บทที่ 5

ซอฟท์แวร์ควบคุมระบบของ PC

ในบทนี้จะกล่าวถึงรายละเอียดของการพัฒนาโปรแกรมซอฟท์แวร์ควบคุมระบบของ PC ภาษาที่ใช้ในการเขียนโปรแกรมพัฒนาซอฟท์แวร์จะใช้ภาษาแอสเซมบลี(Assembly Language) ซึ่งเป็นภาษาที่เหมาะสมที่จะนำมาใช้พัฒนาระบบที่ต้องการความเร็ว เพราะเป็นภาษาที่ซีพียูสามารถทำงานได้เร็วที่สุด การเขียนโปรแกรมจะเขียนในลักษณะเป็นโมดูล (Module) เพื่อให้ง่ายต่อการเพิ่มเติม และแก้ไข

การพัฒนาโปรแกรมควบคุมการทำงานของ PC จะแบ่งออกเป็น 2 ส่วน คือ โปรแกรมหลัก และโปรแกรมจัดการตัวป้อนโปรแกรม (Programming Console)

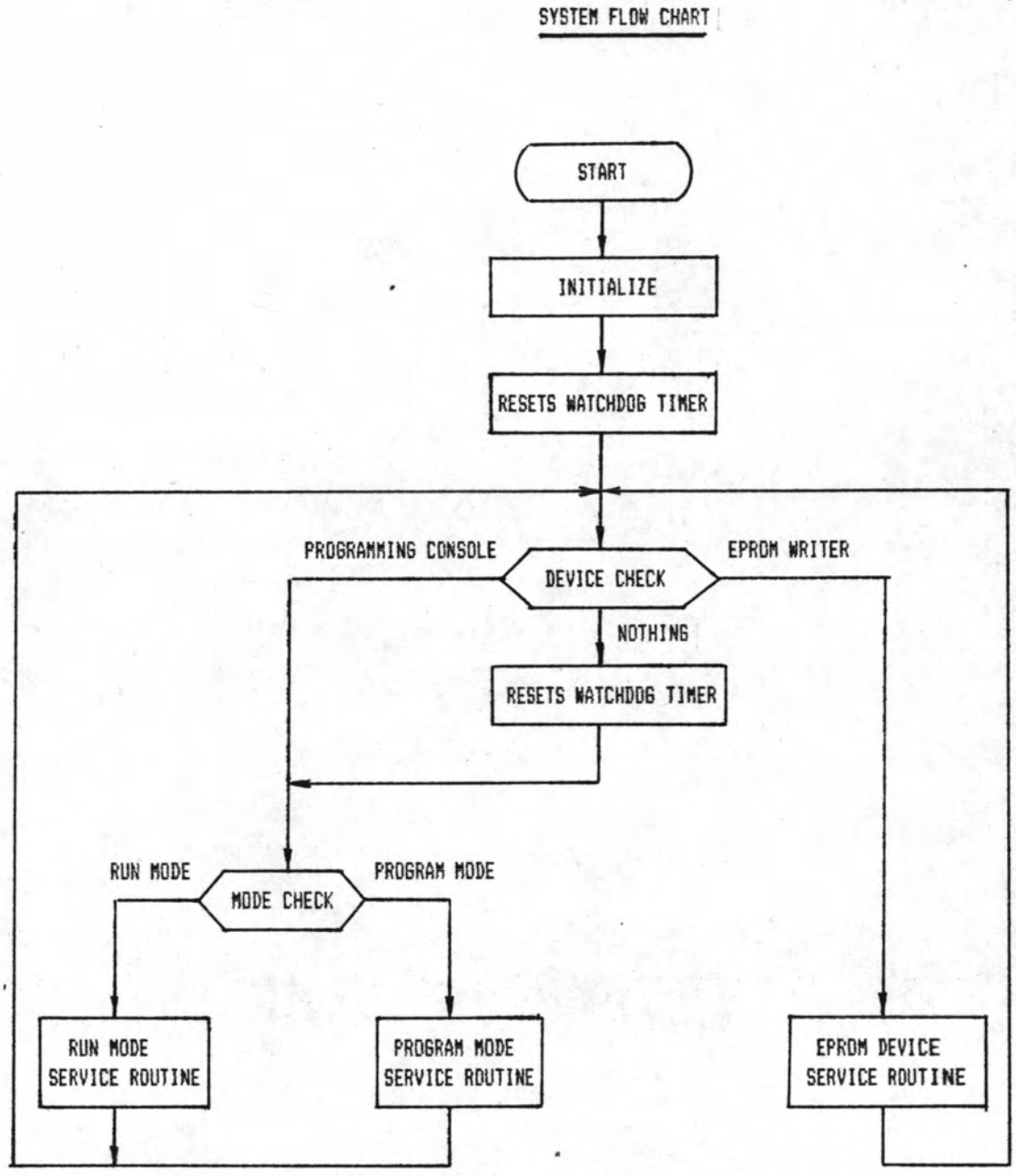
5.1 โปรแกรมหลัก

ทำหน้าที่ควบคุมการทำงานทางฮาร์ดแวร์ของ BASIC UNIT หน้าที่หลักจะทำการตรวจอุปกรณ์ PERIPHERAL ที่จะนำมาใช้กับ PC เช่น ตัวป้อนโปรแกรมและตัวอัดโปรแกรม เป็นต้น นอกจากนี้ยังแบ่งการทำงานของ PC ออกเป็น 2 โหมดคือ โหมดการโปรแกรมและโหมดการทำงาน โปรแกรมหลักจะทำการตรวจเช็คเพื่อทำโปรแกรมบริการตามอุปกรณ์ที่นำมาต่อและโหมดที่ตั้งไว้ โดยจะวนทำเป็นลูป (Loop) ตลอดเวลา ดังแสดงในรูปที่ 5.1

จากโฟล์วชาร์ทของโปรแกรมหลัก ประกอบด้วยองค์ประกอบดังนี้

INITIALIZE เป็นโปรแกรมที่ใช้จัดการเกี่ยวกับฮาร์ดแวร์ดังนี้

1. เคลียร์สภาวะของรีเลย์ภายใน ตัวตั้งเวลา ตัวนับ และแฟล็กต่าง ๆ ภายใน RAM
2. ทำการโปรแกรมเซ็ทพอร์ทของ LSI ที่ใช้เป็น I/O ภายใน BASIC UNIT เอง
3. เช็คโหมดการทำงานว่าให้ทำโปรแกรมผู้ใช้ที่อยู่ภายในหน่วย



รูปที่ 5.1 แสดงโฟล์วชาร์ทของโปรแกรมหลัก

ความจำ EPROM หรือไม่ ถ้ามี EPROM อยู่จะทำการย้ายโปรแกรมผู้ใช้ใน EPROM มาไว้ภายใน RAM ที่ใช้เก็บโปรแกรมผู้ใช้แทน โดยจะย้ายโปรแกรมเมื่อโปรแกรมใน EPROM มีแบบฟอร์มของคำสั่งถูกต้องแล้วเท่านั้น ถ้ามีแบบฟอร์มของคำสั่งที่ผิดไปเครื่องจะแสดง ERROR เพื่อแจ้งให้ผู้ใช้ทราบ ผู้ใช้จะต้องปิดเครื่องและดึง EPROM ออกจากก่อนเครื่องจึงจะทำงานต่อไปได้ มิฉะนั้นเครื่องจะไม่รับคำสั่งใด ๆ ทั้งสิ้น

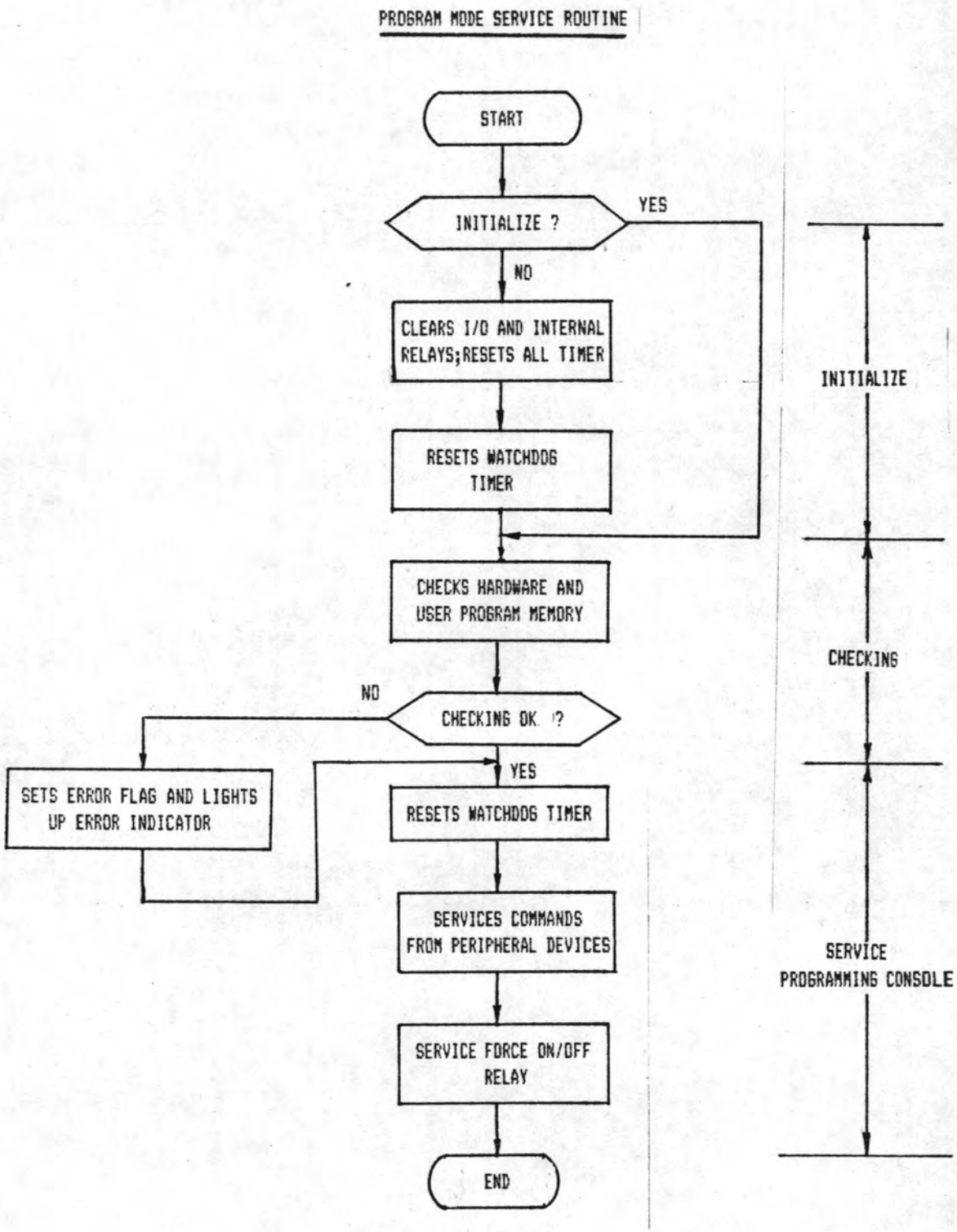
WATCHDOG TIMER — เป็นโปรแกรมที่ใช้ส่งสัญญาณไปรีเซ็ทวงจร WATCHDOG TIMER โดยการดิสชาร์จประจุภายใน WATCHDOG TIMER เพื่อมิให้ถึงจุดที่ WATCHDOG TIMER สามารถทำงานได้

DEVICE CHECK — เป็นโปรแกรมที่ใช้ตรวจสอบอุปกรณ์ PERIPHERAL ที่นำมาใช้กับ PC เพื่อทำโปรแกรมบริการตามชนิดของอุปกรณ์ นั้น ๆ ในวิทยานิพนธ์นี้จะเขียนโปรแกรมบริการเฉพาะ ตัวป้อนโปรแกรมเท่านั้น

MODE CHECK — เป็นโปรแกรมที่ใช้ตรวจเช็คโหมดการทำงานของ PC โดยแบ่งโหมดการทำงานเป็น 2 โหมดคือโหมดโปรแกรม และโหมดทำงาน ซีพียู จะทำโปรแกรมบริการตามโหมดที่ผู้ใช้เครื่องได้ตั้งไว้

5.1.1 โปรแกรมบริการโหมดโปรแกรม

ในโหมดโปรแกรมผู้ใช้สามารถป้อนโปรแกรมขั้นบันไดลงไปในหน่วยความจำของ PC และสามารถทำการแก้ไขโปรแกรมตามต้องการ เช่น การแก้ไขคำสั่งหรือเบอร์ โอเปอแรนด์ (Operand) ในแต่ละขั้นโปรแกรม (Step) การทำงาน การลบคำสั่งและการสอดแทรกคำสั่ง เป็นต้น นอกจากนี้ยังสามารถหาตำแหน่ง(Search) ของคำสั่งว่าอยู่ในขั้นโปรแกรม (Step) ที่เท่าใดบ้างในโปรแกรมและสามารถเซทหรือรีเซทสภาวะการทำงานของรีเลย์ภายในให้เปิดหรือปิดได้อีกด้วยโฟล์วชาร์ทการทำงานของโหมดโปรแกรมแสดงดังรูปที่ 5.2



