บทที่ 2

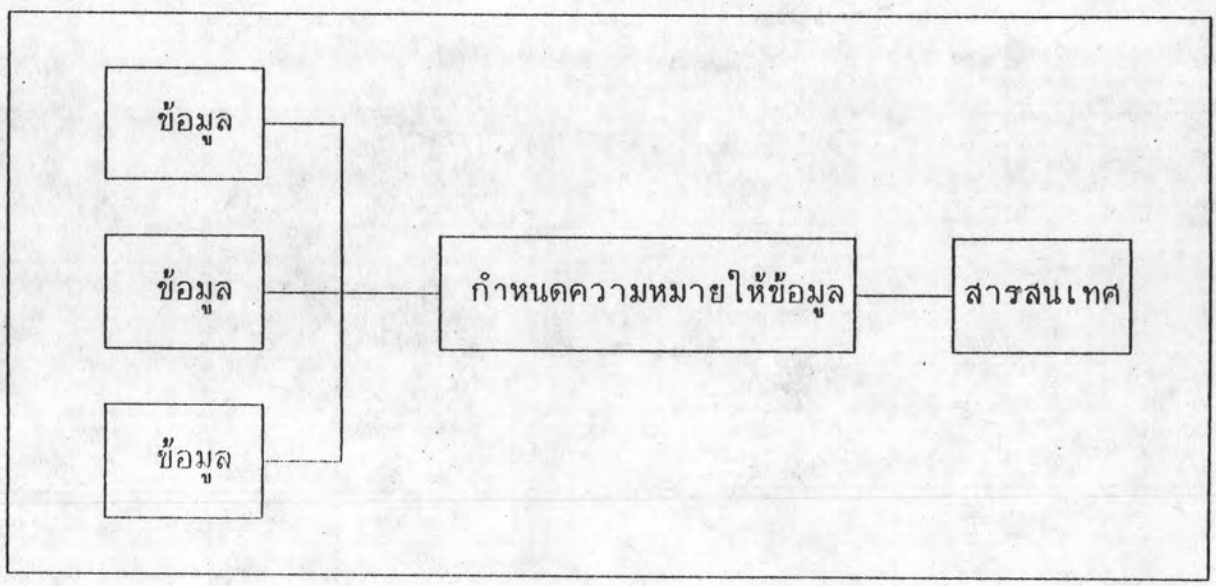
ทฤษฎีของการจัดการข้อมูล โครงสร้างข้อมูล และแฟ้มข้อมูล

ทฤษฎีของการจัดการข้อมูล

ในการบริหารงานใด ๆ ให้ประสพผลสำเร็จด้วยดีนั้น ข้อมูลเป็นปัจจัยสำคัญยิ่งปัจจัยหนึ่งที่ใช้เป็นเครื่องช่วยประกอบการตัดสินใจในการปฏิบัติงาน การบริหารข้อมูลให้ได้ระบบสารสนเทศเพื่อการจัดการถูกพัฒนาให้ทันสมัยตามเทคโนโลยีที่เจริญก้าวหน้าอย่างไม่หยุดยั้ง คอมพิวเตอร์ได้เข้ามามีบทบาทสำคัญในการประมวลผลข้อมูลเพื่อให้ได้สารสนเทศในรูปแบบที่ต้องการ เนื่องจากเครื่องคอมพิวเตอร์มีความสามารถในการจัดการเกี่ยวกับข้อมูลจำนวนมากอย่างมีประสิทธิภาพในการใช้งานด้านต่าง ๆ ทั้งในแง่ของการเข้าถึงข้อมูลและการแปลงข้อมูลให้อยู่ในรูปแบบที่มีความหมาย ซึ่งเราเรียกว่าสารสนเทศ (Information)

สารสนเทศ

สารสนเทศ คือ ข้อมูลที่ได้กำหนดความหมายสอดใส่ไว้ ดังแสดงในภาพที่ 2.1



ภาพที่ 2.1 แสดงลักษณะของสารสนเทศ

สารสนเทศอาจจะอยู่ในรูปแบบที่รวมเอาข้อมูลเกี่ยวกับเรื่องต่าง ๆ เช่น คน เครื่องจักร หรือสิ่งอื่น ๆ ดังตัวอย่างในภาพที่ 2.2 ซึ่งเป็นสารสนเทศของชายคนหนึ่งเกี่ยวกับประวัติส่วนตัวของเขา ดังนี้

"John Jones is an accountant who works for the ABC Company in department 5A. He is 40 years old, and he is married. His salary is $250 per week. He lives at 801 Main Street, Grovers Corners, N.Y. His social security number is 999-99-9999."

ภาพที่ 2.2 แสดงตัวอย่างสารสนเทศประวัติส่วนตัวของชายคนหนึ่ง

จากภาพที่ 2.2 สารสนเทศนี้ประกอบไปด้วยทั้งข้อมูล (data) และบทความ (context) หรือความหมายรวมกันอยู่ ซึ่งถ้าจะเขียนโดยแยกข้อมูลและบทความ(ความหมาย)ออกจากกันจะสามารถเขียนได้ดังในภาพที่ 2.3

"John Jones is the **NAME OF AN EMPLOYEE** whose **OCCUPATION** is Accountant. He works for **EMPLOYER** ABC Company in **DEPARTMENT** 5A. His **AGE** is 40 years, his **MARRITAL STATUS** is married. His **SALARY** is 801 Main Street, Grovers Corners, N.Y. His **SOCIAL SECURITY NUMBER** is 999-99-999."

ภาพที่ 2.3 ตัวอย่างสารสนเทศที่แยกข้อมูลและบทความ(ความหมาย)ออกจากกัน

จากภาพที่ 2.3 บทความ ได้แก่ข้อความที่พิมพ์ด้วยตัวพิมพ์ใหญ่และพิมพ์ตัวเน้น ซึ่งจะเห็นได้ว่า เป็นความหมายของข้อมูลนั่นเอง บทความจะเหมือนกันสำหรับเรื่องเดียวกัน ส่วน ข้อมูล ได้แก่ข้อความที่ไม่ได้พิมพ์ตัวเน้น ซึ่งจะเห็นได้ว่า คือรายละเอียดต่าง ๆ ของคนผู้นั้นตามบทความที่กำกับอยู่นั่นเอง ข้อมูลของคนอื่นก็อาจจะมีค่าเปลี่ยนไปไม่แน่นอน ทั้งนี้ขึ้นอยู่กับสารสนเทศนั้น

นอกจากเรื่องเกี่ยวกับบทความและข้อมูลแล้ว สิ่งที่น่าสนใจอีกอย่างหนึ่งก็คือ ข้อมูลถูกจัดเก็บและแสดงในรูปแบบใด ตัวอย่างเช่น ข้อมูลของบทความ หรือในเขตข้อมูล NAME OF AN EMPLOYEE ในภาพที่ 2.3 ได้แก่ John Jones ซึ่งแสดงในลักษณะที่เป็นตัวอักขระพยัญชนะ 4 อักขระแรกแทนชื่อ แล้วมีช่องว่างหนึ่งช่อง ตามด้วยอักขระพยัญชนะอีก 5 อักขระแทนนามสกุล เป็นต้น ลักษณะการแสดงข้อมูล (representation) ดังกล่าวจะทำให้ข้อมูลที่จัดเก็บอยู่สามารถสื่อความหมายได้ ดังนั้นสรุปได้ว่า สารสนเทศเกี่ยวกับรายการใด ๆ จะประกอบด้วยองค์ประกอบ 3 ส่วน ได้แก่ บทความ (context) ลักษณะการแสดงข้อมูล (representation) และข้อมูล (data) ดังแสดงในภาพที่ 2.4

INFORMATION = CONTEXT + REPRESENTATION + DATA

ภาพที่ 2.4 แสดงองค์ประกอบของสารสนเทศ

จากภาพที่ 2.3 เราสามารถจัดแยกสารสนเทศออกเป็น 3 ส่วน คือ บทความ, ลักษณะการแสดงข้อมูล และ ข้อมูล สรุปได้ดังนี้ มีบทความทั้งสิ้น 9 บทความ ได้แก่ ชื่อพนักงาน (NAME OF AN EMPLOYEE) อาชีพ (OCCUPATION), ชื่อบริษัท (EMPLOYER) แผนกที่ทำงาน (DEPARTMENT) อายุ (AGE) สถานภาพสมรส (MARRITAL STATUS) เงินเดือน (SALARY) ที่อยู่ (ADDRESS) และหมายเลข

ประจำตัว (SOCIAL SECURITY NUMBER) ซึ่งในการเก็บรายละเอียดเกี่ยวกับพนักงานคนใด ๆ แต่ละคนก็จะมีจำนวนบทความเท่ากันคือ 9 บทความ ส่วนลักษณะการแสดงข้อมูลภายในบทความสำหรับแต่ละคนก็จะเหมือนกัน สิ่งที่แตกต่างกันก็คือข้อมูล ลักษณะการแสดงข้อมูลภายในบทความ ส่วนใหญ่มักจะถูกกำหนดในลักษณะชนิดของข้อมูล และความยาวสูงสุดของข้อมูล จากสารสนเทศในภาพที่ 2.2 สามารถจัดแยกเป็นบทความ, ลักษณะการแสดงข้อมูล และข้อมูลได้ดังในภาพที่ 2.5

บทความ	ลักษณะการแสดงข้อมูล	ข้อมูล
NAME OF AN EMPLOYEE	อักขระพยัญชนะ ความยาวสูงสุด 20 อักขระ	John Jones
OCCUPATION	อักขระพยัญชนะ ความยาวสูงสุด 15 อักขระ	Accountant
EMPLOYER	อักขระพยัญชนะ ความยาวสูงสุด 20 อักขระ	ABC Company
DEPARTMENT	อักขระพยัญชนะ ความยาวสูงสุด 3 อักขระ	5A
AGE	อักขระตัวเลข ความยาวสูงสุด 3 อักขระ	40
MARRITAL STATUS	อักขระพยัญชนะ ความยาวสูงสุด 8 อักขระ	Married
SALARY	อักขระตัวเลข 5 หลัก มีทศนิยม 2 ตำแหน่ง	250.00
ADDRESS	อักขระพยัญชนะ ความยาวสูงสุด 45 อักขระ	801 Main Street, Grovers Corners, N.Y.
SOCIAL SECURITY NUMBER	ตัวเลขปนอักขระพิเศษ (-)ความยาว 10 อักขระ	999-99-999

ภาพที่ 2.5 แสดงสารสนเทศของข้อมูลพนักงานโดยแยกออกเป็น 3 ส่วน

<u>การจัดการข้อมูล</u> (Data Management)

การจัดการข้อมูล หมายถึงการกระทำใด ๆ ต่อข้อมูลเพื่อให้สามารถประมวลผลให้ได้สารสนเทศที่มีประสิทธิภาพ ซึ่งรวมถึง การวัด (measurement) การเก็บสะสม (Collection) การขยายความ (transcription) การตรวจสอบความถูกต้อง (validation) การจัดโครงสร้าง (organization) การจัดเก็บ (storage) การรวบรวมเป็นกลุ่ม (aggregation) การปรับปรุงแก้ไข (update) การดึงข้อมูล (retrieval) และการระวังป้องกัน (protection) ซึ่งสามารถสรุปเป็นแนวทางหลักได้ 4 ข้อคือ

1. ข้อมูลต้องถูกกำหนดและจัดเก็บให้อยู่ในรูปที่สามารถเรียกมาใช้งานได้ในภายหลัง (Data must be represented and stored so that they can be accessed later.)

