               Data[8,Node] := $80;
               Data[9,Node] := $80;
             end;
      RemainHigh : begin
               Data[1,Node] := $00;
               Data[2,Node] := $00;
               Data[3,Node] := $00;
               Data[4,Node] := $00;
               Data[5,Node] := $00;
               Data[6,Node] := $00;
               Data[7,Node] := $00;
               Data[8,Node] := $00;
               Data[9,Node] := $FF;
             end;
      RemainLow : begin
               Data[1,Node] := $FF;
               Data[2,Node] := $00;
               Data[3,Node] := $00;
               Data[4,Node] := $00;
               Data[5,Node] := $00;
               Data[6,Node] := $00;
               Data[7,Node] := $00;
               Data[8,Node] := $00;
               Data[9,Node] := $00;
             end;
```

```
      end;
      for ByteCount := 1 to Adjust do
      begin
        if (DrawStep - 1) mod Period = 0 then
        begin
          Write(Device,Chr(Data[ByteCount,Node] or $80));
          Write(Device,Chr(Data[ByteCount,Node]));
        end
        else
        begin
          Write(Device,Chr(Data[ByteCount,Node]));
          Write(Device,Chr(Data[ByteCount,Node]));
        end;
      end;
      for ByteCount := 1 to Adjust do
      begin
        if (DrawStep - 1) mod Period = 0 then
        begin
          Write(Device,Chr($80));
          Write(Device,Chr($00));
        end
        else
        begin
          Write(Device,Chr($00));
          Write(Device,Chr($00));
        end;
      end;
    end;
    if DrawStep mod Period = 0 then
      Write(Device,DrawStep:5);
    Write(Device,Esc,'3',Chr(24),CR,LF);      { 24 = 216/8/72 }
  end { PrintTiming };

  begin
  end.
```

```
Program : Simulation Program Module2
Programmer : Suphan Gulpanich.

{
              +-----------------------------------+
              |                                   |
              | Unit  : Logic Simulation2         |
              | Programmer : Suphan Gulpanich.    |
              | Version    : 1.000A               |
              | Last Updated :  26 June 1988      |
              |                                   |
              +-----------------------------------+
}

unit Simunit2;

interface

uses Crt,Graph,Declare,SimUnit1,Printer;

var
      Number   : word;
      OldColor : Word;
      OffsetX0,OffsetX1,OffsetY0,OffsetY1 : Word;
      BlockX0,BlockX1,BlockY0,BlockY1     : Word;
      BufStr,StrBuf1,StrBuf2,StrBuf3      : string;

Const X_UpLeft = 48;
      Y_UpLeft = 22;
      X_LowRight = 89;
      Y_LowRight = 25;
      StatusCol  = 51;
      StatusLine = 24;

Procedure DeleteProbeName;

Procedure CheckKey_NetWork(Buf: String; Var CheckOk : Boolean);

Procedure ClearDisplay;

procedure WriteStepNo(StartStep : word);

procedure DisPlayName;

procedure LabelFile;

procedure MenuBroder(X0,Y0,X1,Y1 : Word; Color : Byte);
```

```
procedure MoveSimToNet(Step : word);

procedure MoveNetToSim(Step : word);

procedure LoadDataToProbe(SimStep : word);

procedure DrawTiming(SimStep,DrawStep : word; TimingColor : byte);

implementation

procedure MenuBroder(X0,Y0,X1,Y1 : Word; Color : Byte);
begin { MenuBroder }
  OldColor := GetColor;
  X0 := X0 * GraphTextWidth; Y0 := Y0 * GraphTextHeight + 5;
  X1 := X1 * GraphTextWidth; Y1 := Y1 * GraphTextHeight + 5;
  SetCrtColor(Color);
  line(X0,Y0,X1,Y0);
  line(X0,Y1,X1,Y1);
  line(X0,Y0,X0,Y1);
  line(X1,Y0,X1,Y1);
  SetCrtColor(OldColor);
end; { MenuBroder }

procedure LabelFile;
begin { LabelFile }
  OldColor := GetColor;
  SetCrtColor(LightBlue);
  if DataExist then
  begin
    DeleteLine(StatusCol,StatusCol + 16,StatusLine);
    WriteTextXY(StatusCol,StatusLine,'FILE : '+FileName);
  end;
  SetCrtColor(OldColor);
end { LabelFile };