รูปที่ 5.2 แสดงโฟล์วชาร์ทการทำงานของโหมดโปรแกรม

องค์ประกอบของโปรแกรมในโหมดโปรแกรมมีดังนี้

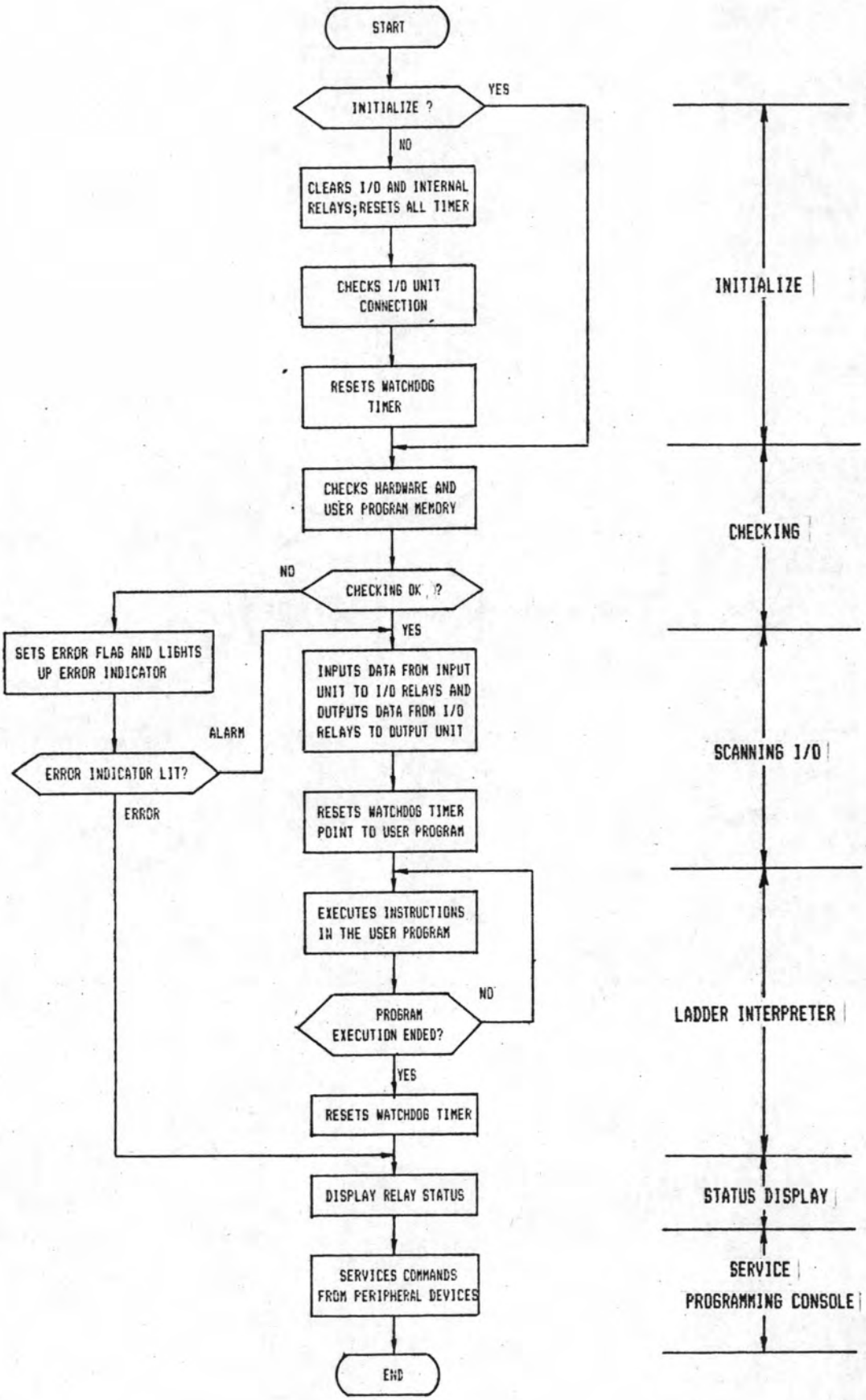
INITIALIZE — ซีพียูจะทำการเคลียร์สภาวะของรีเลย์ ตัวตั้งเวลา ตัวนับ และแฟล็กต่าง ๆ พร้อมทั้งส่งสัญญาณไปรีเซท WATCHDOG TIMER

CHECKING — เป็นโปรแกรมเพื่อเช็คฮาร์ดแวร์ และซอฟท์แวร์ ทางฮาร์ดแวร์ ซีพียูจะทำการตรวจเช็คระดับแรงดันไฟของแหล่งจ่ายไฟสำรอง (Battery) เมื่อตรวจพบว่าระดับแรงดันไฟของแหล่งจ่ายไฟสำรองต่ำกว่าระดับที่จะทำให้วงจรสามารถทำงานได้ถูกต้อง ซีพียูจะส่งสัญญาณไปบอกผู้ใช้ให้ทำการเปลี่ยนแหล่งจ่ายไฟสำรอง ส่วนทางซอฟท์แวร์ ซีพียูจะทำการตรวจสอบแบบฟอร์มของคำสั่งในโปรแกรมขั้นบันไดของผู้ใช้ ที่เก็บไว้ในหน่วยความจำ ถ้าตรวจพบว่าโปรแกรมที่เก็บไว้ในหน่วยความจำมีรูปแบบที่ผิดไปจากที่ควรจะเป็น ซีพียูจะส่งสัญญาณ ERROR บอกผู้ใช้ให้ทราบ ผู้ใช้จะต้องเคลียร์โปรแกรมผู้ใช้ทิ้งก่อนแล้วจึงป้อนโปรแกรมเข้าไปใหม่

SERVICE PROGRAMMING CONSOLE — เป็นโปรแกรมเพื่อทำการตรวจเช็คการกดคีย์จากตัวป้อนโปรแกรมของผู้ใช้ เมื่อมีการกดคีย์จะส่งโค๊ดของคีย์ที่กดไปยังโปรแกรมจัดการตัวป้อนโปรแกรมเพื่อบริการอีกทีหนึ่ง ในโหมดนี้สามารถใช้คำสั่งควบคุม (Command) ได้ทุกคำสั่ง เช่น การอ่าน การเขียน การลบออก การสอดแทรก การค้นหาคำสั่ง และการมอนิเตอร์ เป็นต้น นอกจากนี้ยังสามารถสั่งให้หน้าสัมผัสของรีเลย์ทำงานเปิดและปิดเพื่อทดสอบการทำงานของเครื่องจักรที่ต้องการควบคุมได้อีกด้วย

5.1.2 โปรแกรมบริการโหมดทำงาน

ในโหมดทำงานซีพียูจะทำโปรแกรมขั้นบันไดโดยใช้โปรแกรมตัวแปลคำสั่ง (Ladder Interpreter) วิธี AUTO INCREMENT ดังได้กล่าวมาแล้วในบทที่ 3 ในโหมดนี้สามารถตรวจเช็คมอนิเตอร์สภาวะการทำงานของรีเลย์ภายในได้ ซึ่งทำให้การตรวจซ่อมและแก้ไขโปรแกรมทำได้ง่าย โฟล์วชาร์ทการทำงานของโหมดทำงานดังแสดงในรูปที่ 5.3



รูปที่ 5.3 แสดงโฟล์วชาร์ทการทำงานของโหมดทำงาน

องค์ประกอบของโปรแกรมในโหมดทำงานมีดังนี้

INITIALIZE — โปรแกรมจะเคลียร์สภาวะของรีเลย์ ตัวตั้งเวลา และแฟล็กต่าง ๆ นอกจากนี้ยังทำหน้าที่เช็คการต่อ I/O EXPANSION UNIT ถ้าพบว่ามีการต่อจะทำการโปรแกรมพอร์ทของ IC ที่ใช้เป็น I/O ให้เป็นอินพุทและเอาท์พุทตามต้องการ

CHECKING — เป็นโปรแกรมที่ใช้ในการตรวจเช็คฮาร์ดแวร์ และซอฟท์แวร์ ทางฮาร์ดแวร์จะทำการตรวจเช็คระดับแรงดันของแหล่งจ่ายไฟสำรองเช่นเดียวกับในโหมดโปรแกรม ส่วนทางซอฟท์แวร์จะมีการตรวจสอบแบบฟอร์มของคำสั่งในโปรแกรมขั้นบันได เช่นเดียวกับในโหมดโปรแกรม นอกจากนี้ก่อนที่จะทำโปรแกรมขั้นบันไดยังตรวจสอบหาคำสั่ง END ในโปรแกรมขั้นบันไดก่อนอีกด้วย ถ้าไม่พบคำสั่ง END เครื่องจะไม่ทำงานและแสดงผลให้ผู้ใช้ทราบเพื่อจะได้สอดแทรกคำสั่ง END ลง ในโปรแกรม

SCANNING I/O — เป็นโปรแกรมที่ใช้อ่านอินพุทเข้ามาเก็บไว้ในหน่วยความจำข้อมูล (Data Memory) ในส่วนพื้นที่ของอินพุท และจะนำค่าสภาวะของเอาท์พุทที่เก็บไว้ในหน่วยความจำข้อมูล ในส่วนพื้นที่ของเอาท์พุทออกไปยังอุปกรณ์เอาท์พุท เพื่อใช้ควบคุมการทำงานของเครื่องจักรภายนอก

LADDER INTERPRETER — เป็นโปรแกรมที่ใช้ในการแปลความหมายของคำสั่งโปรแกรมขั้นบันไดของผู้ใช้ที่เก็บในหน่วยความจำผู้ใช้ โปรแกรมขั้นบันไดจะถูกอ่านออกมาทีละขั้น เพื่อแปลความหมายแล้วจึงทำตามคำสั่งที่แปลออกมา จนกระทั่งพบคำสั่ง END จึงหยุดทำโปรแกรม ดังได้กล่าวมาแล้ว

RELAY STATUS DISPLAY — โปรแกรมส่วนนี้ใช้แสดงสภาวะการทำงานของ หน้าสัมผัสของรีเลย์ตามคำสั่งที่ปรากฏบนตัวป้อนโปรแกรม เมื่อตัวป้อนโปรแกรมอยู่ในโหมดคำสั่งควบคุม การอ่าน เท่านั้น เช่น

รีเลย์เบอร์ "000" มีสภาวะการทำงานจริงเปิดอยู่ แต่คำสั่งที่ปรากฏบนตัวป้อนโปรแกรม เป็นคำสั่ง NOT ดังนั้นตัวป้อนโปรแกรมจะแสดงสภาวะของรีเลย์เป็นสภาวะปิด เป็นต้น

SERVICE PROGRAMMING CONSOLE — มีหน้าที่เช่นเดียวกับในโหมดโปรแกรม แต่มีคำสั่งควบคุมที่ไม่สามารถใช้ในโหมดนี้ได้ เช่น คำสั่งควบคุม การเขียน การสอดแทรก การลบออก และการค้นหาคำสั่ง เป็นต้น

5.1.3 โปรแกรมอินเตอร์รัพท์

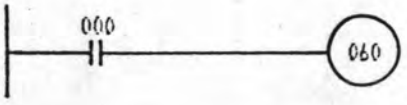
โปรแกรมนี้จะทำงานเฉพาะในโหมดการทำงานเท่านั้น โดยจะอินเตอร์รัพท์ทุก ๆ 100 msec. โปรแกรมจะทำการเช็คว่าตัวตั้งเวลาตัวใดทำงานอยู่ เมื่อถึงเวลาจะทำการลดค่าของตัวตั้งเวลานั้นลงทีละ 1 จนกระทั่ง ค่าของตัวตั้งเวลานั้น ๆ เป็นศูนย์

5.2 การทำงานของคำสั่ง (OPERATION OF INSTRUCTION)

PC ที่ได้ออกแบบนี้สามารถทำการควบคุมซีเควันซ์โดยมีคำสั่งเบื้องต้นที่จำเป็นในการควบคุมดังกล่าวแล้วในบทที่ 2 ในบทนี้จะกล่าวถึงการทำงานของรีจีสเตอร์ภายในของแต่ละคำสั่ง ตัวอย่างการแปลงวงจรขั้นบันได เพื่อเป็นโปรแกรมคำสั่ง (Mnemonic Code)ที่ PC สามารถเข้าใจ การทำงานของแต่ละคำสั่งมีดังนี้

5.2.1 LOAD (LD) และ OUTPUT (OUT) INSTRUCTIONS

จะใช้คำสั่ง LD เมื่อเริ่มต้นด้วยหน้าสัมผัสชนิด NO (NORMALLY OPEN) ในแต่ละวงจรย่อย และใช้คำสั่ง OUT สำหรับคอยล์ของรีเลย์ ดังแสดงในรูปที่ 5.4



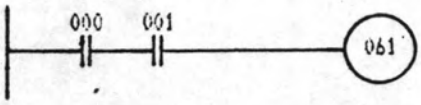
CODING

STEP	OPCODE	OPERAND
000	LD	000
001	OUT	060

รูปที่ 5.4 แสดงการใช้คำสั่ง LOAD และคำสั่ง OUT

5.2.2 AND INSTRUCTION

ใช้คำสั่ง AND เมื่อใช้หน้าสัมผัสชนิด NO ต่ออนุกรม ดังแสดงในรูปที่ 5.5



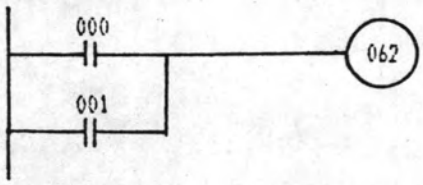
CODING

STEP	OPCODE	OPERAND
002	LD	000
003	AND	001
004	OUT	061

รูปที่ 5.5 แสดงการใช้คำสั่ง AND

5.2.3 OR INSTRUCTION

ใช้คำสั่ง OR เมื่อใช้หน้าสัมผัสชนิด NO ต่อขนาน ดังแสดงในรูปที่ 5.6



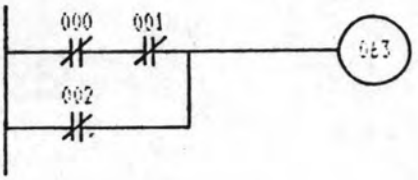
CODING

STEP	OPCODE	OPERAND
005	LD	000
006	OR	001
007	OUT	062

รูปที่ 5.6 แสดงการใช้คำสั่ง OR

5.2.4 LOAD NOT (LD-NOT), AND NOT และ OR NOT INSTRUCTIONS

การใช้คำสั่งนี้มีวิธีใช้เช่นเดียวกับที่กล่าวในข้างต้น โดยใช้เมื่อหน้าสัมผัสเป็นชนิด NC (Normally Close) ดังแสดงในรูปที่ 5.7



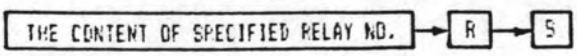
CODING

STEP	OPCODE	OPERAND
008	LD-NOT	000
009	AND-NOT	001
010	OR-NOT	002
011	OUT	063

รูปที่ 5.7 แสดงการใช้คำสั่ง LOAD-NOT, AND-NOT และ OR-NOT

การทำงานของรีเจสเตอร์ภายใน (Operation of Internal Register)

LD -- การใช้คำสั่ง LD จะทำให้ค่าสภาวะของรีเลย์ที่ระบุเบอร์ในคำสั่ง (Operand) จะถูกนำไปเก็บไว้ใน RESULT REGISTER หรือเรียกย่อ ๆ ว่า "R register" และค่าสภาวะเดิมของรีเลย์ ซึ่งแต่เดิมเก็บไว้ใน R register จะถูกย้ายไปเก็บไว้ใน STACK REGISTER หรือเรียกย่อ ๆ ว่า "S register" ดังแสดงในรูปที่ 5.8

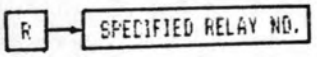


รูปที่ 5.8 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง LOAD

LD-NOT -- การใช้คำสั่ง LD-NOT จะทำให้ค่าสภาวะของรีเลย์ที่ระบุเบอร์ไว้ในคำสั่ง (Operand) จะถูกนำไปเก็บไว้ใน R register โดยจะกลับสภาวะนั้นก่อนนำไปเก็บ ถ้าสภาวะเป็น "1" ก็จะถูกกลับให้เป็น "0" และถ้าสภาวะเป็น "0" ก็จะถูกกลับให้เป็น "1" และในทำนองเดียวกับคำสั่ง LD ค่าสภาวะเดิมใน R register จะถูกย้ายไปเก็บไว้ใน S register ดังแสดงในรูปที่ 5.9

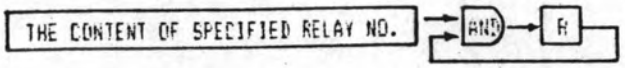
รูปที่ 5.9 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง LOAD-NOT