2. ข้อมูลต้องถูกจัดรูปแบบให้สามารถเข้าถึงได้อย่างผ่านการเลือกได้และมีประสิทธิภาพ (Data must be organized so that they can be selectively and efficiently accessed.)

3. ข้อมูลต้องถูกประมวลผลและแสดงผลลัพธ์ซึ่งเป็นสารสนเทศที่ให้ประโยชน์สูงสุดแก่ผู้ใช้สารสนเทศนั้น (Data must be processed and presented so that they support the user environment effectively.)

4. ข้อมูลต้องถูกปกป้องและจัดการให้คงคุณค่าของข้อมูลนั้น ๆ ไว้ (Data must be protected and managed so that they retain their value.)

การจัดการข้อมูลโดยใช้คอมพิวเตอร์นั้น ประกอบด้วยลักษณะการจัดการข้อมูลในหน่วยความจำของคอมพิวเตอร์ซึ่งสามารถแบ่งออกได้เป็น 2 พวก คือ การจัดการข้อมูลในหน่วยความจำหลัก (Primary Storage Data Management) กับการจัดการข้อมูลในหน่วยความจำภายนอก (Secondary Storage Data Management) ซึ่งมีความแตกต่างกัน เนื่องจากเวลาที่ใช้ในการเข้าถึง (access)

ข้อมูลที่อยู่คนละตำแหน่งในหน่วยความจำหลักนั้นเท่ากัน แต่ถ้าเป็นข้อมูลที่อยู่คนละตำแหน่งในหน่วยความจำภายนอกจะใช้เวลาในการเข้าถึงแตกต่างกัน

โครงสร้างข้อมูล (Data Structure)

การใช้คอมพิวเตอร์ในการประมวลผลข้อมูลให้ได้สารสนเทศที่ต้องการนั้น เพื่อให้ตรงตามแนวทางหลักของการจัดการข้อมูล จะต้องจัดเก็บข้อมูลให้อยู่ในโครงสร้างข้อมูลที่เหมาะสม โครงสร้างข้อมูล หมายถึง การจัดโครงสร้าง ลำดับชั้น และรูปแบบของข้อมูลโดยวิธีต่าง ๆ ที่เหมาะสม เพื่อให้สามารถประมวลผลข้อมูลนั้น ๆ ได้อย่างมีประสิทธิภาพ ภาพที่ 2.6 แสดงถึงโครงสร้างข้อมูลชนิดต่าง ๆ ซึ่งบางโครงสร้างข้อมูลจัดเป็นพวกพื้นฐาน นั่นคือ เป็นโครงสร้างข้อมูลที่ไม่ได้ประกอบด้วยโครงสร้างข้อมูลอื่น ๆ เลย ปกติเรียกว่า ชนิดข้อมูลพื้นฐาน (primitive data types) ได้แก่ โครงสร้างข้อมูลชนิด ตัวเลข (integers) บูลีน (booleans) และอักขระ (characters) ส่วนโครงสร้างข้อมูลที่ได้จากการนำชนิดข้อมูลพื้นฐานมาประกอบกัน เรียกว่า ชนิดข้อมูลประกอบ (compound data type) ได้แก่ สตริงค์ (string)

ชนิดข้อมูลสามารถนำมาจัดได้หลายวิธี เพื่อสร้างเป็นโครงสร้างข้อมูลแบบต่าง ๆ ขึ้น โครงสร้างข้อมูลสามารถแบ่งออกได้เป็น 2 พวก ได้แก่ โครงสร้างข้อมูลทั่วไป (Simple Data Structure) และโครงสร้างข้อมูลประกอบ (Compound Data Structure)

โครงสร้างข้อมูลทั่วไป เป็นโครงสร้างข้อมูลพื้นฐานที่ประกอบด้วยชนิดข้อมูลต่าง ๆ ได้แก่ แถวลำดับ (Array) และระเบียน (Record)

โครงสร้างข้อมูลประกอบ เป็นการผสมผสานกันของโครงสร้างข้อมูลทั่วไปให้ได้โครงสร้างข้อมูลที่ซับซ้อนยิ่งขึ้น แบ่งออกได้เป็น 2 ประเภท คือ โครงสร้างข้อมูลประกอบเชิงเส้น (Linear Compound Data Structure) ได้แก่ ลิสต์เส้นตรง (Linear List) แสตก (Stack) แถวคอย (Queue) และ ลิงค์ลิสต์ (Linked

List) และ โครงสร้างข้อมูลประกอบไม่เชิงเส้น (Nonlinear Compound Data Structure) ได้แก่ กราฟ (Graph) ทรี (Tree) ไบนารีทรี (Binary Tree) บีทรี (B-TREE) และ บี*ทรี (B*TREE) เป็นต้น

โครงสร้างข้อมูลที่จัดสำหรับข้อมูลที่เก็บในหน่วยความจำภายนอก เรียกว่า การจัดองค์กรแฟ้มข้อมูล (File Organization) ที่เป็นที่นิยมใช้ในปัจจุบันได้แก่ การจัดองค์กรแฟ้มข้อมูลแบบเรียงลำดับ (Sequential File Organization) การจัดองค์กรแฟ้มข้อมูลแบบเข้าถึงโดยตรง (Relative File Organization) การจัดองค์กรแฟ้มข้อมูลแบบเรียงลำดับดัชนี (Indexed Sequential File Organization) และ การจัดองค์กรแฟ้มข้อมูลแบบหลายคีย์ (Multi-key File Organization)

การศึกษาโครงสร้างข้อมูล เป็นการศึกษาความสัมพันธ์ของ เขต ระเบียน ต่าง ๆ ทั้งภายในและระหว่างแฟ้มข้อมูล ในเชิงกายภาพ (Physical) และเชิงตรรก (Logical) เพื่อออกแบบให้โครงสร้างเชิงตรรกของข้อมูลถูกจัดเก็บในเชิงกายภาพของสื่อข้อมูลคอมพิวเตอร์อย่างมีประสิทธิภาพ เป็นการศึกษาคุณสมบัติของโครงสร้างข้อมูลต่าง ๆ การจัดสรรหน่วยความจำ การสร้างความสัมพันธ์ระหว่างข้อมูล ขั้นตอนวิธีในการสร้าง บันทึก เปลี่ยนแปลง เรียกใช้ ตลอดจนลบข้อมูล

ภาพที่ 2.6 แสดงโครงสร้างข้อมูลชนิดต่าง ๆ จะเห็นได้ว่า ทั้งชนิดข้อมูล โครงสร้างข้อมูล และโครงสร้างแฟ้มข้อมูลมีความสัมพันธ์กัน

1. <u>ชนิดข้อมูลพื้นฐาน</u>

ชนิดข้อมูลพื้นฐาน เป็นโครงสร้างข้อมูลอิสระที่ไม่ได้ประกอบขึ้นด้วยโครงสร้างข้อมูลต่างประเภทกันเลย ใช้สำหรับประกอบกันเป็นโครงสร้างข้อมูลที่ซับซ้อนเพื่อให้สื่อความหมายถึงสารสนเทศได้อย่างถูกต้องและมีประสิทธิภาพ ตัวอย่าง เช่น เลขจำนวนเต็ม บูลีน และอักขระ เป็นต้น

ชนิดข้อมูล (DATA TYPES)	พื้นฐาน (Primitive)	ตัวเลข (Integer) บูลีน (Boolean) อักขระ (Character)	
	ประกอบ (Compound)	สตริงค์ (String)	
โครงสร้างข้อมูล (Data Structure)	ทั่วไป (General)	แถวลำดับ (Array) ระเบียน (Record)	
	ประกอบ (Compound)	เชิงเส้น (Linear)	ลิสต์เส้นตรง (Linear List) แสตก (Stack) แถวคอย (Queue) ลิงค์ลิสต์ (Linked List)
		ไม่เชิงเส้น (Non-Linear)	กราฟ (Graph) ไบนารีทรี (BinaryTree) บีทรี (B-TREE) บี*ทรี (B*tree)

โครงสร้างแฟ้มข้อมูล (File Structure)	การจัดองค์กรแฟ้มข้อมูลแบบเรียงลำดับ (Sequential File Organization) การจัดองค์กรแฟ้มข้อมูลแบบเข้าถึงโดยตรง (Relative File Organization) การจัดองค์กรแฟ้มข้อมูลแบบเรียงลำดับดัชนี (Indexed Sequential File Organization) การจัดองค์กรแฟ้มข้อมูลแบบหลายคีย์ (Multi-key File Organization)

ภาพที่ 2.6 แสดงโครงสร้างข้อมูลชนิดต่าง ๆ

1.1 เลขจำนวนเต็ม

เลขจำนวนเต็ม เป็นชนิดข้อมูลพื้นฐานที่ประกอบด้วยเซทซึ่งมีสมาชิกประกอบด้วยตัวเลขต่อไปนี้

[..., -(n-1),-n,...,-2,-1,0,1,2,...,n,n+1,...]

การกระทำขั้นพื้นฐานกับชนิดข้อมูลเลขจำนวนเต็ม ได้แก่ การบวก การลบ การคูณ การหาร และ การยกกำลัง

1.2 บูลีน

ชนิดข้อมูลบูลีน บางครั้งเรียกว่าชนิดข้อมูลตรรก (Logical Data Type) เป็นโครงสร้างข้อมูลที่ประกอบด้วยข้อมูลที่หน่วยข้อมูลมีค่าได้เพียงค่าเดียวในสองค่าเท่านั้น คือ จริง (True) หรือ เท็จ (False) การกระทำขั้นพื้นฐานกับชนิดข้อมูลนี้ ได้แก่ การเชื่อมด้วยตัวกระทำ NOT, AND หรือ OR ซึ่งผลของการกระทำจะเป็นดังข้อมูลในตารางที่ 2.1 ซึ่งจะเห็นได้ว่า สำหรับตัวกระทำ AND นั้น ค่าข้อมูลทั้งสองค่าจะต้องเป็นจริง จึงจะทำให้ผลลัพธ์รวมเป็นจริง ส่วนตัวกระทำ OR นั้น มีค่าข้อมูลเพียงค่าหนึ่งเป็นจริงก็จะได้ผลลัพธ์รวมเป็นจริง ส่วนตัวกระทำ NOT นั้น จะกลับค่าของข้อมูลให้เป็นตรงกันข้าม เช่นจากจริงให้เป็นเท็จ หรือจากเท็จให้เป็นจริง เป็นต้น

นอกจากนี้ ผลลัพธ์จริง หรือ เท็จ ยังได้จากการเปรียบเทียบด้วยตัวกระทำเชิงความสัมพันธ์ (Relational Operator) ได้แก่เครื่องหมาย <, >, =, <=, >= และ <> ซึ่งมีความหมายดังนี้