procedure DisPlayName;
begin
  OldColor := GetColor;
  SetCrtColor(LightBlue);
  DeleteLine(StatusCol,StatusCol + 6,StatusLine);
  WriteTextXY(StatusCol,StatusLine,'FILE : ');
  WriteTextXY(X_UpLeft + 3,StatusLine + 1,'LEFT STEP : ');
  DeleteLine(X_UpLeft + 16,X_UpLeft + 19,StatusLine + 1);
  Str(LeftStep,StrBuf1);
  WriteTextXY(X_UpLeft + 17,StatusLine + 1,StrBuf1);
  SetCrtColor(LightRed);
  WriteTextXY(X_UpLeft + 29,StatusLine,'STEP : ');
  Str(StartStep,StrBuf2);
  WriteTextXY(X_UpLeft + 36,StatusLine,StrBuf2);
  DeleteLine(X_UpLeft + 22,X_UpLeft + 35,StatusLine + 1);
  WriteTextXY(X_UpLeft + 22,StatusLine + 1,'RIGHT STEP : ');
  DeleteLine(X_UpLeft + 35,X_UpLeft + 40,StatusLine + 1);
  Str(RightStep,StrBuf3);       { Show RightStep }
  WriteTextXY(X_UpLeft + 36,StatusLine + 1,StrBuf3);
```

```
  SetCrtColor(OldColor);
end;

procedure WriteStepNo(StartStep : word);
begin { WriteStepNo }
  StartStep := StartStep;
  DeleteLine(X_UpLeft + 35,X_UpLeft + 40,StatusLine);
  SetCrtColor(LightRed);
  WriteTextXY(X_UpLeft + 29,StatusLine,'STEP : ');
  Str(StartStep,BufStr);
  WriteTextXY(X_UpLeft + 36,StatusLine,BufStr);
end { WriteStepNo };

Procedure ClearDisplay;
begin { ClearDisplay }
  SetGraphMode(GraphMode);
  DrawAxis;
  LabelFile;
  if GridOn then Grid(On);
  SetCrtColor(LightCyan);
  MenuBroder(X_UpLeft,Y_UpLeft,X_LowRight,Y_LowRight,Yellow);
  DisPlayName;
end { ClearDisplay };

Procedure CheckKey_NetWork(Buf: String; Var CheckOk : Boolean);
Var Count : Byte;
begin { CheckKey_NetWork }
  Count := 0;
  CheckOk := False;
  repeat
    Inc(Count);
    if Buf = Network[Count]^.Name then
      CheckOk := True
    else
      CheckOk := False;
  until (Count = MaxNetWork) or CheckOK or (NetWork[Count]^.Name = '');
end; { CheckKey_NetWork }

Procedure DeleteProbeName;
begin
  SetFillPattern(Hidden,Black);
  Bar((XCharOffset-10)*GraphTextWidth,2*GraphTextHeight,
      XCharOffset * GraphTextWidth,20*GraphTextHeight);
end;

procedure LoadDataToProbe(SimStep : word);
begin { LoadDataToProbe }
  for Number := 1 to MaxDisplayProbe do
    Probe[Number].Status := Simulate[Probe[Number].Node]^.StepNo[SimStep];
end { LoadDataToProbe };

procedure DrawTiming(SimStep,DrawStep : word; TimingColor : byte);
var
  Node : word;
```

```
  X0,Y0,X1,Y1 : word;
begin { DrawTiming }
  SetCrtColor(TimingColor);
  X0 := DrawStep * GraphTextHeight + OffsetX0;
  X1 := (DrawStep+1) * GraphTextHeight + OffsetX1;
  for Node := 1 to MaxDisplayProbe do
  begin
    Y0 := Node * (GraphTextHeight) + OffsetY0;
    Y1 := Node * (GraphTextHeight) + OffsetY1;
    if Probe[Node].Status = On then
      Line(X0,Y0,X1,Y0)
    else
      Line(X0,Y1,X1,Y1);
    if DrawStep > 1 then
    begin
      if Simulate[Probe[Node].Node]^.StepNo[SimStep+DrawStep-2] <> Probe[Node].Status then
        Line(X0,Y0,X0,Y1);
    end;
  end;
end { DrawTiming };