OUT -- การใช้คำสั่ง OUT จะทำให้ค่าสภาวะที่เก็บไว้ใน R register จะถูกส่งออกไปที่ OUTPUT ของรีเลย์ที่ถูกระบุเบอร์ไว้ในคำสั่ง ในภาวะนี้ค่าสภาวะของ R register จะไม่เปลี่ยน ดังแสดงในรูปที่ 5.10



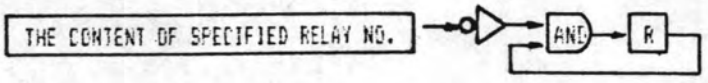
รูปที่ 5.10 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง OUT

AND -- การใช้คำสั่ง AND จะเป็นการกระทำทางลอจิก AND ระหว่างสภาวะของรีเลย์ที่ถูกระบุเบอร์ในคำสั่งกับค่าสภาวะของ R register ผลลัพธ์ของการกระทำลอจิก AND จะถูกนำไปเก็บไว้ใน R registerเช่นเดิม ดังแสดงในรูปที่ 5.11



รูปที่ 5.11 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง AND

AND-NOT -- การใช้คำสั่ง AND-NOT จะเป็นการนำค่าสภาวะของรีเลย์ที่ถูกระบุเบอร์ในคำสั่งนำมากลับค่าสภาวะก่อนแล้วจึงนำมาทำลอจิก AND กับค่าสภาวะใน R register ผลลัพธ์ของการกระทำลอจิกจะถูกเก็บไว้ใน R register ดังแสดงในรูปที่ 5.12

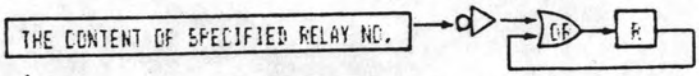


รูปที่ 5.12 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง AND-NOT

OR -- การใช้คำสั่ง OR จะเป็นการกระทำทางลอจิก OR ระหว่างสภาวะของรีเลย์ที่ถูกระบุเบอร์ในคำสั่งกับค่าสภาวะของ R register ผลลัพธ์ของการกระทำลอจิก OR จะถูกนำไปเก็บไว้ใน R register เช่นเดิม ดังแสดงในรูปที่ 5.13

รูปที่ 5.13 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง OR

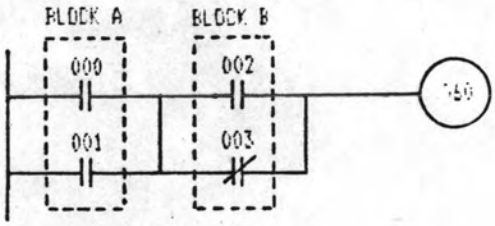
OR-NOT -- การใช้คำสั่ง OR-NOT จะเป็นการนำค่าสภาวะของรีเลย์ที่ถูกระบุเบอร์ในคำสั่งนำมากลับค่าของสภาวะก่อนแล้ว จึงมากระทำลอจิก OR กับค่าสถานะที่เก็บใน R register ผลลัพธ์ของการกระทำทางลอจิกจะถูกเก็บไว้ใน R register ดังแสดงในรูปที่ 5.14



รูปที่ 5.14 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง OR-NOT

5.2.5 AND-BLOCK (AND-BLK) INSTRUCTION

ใช้คำสั่ง AND-BLOCK เมื่อมีการ AND กันระหว่าง 2 บล็อคหรือมากกว่า ดังแสดงในรูปที่ 5.15



CODING

STEP	OPCODE	OPERAND
020	LD	000
021	OR	001
022	LD‼	002
023	OR-NOT	003
024	AND-BLK‼	-
025	OUT	060

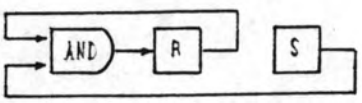
รูปที่ 5.15 แสดงการใช้คำสั่ง AND-BLOCK

หมายเหตุ 1. * ให้ใช้คำสั่ง LD เป็นคำสั่งแรกสำหรับบล็อคถัดไปที่จะมา AND กับบล็อคก่อนหน้า

2. ** ให้ใช้คำสั่ง AND-BLK สำหรับบล็อคที่ต่ออนุกรมกัน 2 บล็อค (บล็อค A และ B)

การทำงานของรีจีสเตอร์ภายใน (Operation of Internal Register)

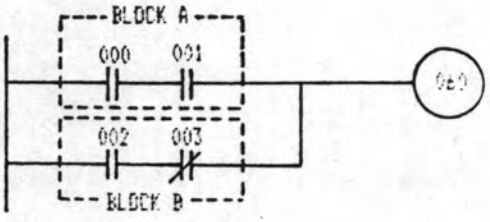
1. จากคำสั่ง LD 000 และ OR 001 จะได้ผลลัพธ์จากการทำลอจิก OR ในบล็อค A และจะเก็บผลลัพธ์ที่ได้ใน R register
2. จากคำสั่ง LD 002 ในบล็อค B จะทำให้ผลลัพธ์ของบล็อค A ที่เก็บไว้ใน R register ถูกย้ายไปเก็บที่ S register และผลลัพธ์จากการทำลอจิก OR ระหว่างคำสั่ง LD 002 และ OR-NOT 003 ในบล็อค B จะถูกนำไปเก็บใน R register
3. คำสั่ง AND-BLK จะเป็นการกระทำทางลอจิก AND ระหว่างกับ R register กับ S register ผลลัพธ์ของการทำลอจิก AND จะถูกนำไปเก็บไว้ใน R register เดิม ดังแสดงในรูปที่ 5.16
   จำนวนบล็อคที่สามารถนำมากระทำลอจิก AND สามารถมีได้ไม่จำกัด



รูปที่ 5.16 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง AND-BLOCK

5.2.6 OR-BLOCK (OR-BLK) INSTRUCTION

การใช้คำสั่ง OR-BLOCK จะใช้เมื่อมีการ OR กันระหว่าง 2 บล็อคหรือมากกว่าดังแสดงในรูปที่ 5.17



CODING

STEP	OPCODE	OPERAND
020	LD	000
021	AND	001
022	LD*	002
023	AND-NOT	003
024	OR-BLK**	-
025	OUT	060

รูปที่ 5.17 แสดงการใช้คำสั่ง OR-BLOCK

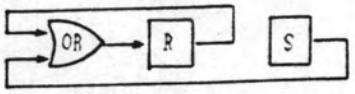
<u>หมายเหตุ</u> 1. * ให้ใช้คำสั่ง LD เป็นคำสั่งแรกสำหรับบล็อคถัดไปที่จะนำมา OR กับบล็อคก่อนหน้า

2. ** ให้ใช้คำสั่ง OR-BLK สำหรับบล็อคที่นำมาต่อขนานกัน 2 บล็อค (บล็อค A และ B)

การทำงานของรีเจสเตอร์ภายใน (Operation of Internal Register)

1. จากคำสั่ง LD 000 และ AND 001 จะได้ผลลัพธ์จากการทำลอจิก AND ในบล็อค A และจะเก็บผลลัพธ์ที่ได้ใน R register
2. คำสั่ง LD 002 ในบล็อค B จะทำให้ผลลัพธ์ของบล็อค A ที่เก็บไว้ใน R register ถูกย้ายไปเก็บใน S register และผลลัพธ์จากการทำลอจิก AND ระหว่างคำสั่ง LD 002 และ AND-NOT 003 ในบล็อค B จะถูกเก็บใน R register
3. คำสั่ง OR-BLK จะเป็นการกระทำทางลอจิก OR ระหว่าง R register และ S register ผลลัพธ์ของการทำลอจิก OR จะถูกเก็บลงใน R register เดิม ดังแสดงในรูปที่ 5.18

จำนวนบล็อคที่สามารถนำมากระทำลอจิก OR สามารถมีได้ไม่จำกัด



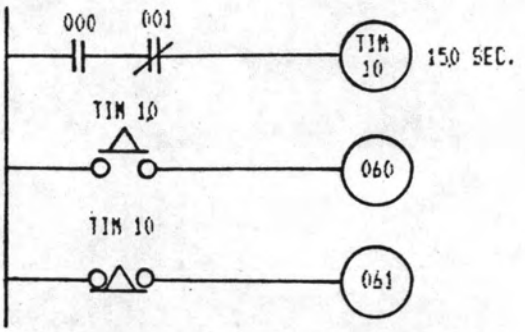
รูปที่ 5.18 แสดงการทำงานของรีจีสเตอร์ในคำสั่ง OR-BLOCK

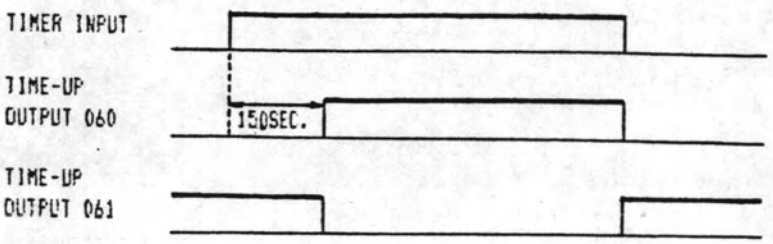
5.2.7 TIMER (TIM) INSTRUCTION

คำสั่ง TIM ที่ใช้จะเป็นตัวตั้งเวลาชนิด ON-DELAY TIMER คือหน้าสัมผัสของตัว TIMER จะเริ่มทำงานเมื่อถึงเวลาที่ตั้งไว้ ถ้าเป็นหน้าสัมผัสชนิด NO เมื่อถึงเวลาที่ตั้งไว้หน้าสัมผัสก็จะปิดลง และถ้าหน้าสัมผัสเป็นชนิด NC เมื่อถึงเวลาที่ตั้งไว้ก็จะเปิดออก ดังแสดงในรูปที่ 5.19

หมายเหตุ

1. * TIMER NUMBER จะเริ่มจาก 00-29
   ** ค่าเวลาของ TIMER เริ่มจาก 0000-9999 x 0.1 SEC
2. TIMER NUMBER จะใช้ร่วมกับ COUNTER ดังนั้นเมื่อ NUMBER ใดถูกใช้แล้วจะนำมาใช้ซ้ำกันอีกไม่ได้





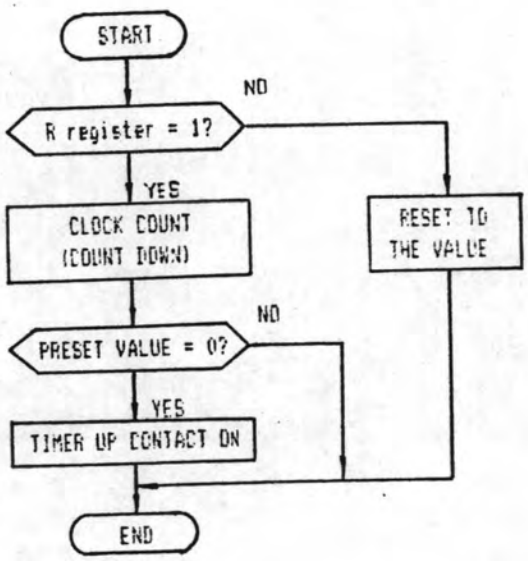
CODING

STEP	OPCODE	OPERAND
020	LD	000
021	AND-NOT	001
022	TIM	10#
023	DS	0150##
024	LD	TIM 10
025	OUT	060
026	LD-NOT	TIM 10
027	OUT	061

รูปที่ 5.19 แสดงการใช้คำสั่ง TIMER

การทำงานของรีจีสเตอร์ภายใน (Operation of Internal Register)

TIMER จะเริ่มนับโดยจะนับถอยหลัง (Count Down) เมื่อค่าใน R register มีสถานะเป็น "1" เมื่อค่านับมีค่าเป็น "0" หน้าสัมผัสของ TIMER จะเริ่มทำงานหรือจะรีเซทเมื่อค่าสภาวะใน R register เป็น "0" รูปที่ 5.20 แสดงโฟล์วชาร์ทการทำงานของตัวตั้งเวลา (TIMER)



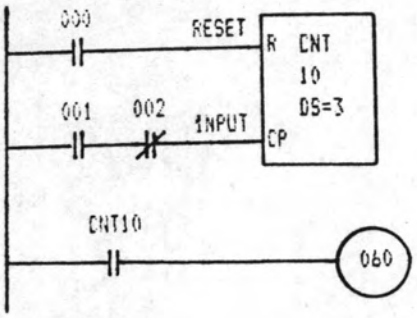
รูปที่ 5.20 แสดงโฟล์วชาร์ทของตัวตั้งเวลา

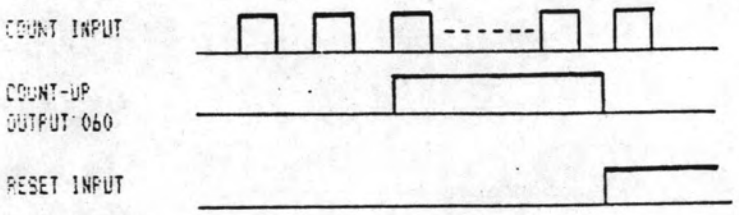
5.2.8 COUNTER (CNT) INSTRUCTION

คำสั่ง CNT ที่ใช้ เป็นตัวนับชนิด PRESET COUNTER เช่นเดียวกับตัวนับที่ใช้ในวงจรรีเลย์ทั่ว ๆ ไป ดังแสดงในรูปที่ 5.21

หมายเหตุ

1. COUNTER NUMBER เริ่มจาก 00-29 เช่นเดียวกับตัวตั้งเวลา
2. ค่านับของ COUNTER สามารถตั้งค่าได้จาก 0000-9999
3. COUNTER NUMBER จะใช้ร่วมกับ TIMER ดังนั้นเมื่อ NUMBER ใดถูกใช้แล้ว ห้ามนำมาใช้ซ้ำอีก





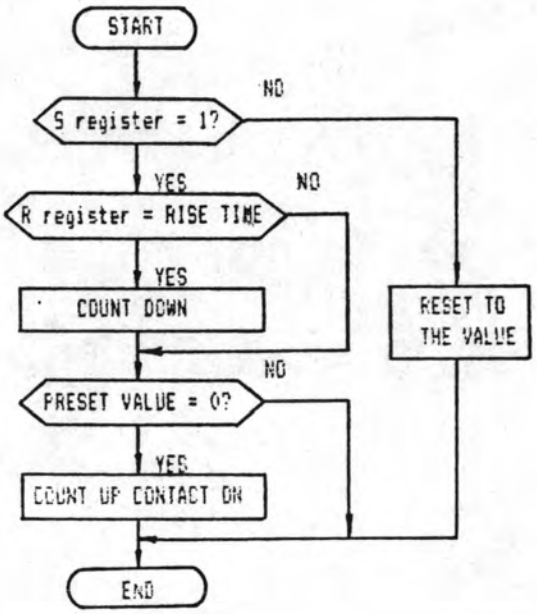
CODING

STEP	OPCODE	OPERAND
020	LD	000
021	LD	001
022	AND-NOT	002
023	CNT	10
024	DS	0003
025	LD	CNT 10
026	OUT	060

รูปที่ 5.21 แสดงการใช้คำสั่ง COUNTER

การทำงานของรีจีสเตอร์ภายใน (Operation of Internal Register)