เครื่องหมาย < หมายถึง น้อยกว่า
เครื่องหมาย > หมายถึง มากกว่า
เครื่องหมาย = หมายถึง เท่ากับ
เครื่องหมาย <= หมายถึง น้อยกว่า หรือ เท่ากับ
เครื่องหมาย >= หมายถึง มากกว่า หรือ เท่ากับ
เครื่องหมาย <> หมายถึง ไม่เท่ากับ

ค่าของตัวถูกกระทำตัวแรก	ค่าของตัวถูกกระทำตัวที่สอง	ตัวกระทำ		
		AND	OR	NOT*
จริง	จริง	จริง	จริง	เท็จ
จริง	เท็จ	เท็จ	จริง	เท็จ
เท็จ	จริง	เท็จ	จริง	จริง
เท็จ	เท็จ	เท็จ	เท็จ	จริง

* ข้อมูลจากตัวถูกกระทำตัวแรก

ตารางที่ 2.1 แสดงชนิดข้อมูลบูลีน

1.3 อักขระ

ชนิดข้อมูลอักขระ คือชนิดข้อมูลที่มีสมาชิกของเซทประกอบด้วยตัวเลข 0 - 9, ตัวอักษร a - z, A - Z และอักขระพิเศษอื่น ๆ เช่น ?, (,), ., *, +, -, /,... ตัวอย่างของชนิดข้อมูลอักขระได้แก่

```
[0, 1, 2,...,8, 9, a, b, c,...,z, A, B,...,Z, ?, ., *,...]
```

2. ชนิดข้อมูลประกอบ

ชนิดข้อมูลประกอบ คือชนิดข้อมูลที่เกิดจากการนำเอาชนิดข้อมูลพื้นฐานหลาย ๆ แบบมาประกอบกันขึ้นเป็นชนิดข้อมูล ได้แก่ ชนิดข้อมูลสตริงค์

2.1 สตริงค์

สตริงค์ คือ ชนิดข้อมูลที่เกิดจากการนำอักขระใด ๆ มาต่อกัน ปกติอยู่ภายในเครื่องหมายคำพูด ใช้แทนสารสนเทศใด ๆ นอกจากนี้ ยังใช้เป็นสื่อ

กลางในการเขียนโปรแกรมสั่งงานคอมพิวเตอร์ และมนุษย์ใช้สื่อสารข้อมูลถึงกัน ตัวอย่างสตริงค์ ได้แก่ "Computer", "13-59", "abcd" เป็นต้น การกระทำพื้นฐานของสตริงค์ ได้แก่ การหาขนาดสตริงค์ (Length) การรวมสตริงค์ (Concatenation) และ การแยกสตริงค์ (Substring)

2.1.1 การหาขนาดสตริงค์ เป็นการนับจำนวนอักขระที่ประกอบขึ้นเป็นสตริงค์ว่ามีทั้งสิ้นกี่ตัว โดยนับอักขรุทุกตัวในเครื่องหมายคำพูด รวมทั้งช่องว่างด้วย ตัวอย่างเช่น ขนาดของสตริงค์ "a2 3ycz" คือ 7 ขนาดของสตริงค์ "" คือ 0 เป็นต้น

2.1.2 การรวมสตริงค์ เป็นการนำสตริงค์สองสตริงค์มาเชื่อมต่อกันเป็นสตริงค์ใหม่ ที่มีขนาดเท่ากับขนาดของสตริงค์ทั้งสองที่มารวมกัน ตัวอย่างเช่น ผลจากการรวมสตริงค์ "xyz" กับ "5678" คือ "xyz5678" เป็นต้น

2.1.3 การแยกสตริงค์ เป็นการหยิบแยกสตริงค์ออกมาจากสตริงค์ที่กำหนดให้ เช่น ถ้าจะหยิบสตริงค์จำนวน 4 อักขระจากสตริงค์ "computer" โดยเริ่มหยิบตั้งแต่อักขระตัวแรกของสตริงค์ จะได้ผลลัพธ์ "comp" เป็นต้น

3. โครงสร้างข้อมูลทั่วไป

โครงสร้างข้อมูลทั่วไปเป็นโครงสร้างข้อมูลพื้นฐานที่ใช้แสดงสารสนเทศทั่ว ๆ ไป แบ่งเป็น 2 พวก ได้แก่ แถวลำดับ และระเบียน

3.1 แถวลำดับ

แถวลำดับเป็นโครงสร้างข้อมูลแบบหนึ่ง ที่มีลักษณะคล้ายเซทในวิชาคณิตศาสตร์ คือ ประกอบด้วยสมาชิก (element) ที่มีคุณสมบัติเหมือนกัน (ชนิดข้อมูลเดียวกัน) เก็บในลักษณะเรียงลำดับ โดยสามารถอ้างอิงถึงสมาชิกแต่ละตัวได้โดยการระบุชื่อและตัวเลขบอกลำดับว่าเป็นสมาชิกตัวใด เช่น แถวลำดับชื่อ A ซึ่งมีสมาชิก 6 ตัว อาจเขียนแทนด้วย A(1), A(2), A(3), A(4), A(5) และ A(6)

ดังแสดงในภาพที่ 2.7 เป็นต้น

A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
7	56	423	0	3342	590

ภาพที่ 2.7 แสดงแถวลำดับ A ซึ่งมีสมาชิก 6 ตัว

ตัวเลขในวงเล็บที่ระบุตำแหน่งของข้อมูลนั้น เรียกว่า ดัชนี (Index) หรือ สัปสคริปท์ (Subscript) แถวลำดับอาจมีดัชนีได้มากกว่า 1 ตัว ภายในวงเล็บ จำนวนของดัชนีของแถวลำดับเรียกว่า มิติ (Dimension) ของแถวลำดับ แถวลำดับมิติเดียวโดยทั่วไปมักเรียกว่า เวคเตอร์ (Vector) ส่วนแถวลำดับที่มีดัชนีมากกว่า 1 ตัวขึ้นไป เรียกว่า แถวลำดับหลายมิติ (Multi-dimensional Array) หรือ เมตริกซ์ (Matrix) การอ้างถึงข้อมูลในแถวลำดับ จะต้องอ้างถึงทั้งชื่อ และดัชนี ในกรณีที่เป็นแถวลำดับหลายมิติ จะต้องระบุดัชนีให้ครบทุกค่า

จากภาพที่ 2.7 เป็นแถวลำดับ 1 มิติ คือ แถวลำดับ A มีสมาชิก 6 ค่า คือ

A(1) เก็บข้อมูลชนิดตัวเลขคือ 7
A(2) เก็บข้อมูลชนิดตัวเลขคือ 56
A(3) เก็บข้อมูลตัวเลขคือ 423
A(4) เก็บข้อมูลตัวเลขคือ 0
A(5) เก็บข้อมูลตัวเลขคือ 3342
A(6) เก็บข้อมูลตัวเลขคือ 590

เป็นต้น

	B(1,1)	B(1,2)	B(1,3)	B(1,4)	
	ก	ข	ค	ง	
B(2,1)	จ	ฉ	ช	ซ	B(2,4)
B(3,1)	ฌ	ญ	บ	ป	B(3,4)
		B(3,2)	B(3,3)		

ภาพที่ 2.8 แสดงแถวลำดับ 2 มิติ ชนิด 3 X 4 (3 แถว 4 คอลัมน์)

จากภาพที่ 2.8 เป็นแถวลำดับ 2 มิติ คือ แถวลำดับ B เป็นชนิด 3 แถว ๆ ละ 4 คอลัมน์ มีสมาชิก 12 ค่า การอ้างถึงสมาชิก จะต้องระบุทั้งแถวและคอลัมน์ โดยดัชนีลำดับแรกในวงเล็บหมายถึงแถว และดัชนีลำดับที่สองในวงเล็บหมายถึงคอลัมน์ สมาชิกทั้ง 12 ค่าได้แก่

B(1,1) หมายถึงข้อมูลในแถวที่ 1 คอลัมน์ที่ 1 คืออักขระ ก
B(1,2) หมายถึงข้อมูลในแถวที่ 1 คอลัมน์ที่ 2 คืออักขระ ข
B(1,3) หมายถึงข้อมูลในแถวที่ 1 คอลัมน์ที่ 3 คืออักขระ ค
B(1,4) หมายถึงข้อมูลในแถวที่ 1 คอลัมน์ที่ 4 คืออักขระ ง
B(2,1) หมายถึงข้อมูลในแถวที่ 2 คอลัมน์ที่ 1 คืออักขระ จ
B(2,2) หมายถึงข้อมูลในแถวที่ 2 คอลัมน์ที่ 2 คืออักขระ ฉ
B(2,3) หมายถึงข้อมูลในแถวที่ 2 คอลัมน์ที่ 3 คืออักขระ ช
B(2,4) หมายถึงข้อมูลในแถวที่ 2 คอลัมน์ที่ 4 คืออักขระ ซ
B(3,1) หมายถึงข้อมูลในแถวที่ 3 คอลัมน์ที่ 1 คืออักขระ ฌ
B(3,2) หมายถึงข้อมูลในแถวที่ 3 คอลัมน์ที่ 2 คืออักขระ ญ
B(3,3) หมายถึงข้อมูลในแถวที่ 3 คอลัมน์ที่ 3 คืออักขระ บ
B(3,4) หมายถึงข้อมูลในแถวที่ 3 คอลัมน์ที่ 4 คืออักขระ ป

เป็นต้น

3.2 ระเบียน

ระเบียน เป็นโครงสร้างข้อมูลที่เกิดจากการนำหน่วยข้อมูลที่มีความสัมพันธ์กันในเรื่องเดียวกันมารวมเข้าเป็นกลุ่มเรียงกันไป โดยหน่วยข้อมูลที่นำมารวมกันนั้นไม่จำเป็นต้องเป็นชนิดข้อมูลเดียวกัน หน่วยข้อมูลแต่ละหน่วยที่ประกอบขึ้นเป็นระเบียนนั้น เรียกว่า เขต (Field)

ตำแหน่ง	รหัสพนักงาน	อัตราค่าจ้าง
นักวิเคราะห์ระบบ-1	1278359	8000

ภาพที่ 2.9 แสดงตัวอย่างโครงสร้างข้อมูลระเบียนที่ประกอบด้วย 3 เขต

ระเบียนในภาพที่ 2.9 ประกอบด้วย 3 เขต ดังนี้

เขต **ตำแหน่ง** เก็บข้อมูลชนิดสตริงค์ ได้แก่ "นักวิเคราะห์ระบบ-1"
เขต **รหัสพนักงาน** เก็บข้อมูลชนิดตัวเลข ได้แก่ 1278359
เขต **อัตราค่าจ้าง** เก็บข้อมูลชนิดตัวเลข ได้แก่ 8000

เป็นต้น

ตำแหน่ง	รหัสพนักงาน	อัตราค่าจ้าง	รหัสโครงการ				
เสมียน	3549864	4000	18	41	50	54	59