procedure MoveSimToNet(Step : word);       { Must be move Input Only }
var
  Number : word;
begin { MoveSimToNet }
  Number := 0;
  repeat
    Inc(Number);
    Network[ClkData[Number].Node]^.Status := Simulate[ClkData[Number].Node]^.StepNo[Step];
  Until (Number = ClkDataNumber) or (Number = MaxClkData);
end { MoveSimToNet };

procedure MoveNetToSim(Step : word);
var
  Node : word;
begin { MoveNetToSim }
  for Node := 1 to MaxNetwork do
  begin
    if Node <> ClkData[Node].Node then
      Simulate[Node]^.StepNo[Step] := Network[Node]^.Status;
  end;
end { MoveNetToSim };

begin
  OffsetX0 := AxisX0-(GraphTextHeight);
  OffsetX1 := AxisX0-(GraphTextHeight);
  OffsetY0 := AxisY0-(GraphTextHeight)+1;
  OffsetY1 := AxisY0-(GraphTextHeight div 2) + 1;
end { SimUnit2 }.
```

```
Program : Logic Simulation Program
Programmer : Suphan Gulpanich.

{
              +-------------------------------------+
              |                                     |
              | Unit  : Simulation.                 |
              | Programmer : Suphan  Gulpanich.     |
              | Version    : 1.000A                 |
              | Last Updated :  24 June  1988       |
              |                                     |
              +-------------------------------------+
}
Program LogicSimulator;

uses SimUnit3;

begin
  LogicSimulation;
end { Main }.
```

File : A:PRESIM.BAT
Current Date : Sunday September 4, 1988
Current Time : 12:08 PM

Program : Pre - Simulation [Batch File]
Programmer : Suphan Gulpanich.

```
echo off
\tpc datatxrx
\tpc simunit3
\tpc logsim
echo on
```

```
Program : Logic Simulation Module1
Programmer : Suphan Gulpanich.

{
              +-----------------------------------+
              |                                   |
              | Unit  : Simulation I              |
              | Programmer : Suphun Gulpanich.    |
              | Version    : 1.00A                |
              | Last Updated :  24 June 1988      |
              |                                   |
              +-----------------------------------+
}
Unit SimUnit1;

interface

uses Crt,Graph,Declare;

const
  Hidden : FillPatternType = ($FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF);
  XCharOffset = 10; { x,y lines offset from upper left corner }
  YCharOffset = 2;

var
  GraphDriver : integer;  { The Graphics device driver }
  GraphMode   : integer;  { The Graphics mode value }
  MaxX, MaxY  : word;     { The maximum resolution of the screen }
  MaxColor    : word;     { The maximum color value available }
  ViewPort : ViewPortType;
  GraphTextWidth : byte;
  GraphTextHeight : byte;
  AxisX0,AxisY0,AxisX1,AxisY1 : word;
  Command : string;
  GridOn  : boolean;
  MenuSelect : byte;
  MaxNetwork : word;
  NetFile : FileType;
  WindowX0 : word;
  WindowY0 : word;
  WindowX1 : word;
  WindowY1 : word;
  TextWindow : pointer;
  Ch : Char;

Procedure SetCrtColor (Color : byte);

procedure WriteTextXY (x,y : byte; Str : string);
```

```
procedure ReadTextXY (x,y : word; NoChar : Byte; var Str : string);

procedure DrawAxis;

procedure SelectMenu;

procedure Grid (Status : boolean);

procedure DeleteLine (X0,X1,Y : byte);

procedure MenuBlock (MenuBlockOn : boolean);

procedure SaveScreen;

procedure LoadScreen;

function OpenNetFileOk : boolean;

procedure ReadData;

procedure WriteProbeName;

function NodeOf (ProbeName : string) : word;

procedure OpenWindow (X0,Y0,X1,Y1 : byte);

procedure CloseWindow;

procedure ClearWindow;

implementation

var
  ErrorCode   : integer;  { Reports any graphics errors }
  Buffer : pointer;
  Size : word;
  Number : word;

Const
  ScrollStatusCol = 20;

Procedure SetCrtColor (Color : byte);
Begin
  Case GraphDriver of
    EGA : SetColor(Color);
    HercMono : Begin
                 if Color = Black then
                   Color := Color
                 else
                   Color := White;
                 SetColor (Color);
               end;
  end; { Case }
```