COUNTER จะทำงานเมื่อค่าใน S register เป็น "0" และค่านับของ COUNTER จะนำมาจากค่าสภาวะใน R register และจะนับเมื่อสัญญาณเป็นขอบขาขึ้นเท่านั้น การนับจะเป็นแบบนับถอยหลัง (Count Down) เช่นเดียวกับ TIMER COUNTER จะถูกรีเซทเมื่อค่าสภาวะใน S register เป็น "1" รูปที่ 5.22 แสดงโฟล์วชาร์ทการทำงานของตัวนับ (COUNTER)



รูปที่ 5.22 แสดงโฟล์วชาร์ทของตัวนับ

5.2.9 DATA SET (DS) INSTRUCTION

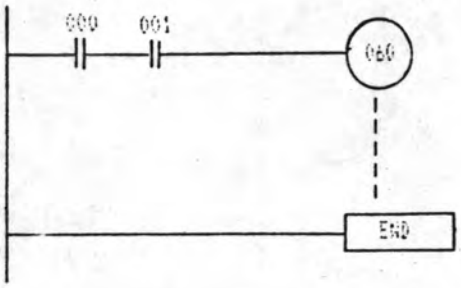
คำสั่ง DS ใช้ในการ PRESET ค่าให้แก่ ตัวนับ (COUNTER) และตัวตั้งเวลา (TIMER) ซึ่งสามารถตั้งค่าได้จาก 0000-9999 คำสั่งนี้จะต้องใช้ร่วมกับคำสั่ง COUNTER หรือ TIMER เสมอ

การทำงานของรีจิสเตอร์ (Operation of Internal Register)

ค่าของเวลาหรือค่าของตัวนับ (Set Value) จะถูกนำไปเก็บไว้ยังบัฟเฟอร์เพื่อให้ตัวตั้งเวลาหรือตัวนับนำค่าตั้ง (Set Value) ในบัฟเฟอร์ไปใช้อีกทีหนึ่ง

5.2.10 END INSTRUCTION

คำสั่ง END จะต้องใส่เพื่อจบโปรแกรมคำสั่งแล้วทุกครั้ง มิฉะนั้นจะไม่สามารถ RUN โปรแกรมได้ ดังแสดงในรูปที่ 5.23

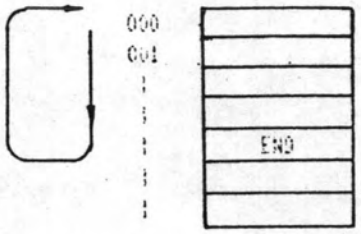


CODING

STEP	OPCODE	OPERAND
000	LD	000
001	AND	001
002	OUT	060
:	:	:
100	END	-

รูปที่ 5.23 แสดงการใช้คำสั่ง END

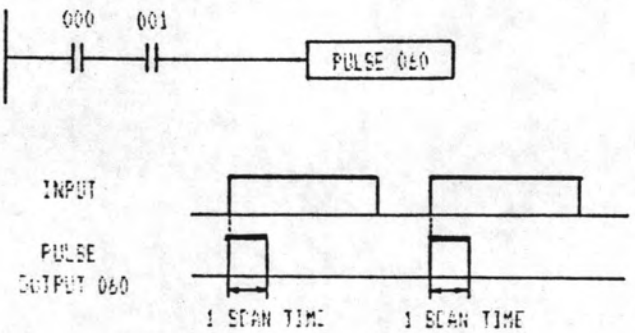
รูปที่ 5.24 แสดงการทำงานของคำสั่ง END CPU จะทำโปรแกรมของผู้ใช้ โดยจะเริ่มทำจาก STEP 000 จนกระทั่งพบคำสั่ง END CPU ก็จะหยุดทำคำสั่งที่ต่อจากคำสั่ง END แล้วจะวนทำ STEP 000 ใหม่ตลอด เมื่อยังอยู่ในโหมด RUN ดังนั้นเราจึงสามารถทดสอบโปรแกรมที่เขียนขึ้นได้ว่าถูกต้องหรือไม่ โดยการแทรกคำสั่ง END เข้าไปในโปรแกรมของแต่ละวงจรย่อย เพื่อทดสอบความถูกต้อง ถ้าการทำงานของโปรแกรมถูกต้องแล้วจึงทำการลบออกเพื่อทดสอบต่อไป ในที่สุดแล้วจะมีคำสั่ง END เพียงคำสั่งเดียวตอนท้ายโปรแกรม



รูปที่ 5.24 แสดงการทำงานของคำสั่ง END

5.2.11 PULSE INSTRUCTION

การใช้คำสั่ง PULSE จะให้เอาท์พุทของรีเลย์ที่ถูกระบุเบอร์ทำงานเพียงการสแกน 1 รอบ ตามเงื่อนไขของอินพุท ดังแสดงในรูปที่ 5.25



CODING

STEP	OPCODE	OPERAND
000	LD	000
001	AND	001
002	PLS	060

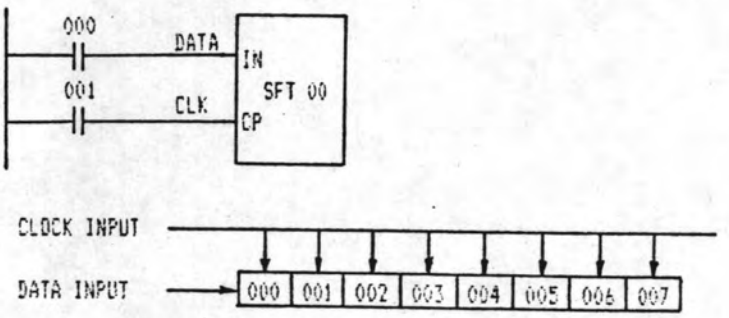
รูปที่ 5.25 แสดงการทำงานของคำสั่ง PULSE

การทำงานของรีจีสเตอร์ภายใน (Operation of Internal Register)

คำสั่ง PULSE จะเซทให้เอาท์พุทที่ถูกระบุเบอร์ทำงานเพียงการสแกน 1 รอบ เมื่อค่าสภาวะใน R register มีค่าเป็น "1"หรือทำงานเมื่อค่าสภาวะใน R register เป็นขอบขาขึ้น

5.2.12 SHIFT INSTRUCTION

คำสั่ง SHIFT เป็นคำสั่งที่ใช้เลื่อนข้อมูลที่เข้ามาอย่างอนุกรมที่อินพุท โดยข้อมูลจะเลื่อนอยู่เฉพาะภายในแชลแนล (CHANNEL) ที่ระบุเท่านั้น ดังแสดงในรูปที่ 5.26



CODING

STEP	OPCODE	OPERAND
000	LD	000
001	LD	001
002	SFT	00

รูปที่ 5.26 แสดงการทำงานของคำสั่ง SHIFT

การทำงานของรีจีสเตอร์ภายใน (Operation of Internal Register)

ค่าสภาวะที่เก็บใน R register จะเป็นข้อมูลที่ต้องการเลื่อนภายในแชลแนล (CHANNEL) ข้อมูลจะถูกเลื่อนไป 1 ตำแหน่งทุก ๆ ขอบขาขึ้นของสัญญาณใน S register

5.3 โปรแกรมจัดการตัวป้อนโปรแกรม

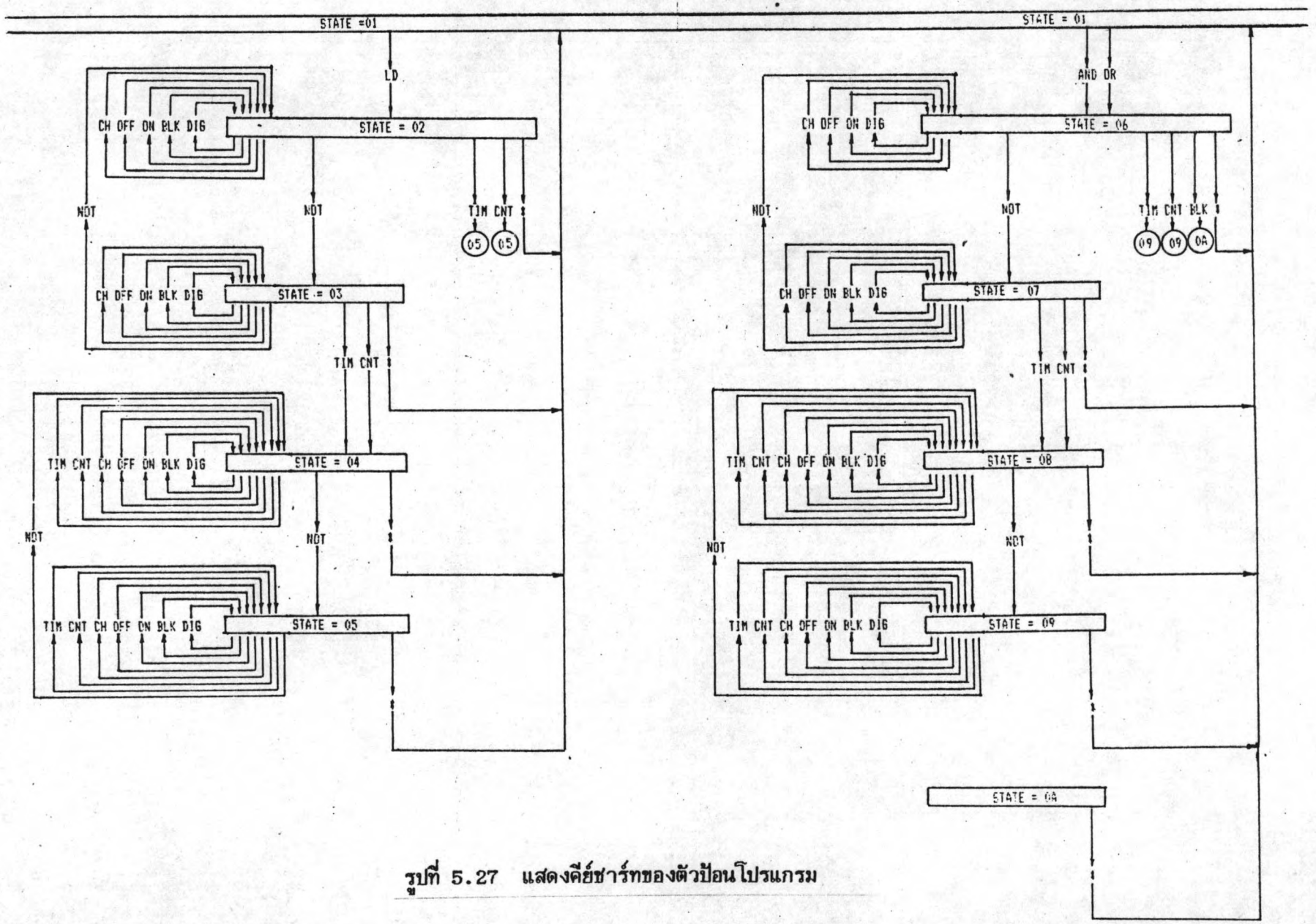
ตัวป้อนโปรแกรมเป็นตัวที่ใช้ติดต่อระหว่างผู้ใช้กับเครื่องอุปกรณ์เครื่องมือทางอินสทรูเมนท์ (Instrument) ส่วนใหญ่มักจะมีการตั้งค่าพารามิเตอร์ (Parameter) ต่าง ๆ ซึ่งจะต้องมีการป้อนตัวพารามิเตอร์แบบเป็นลำดับขั้น (Sequence) ในทำนองเดียวกัน PC ถือว่าเป็นอินสทรูเมนท์ชนิดหนึ่ง ซึ่งมีการป้อนคำสั่งแบบเป็นลำดับขั้น ดังนั้นในการพัฒนาโปรแกรมของตัวป้อนโปรแกรมจะใช้อัลกอริธึมที่เรียกว่า KEYBOARD PARSING [8] ซึ่งเป็นอัลกอริธึมที่ใช้ในอุปกรณ์อินสทรูเมนท์ที่มีการป้อนค่าพารามิเตอร์แบบลำดับ

KEYBOARD PARSER

เป็นการตอบสนองของเครื่องมืออย่างเป็นระบบ เมื่อมีอินพุทป้อนเข้ามาอย่างลำดับ เช่น ในการป้อนคำสั่ง LD จะต้องตามด้วย OPERAND หรือ NOT เครื่องจึงจะได้รับการตอบสนองจากโปรแกรม แต่ถ้าตามด้วย BLOCK เครื่องจะไม่ได้รับการตอบสนองจากโปรแกรม เป็นต้น หน้าที่หลักของ KEYBOARD PARSER แบ่งออกเป็น 2 ส่วนคือ

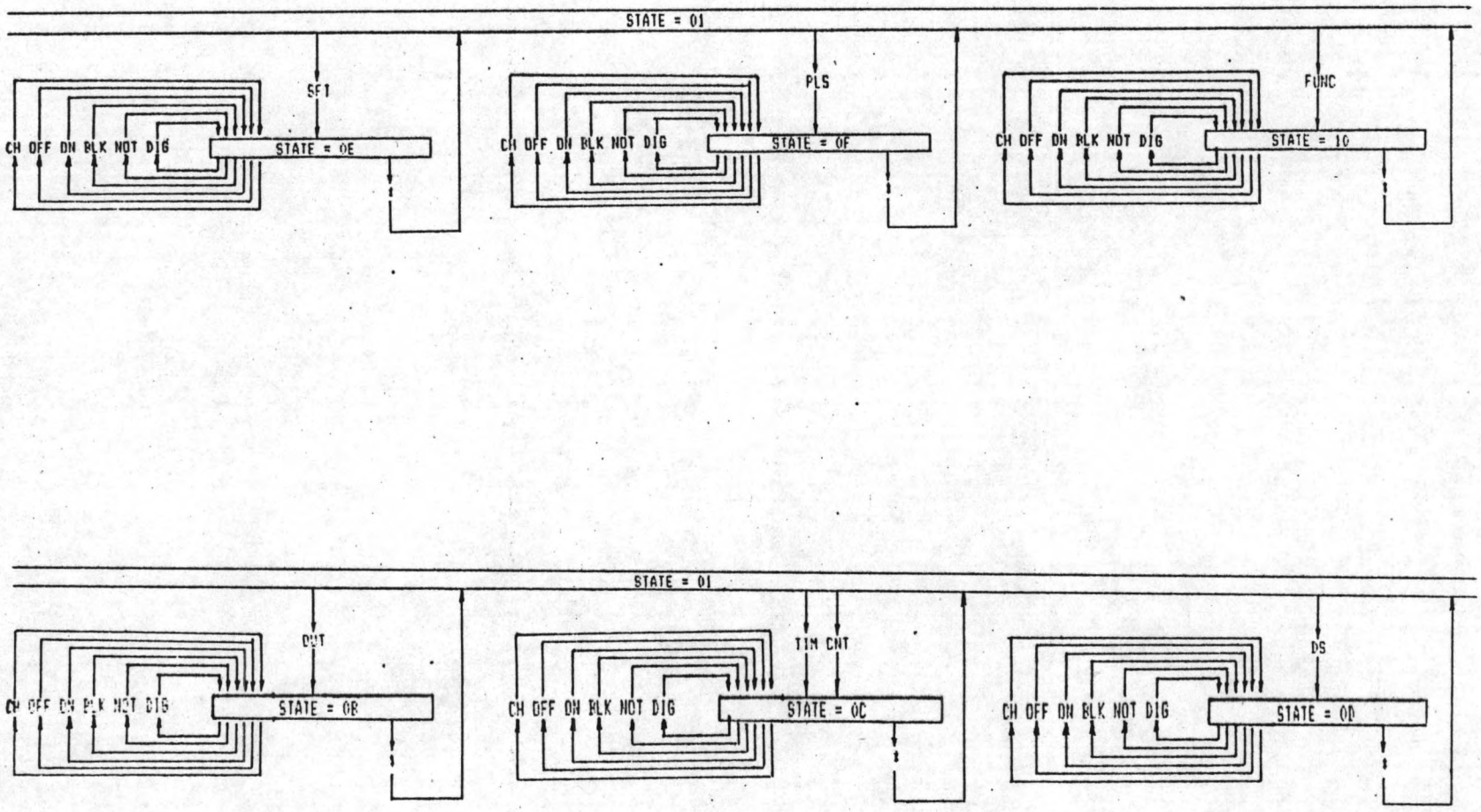
1. กำหนดลำดับสภาวะของคีย์ที่จะต้องกดตามมา เมื่อมีการกดคีย์หลักแล้ว
2. ทำโปรแกรมตอบสนองการกดคีย์