ภาพที่ 2.10 แสดงตัวอย่างระเบียนที่ประกอบด้วยโครงสร้างข้อมูลหลาย ๆ ประเภท

ระเบียนอาจจะประกอบด้วยโครงสร้างข้อมูลหลาย ๆ ประเภท ได้ ตัวอย่างเช่น จากภาพที่ 2.10 ระเบียนตัวอย่างนี้นอกจากจะประกอบด้วยชนิดข้อมูลพื้นฐานต่าง ๆ แล้ว ยังประกอบด้วยแถวลำดับ รหัสโครงการอีกด้วย ซึ่งเป็นแถวลำดับ 1 มิติ ขนาด 5 ตัว ประกอบด้วยข้อมูลชนิดตัวเลข ได้แก่ 18, 41, 50, 54 และ 59 เป็นต้น

ระเบียนแบ่งออกได้เป็น 2 พวกใหญ่ ๆ คือ ระเบียนความยาวคงที่ (Fixed-length record) และ ระเบียนความยาวแปรผัน (Variable-length record)

3.2.1 ระเบียนความยาวคงที่

ระเบียนความยาวคงที่ คือ ระเบียนที่มีเขตและระเบียนที่มีขนาดไม่เปลี่ยนแปลง ตำแหน่งของเขตต่าง ๆ ได้ถูกกำหนดไว้คงที่ในแต่ละระเบียนของแฟ้ม ระเบียนความยาวคงที่เป็นแบบที่นิยมใช้กัน เพราะง่ายต่อการประมวลผลและควบคุม ตัวอย่างดังในภาพที่ 2.10 เป็นต้น

3.2.2 ระเบียนความยาวแปรผัน

ระเบียนความยาวแปรผัน คือระเบียนที่มีขนาดไม่คงที่ สามารถแบ่งออกได้เป็น 2 ลักษณะ คือ

3.2.2.1 ระเบียนประกอบด้วยหนึ่งหรือมากกว่าหนึ่งเขตที่มีความยาวแปรผัน

สำหรับงานที่มีขนาดของแฟ้มใหญ่มาก ๆ อาจจะมีความจำเป็นในการลดขนาดของแฟ้มลงเพื่อให้ประหยัดเนื้อที่เก็บข้อมูล อาจใช้ระเบียนที่ประกอบด้วยเขตที่มีความยาวแปรผัน โดยเก็บข้อมูลเท่าที่เป็นจริง ไม่ปล่อยที่ว่างให้สูญเปล่า ตัวอย่างดังภาพที่ 2.11 จะเห็นว่าแต่ละเขตของแต่ละระเบียนใช้เนื้อที่แตกต่างกันขึ้นกับข้อมูลจริง ๆ ว่าใช้เนื้อที่เท่าไร

1		2		3		4		5	
12	สุดาวัลย์	16	45	วิชัย	8	56	นพ	12	164	สันติ	18	273	ดำรง	15

ภาพที่ 2.11 แสดงระเบียนที่เขตมีความยาวแปรผัน

3.2.2.2 <u>ระเบียนประกอบด้วยเขตที่มีความยาวคงที่แต่จำนวนแปรผัน</u>

ในบางครั้ง ลักษณะของสารสนเทศที่จะจัดเก็บ มีขนาดของเขตแต่ละเขตเท่ากัน แต่จำนวนเขตบางเขตไม่แน่นอน ต้องใช้ลักษณะระเบียนที่ประกอบด้วยเขตข้อมูลที่มีความยาวคงที่แต่จำนวนแปรผัน ดังตัวอย่างในภาพที่ 2.12

ตำแหน่ง	รหัสพนักงาน	อัตราค่าจ้าง	รหัสโครงการ									
เสมียน	3549864	4000	18	41	50	54	59	.	.	.	.	n

ภาพที่ 2.12 แสดงตัวอย่างระเบียนที่ประกอบด้วยเขตที่มีความยาวคงที่แต่จำนวนแปรผัน

การรวมหลาย ๆ ระเบียนที่เป็นเรื่องเดียวกันเข้าด้วยกัน จะได้หน่วยข้อมูลใหม่ เรียกว่า แฟ้มข้อมูล

การกระทำใด ๆ กับระเบียน ก็ขึ้นอยู่กับว่าระเบียนนั้น ๆ ประกอบด้วยชนิดข้อมูลอะไร ก็จะสามารถกระทำได้ตามชนิดข้อมูลนั้น ๆ

4. โครงสร้างข้อมูลประกอบ

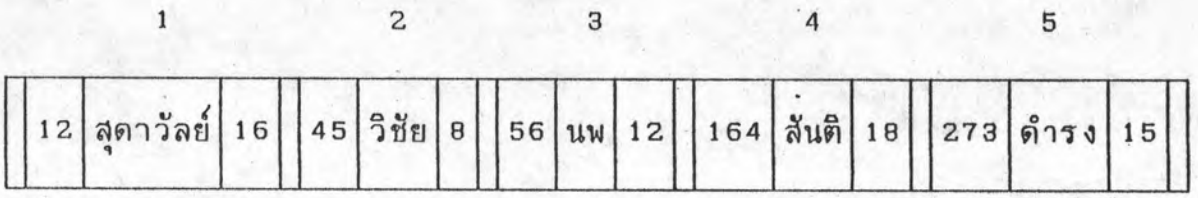
โครงสร้างข้อมูลประกอบ เกิดจากการนำโครงสร้างข้อมูลพื้นฐานมาประกอบกันขึ้นเพื่อให้สามารถนำไปใช้งานได้อย่างเหมาะสม แบ่งได้เป็น 2 พวกตามความซับซ้อน คือ ชนิดเชิงเส้น (Linear) และชนิดไม่เชิงเส้น (Nonlinear)

4.1 โครงสร้างข้อมูลประกอบเชิงเส้น

โครงสร้างข้อมูลประกอบเชิงเส้น ได้แก่ โครงสร้างข้อมูลที่เกิดจากการประกอบกันของโครงสร้างข้อมูลพื้นฐาน ในลักษณะที่ความสัมพันธ์เชิงตรรกของหน่วยข้อมูลมีลักษณะเรียงลำดับกันไป ที่ใช้กันแพร่หลายได้แก่ ลิสต์เส้นตรง แสตก แถวคอย และ ลิงค์ลิสต์

4.1.1 ลิสต์เส้นตรง

เป็นลักษณะโครงสร้างข้อมูลที่ประกอบด้วยหน่วยข้อมูลขนาดไม่ตายตัวเรียงลำดับกันไป ข้อมูลตัวที่ 2 จะอยู่ต่อจากข้อมูลตัวแรก ข้อมูลตัวที่ 3 จะอยู่ต่อจากข้อมูลตัวที่ 2 เช่นนี้ไปเรื่อย ๆ การเข้าถึงข้อมูลจะต้องผ่านข้อมูลที่อยู่ก่อนหน้านั้นมาเป็นลำดับ ดังตัวอย่างในภาพที่ 2.13



ภาพที่ 2.13 แสดงลิสต์เส้นตรงที่ประกอบด้วยระเบียน 5 ระเบียน

จากภาพที่ 2.13 เป็นลิสต์เส้นตรงที่ประกอบด้วยโครงสร้างข้อมูลระเบียนจำนวน 5 ระเบียน แต่ละระเบียนประกอบด้วย 3 เขตข้อมูล จะเห็นได้ว่าขนาดของเขตข้อมูลไม่จำเป็นต้องเท่ากัน และการเข้าถึงระเบียนจะต้องผ่านระเบียนเรียงตามลำดับที่อยู่ติด ๆ กันเป็นเสมือนเส้นตรง การปฏิบัติการกับลิสต์ทั่ว ๆ ไปมี 4 แบบ คือ การเพิ่มข้อมูลเข้าสู่ลิสต์ การลบข้อมูลออกจากลิสต์ การค้นหาข้อมูลในลิสต์ และการเปลี่ยนแปลงข้อมูลในลิสต์

4.1.1.1 <u>การเพิ่มข้อมูลเข้าสู่ลิสต์เส้นตรง</u>

ในกรณีที่เป็นการเพิ่มข้อมูลเข้าที่ท้ายลิสต์ก็จะไม่มีปัญหาอะไร สามารถกระทำได้ง่ายและไม่เสียเวลา แต่ถ้าเป็นการเพิ่มข้อมูลใหม่เข้าตรงกลางลิสต์เส้นตรง จะต้องเลื่อนข้อมูลตรงกลางขยับให้มีที่ว่างสำหรับข้อมูลใหม่ที่จะแทรกเข้ามา ซึ่งโดยเฉลี่ยแล้ว จำนวนครั้งของการเลื่อนข้อมูลจะเท่ากับครึ่งหนึ่งของขนาดของลิสต์เส้นตรง ดังนั้น ยิ่งข้อมูลมีปริมาณมากก็จะต้องเสียเวลามาก

4.1.1.2 <u>การลบข้อมูลออกจากลิสต์เส้นตรง</u>

ในกรณีที่เป็นการลบข้อมูลที่ท้ายลิสต์ก็จะไม่มีปัญหาอะไร สามารถกระทำได้ง่ายและไม่เสียเวลา แต่ถ้าเป็นการลบข้อมูลตรงกลางลิสต์เส้นตรง จะต้องเลื่อนข้อมูลตรงกลางขยับเข้าแทนที่ข้อมูลที่ลบทิ้งไป ซึ่งโดยเฉลี่ยแล้ว จำนวนครั้งของการเลื่อนข้อมูลจะเท่ากับครึ่งหนึ่งของขนาดของลิสต์เส้นตรง ดังนั้น ยิ่งข้อมูลมีปริมาณมากก็จะต้องเสียเวลามากเช่นเดียวกับการเพิ่มข้อมูล

4.1.1.3 <u>การค้นหาข้อมูลในลิสต์เส้นตรง</u>

การค้นหาข้อมูลในลิสต์ โดยทั่วไปก็เพื่อค้นหาข้อมูลใดข้อมูลหนึ่งที่มีคุณสมบัติตรงตามเงื่อนไขที่ต้องการ เช่นจากตัวอย่างในภาพที่ 2.11 เป็นลิสต์เส้นตรงที่เก็บข้อมูล รหัส ชื่อ และอายุ ของนักเรียน 6 คน ซึ่งเราอาจจะต้องการทราบว่าสุดาอายุเท่าไร เป็นต้น ลักษณะการค้นหาข้อมูลในลิสต์เส้นตรง จะต้องเริ่มทำการค้นหาโดยดูจากข้อมูลแรกสุดไล่ไปเรื่อย ๆ จนกว่าจะพบหรือหมดข้อมูลในลิสต์ ถ้าหมดข้อมูลแล้วยังไม่พบแสดงว่าไม่มีข้อมูลนั้น ๆ ในลิสต์ ลักษณะการค้นหาเช่นนี้ เรียกว่า การค้นหาแบบเส้นตรง (Linear Search) หรือการค้นหาแบบเรียงลำดับ (Sequential Search) โดยเฉลี่ยแล้ว จะต้องทำการ