```
end; { SetCrtColor }

procedure InitialGraphics;
begin { InitialGraphics }
{ Initialize graphics and report any errors that may occur }
{ when using Crt and graphics, turn off Crt's memory-mapped writes }
  DetectGraph (GraphDriver,GraphMode);
  case GraphDriver of
     EGA : begin
             DirectVideo := True;
             GraphMode := EGAHi;
             MaxStepOnScreen := 42;
             MaxGridOnScreen := 42;
           end;
HercMono : begin
             DirectVideo := True;
             GraphMode := HercMonoHi;
             MaxStepOnScreen := 42;
             MaxGridOnScreen := 42;
           end;
  end { case };
  InitGraph(GraphDriver,GraphMode, ''); { activate graphics }
  SetGraphMode(GraphMode);
  ErrorCode := GraphResult;             { error? }
  if ErrorCode <> grOk then
  begin
    Writeln('Graphics error: ', GraphErrorMsg(ErrorCode));
    Halt(1);
  end;
  MaxColor := GetMaxColor; { Get the maximum allowable drawing color }
  MaxX := GetMaxX;         { Get screen resolution values }
  MaxY := GetMaxY;
  SetCrtColor(MaxColor);
  SetLineStyle(SolidLn, 0, NormWidth);
  SetViewPort(0,0,MaxX,MaxY,ClipOn);
  GetViewSettings(ViewPort);
  case GraphDriver of
    EGA      : begin
                 GraphTextHeight := MaxY div 25;
               end;
    HercMono : begin
                 GraphTextHeight := MaxY div 25;
               end;
  end { case };
  GraphTextWidth := MaxX div 80;
end { InitialGraphics };

procedure WriteTextXY (x,y : byte; Str : string);
begin { WriteTextXY }
  Dec (X);
  Dec (Y);
  OutTextXY(X * GraphTextWidth,Y * GraphTextHeight,Str);
end { WriteTextXY };
```

```
procedure ReadTextXY (x,y : word; NoChar : Byte; var Str : string);
var
  OldColor : word;
begin { ReadTextXY }
  OldColor := GetColor;
  Str := '';
  Dec(x);
  Dec(y);
  x := x * GraphTextWidth;
  y := y * GraphTextHeight;
  SetCrtColor(White);
  OutTextXY(x,y,'_');
  Ch := #0;
  repeat
    if not KeyPressed then
    begin
      { User Task }
    end
    else
    begin  { else KeyPressed }
      Ch := ReadKey;
      Ch := UpCase(Ch);
      if (Ch = ^H) and (Length(Str) > 0) then
      begin
        Delete(Str,Length(Str),1);
        Dec(x,GraphTextWidth);
        SetFillPattern(Hidden,Black);
        Bar(x,y,x+GraphTextWidth,y+GraphTextHeight);
        SetCrtColor(Black);
        OutTextXY(x+GraphTextWidth,y,'_');
        SetCrtColor(White);
        OutTextXY(x,y,'_');
      end
      else
      begin
        if (Ch <> ^H) and (Ch <> #$0D) then
          begin
            case NoChar of
                3 : begin
                      if Length(Str) <= 2 then
                      begin
                        Str := Str + Ch;
                        SetCrtColor(Black);
                        OutTextXY(x,y,'_');
                        SetCrtColor(OldColor);
                        OutTextXY(x,y,Ch);
                        SetCrtColor(White);
                        OutTextXY(x+GraphTextWidth,y,'_');
                        Inc(x,GraphTextWidth);
                      end;
                    end;
                8 : begin
                      if Length(Str) <= 7 then
                      begin
```

```
                        Str := Str + Ch;
                        SetCrtColor(Black);
                        OutTextXY(x,y,'_');
                        SetCrtColor(OldColor);
                        OutTextXY(x,y,Ch);
                        SetCrtColor(White);
                        OutTextXY(x+GraphTextWidth,y,'_');
                        Inc(x,GraphTextWidth);
                      end;
                    end;
                12 : begin
                      if Length(Str) <= 11 then
                      begin
                        Str := Str + Ch;
                        SetCrtColor(Black);
                        OutTextXY(x,y,'_');
                        SetCrtColor(OldColor);
                        OutTextXY(x,y,Ch);
                        SetCrtColor(White);
                        OutTextXY(x+GraphTextWidth,y,'_');
                        Inc(x,GraphTextWidth);
                      end;
                    end;
              end;  { Case }
          end;
      end;
    end; { else KeyPressed }
  until Ch = #$0D;
  SetCrtColor(Black);
  OutTextXY(x+GraphTextWidth,y,'_');
  SetCrtColor(OldColor);
end { ReadTextXY };