พิจารณารูปที่ 5.27 แสดงคีย์ชาร์ทของตัวป้อนโปรแกรม จะเห็นลำดับของคีย์ที่จะต้องกดตามมา และสภาวะของคีย์ในลำดับต่อมา คีย์ชาร์ทจะประกอบด้วย STATE BOX ซึ่งเป็นตัวกำหนดสภาวะของคีย์อินพุทที่จะต้องกดในลำดับต่อไป และจะประกอบด้วย KEY PATH ซึ่งเป็นทางเดินที่อยู่ระหว่าง STATE BOX การเริ่มต้นทำงานจะต้องกำหนดสภาวะเริ่มต้น (Initial State) ในกรณีนี้ให้สภาวะเริ่มต้นเป็น "0" จากสภาวะเริ่มต้น "0" จะเห็นทางเดินของคีย์ในลำดับต่อไปคือคีย์ CLEAR และสภาวะต่อไปจะเป็น "13" การกดคีย์นี้จะให้ผลตอบสนองต่อผู้ใช้ตามโปรแกรมบริการของ CLEAR ที่ได้เขียนไว้ การกดคีย์ใด ๆ ซึ่งแทนด้วยเครื่องหมาย "*" เครื่องจะไม่ตอบสนองการกด และสภาวะต่อไปจะกลับมาอยู่ที่สภาวะ "0" เช่นเดิม จนกว่าผู้ใช้กดคีย์ CLEAR ก่อนเริ่มต้น จึงสามารถกดคีย์หลักใด ๆ ได้ ในการกดแต่ละครั้งจะมีการกระทำ ACTION ROUTINE เกิดขึ้น เช่น การกดคีย์ CLEAR ACTION ROUTINE จะทำการเคลียร์ตัวแสดงผล และให้แสดง STEP เริ่มต้นที่ "000" และเมื่อกดคีย์ใด ๆ (*) จะไม่ได้รับการตอบสนอง (No Action) เป็นต้น



รูปที่ 5.27 แสดงคีย์ชาร์ทของตัวป้อนโปรแกรม

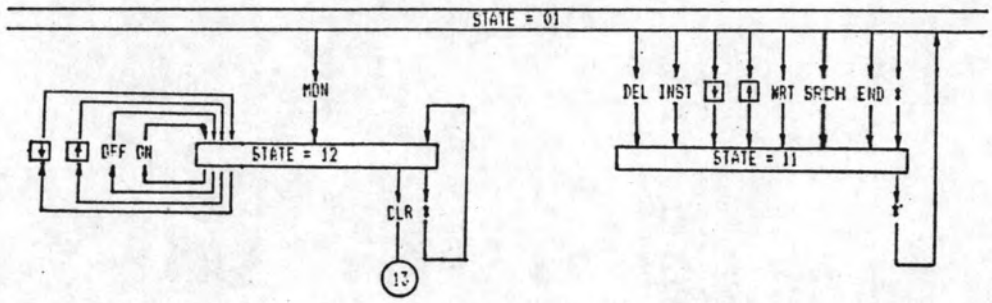
KEY CHART FOR INSTRUCTION



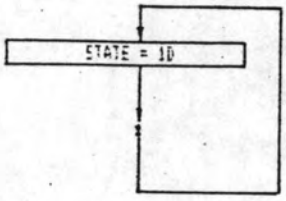
รูปที่ 5.27 แสดงคีย์ชาร์ทของตัวป้อนโปรแกรม(ต่อ)

รูปที่ 5.27 แสดงคีย์ชาร์ทของตัวป้อนโปรแกรม(ต่อ)

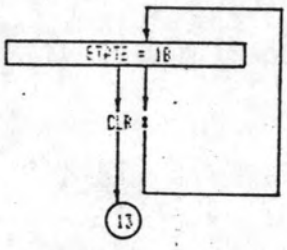
KEY CHART FOR COMMAND



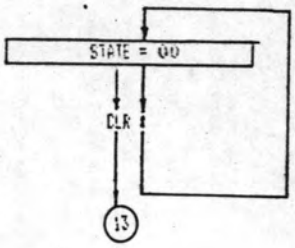
KEY CHART FOR EPROM ERROR



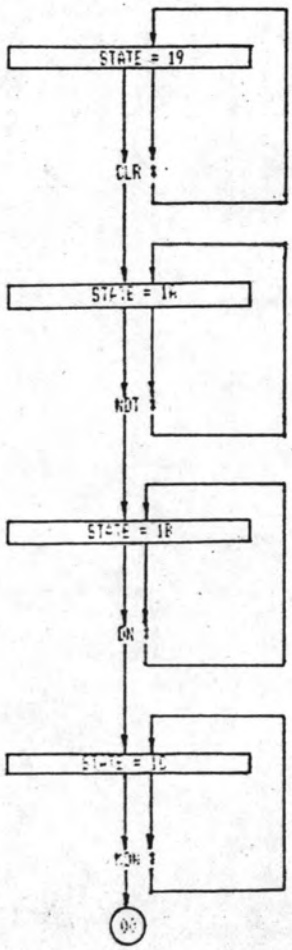
KEY CHART FOR HANDLING ERROR

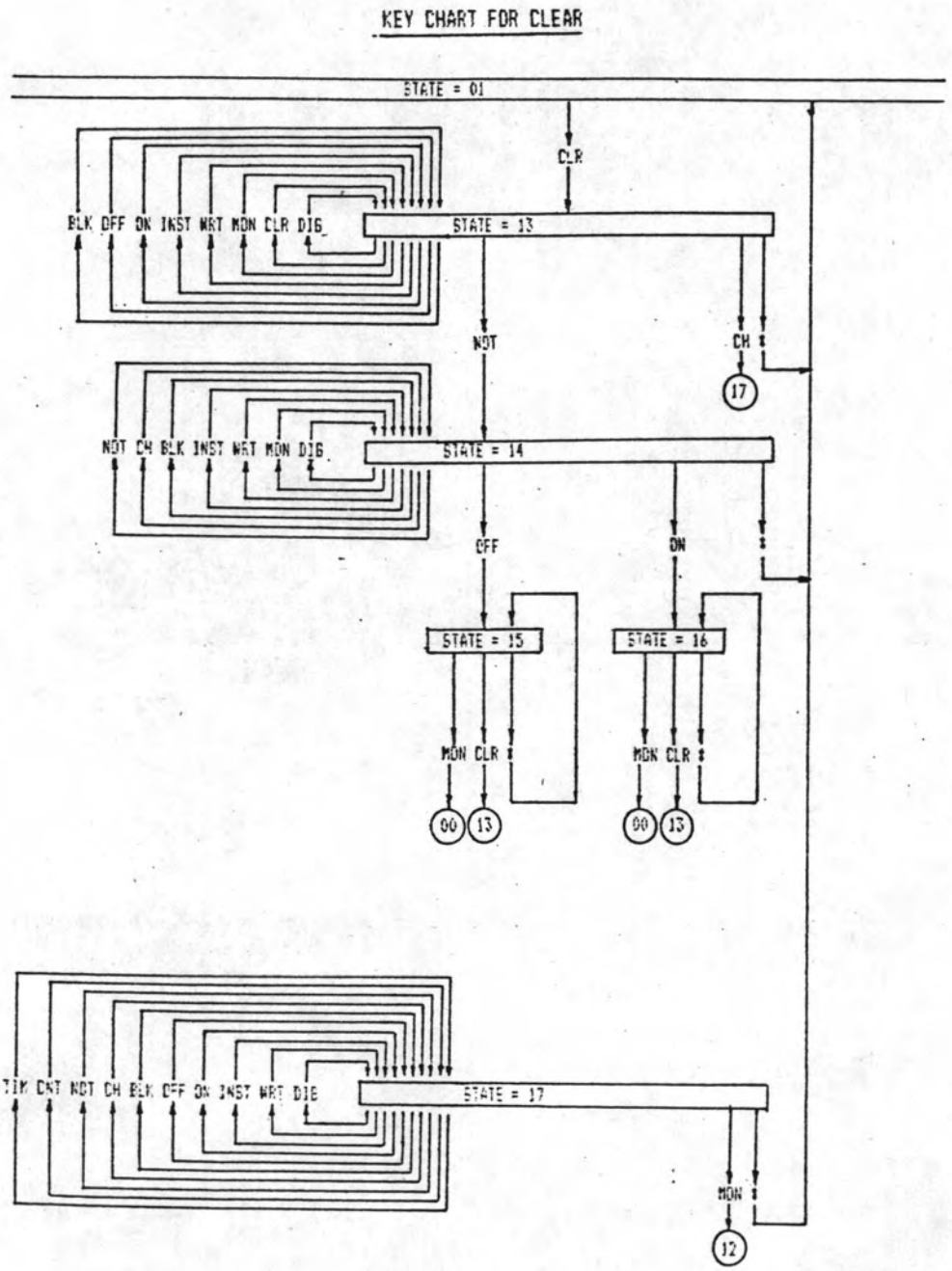


KEY CHART FOR INITIAL STATE



KEY CHART FOR MEMORY ERROR





รูปที่ 5.27 แสดงคีย์ชาร์ทของตัวป้อนโปรแกรม(ต่อ)

การทำงานของคีย์ชาร์ทจะถูกแบ่งออกเป็นส่วน ๆ ดังนี้

1. KEYCHART FOR INSTRUCTION — เป็นคีย์ชาร์ทที่กำหนดสภาวะต่อไปของคำสั่ง เช่น คำสั่ง LD, AND, OR, OUT, TIM และ CNT เป็นต้น

2. KEYCHART FOR COMMAND — เป็นคีย์ชาร์ทที่กำหนดสภาวะต่อไปของคำสั่งควบคุมเครื่อง เช่น DEL, INST, WRT, SRCH, MON เป็นต้น

3. KEYCHART FOR EPROM ERROR — เป็นคีย์ชาร์ทที่กำหนดสภาวะของคีย์ต่อไปที่จะกด ซึ่งจะเห็นว่าสภาวะต่อไปเป็นสภาวะเดิมไม่มีทางหลุดออกจากลูป นั่นคือในกรณีที่เกิด EPROM ERROR ให้ปิดเครื่อง แล้วถอด EPROM ออก เครื่องจึงจะสามารถทำงานต่อไปได้

4. KEYCHART FOR HANDLING ERROR — เป็นคีย์ชาร์ทที่กำหนดสภาวะของคีย์ที่จะกดต่อไป เมื่อเครื่องมี ERROR เกิดขึ้น ในที่นี้คือการกดคีย์ CLR

5. KEYCHART FOR INITIAL STATE — ในสภาวะเริ่มแรกเมื่อเปิดเครื่องจะถูกกำหนดให้มีสภาวะเป็น "0" และคีย์ต่อมาที่จะสามารถกดได้คือคีย์ CLR

6. KEYCHART FOR MEMORY ERROR — เมื่อโปรแกรมผู้ใช้ผิดรูปแบบฟอร์มจากที่กำหนดไว้ สภาวะของคีย์จะถูกกำหนดให้เป็นสภาวะที่ "19" และจะต้องกดคีย์ตามลำดับในคีย์ชาร์ทนี้ เพื่อทำการเคลียร์โปรแกรมในหน่วยความจำ เครื่องจึงจะสามารถทำงานต่อเป็นปกติได้

7. KEYCHART FOR CLEAR — เป็นคีย์ชาร์ทที่ใช้เคลียร์หน่วยแสดงผลบนตัวป้อนโปรแกรม และยังใช้ในการเคลียร์ PROGRAM MEMORY และ DATA MEMORY อีกด้วย

จากคีย์ชาร์ทที่ได้นี้จะพัฒนาให้อยู่ในรูปของตารางที่ 5.1 เป็นตาราง PARSER STATE ซึ่งเป็นหัวใจของ KEYBOARD PARSER ประกอบด้วย PRESENT STATE คีย์ที่กดในแต่ละ PRESENT STATE NEXT STATE ซึ่งเป็น STATE ต่อไป หรือคือ PRESENT STATE ของคีย์ที่จะกดในครั้งต่อไป และ ACTION ROUTINE NUMBER ซึ่งเป็นหมายเลขของ ACTION ROUTINE ที่จะตอบสนองการกดคีย์ใน PRESENT STATE นั้นๆ

จากรูปที่ 5.28 แสดงโฟล์วชาร์ทของโปรแกรม KEYBOARD PARSER โดยจะใช้ PRESENT STATE และคีย์ที่กดเพื่อใช้หาสภาวะต่อไป ถ้าสภาวะต่อไปเป็น "1" โปรแกรมจะเริ่มกลับไปวนลูปใหม่ โดยจะใช้สภาวะ "1" และคีย์ที่กดอยู่ก่อนแล้วในการหาสภาวะต่อไป ถ้าสภาวะต่อไปไม่ใช่ "1" ก็จะมีการทำโปรแกรม ACTION ROUTINE ตามหมายเลขของ ACTION ROUTINE NUMBER ที่ปรากฏในตารางที่ 5.1