ค้นหาประมาณครึ่งหนึ่งของขนาดของลิสต์ ทั้งนี้เพราะข้อมูลที่กำลังค้นหามีโอกาสที่จะถูกค้นพบได้ทั้งในครึ่งแรกและครึ่งหลังของลิสต์ ทำให้เสียเวลามากถ้าลิสต์มีขนาดใหญ่ จึงมีวิธีการค้นหาแบบไบนารี (Binary Search) ที่ช่วยลดเวลาในการค้นหาลงมาก โดยเรียงลำดับข้อมูลในลิสต์ตามข้อมูลที่จะค้นหาไว้ล่วงหน้า จากนั้น เวลาจะค้นหาข้อมูลใด ก็ทำการแบ่งลิสต์เป็นสองส่วนที่กลางลิสต์ แล้วเอาค่าตรงกลางลิสต์ไปเปรียบเทียบกับค่าที่ต้องการค้นหาเพื่อดูว่าค่าที่ต้องการค้นหานั้นตกอยู่ในส่วนใดของลิสต์ เช่น ถ้าพบว่ามากกว่าค่าตรงกลางลิสต์ แสดงว่าข้อมูลอยู่ในลิสต์ส่วนทางขวา ก็ไม่ต้องเสียเวลาไปค้นหาในลิสต์ส่วนซ้าย แต่มาแบ่งครึ่งลิสต์ขวาเป็นสองส่วนแล้วเปรียบเทียบ ทำแบบนี้ไปเรื่อย ๆ จนพบว่าค่าที่แบ่งครึ่งเท่ากับค่าที่ค้นหา ก็จะได้ข้อมูลอื่น ๆ ตามต้องการ จะพบว่า ถ้าข้อมูลมีทั้งสิ้น n หน่วย จะใช้เวลาในการค้นหาทั้งสิ้นเพียง $\log_2 n$ ครั้งเท่านั้น นั่นคือ ถ้าลิสต์มี 100 หน่วยข้อมูล จำนวนครั้งของการเปรียบเทียบจะมีได้มากที่สุด 10 ครั้งเท่านั้น ก็จะพบว่าข้อมูลที่กำลังค้นหามีอยู่ในลิสต์หรือไม่

4.1.1.4 การเปลี่ยนแปลงข้อมูลในลิสต์เส้นตรง

การเปลี่ยนแปลงข้อมูลในลิสต์ เป็นการค้นหาข้อมูลในลิสต์ แล้วบันทึกข้อมูลใหม่เข้าไปแทนนั่นเอง ดังนั้นจึงประกอบด้วยขั้นตอนการค้นหาแล้วบันทึกข้อมูลใหม่เข้าไปแทนที่ของเดิมนั่นเอง

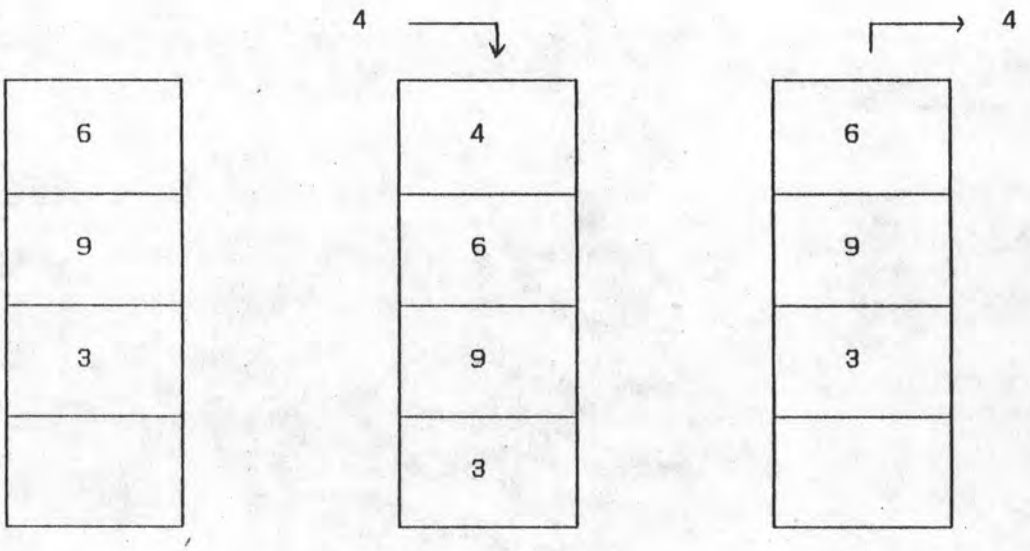
4.1.2 แสตก (Stack)

แสตก คือ ลิสต์เส้นตรงที่มีลักษณะพิเศษในด้านการเพิ่มข้อมูลและการลบข้อมูล จะไม่มีการเพิ่มหรือลบข้อมูลที่อยู่ตรงกลาง ทำได้เฉพาะที่ท้ายลิสต์เท่านั้น โดยจะกระทำเพียงด้านเดียวเท่านั้น ซึ่งเรียกว่า ยอดแสตก (Top of stack) ดังนั้น ข้อมูลจะถูกกระทำหรือเข้าถึงในลักษณะ "มาหลังไปก่อน" (Last in, First out) นั่นคือ ข้อมูลสุดท้ายที่เพิ่มเข้ามาในลิสต์ก็จะเป็นข้อมูลแรกที่ถูกดึงออกมาก่อนเสมอ

ตัวอย่างเครื่องใช้ที่ทำงานด้วยระบบการปฏิบัติการแบบแสตกที่แสดงการทำงานได้ชัดเจน ได้แก่ ช่องวางถาดอาหารแบบวางซ้อนกันเป็นกองในโรงอาหาร ซึ่งภายในช่องวางถาดนี้ จะมีขดลวดอยู่ข้างล่าง ซึ่งจะดันให้ถาดอาหารใบที่อยู่ข้างบนสุดในกองลอยอยู่ที่ระดับคงที่บนช่องวางถาดอาหารเสมอ เมื่อวาง

ถาดลงไปบนกองถาด ขดลวดจะหดสั้นลง และเมื่อหยิบถาดบนสุดออก ขดลวดจะขยายตัวออก ถาดอาหารที่วางซ้อนกันในช่องวางถาดนี้เปรียบเสมือนข้อมูลของแสตก การเพิ่มข้อมูลใหม่ที่ยอดแสตกก็คือ การวางถาดใหม่ลงบนกองถาดเก่า ข้อมูลเก่าในแสตกจะถูกดันลงไปข้อมูลละตำแหน่ง เพื่อให้ข้อมูลใหม่อยู่ในตำแหน่งยอดแสตก

ขบวนการเพิ่มข้อมูลเข้าแสตกเรียกว่า การพุชค่าลงแสตก (Push) ซึ่งหมายถึง ดันลงไป และการดึงข้อมูลออกจากยอดแสตก ก็เปรียบเสมือนการหยิบถาดอาหารออกจากกอง ข้อมูลภายในแสตกก็จะลอยตัวสูงขึ้นมาข้อมูลละหนึ่งตำแหน่ง ขบวนการดึงข้อมูลออกจากแสตกเรียกว่า การพอพ (Pop) ตัวอย่างแสตกจะเป็นดังภาพที่ 2.14 ซึ่งจะประกอบด้วย ภาพย่อย ๆ 3 ภาพ คือ ภาพ ก. เป็นตัวอย่างแสตกในขณะหนึ่ง สมมติว่ามีข้อมูลอยู่แล้ว 3 ค่า คือ 3, 9 และ 6 ตามลำดับ จากภาพ แสดงว่า 3 ถูกพุชลงแสตกเป็นลำดับแรก ตามด้วย 9 และ 6 ทำให้ค่า 6 อยู่ที่ยอดแสตก ภาพ ข. เป็นภาพที่แสดงการพุชข้อมูล 4 เพิ่มลงไปในแสตก จะเห็นว่ามีการเลื่อนข้อมูลอื่น ๆ ลงไปข้างล่าง ส่วนภาพ ค. เป็นภาพที่แสดงการพอพค่า 4 ออกจากแสตก จะเห็นว่ามีการเลื่อนข้อมูลขึ้นหนึ่งตำแหน่ง และ ข้อมูล 6 จะอยู่ที่ยอดแสตกแทนที่ 4 เดิมที่ถูกดึงค่าออกไปจากแสตกแล้ว ถ้ามีการดึงข้อมูลออกจากแสตกอีกก็จะได้ค่า 6, 9 และ 3 ตามลำดับ



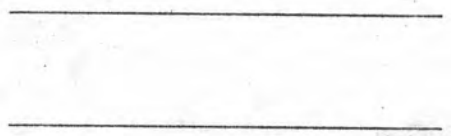
ก. แสตกขณะหนึ่ง ข. เมื่อพุชค่า 4 ค. เมื่อพอพค่า 4

ภาพที่ 2.14 ภาพแสดงโครงสร้างข้อมูลแบบแสตก

4.1.3 แถวคอย

แถวคอย เป็นโครงสร้างข้อมูลแบบลิสต์ชนิดที่การเพิ่มข้อมูลจะถูกกระทำที่ปลายข้างหนึ่ง และการดึงข้อมูลจะถูกกระทำที่ปลายอีกข้างหนึ่ง จะไม่มีการเพิ่มหรือลบข้อมูลที่อยู่ตรงกลางลิสต์ ดังนั้น ข้อมูลจะถูกกระทำในลักษณะ "มาก่อน ไปก่อน" (First-in, First-out, FIFO) นั่นคือ ข้อมูลที่ถูกเพิ่มเข้าในลิสต์เป็นลำดับแรกก็จะถูกดึงออกก่อนข้อมูลอื่นเสมอ โครงสร้างข้อมูลแบบแถวคอยใช้มากในส่วนต่าง ๆ ของระบบปฏิบัติการของเครื่องคอมพิวเตอร์ เช่น แถวคอยจัดระบบงาน (Job Queue) แถวคอยรอพิมพ์ผลลัพธ์จากเครื่องพิมพ์ (Printer Output Queue) เป็นต้น ปลายข้างที่เพิ่มข้อมูลของแถวคอย เรียกว่า หางแถวคอย (tail) และปลายข้างที่ดึงข้อมูลของแถวคอย เรียกว่า หัวแถวคอย (head)

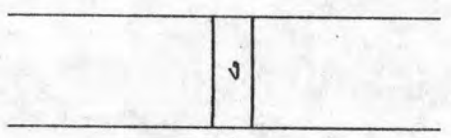
นั่นคือ ถ้าเขียนในรูปเซทคณิตศาสตร์ แถวคอย Q ประกอบด้วยสมาชิก $Q_1, Q_2, \ldots, Q_T$ เมื่อ หัวแถวคอยอยู่ที่ Q_1 และท้ายแถวคอยอยู่ที่ Q_T โดยจำนวนสมาชิกแถวคอย คือ T



จำนวนสมาชิกในแถวคอย = 0

ภาพที่ 2.15 แสดงแถวคอยว่างเปล่า

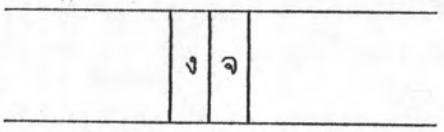
ภาพที่ 2.15 แสดงแถวคอยว่างเปล่าที่ยังไม่มีข้อมูล เมื่อมีการเพิ่มข้อมูลในแถวคอย เช่น เพิ่ม ง จำนวนสมาชิกจะเป็น 1 และหัวแถวคอยจะถูกกำหนดให้ชี้ที่ข้อมูล ง เช่นเดียวกับหางแถวคอย ดังภาพที่ 2.16