procedure ErrorBeep;
begin { ErrorBeep }
  Sound(2000);
  Delay(1000);
  NoSound;
end { ErrorBeep };

procedure CursorBeep;
begin { CursorBeep }
  Sound (5000);
  Delay (100);
  NoSound;
end { CursorBeep };

procedure SelectMenuBeep;
begin { SelectMenuBeep }
  Sound (3000);
  Delay (100);
  NoSound;
end { SelectMenuBeep };
```

```
procedure Title;
var
  Count : byte;
  PassCode : boolean;

procedure GraphWritelnCenter (Y : byte; Str : string);
begin { GraphWritelnCenter }
  OutTextXY (MaxX div 2,Y * GraphTextHeight,Str);
end { GraphWritelnCenter };

begin { Title }
  PassCode := true;
  with ViewPort do
    Rectangle(0, 0, x2-x1, y2-y1);
  SetTextJustify(CenterText,CenterText);
  GraphWritelnCenter (3,'KMIT LADKRABANG');
  GraphWritelnCenter (5,'Faculty of Engineering');
  GraphWritelnCenter (7,'Department of Industial Instrumentation Technology');
  GraphWritelnCenter (10,'Logic Circuit Simulation Program');
  GraphWritelnCenter (12,'L O G C A D');
  GraphWritelnCenter (14,'Version 1.000A');
  GraphWritelnCenter (18,'Advisor Assc. Prof. Dr. Sittichai POOKAIYAUDOM');
  GraphWritelnCenter (21,'Suphan GULPANICH');
  GraphWritelnCenter (23,'Last Updated : 15 July 1988');
  for Count := 1 to 6 do
  begin
    Ch := ReadKey;
    case Count of
      1 : if Upcase(Ch) <> 'L' then PassCode := False;
      2 : if Upcase(Ch) <> 'O' then PassCode := False;
      3 : if Upcase(Ch) <> 'G' then PassCode := False;
      4 : if Upcase(Ch) <> 'C' then PassCode := False;
      5 : if Upcase(Ch) <> 'A' then PassCode := False;
      6 : if Upcase(Ch) <> 'D' then PassCode := False;
    end;
  end;
  if PassCode = False then Halt;
end { Title };

procedure ReadData;
begin { ReadData }

  { Read NetList File }

  Number := 1;
  while not EOF(NetFile) do
  begin
    Readln(NetFile,Network[Number]^.Name);
    Network[Number]^.Node := Number;
    Inc(Number);
  end;
  Close(NetFile);
  MaxNetwork := Number - 1;
  DataExist := True;
```

```
end { ReadData };

procedure Grid (Status : boolean);
var
  ProbePixel,StepPixel : word;
  Count,CountX,CountY : byte;
begin { Grid }
  GridOn := Status;
  ProbePixel := AxisY0;
  CountY := 1;
  while CountY <= MaxProbe do
  begin
    StepPixel := AxisX0;
    CountX := 0;
    Count := 0;
    while CountX <= MaxGridOnScreen do
    begin
      if GridOn then
      begin
        if Count = 5 then
        begin
          PutPixel (StepPixel,ProbePixel,White);
          Count := 0;
        end
        else
          PutPixel (StepPixel,ProbePixel,LightBlue);
      end
      else
        PutPixel (StepPixel,ProbePixel,Black);
      Inc(StepPixel,GraphTextHeight);
      Inc(CountX);
      Inc(Count);
    end;
    Inc(ProbePixel,GraphTextHeight);
    Inc(CountY);
  end;
end { Grid };

procedure DrawAxis;
begin { DrawAxis }
  AxisX0 := GraphTextWidth * XCharOffset + GraphTextWidth div 2;
  AxisY0 := GraphTextHeight * YCharOffset;
  AxisX1 := GraphTextHeight * MaxStepOnScreen + AxisX0;
  AxisY1 := MaxProbe * GraphTextHeight + AxisY0;
  SetCrtColor (LightGreen);
  Line (AxisX0,AxisY0,AxisX0,AxisY1);
  Line (AxisX0,AxisY1,AxisX1,AxisY1);
end { DrawAxis };

procedure WriteProbeName;
begin { WriteProbeName }
  SetTextJustify(RightText,TopText);
  SetCrtColor(LightCyan);
  for Number := 1 to MaxDisplayProbe do
```