ตารางที่ 5.1 แสดงตาราง PARSER STATE

PARSER STATE TABLE

PRESENT STATE	KEYBROAD	FNKYT	NEXT	ACTION ROUTINE NO.
PREST = 0	CLR	2	13	1 (BLK-DIG)
	*	0	0	0 (ACT-MON1)
PREST = 1	CLR	2	13	26 (CLEAR26)
	LD	3	2	3 (DSP-LD)
	AND	4	6	4 (DSP-AND)
	OR	5	6	5 (DSP-OR)
	OUT	6	B	6 (DSP-OUT)
	TIM	7	C	7 (DSP-TIM)
	CNT	8	C	B (DSP-CNT)
	DS	9	D	9 (DSP-DS)
	END	A	11	A (DSP-END)
	SFT	B	E	B (DSP-SFT)
	PLS	C	F	C (DSP-PLS)
	FUNC	D	10	D (DSP-FUNC)
	INST	E	11	E (INSERT)
	DEL	F	11	F (DELETE)
	↓	10	11	10 (DCRMENT)
	↑	11	11	11 (INRMENT)
	WR	12	11	12 (WRITE)
	SRCH	13	11	13 (SRCH)
	MON	14	12	14 (CALL)
	*	0	1	0 (ACT-MON1)
PREST = 2	DIG	1	2	15 (DISP#1)
	NOT	15	3	1C (DSP-NOT)
	TIM	7	5	16 (DSP-TIM2)
	CNT	8	5	17 (DSP-CNT2)
	BLK	17	2	0 (ACT-MON1)
	CH	1A	2	0 (ACT-MON1)
	ON	18	2	0 (ACT-MON1)
	OFF	19	2	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = 3	DIG	1	3	15 (DISP#1)
	NOT	15	2	1D (DSP-BKNOT)
	TIM	7	4	16 (DSP-TIM2)
	CNT	8	4	17 (DSP-CNT2)
	BLK	17	3	0 (ACT-MON1)
	CH	1A	3	0 (ACT-MON1)
	ON	18	3	0 (ACT-MON1)
	OFF	19	3	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)

ตารางที่ 5.1 แสดงตาราง PARSER STATE(ต่อ)

PRESENT STATE	KEYBROAD	FNKYT	NEXT	ACTION ROUTINE NO.
PREST = 4	DIG	1	4	18 (DISP#2)
	NOT	15	5	1D (DSP-BKNOT)
	TIM	7	4	16 (DSP-TIM2)
	CNT	8	4	17 (DSP-CNT2)
	BLK	17	4	0 (ACT-MON1)
	CH	1A	4	0 (ACT-MON1)
	ON	18	4	0 (ACT-MON1)
	OFF	19	4	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = 5	DIG	1	5	18 (DISP#2)
	NOT	15	4	1C (DSP-NOT)
	TIM	7	5	16 (DSP-TIM2)
	CNT	8	5	17 (DSP-CNT2)
	BLK	17	5	0 (ACT-MON1)
	CH	1A	5	0 (ACT-MON1)
	ON	18	5	0 (ACT-MON1)
	OFF	19	5	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = 6	DIG	1	6	15 (DISP#1)
	NOT	15	7	1C (DSP-NOT)
	TIM	7	9	16 (DSP-TIM2)
	CNT	8	9	17 (DSP-CNT2)
	BLK	17	A	27 (DSP-BLK)
	CH	1A	6	0 (ACT-MON1)
	ON	18	6	0 (ACT-MON1)
	OFF	19	6	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = 7	DIG	1	7	15 (DISP#1)
	NOT	15	6	1D (DSP-BKNOT)
	TIM	7	B	16 (DSP-TIM2)
	CNT	8	B	17 (DSP-CNT2)
	BLK	17	7	0 (ACT-MON1)
	CH	1A	7	0 (ACT-MON1)
	ON	18	7	0 (ACT-MON1)
	OFF	19	7	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = 8	DIG	1	8	18 (DISP#2)
	NOT	15	9	1D (DSP-BKNOT)
	TIM	7	8	16 (DSP-TIM2)
	CNT	8	8	17 (DSP-CNT2)
	BLK	17	8	0 (ACT-MON1)
	CH	1A	8	0 (ACT-MON1)
	ON	18	8	0 (ACT-MON1)
	OFF	19	8	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)

ตารางที่ 5.1 แสดงตาราง PARSER STATE(ต่อ)

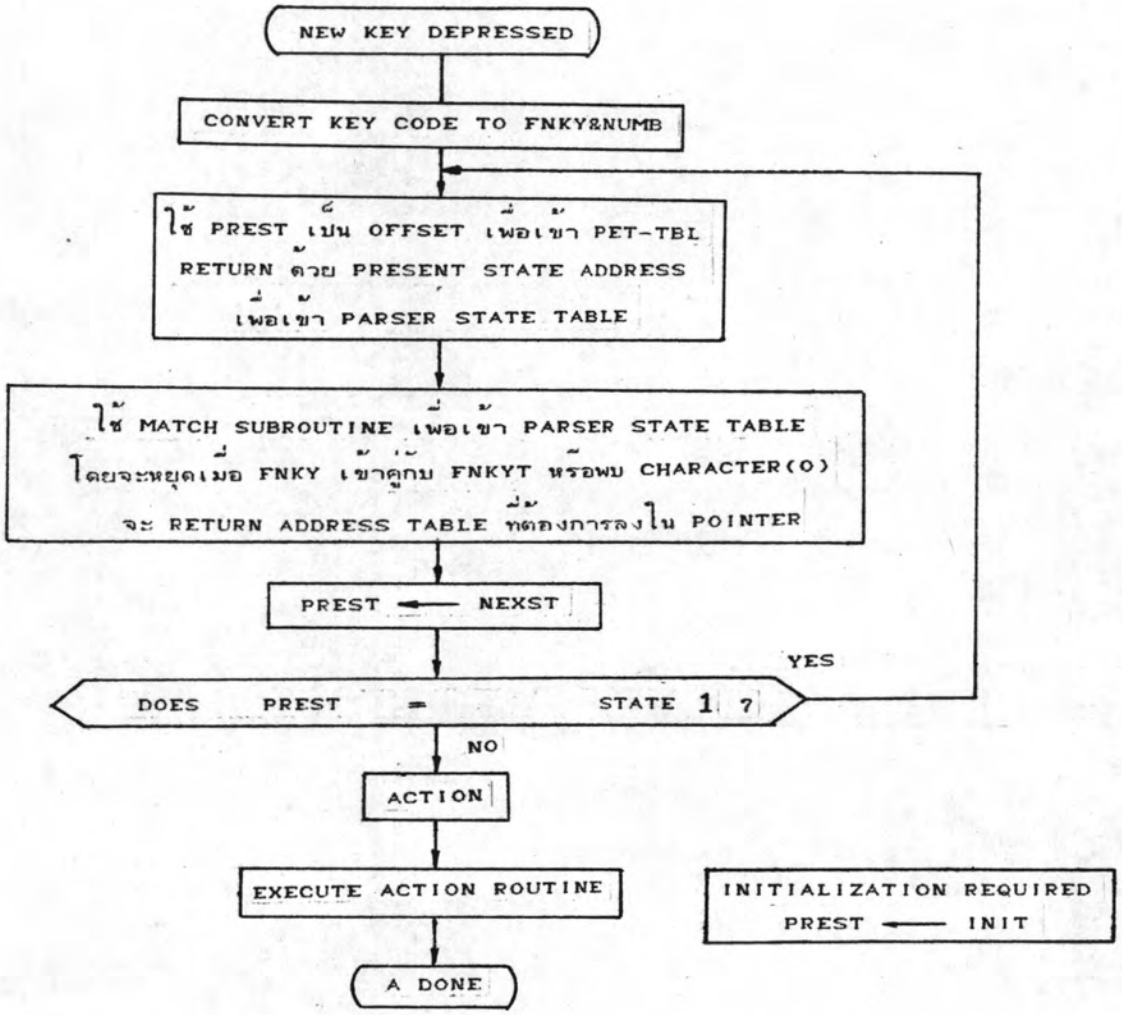
PRESENT STATE	KEYBROAD	FNKYT	NEXT	ACTION ROUTINE NO.
PREST = 9	DIG	1	9	18 (DISP#2)
	NOT	15	8	1C (DSP-NOT)
	TIM	7	9	16 (DSP-TIM2)
	CNT	8	9	17 (DSP-CNT2)
	BLK	17	9	0 (ACT-MON1)
	CH	1A	9	0 (ACT-MON1)
	ON	18	9	0 (ACT-MON1)
	OFF	19	9	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = A	*	0	1	0 (ACT-MON1)
PREST = B	DIG	1	B	15 (DISP#1)
	NOT	15	B	0 (ACT-MON1)
	BLK	17	B	0 (ACT-MON1)
	CH	1A	B	0 (ACT-MON1)
	ON	18	B	0 (ACT-MON1)
	OFF	19	B	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = C	DIG	1	C	18 (DISP#2)
	NOT	15	C	0 (ACT-MON1)
	BLK	17	C	0 (ACT-MON1)
	CH	1A	C	0 (ACT-MON1)
	ON	18	C	0 (ACT-MON1)
	OFF	19	C	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = D	DIG	1	D	19 (DISP#3)
	NOT	15	D	0 (ACT-MON1)
	BLK	17	D	0 (ACT-MON1)
	CH	1A	D	0 (ACT-MON1)
	ON	1B	D	0 (ACT-MON1)
	OFF	19	D	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = E	DIG	1	E	18 (DISP#2)
	NOT	15	E	0 (ACT-MON1)
	BLK	17	E	0 (ACT-MON1)
	CH	1A	E	0 (ACT-MON1)
	ON	1B	E	0 (ACT-MON1)
	OFF	19	E	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = F	DIG	1	F	15 (DISP#1)
	NOT	15	F	0 (ACT-MON1)
	BLK	17	F	0 (ACT-MON1)
	CH	1A	F	0 (ACT-MON1)
	ON	18	F	0 (ACT-MON1)
	OFF	19	F	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)

ตารางที่ 5.1 แสดงตาราง PARSER STATE(ต่อ)

PRESENT STATE	KEYBRDAD	FNKYT	NEXT	ACTION ROUTINE NO.
PREST = 10	DIG	1	10	1A (DISP#5)
	NOT	15	10	0 (ACT-MON1)
	BLK	17	10	0 (ACT-MON1)
	CH	1A	10	0 (ACT-MON1)
	ON	18	10	0 (ACT-MON1)
	OFF	19	10	0 (ACT_MON1)
	*	0	1	0 (ACT-MON1)
PREST = 11	*	0	1	0 (ACT-MON1)
PREST = 12	ON	18	12	1E (ON)
	OFF	19	12	1F (OFF)
	:	10	12	20 (DCR-CALL)
	;	11	12	21 (INC-CALL)
	CLR	2	13	1 (BLK-DIG)
	*	0	12	0 (ACT-MON1)
PREST = 13	DIG	1	13	1B (DISP#4)
	NOT	15	14	28 (NO-CLR1)
	BLK	17	13	0 (ACT-MON1)
	CH	1A	17	0 (ACT-MON1)
	ON	18	13	0 (ACT-MON1)
	OFF	19	13	0 (ACT_MON1)
	WR	12	13	0 (ACT-MON1)
	MON	14	13	0 (ACT-MON1)
	CLR	2	13	2 (CLEAR2)
	INST	E	13	0 (ACT-MON1)
	*	0	1	0 (ACT-MON1)
PREST = 14	DIG	1	14	0 (ACT-MON1)
	NOT	15	14	0 (ACT-MON1)
	BLK	17	14	0 (ACT-MON1)
	CH	1A	14	0 (ACT-MON1)
	ON	18	16	22 (CLR1-DSP)
	OFF	19	15	23 (CLR2-DSP)
	WR	12	14	0 (ACT-MON1)
	MON	14	14	0 (ACT-MON1)
	INST	E	14	0 (ACT-MON1)
	*	0	1	0 (ACT-MON1)
PREST = 15	CLR	2	13	26 (CLEAR26)
	MON	14	0	24 (DAT-CLR)
	*	0	15	0 (ACT-MON1)
PREST = 16	CLR	2	13	26 (CLEAR26)
	MON	14	0	25 (USER-CLR)
	*	0	15	0 (ACT-MON1)

ตารางที่ 5.1 แสดงตาราง PARSER STATE(ต่อ)

PRESENT STATE	KEYBROAD	FNKYT	NEXT	ACTION ROUTINE NO.
PREST = 17	DIG	1	17	15 (DISP#1)
	NOT	15	17	0 (ACT-MON1)
	BLK	17	17	0 (ACT-MON1)
	CH	1A	17	0 (ACT-MON1)
	ON	18	17	0 (ACT-MON1)
	OFF	19	17	0 (ACT-MON1)
	WR	12	17	0 (ACT-MON1)
	MON	14	12	0 (ACT-MON1)
	TIM	7	17	0 (ACT-MON1)
	CNT	8	17	0 (ACT-MON1)
	INST	E	17	0 (ACT-MON1)
	*	0	1	0 (ACT-MON1)
PREST = 18	CLR	2	13	26 (CLEAR26)
	*	0	18	0 (ACT-MON1)
PREST = 19	CLR	2	1A	29 (NO-CLR2)
	*	0	19	0 (ACT-MON1)
PREST = 1A	NOT	15	1B	0 (ACT-MON1)
	*	0	1A	0 (ACT-MON1)
PREST = 1B	ON	18	1C	22 (CLR1-DSP)
	*	0	1B	0 (ACT-MON1)
PREST = 1C	MON	14	0	2A (CLR2-FG)
	*	0	1C	0 (ACT-MON1)
PREST = 1D	*	0	1D	0 (ACT-MON1)



รูปที่ 5.28 แสดงโฟล์วชาร์ทของโปรแกรม KEYBOARD PARSER

การเข้าในตารางของ PARSER STATE เพื่อกำหนดสภาวะต่อไปและทำ ACTION ROUTINE จำเป็นจะต้องใช้ตัวแปรช่วยอยู่หลายตัว ดังแสดงในตารางที่ 5.2

ตารางที่ 5.2 แสดงตัวแปรที่ใช้ในการเข้าตาราง PARSER STATE

PREST	เป็นตัวแปรที่ใช้เก็บ PRESENT STATE เพื่อใช้ในการเข้าไปในตาราง PARSER STATE ในสภาวะเริ่มต้น จำเป็นต้องกำหนดให้มีสภาวะเป็น "0" ก่อน
FNKY	เป็นตัวแปรที่ใช้ในการเก็บโค๊ดจากการกดคีย์ของตัวป้อนโปรแกรม
NUMB	เป็นตัวแปรที่ใช้ในการเก็บค่าตัวเลข จากการกดคีย์ตัวเลข 0-9
FNKYT	เป็นคีย์โค๊ดที่เก็บไว้ภายในตาราง PARSER STATE ซึ่งค่าที่เก็บนี้จะถูกนำไปเปรียบเทียบกับค่าตัวแปรที่เก็บใน FNKY ถ้าค่าที่นำมาเปรียบเทียบกันสามารถจับคู่กันได้แอดเดรสที่อยู่ของ FNKYT นั้นคือค่าที่ต้องการ
NEXST	เป็นค่าสภาวะต่อไปที่เก็บไว้ในตาราง PARSER STATE และค่าสภาวะนี้จะใช้เป็นค่า PRESENT STATE ในครั้งต่อไป
ARNO	เป็น ACTION ROUTINE NUMBER ที่เก็บไว้ในตาราง PARSER STATE เป็นตัวชี้ไปยังโปรแกรม ACTION ที่ต้องการตอบสนองการกดคีย์นั้น