จำนวนสมาชิกในแถวคอย = 1
หัวแถวคอย = ง
ท้ายแถวคอย = ง

ภาพที่ 2.16 แสดงแถวคอยเมื่อเพิ่มข้อมูล ง

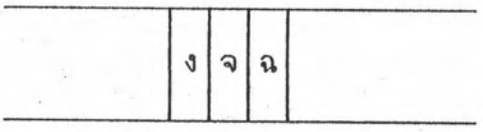
เมื่อเพิ่มข้อมูล จ เข้าไปในแถวคอย หางแถวคอยจะเลื่อนมาอยู่ที่ จ ดังภาพที่ 2.17



จำนวนสมาชิกในแถวคอย = 2
หัวแถวคอย = ง
ท้ายแถวคอย = จ

ภาพที่ 2.17 แสดงแถวคอยเมื่อเพิ่มข้อมูล จ

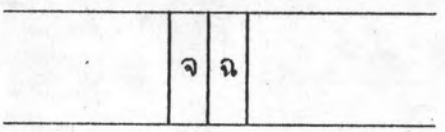
เมื่อเพิ่มข้อมูล ฉ จะได้แถวคอยดังภาพที่ 2.18



จำนวนสมาชิกในแถวคอย = 3
หัวแถวคอย = ง
ท้ายแถวคอย = ฉ

ภาพที่ 2.18 แสดงแถวคอยเมื่อเพิ่มข้อมูล ฉ

ถ้าลบข้อมูลในแถวคอย ข้อมูลที่ถูกลดลำดับแรกคือ ข้อมูลที่หัวแถวคอย นั่นคือ ง จะเหลือแถวคอยดังภาพที่ 2.19



จำนวนสมาชิกในแถวคอย = 2
หัวแถวคอย = จ
ท้ายแถวคอย = ฉ

ภาพที่ 2.19 แสดงแถวคอยเมื่อลบข้อมูล ง

การปฏิบัติการขั้นพื้นฐานกับแถวคอยมี 4 แบบ คือ การสร้างแถวคอย การตรวจสอบว่ามีข้อมูลในแถวคอยหรือไม่ การเพิ่มข้อมูลในแถวคอย และ การลบข้อมูลออกจากแถวคอย

4.1.3.1 <u>การสร้างแถวคอย</u>

กจรสร้างแถวคอย เป็นการกำหนดให้สร้างแถวคอยที่ว่างเปล่าขึ้นมาใช้งาน ตามข้อกำหนดดังนี้

CREATE(Q) จะสร้างแถวคอย Q ซึ่งมีคุณสมบัติดังภาพที่ 2.15

จำนวนข้อมูล = 0
หัวแถวคอย และหางแถวคอย ยังไม่กำหนด

4.1.3.2 <u>การตรวจสอบแถวคอยว่ามีข้อมูลหรือไม่</u>

การตรวจสอบแถวคอยว่ามีข้อมูลหรือไม่ เป็นการกำหนดให้ทำการทดสอบดูว่าแถวคอยที่ระบุ มีข้อมูลอยู่ หรือว่างเปล่า ถ้าว่างเปล่า ผลการตรวจสอบจะเป็นจริง มิฉะนั้น ผลการตรวจสอบจะเป็นเท็จ

4.1.3.3 <u>การเพิ่มข้อมูลในแถวคอย</u>

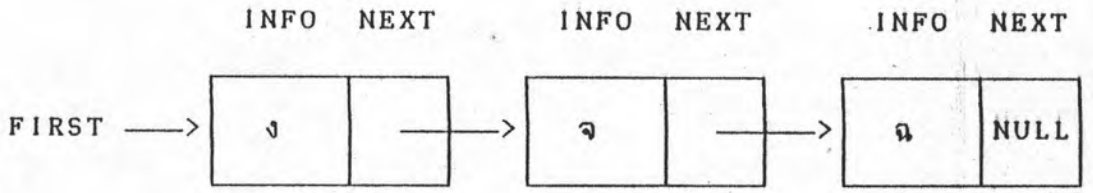
การเพิ่มข้อมูลในแถวคอย เป็นการเพิ่มข้อมูลเข้าที่ท้ายแถวคอย และต้องเพิ่มค่าในจำนวนข้อมูลของแถวคอย ดังตัวอย่างในภาพที่ 2.16

4.1.3.4 <u>การลบข้อมูลในแถวคอย</u>

การลบข้อมูลในแถวคอย เป็นการลบข้อมูลออกจากหัวแถวคอย และต้องลดค่าในจำนวนข้อมูลของแถวคอย ดังตัวอย่างในภาพที่ 2.19

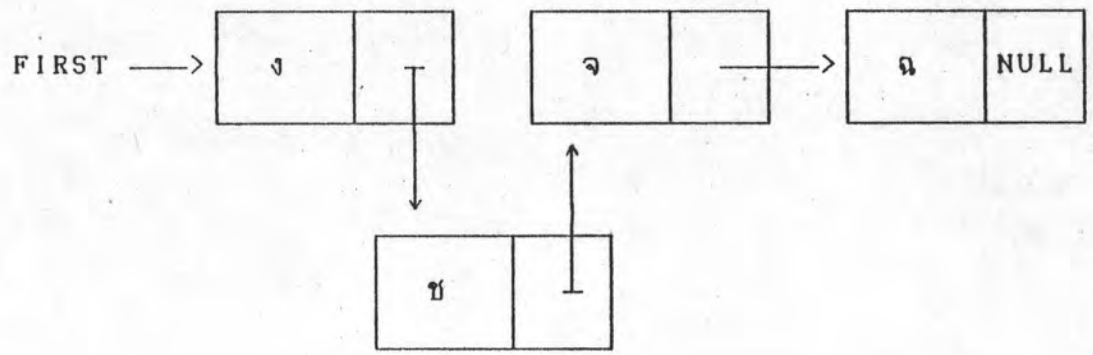
4.1.4 ลิงค์ลิสต์

ลิงค์ลิสต์ เป็นโครงสร้างข้อมูลแบบลิสต์ชนิดที่ความสัมพันธ์ของข้อมูลยังคงเรียงลำดับกัน แต่ตำแหน่งของข้อมูลจริงไม่ได้อยู่เรียงลำดับกันเสมอไป โดยมีตัวชี้เพื่อชี้ข้อมูลตัวถัดไปแทน เพื่อแก้ปัญหาของการเพิ่มหรือลดข้อมูลของโครงสร้างข้อมูลเชิงเส้นอื่น ๆ ที่ข้อมูลจริงจะต้องอยู่เรียงลำดับกัน ทำให้มีการขยับข้อมูลมาก ๆ ในกรณีที่มีการเพิ่ม หรือลดข้อมูลที่อยู่ช่วงกลางลิสต์



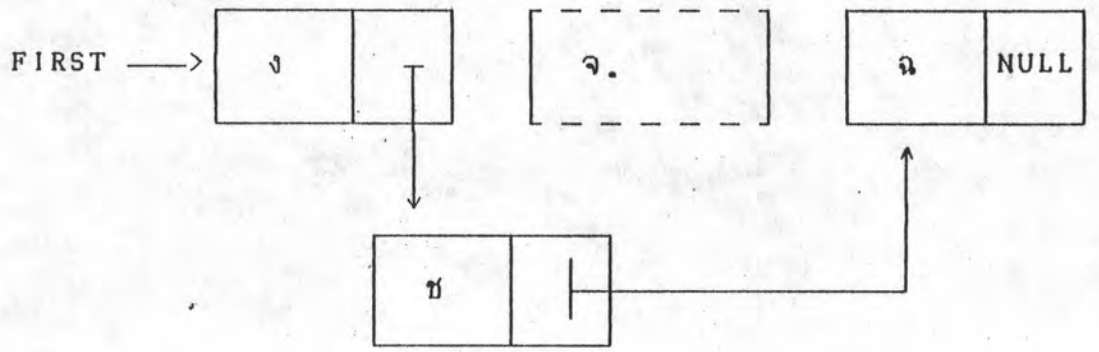
ภาพที่ 2.20 แสดงตัวอย่างลิงค์ลิสต์ที่มีข้อมูล 3 ค่า

ภาพที่ 2.20 แสดงตัวอย่างลิงค์ลิสต์ที่มีข้อมูล ง, จ และ ฉ จะเห็นได้ว่า มีตัวชี้ชื่อ FIRST เป็นหน่วยข้อมูลแรกของลิสต์ แต่ละหน่วยข้อมูลจะเป็นระเบียนที่ประกอบไปด้วย 2 ส่วน คือ ข้อมูล (INFO) และตัวชี้ไปยังระเบียนถัดไป (NEXT) ที่ระเบียนสุดท้าย ตัวชี้ไปยังระเบียนถัดไปจะเป็น NULL ซึ่งแสดงว่าไม่มีข้อมูลต่อจากนี้อีกแล้ว



ภาพที่ 2.21 แสดงการเพิ่มข้อมูล ช ระหว่างข้อมูล ง กับ จ

ภาพที่ 2.21 แสดงตัวอย่างลิงค์ลิสต์ที่มีข้อมูล ง ช จ และ ฉ โดยการเพิ่มข้อมูล ช แทรกระหว่างข้อมูล ง กับ จ จะเห็นได้ว่าไม่มีการขยับตำแหน่งข้อมูลเดิมเลย เพียงแต่ย้ายตัวชี้ข้อมูลของ ง ให้ชี้ไปที่ ช แทนที่จะเป็น จ แล้วให้ตัวชี้ของ ช ชี้ไปที่ จ แทน



ภาพที่ 2.22 แสดงการลบข้อมูล จ ออกจากลิงค์ลิสต์

ภาพที่ 2.22 แสดงตัวอย่างการลบข้อมูล จ ออกจากลิงค์ลิสต์ จะเห็นได้ว่า ไม่มีการขยับตำแหน่งข้อมูลเดิม เพียงแต่ย้ายตัวชี้ข้อมูลของ ช ให้ชี้ไปที่ ฉ แทนที่จะเป็น จ เท่านั้น

การปฏิบัติการขั้นพื้นฐานกับลิงค์ลิสต์ได้แก่ การลบข้อมูลในลิงค์ลิสต์ และการเพิ่มข้อมูลในลิงค์ลิสต์ โดยกำหนดให้ P เป็นตัวแปรที่เป็นตัวชี้ซึ่งเก็บข้อมูลตำแหน่ง การปฏิบัติการกับตัวแปรที่เป็นตัวชี้กระทำได้ 4 แบบ คือ การการตรวจสอบว่ามีข้อมูลเป็น NULL หรือไม่ การทดสอบค่าข้อมูลตัวแปรที่เป็นตัวชี้ การกำหนดให้เป็น NULL และ การกำหนดตัวชี้ไปยังข้อมูล กำหนดโดย

NODE(P) คือ ระเบียนที่ถูกชี้โดยตัวแปร P ซึ่งแบ่งออกเป็น 2 ส่วน คือ INFO(P) กับ NEXT(P)