```
    WriteTextXY (XCharOffset + 1,Number + 2,Probe[Number].Name);
  SetTextJustify(LeftText,TopText);
end { WriteProbeName };

function NodeOf (ProbeName : string) : word;
var
  Found : boolean;
  NodeNumber : word;
begin { NodeOf }
  NodeNumber := 0;
  Found := False;
  repeat
    Inc(NodeNumber);
    if ProbeName = Network[NodeNumber]^.Name then
      Found := True;
  until Found or (NodeNumber = MaxNetwork);
  if Found then
    NodeOf := Network[NodeNumber]^.Node
  else
    NodeOf := 0;
end { NodeOf };

procedure SaveScreen;
begin { SaveScreen }
  GetImage(AxisX0,AxisY0,AxisX1,AxisY1,Buffer^);
end { SaveScreen };

procedure LoadScreen;
begin { LoadScreen }
  PutImage(AxisX0,AxisY0,Buffer^,NormalPut);
end { LoadScreen };

procedure OpenWindow (X0,Y0,X1,Y1 : byte);
begin { OpenWindow }
  Dec(X0);
  Dec(Y0);
  WindowX0 := X0 * GraphTextWidth;
  WindowY0 := Y0 * GraphTextHeight;
  WindowX1 := X1 * GraphTextWidth;
  WindowY1 := Y1 * GraphTextHeight;
  GetImage(WindowX0,WindowY0,WindowX1,WindowY1,TextWindow^);
  SetFillPattern(Hidden,Black);
  Bar(WindowX0,WindowY0,WindowX1,WindowY1);
  SetCrtColor(LightRed);
  Rectangle(WindowX0,WindowY0,WindowX1,WindowY1);
end { OpenWindow };

procedure CloseWindow;
begin { CloseWindow }
  PutImage(WindowX0,WindowY0,TextWindow^,NormalPut);
end { CloseWindow };

procedure ClearWindow;
begin { ClearWindow }
```

```
  SetFillPattern(Hidden,Black);
  Bar(WindowX0+1,WindowY0+1,WindowX1-1,WindowY1-1);
end { ClearWindow };

function OpenNetFileOk : boolean;
begin { OpenNetFileOk }
  Assign(NetFile,FileName);
  {$I-} Reset(NetFile); {$I+}
  OpenNetFileOk := (IOResult = 0);
end { OpenNetFileOk };

procedure DeleteLine (X0,X1,Y : byte);
begin { DeleteLine }
  Dec(Y);
  SetFillPattern(Hidden,Black);
  Bar (X0 * GraphTextWidth,Y * GraphTextHeight,
       X1 * GraphTextWidth,Y * GraphTextHeight + 10);
end { DeleteLine };

procedure MenuBlock (MenuBlockOn : boolean);
begin { MenuBlock }
  if MenuBlockOn then
    SetCrtColor(White)
  else
    SetCrtColor(Black);
  Line((MenuSelect - 1) * 48 + 7,23 * GraphTextHeight + 9,
          (MenuSelect - 1) * 48 + 45,23 * GraphTextHeight + 9);
  SetCrtColor(LightCyan);
end { MenuBlock };

procedure SelectMenu;
begin { SelectMenu }
  repeat
    Ch := ReadKey;
    if Ch = #0 then
    begin
      Ch := ReadKey;
      if Ch in [Left,Right] then
        MenuBlock(Off);
      case Ch of
        Left : begin
                 if MenuSelect > 1 then
                   Dec(MenuSelect)
                 else
                   MenuSelect := 6;
               end;
       Right : begin
                 if MenuSelect < 6 then
                   Inc(MenuSelect)
                 else
                   MenuSelect := 1;
               end;
      end { case };
      CursorBeep;
```

```
      MenuBlock(On);
    end;
  until Ch in [#$0D,' '];
  SelectMenuBeep;
end { SelectMenu };

begin { unit Simunit1 }
  InitialGraphics;
  Title;
  MenuSelect := 1;
  MaxNetwork := 1;
  SetGraphMode(GraphMode);
  DrawAxis;
  GetMem(TextWindow,20000);
  Grid (On);
end { unit Simunit1 }.
```