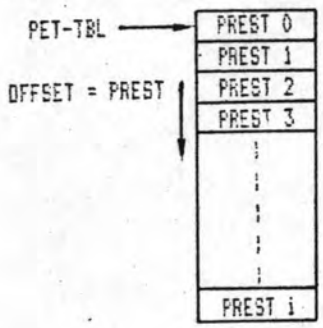
เมื่อมีการกดคีย์ผ่านตัวป้อนโปรแกรม โค๊ดของคีย์ที่กดจริงจะถูกแปลงให้เป็นโค๊ดเก็บลงไปยังตัวแปร 2 ตัวคือ FNKY และ NUMB ดังแสดงในตารางแปลงโค๊ดที่ 5.3

ตารางที่ 5.3 แสดงการแปลงโค๊ดจากการกดคีย์

USE OF FNKY AND NUMB			
ACTUAL KEYBROAD LABEL	ACTUAL HEX KEY CODE	FNKYT	NUMB
0	04	1	0
1	03	1	1
2	0B	1	2
3	13	1	3
4	02	1	4
5	0A	1	5
6	12	1	6
7	00	1	7
8	08	1	8
9	10	1	9
CLR	2B	2	0
LD	19	3	0
AND	21	4	0
OR	29	5	0
OUT	31	6	0
TIM	18	7	0
CNT	20	8	0
DS	28	9	0
END	2A	A	0
SFT	1A	B	0
PLS	22	C	0
FUNC	0C	D	0
INST	1C	E	0
DEL	24	F	0
↑	1B	10	0
↓	23	11	0
WR	34	12	0
SRCH	33	13	0
MON	2C	14	0
NOT	30	15	0
TR	--	16	0
BLK	32	17	0
ON	01	18	0
OFF	09	19	0

การเข้าในตาราง PARSER STATE จำเป็นต้องอาศัยตัวแปร FNKY NUMB และ PREST ซึ่งจะต้องมีค่าสภาวะ (State) ที่ถูกกำหนดแล้วไว้ภายใน จากค่าสภาวะในตัวแปร PREST จะถูกนำมาเป็น OFFSET เพื่อหาแอดเดรสจริงของ PRESENT STATE ที่อยู่ในตาราง PRASER STATE การหาแอดเดรสนี้จำเป็นจะต้องมีตารางที่เก็บแอดเดรสจริงของ PRESENT STATE (PREST 0, PREST 1, PREST2 ....PRESTi) ตารางที่ 5.4 ใช้เพื่อหาแอดเดรสของ PRESENT STATE ที่อยู่บนตาราง PARSER STATE โดยมี PET-TBL เป็นเบส (Base) แอดเดรส และ PREST เป็น OFFSET

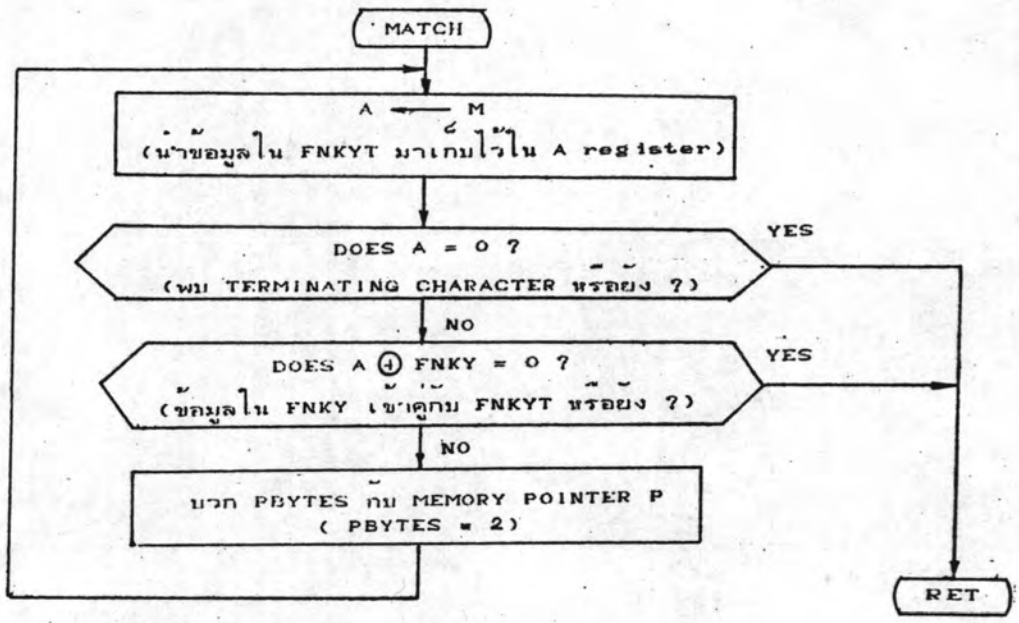
ตารางที่ 5.4 แสดงแอดเดรสของ PRESENT STATE (PARSER ENTRY TABLE)



เมื่อพบแอดเดรสของ PRESENT STATE แล้ว จำเป็นจะต้องหาสภาวะต่อไป และ ACTION ROUTINE ที่ต้องการ ซึ่งในแต่ละ PRESENT STATE จะประกอบด้วย สภาวะต่อไป และ ACTION ROUTINE หลายค่า ดังนั้นการที่จะรู้ว่าเป็นสภาวะใดจะต้องขึ้นอยู่กับเงื่อนไขดังนี้

1. เมื่อพบ CHARACTER "0" ของแต่ละ PRESENT STATE ในตาราง PARSER STATE ให้ถือว่าสภาวะต่อไปคือแอดเดรสนั้น
2. เมื่อ FNKY จับคู่กับ FNKYT ก็ให้ถือว่าสภาวะต่อไปคือแอดเดรสนั้น

เพื่อให้บรรลุวัตถุประสงค์ตามเงื่อนไขที่กล่าวมาทั้ง 2 ข้อ จำเป็นจะต้องมีโปรแกรมช่วยในการนี้ โปรแกรมดังกล่าวแสดงในรูปที่ 5.29



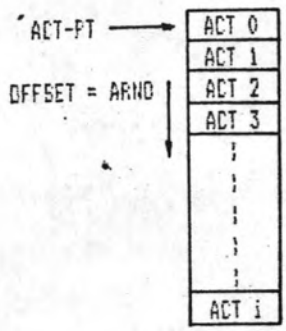
รูปที่ 5.29 แสดงโฟล์วชาร์ทของโปรแกรมจับคู่ (MATCH SUBROUTINE)

การทำงานของโปรแกรมจับคู่เริ่มจากมีตัวชี้ไปยัง PRESENT STATE ภายในตาราง PARSER STATE หลังจากนั้นจะนำค่าโค๊ดจาก FNKYT นำมาเปรียบเทียบกับค่าโค๊ดที่เก็บไว้ในตัวแปร FNKY ถ้าการเปรียบเทียบพบ TERMINATING CHARACTER "0" ก็จะได้แอดเดรสของสภาวะต่อไป และได้หมายเลขของ ACTION ROUTINE หรือถ้าการเปรียบเทียบพบว่าค่าใน FNKY จับคู่ได้กับ FNKYT ก็จะได้แอดเดรสของสภาวะต่อไป และหมายเลขของ ACTION ROUTINE เช่นกัน

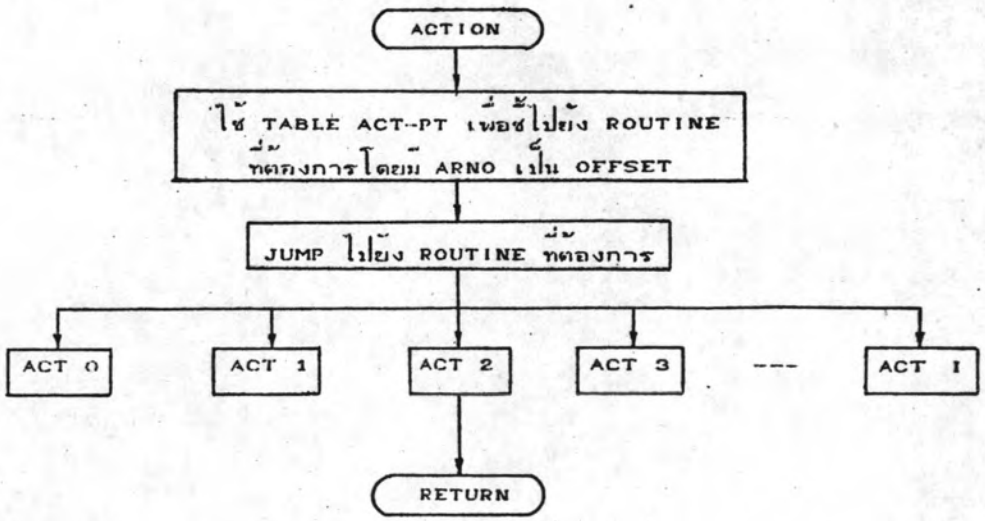
ในการทำโปรแกรม ACTION ROUTINE ที่ต้องการในแต่ละ PRESENT STATE จะต้องนำหมายเลขของ ACTION ROUTINE ที่ได้จากโปรแกรมจับคู่มาเป็น OFFSET เพื่อใช้ใน

การหาแอดเดรสของ ACTION ROUTINE (ACT0, ACT1, ACT2 ..... ACTi) ตารางที่ 5.5 ใช้เพื่อชี้ไปยัง ACTION ROUTINE โดยมี ACT-PT เป็นเบสแอดเดรส และมี ARNO เป็น OFFSET

ตารางที่ 5.5 แสดงการชี้แอดเดรสของ ACTION ROUTINE



พิจารณารูปที่ 5.30 แสดงการทำงานของโฟล์วชาร์ทของ ACTION ROUTINE และตารางที่ 5.6 แสดงหน้าที่ของแต่ละ ACTION ROUTINE



รูปที่ 5.30 แสดงโฟล์วชาร์ทของการทำ ACTION ROUTINE

ตารางที่ 5.6 แสดงการทำงานของโปรแกรม ACTION ROUTINE

ACTION ROUTINE NO.	หน้าที่
0 (ACT-MON1)	ให้รีเทิร์นกลับทันที (NO ACTION)
1 (BLK-DIG)	ดับตัวแสดงผลทั้งหมด ยกเว้นตัวแสดงผล LED 7 ขีดที่ใช้แสดงหมายเลขของ STEP จะแสดง STEP ที่ "000"
2 (CLEAR2)	ให้ LED 7 ขีดที่ใช้แสดงหมายเลขของ STEP แสดง STEP ที่ "000"
3 (DSP-LD)	ให้ LED ของคำสั่ง LD ติด และ LED และ LED 7 ขีดของ OPERAND แสดงผล "000"
4 (DSP-AND)	ให้ LED ของคำสั่ง AND ติด และ LED 7 ขีดของ OPERAND แสดงผล "000"
5 (DSP-OR)	ให้ LED ของคำสั่ง OR ติด และ LED 7 ขีดของ OPERAND แสดงผล "000
6 (DSP-OUT)	ให้ LED ของคำสั่ง OUT ติด และ LED 7 ขีดของ OPERAND แสดงผล "000"
7 (DSP-TIM)	ให้ LED ของคำสั่ง TIM ติด และ LED 7 ขีดของ OPERAND แสดงผล "000"
8 (DSP-CNT)	ให้ LED ของคำสั่ง CNT ติด และ LED 7 ขีดของ OPERAND แสดงผล "000"
9 (DSP-DS)	ให้ LED ของคำสั่ง DS ติด และ LED 7 ขีดของ OPERAND แสดงผล "000"

A (DSP-END) ให้ LED ของคำสั่ง END ติด และ LED 7 ขีดของ OPERAND ดับทั้งหมด

B (DSP-SFT) ให้ LED ของคำสั่ง SFT ติด และ LED 7 ขีดของ OPERAND แสดงผล "000"

C (DSP-PLS) ให้ LED ของคำสั่ง PLS ติด และ LED 7 ขีดของ OPERAND แสดงผล "000"

D (DSP-FUNC) ให้ LED ของคำสั่ง FUNC ติด และ LED 7 ขีดของ OPERAND ดับทั้งหมด และให้ LED 7 ขีดที่ใช้แสดงหมายเลขของ FUNC แสดงผล "00"

E (INSERT) จะทำการเช็คแบบฟอร์มของคำสั่งที่ต้องการสอดแทรกว่าถูกต้องหรือไม่ ถ้าผิดจะแสดงผล ERROR ถ้าถูก LED INST จะติด และจะสอดแทรกโปรแกรมต่อเมื่อมีการกดคีย์ลูกศรลงเท่านั้น

F (DELETE) จะทำการเช็คคอมมานด์ ถ้าเป็น RD จึงจะทำการลบคำสั่งออกได้ และจะแสดงผลของคำสั่งในขั้นโปรแกรม (STEP) ต่อไปต่อเมื่อมีการกดคีย์ลูกศรขึ้นเท่านั้น

10 (DCRMENT) จะทำการแสดงผลของคำสั่งในขั้นโปรแกรม (STEP) ก่อนหน้านี้

11 (INRMENT) จะทำการแสดงผลของคำสั่งในขั้นโปรแกรม (STEP) ต่อไป

12 (WRITE) โปรแกรมจะทำการเช็คโหมด ถ้าอยู่ในโหมดโปรแกรม การทำงานต้องอยู่ใน RAM เท่านั้น จึงจะทำคำสั่ง มิฉะนั้นจะไม่รับคำสั่ง การเขียนคำสั่งจะต้องมีการเช็คแบบฟอร์มว่าถูกต้องจึงเขียนได้ ถ้าผิดจะแสดง ERROR บอกผู้ใช้ให้ทราบ เมื่อเขียนคำสั่งแล้ว จะแสดงผลในขั้นโปรแกรม (STEP) ต่อไป

13 (SRCH) โปรแกรมจะทำงานเฉพาะในโหมดโปรแกรมเท่านั้น โดยจะนำคำสั่งที่ปรากฏบนตัวป้อนโปรแกรม นำไปเปรียบเทียบหาคำสั่งในโปรแกรมผู้ใช้ เมื่อพบจะแสดง STEP และคำสั่งนั้น เมื่อไม่พบจะแสดงคำว่า END

14 (CALL) เป็นโปรแกรมที่ใช้มอนิเตอร์ดูสภาวะของรีเลย์ว่าเปิด หรือปิด และยังใช้ดูสภาวะการทำงานของเวลาในตัวตั้งเวลา ค่านับในตัวนับอีกด้วย