INFO(P) คือ ค่าข้อมูลที่ถูกชี้โดยตัวแปร P

NEXT(P) คือ ค่าตัวชี้ข้อมูลของระเบียนที่ถูกชี้โดยตัวแปร P

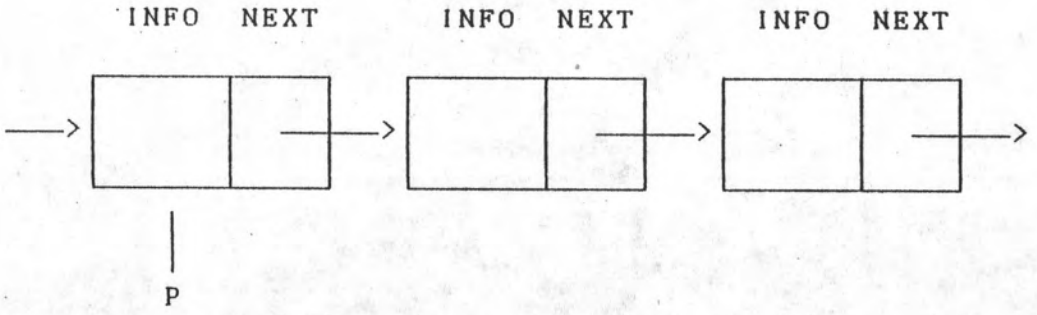
4.1.4.1 การลบข้อมูลในลิงค์ลิสต์

การลบข้อมูลในลิงค์ลิสต์ โดยลบข้อมูลที่อยู่ถัดจากที่ถูกระบุโดย P (ในตัวอย่างนี้ ใช้ Q เป็นตัวแปรที่เป็นตัวชี้ชั่วคราว) มีขั้นตอนดังนี้

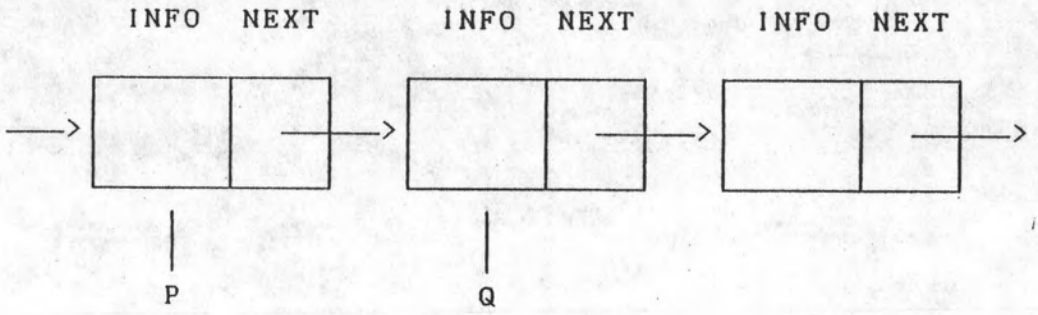
ขั้นตอนที่ 1 Q := NEXT(P)

ขั้นตอนที่ 2 NEXT(P) := NEXT(Q)

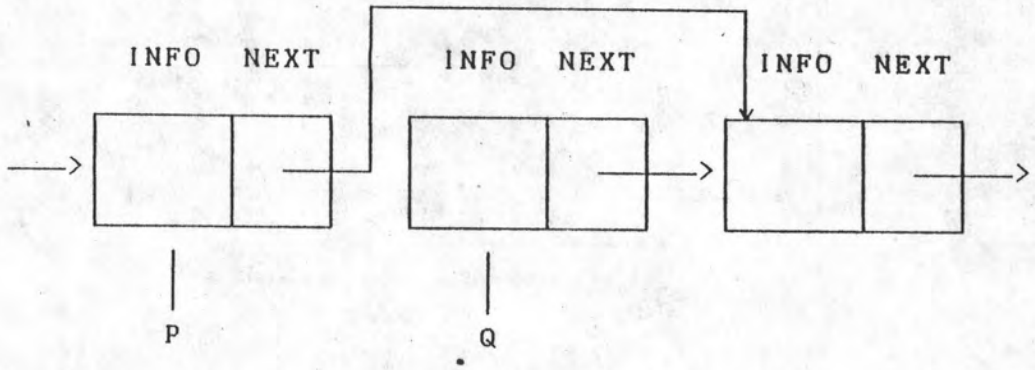
ขั้นตอนที่ 3 FREENODE(Q)



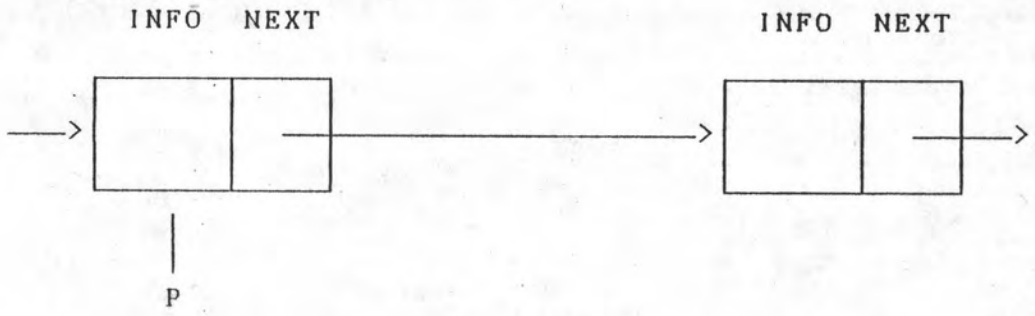
ภาพที่ 2.23 แสดงตัวอย่างลิงค์ลิสต์ก่อนการย้ายค่า



ภาพที่ 2.24 แสดงลิงค์ลิสต์ภายหลังทำงานขั้นตอนที่ 1



ภาพที่ 2.25 แสดงลิงค์ลิสต์ภายหลังทำงานขั้นตอนที่ 2



ภาพที่ 2.26 แสดงลิงค์ลิสต์ภายหลังทำงานขั้นตอนที่ 3

ขั้นตอนที่ 3 เป็นการเรียกฟังก์ชัน FREENODE ซึ่งสมมติให้เป็นการเอาโหนดข้อมูลที่ระบุออกจากระบบ นั่นคือ โปรแกรมที่จะนำโครงสร้างข้อมูลแบบลิงค์ลิสต์มาใช้จะต้องเตรียมแอ่ง (pool) สำหรับเก็บโหนดข้อมูลอิสระที่ยังไม่ถูกใช้หรือที่เลิกใช้เอาไว้ เวลาจะเพิ่มข้อมูลก็จะต้องมาเอาโหนดว่างจากแอ่งไปใส่ข้อมูล โดยสมมติว่าใช้ฟังก์ชัน GETNODE และเวลาลบโหนดใด ๆ ออกจากลิงค์ลิสต์ ก็จะต้องใช้ฟังก์ชัน FREENODE เพื่อคืนโหนดข้อมูลที่ไม่ใช้มาเก็บไว้ในแอ่งตามเดิม

4.1.4.2 การเพิ่มข้อมูลในลิงค์ลิสต์

การเพิ่มข้อมูลในลิงค์ลิสต์ โดยเพิ่มข้อมูล

ตัวแปร NAME เข้าไปในลิงค์ลิสต์ ตามหลังระเบียนที่ถูกระบุโดย P (ในตัวอย่างนี้ใช้ Q เป็นตัวแปรที่เป็นตัวชี้ชั่วคราว) มีขั้นตอนดังนี้

ขั้นตอนที่ 1 NEW := GETNODE

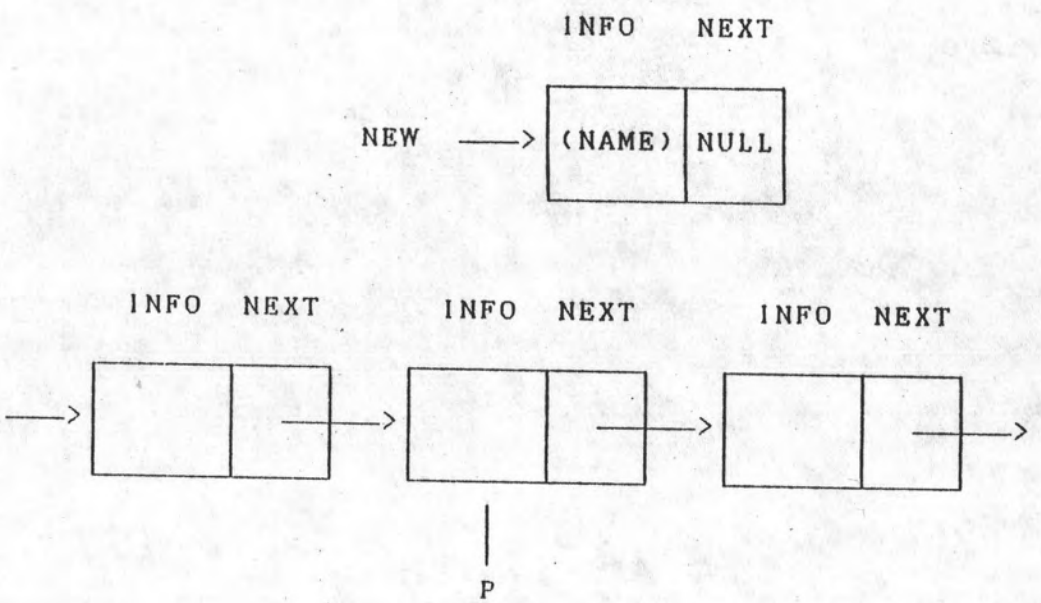
ขั้นตอนที่ 2 INFO(NEW) := NAME

ขั้นตอนที่ 3 Q := NEXT(P)

ขั้นตอนที่ 4 NEXT(P) := NEW

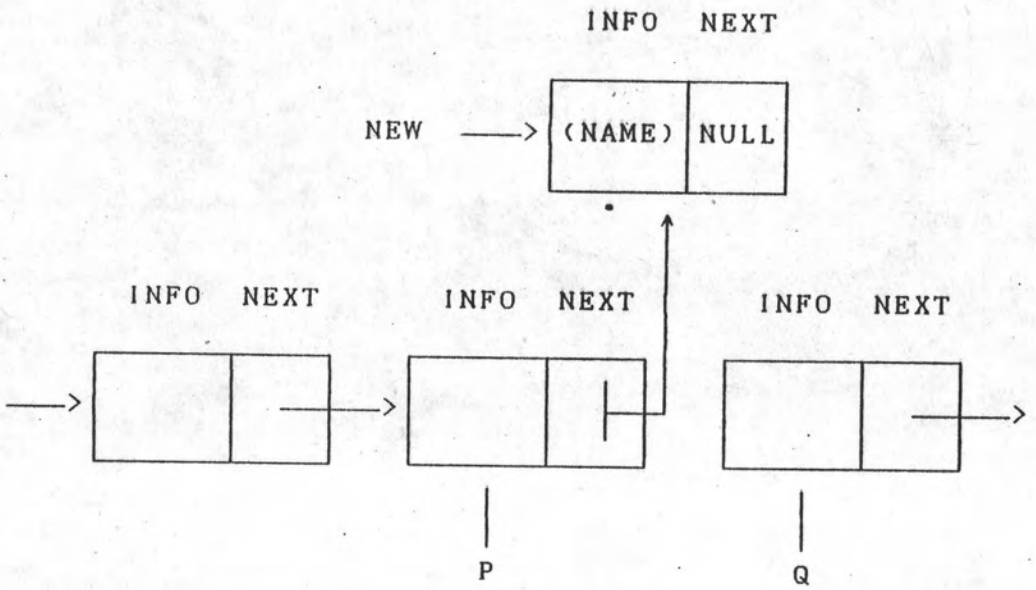
ขั้นตอนที่ 5 NEXT(NEW) := Q

ในขั้นตอนที่ 1 ผลของฟังก์ชัน GETNODE คือ จะได้ NEW เป็นตำแหน่งของโหนดว่าง ซึ่งจะต้องถูกกำหนดค่าให้เป็น NAME ในขั้นตอนที่ 2 เมื่อผ่าน 2 ขั้นตอนนี้ จะได้โครงสร้างดังแสดงในภาพที่ 2.27