15 (DISP # 1) เป็นโปรแกรมที่ใช้แสดงผลเบอร์ของ OPERAND ด้วย LED 7 ขีด ตามคีย์ตัวเลขที่กด 0-9 จะถูกเรียกใช้เมื่อมีการเรียกใช้คำสั่ง LD, AND, OR และ OUT ก่อนแล้วเท่านั้น

16 (DSP-TIM2) เป็นโปรแกรมแสดงผลเบอร์ TIMER เป็น "00" ด้วย LED 7 ขีด 2 หลัก จะถูกเรียกใช้เมื่อมี OPERAND เป็นหน้าสัมผัสของ TIMER

17 (DSP-CNT2) เป็นโปรแกรมแสดงผลเบอร์ของตัวนับ (COUNTER) เป็น "00" ด้วย LED 7 ขีด 2 หลัก จะถูกเรียกใช้เมื่อมี OPERAND เป็นหน้าสัมผัสตัวนับ (COUNTER)

18 (DISP#2) เป็นโปรแกรมใช้แสดงผลเบอร์ของตัวตั้งเวลา (TIMER) และตัวนับ (COUNTER) ด้วย LED 7 ขีด 2 หลัก ตามคีย์ตัวเลขที่ได้กด 0-9 จะถูกเรียกใช้เมื่อมีการเรียกใช้คำสั่ง TIM หรือ CNT ก่อนแล้วเท่านั้น

19 (DISP#3) เป็นโปรแกรมใช้แสดงผลค่านับของตัวนับ และค่าเวลาของตัวตั้งเวลาด้วย LED 7 ขีด 4 หลัก จะถูกเรียกใช้เมื่อมีการเรียกใช้คำสั่ง DS ก่อนแล้วเท่านั้น

1A (DISP#5) เป็นโปรแกรมใช้แสดงผลเบอร์ของฟังก์ชัน ด้วย LED 7 ขีด 2 หลัก จะถูกเรียกใช้เมื่อมีการเรียกใช้คำสั่ง FUNC ก่อนแล้วเท่านั้น

1B (DISP#4) เป็นโปรแกรมใช้แสดงผลเบอร์ STEP ด้วย LED 7 ขีด 2 หลัก จะถูกเรียกใช้เมื่อมีการกดคีย์ CLEAR ก่อนแล้วเท่านั้น

1C (DSP-NOT) LED ที่ใช้แสดงผลของคำสั่ง NOT จะติด ใช้ร่วมกับคำสั่ง ดังนี้ LD-NOT, AND-NOT, OR-NOT, LD-NOT, TIM, LD-NOT CNT OR-NOT TIM, OR-NOT CNT, AND-NOT TIM, AND-NOT CNT เป็นต้น

1D (DSP-BKNOT) LED ที่ใช้แสดงคำสั่ง NOT จะดับ ใช้ตรงข้ามกับโปรแกรม DSP-NOT นั่นคือการกดคีย์ NOT จะเป็นเหมือน TOGGLE SWITCH โดยการกดครั้งแรก LED ที่ใช้แสดงคำสั่ง NOT จะติด การกดคีย์ NOT อีกครั้งจะทำให้ LED ดับ

1E (ON) เป็นโปรแกรมที่เซทให้หน้าสัมผัสปิด และจะแสดงผลบน LED 7 ขีดเป็น "ON" จะถูกเรียกใช้ เมื่อมีการสั่งให้มอนิเตอร์ดูสภาวะของรีเลย์ก่อนแล้ว

1F (OFF) เป็นโปรแกรมรีเซทให้หน้าสัมผัสเปิด ใช้ตรงข้ามกับ ON

20 (DCR-CALL) เป็นโปรแกรมที่จะถูกเรียกใช้ภายหลังจากการใช้คำสั่ง MON แล้ว โดยโปรแกรมจะลดเบอร์ OPERAND ของรีเลย์ที่กำลังมอนิเตอร์ดูสภาวะอยู่ เพื่อดูสภาวะของรีเลย์ตัวก่อนหน้านี้

21 (INC-CALL) ทำหน้าที่คล้าย DCR-CALL แต่ใช้เพื่อดูสภาวะของรีเลย์ตัวถัดไป

22 (CLR1-DSP) เป็นโปรแกรมที่ใช้แสดงผลเป็น "CLR1" เมื่อมีการเคลียร์โปรแกรมผู้ใช้

23 (CLR2-DSP) เป็นโปรแกรมที่ใช้แสดงผลเป็น "CLR2" เมื่อมีการเคลียร์หน่วยความจำข้อมูล

24 (DAT-CLR) เป็นโปรแกรมที่ใช้เคลียร์หน่วยความจำข้อมูลให้เป็น "0"

25 (USER-CLR) เป็นโปรแกรมที่ใช้เคลียร์โปรแกรมผู้ใช้ให้เป็น "NOP"

26 (CLEAR 26) เป็นโปรแกรมที่ใช้ดับ LED ของคำสั่ง และตัวแสดงผลของ OPERAND

27 (DSP-BLK) LED ที่ใช้แสดงคำสั่ง BLK จะติด ใช้ร่วมกับคำสั่ง AND และ OR เท่านั้น

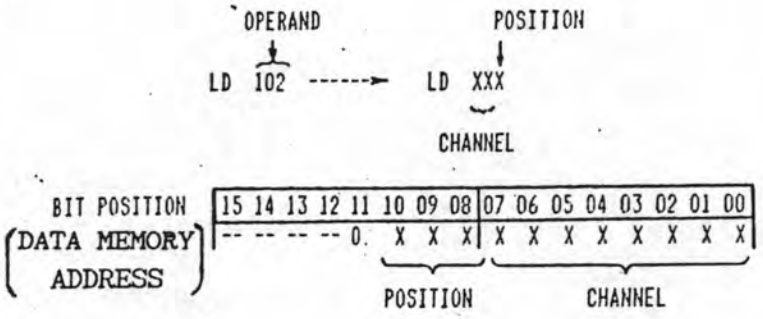
28 (NO-CLR1) เป็นโปรแกรมทำหน้าที่เช่นเดียวกับ CLEAR2 จะถูกเรียกใช้เฉพาะในโหมดโปรแกรม

29 (NO-CLR2) เป็นโปรแกรมที่ทำหน้าที่เช่นเดียวกับ CLEAR 26 จะถูกเรียกใช้เฉพาะในโหมดโปรแกรม

2A (CLR2-FG) เป็นโปรแกรมที่ทำหน้าที่เช่นเดียวกับ USER-CLR และเคลียร์ ERROR FLAG

5.4 โครงสร้างข้อมูลของระบบ (SYSTEM DATA STRUCTURE)

เพื่อให้การทำงานของระบบเป็นไปด้วยความรวดเร็ว ดังนั้นจึงจำเป็นต้องมีการจัดระบบโครงสร้างข้อมูลให้เหมาะสม เพื่อลดความยุ่งยากในการดึงข้อมูลมาใช้ประมวลผลจาก PC การจัดโครงสร้างข้อมูลจะพิจารณาจากระบบเลขฐานที่ใช้ ในบทที่ 2 ได้กล่าวถึงระบบเลขฐานที่จะนำมาใช้ใน PC คือระบบเลขฐานแปด และระบบเลข BCD ซึ่งเป็นระบบเลขที่คนส่วนใหญ่รู้จักคุ้นเคยดี โดยจะนำมาใช้ในการบอกเบอร์ของ OPERAND ซึ่งใช้ระบบเลขทั้ง 2 ระบบผสมกัน



รูปที่ 5.31 แสดงการจัดแอดเดรสของ OPERAND ใน DATA MEMORY

พิจารณารูปที่ 5.31 แสดงการเขียนคำสั่งและการแบ่ง OPERAND ออกเป็น 2 ส่วนคือ

1. OPERAND หลักแรกใช้แสดงตำแหน่ง (POSITION) โดยใช้ระบบเลขฐานแปด ซึ่งมีค่าได้จาก 0-7

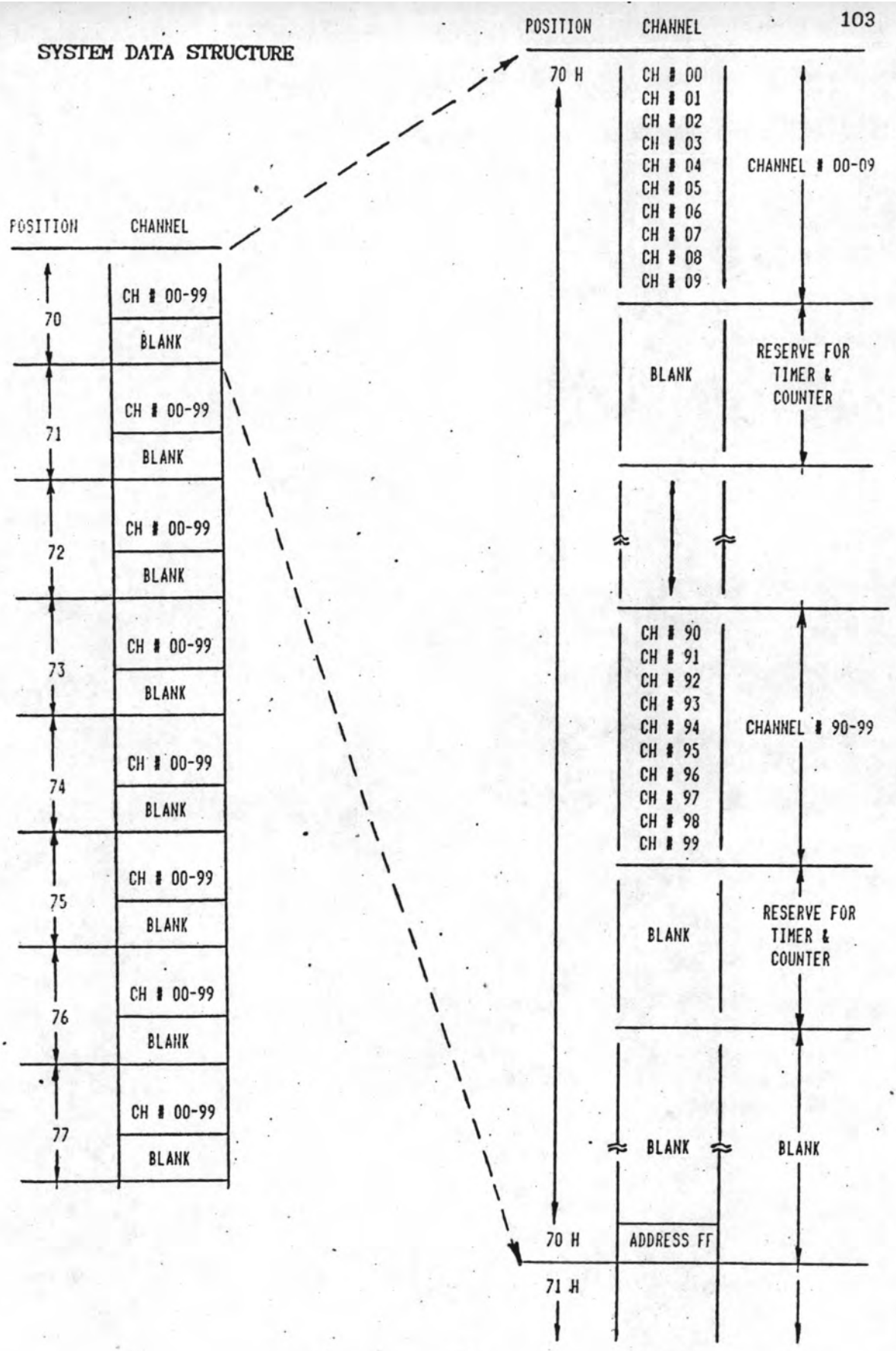
2. OPERAND หลักที่ 2 และ 3 ใช้แสดงแชลแนล (CHANNEL) ใช้ระบบเลขฐานเป็น BCD ซึ่งมีค่าได้จาก 00-99 จากค่าเบอร์ OPERAND ที่ได้ จะถูกนำมาจัดเรียงใหม่เป็นแอดเดรสที่ใช้เก็บสภาวะจริงของ OPERAND นั้น

จากคำสั่งเบื้องต้นที่แสดงในรูปที่ 5.31 OPERAND จะอยู่ในแชลแนลที่ 10 และมีตำแหน่งอยู่ที่ 2 BIT POSITION จะเป็นแอดเดรสของหน่วยความจำข้อมูลที่ใช้เก็บสภาวะ ซึ่งมีค่าจาก 0-15 โดย BIT ที่ 0-7 จะแสดงแชลแนล BIT ที่ 8-10 จะแสดงตำแหน่ง (POSITION) ดังนั้นในการหาแอดเดรสของ OPERAND จะทำได้ง่ายโดยใช้ค่าจาก OPERAND ที่ป้อนจากตัวป้อนโปรแกรมเลย ทำให้ลดความยุ่งยากในการแปลงเลขกลับไปมาระหว่าง BCD และเลขฐานสอง (BINARY) เพื่อหาแอดเดรสจริงของ OPERAND

เหตุผลในการเลือกให้ค่าแชลแนลอยู่ที่ LOW BYTE ของแอดเดรสหน่วยความจำข้อมูล เพราะสามารถใช้ LOW BYTE เป็น BCD ซึ่งแสดงค่าได้สูงสุด 99 แชลแนล ส่วนตำแหน่งจะอยู่นิบเบิล (NIBBLE) ล่างของ HIGH BYTE ของแอดเดรสหน่วยความจำข้อมูล เพราะนิบเบิลของ HIGH BYTE อาจไม่สามารถใช้ได้เนื่องจากถูกจำกัดด้วย โครงสร้างทางฮาร์ดแวร์ของการจัดหน่วยความจำ ดังนั้นนิบเบิลล่างจึงสามารถแทนตำแหน่งได้ 16 ตำแหน่งด้วยระบบเลขฐานสิบหก คือจาก 0-F ซึ่งใช้ตัวแสดงผลเพียงหลักเดียว เช่นเดียวกับเลขฐานแปด แต่เนื่องจากความไม่คุ้นเคยจึงใช้ระบบเลขฐานแปดแทน และเป็นการสะดวกที่ให้ในแต่ละแชลแนลมี

8 ตำแหน่ง เพราะซีพียูที่ใช้เป็นขนาด 8 บิท การอ่านอินพุทและให้เอาท์พุทจะกระทำครั้งละ 8 บิท จากตัวอย่างที่ยกมากล่าวจะได้แอดเดรสจริงของ OPERAND ของคำสั่ง LD 102 ดังนี้

สมมติให้แอดเดรสของ หน่วยความจำข้อมูล (DATA MEMORY) เริ่มต้นที่ 7000H จากคำสั่ง LD 102 จะได้แอดเดรสจริงเป็น 7210H เป็นต้น รูปที่ 5.32 แสดงโครงสร้างข้อมูลของระบบที่ใช้ในการทำวิทยานิพนธ์นี้



รูปที่ 5.32 แสดงโครงสร้างข้อมูลของระบบ