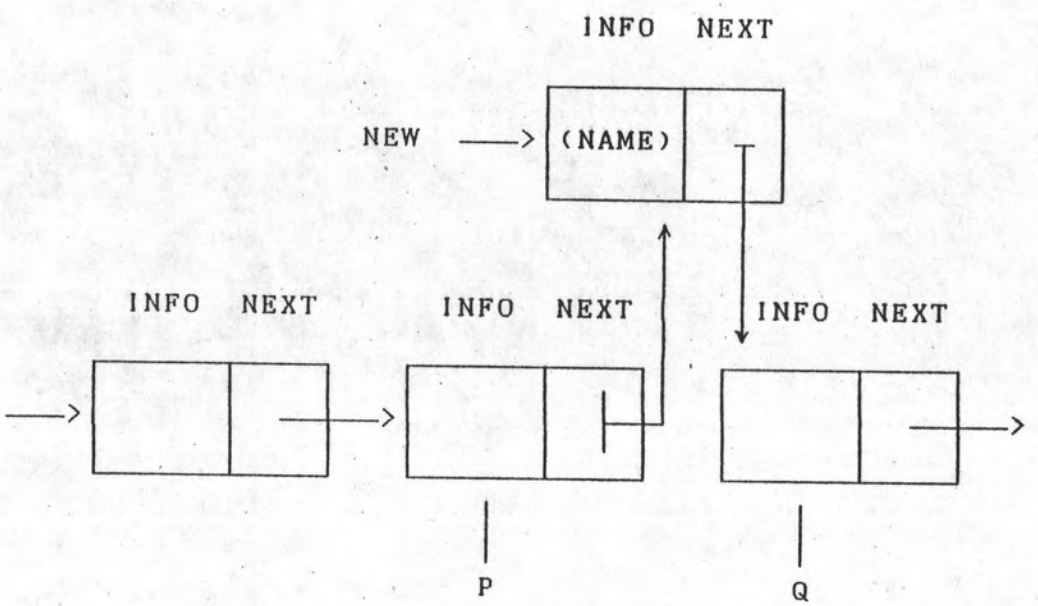
ภาพที่ 2.27 แสดงลิงค์ลิสต์เมื่อผ่านขั้นตอนที่ 1 และ 2

ในขั้นตอนที่ 3 และ 4 จะเป็นการกำหนดตัวชี้ใหม่ ได้โครงสร้างดังภาพที่ 2.28



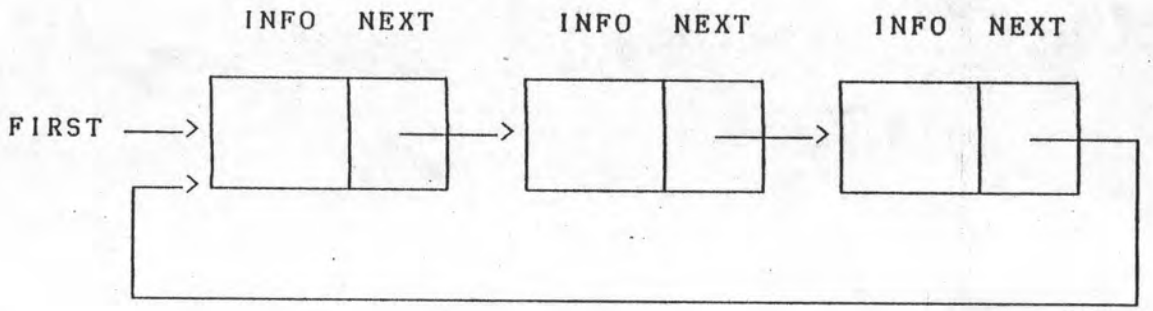
ภาพที่ 2.28 แสดงลิงค์ลิสต์เมื่อผ่านขั้นตอนที่ 3 และ 4

ในขั้นตอนที่ 5 เป็นการปรับตัวชี้ของโหนดใหม่ให้ชี้ไปยังโหนดที่อยู่ถัดจาก P ก็จะได้โครงสร้างดังแสดงในภาพที่ 2.29



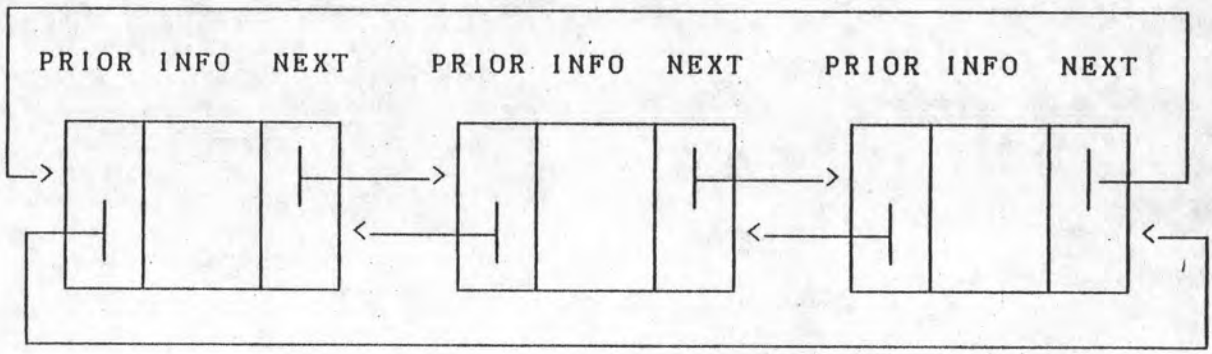
ภาพที่ 2.29 แสดงลิงค์ลิสต์เมื่อผ่านขั้นตอนที่ 5

ถ้าต้องการปฏิบัติการกับทุก ๆ โหนดในลิงค์ลิสต์ จะต้องเริ่มต้นที่ต้นลิสต์เสมอ เพราะถ้าเริ่มที่ตรงกลาง จะปฏิบัติการกับโหนดไปได้เรื่อย ๆ จนถึงโหนดสุดท้ายก็จะจบแค่นั้น (ตัวชี้มีค่าเป็น NULL) ไม่สามารถย้อนมาเริ่มที่ต้นลิสต์ได้ จึงมีการโยงข้อมูลจากโหนดสุดท้าย คือแทนที่ตัวชี้จะมีค่าเป็น NULL ก็ให้ชี้มาที่ตำแหน่งเริ่มต้นลิสต์แทน มีลักษณะเชื่อมกันเป็นวงกลม เรียกว่า เซอร์คิวลาร์ลิสต์ (Circular List) ดังตัวอย่างในภาพที่ 2.30



ภาพที่ 2.30 แสดงตัวอย่างเซอร์คิวลาร์ลิสต์

ลิงค์ลิสต์มีข้อจำกัดตรงที่ว่า ถ้าต้องการทราบว่าโหนดที่อยู่ลำดับก่อนหน้านี้มีข้อมูลอะไร จะไม่มีทางทราบได้ทันที แต่จะต้องทำการค้นหาโดยเริ่มตั้งแต่โหนดแรกในลิสต์ใหม่ ซึ่งอาจจะเสียเวลามาก สามารถแก้ไขได้โดยใช้ลิงค์ลิสต์ชนิดพิเศษ เรียกว่า ดับบลิลิงค์ลิสต์ (Doubly Linked List) ซึ่งเป็นลิงค์ลิสต์ชนิดพิเศษที่แต่ละโหนดมีตัวชี้ 2 ตัว ตัวชี้ตัวแรกชี้กลับไปยังโหนดที่อยู่ก่อนหน้า และตัวชี้ตัวที่สองชี้ไปยังโหนดที่อยู่ถัดไป ดังตัวอย่างในภาพที่ 2.31

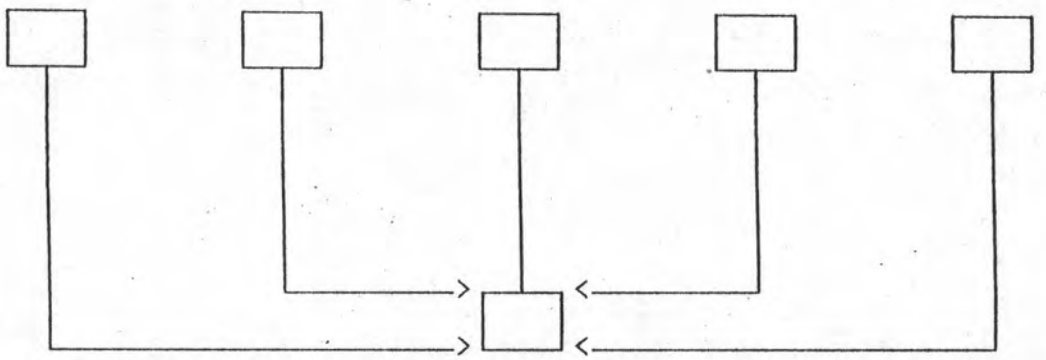


ภาพที่ 2.31 แสดงตัวอย่างดับบลิลิงค์ลิสต์

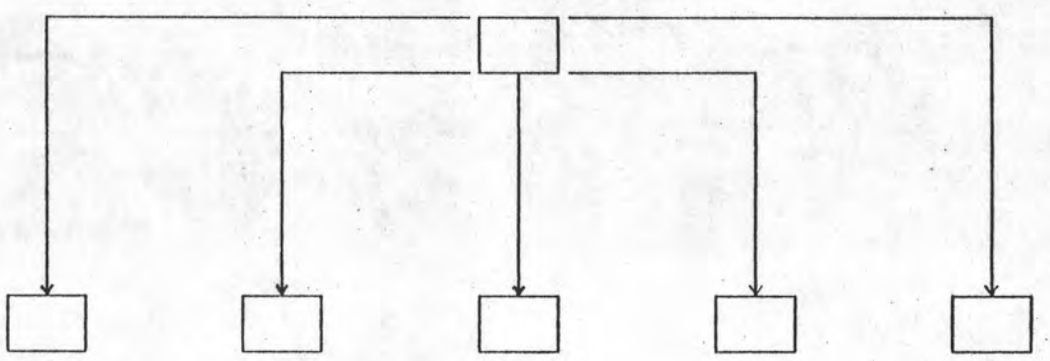
4.2 โครงสร้างข้อมูลประกอบไม่เชิงเส้น

โครงสร้างข้อมูลประกอบไม่เชิงเส้น ได้แก่ โครงสร้างข้อมูลที่เกิดจากการประกอบกันของโครงสร้างข้อมูลพื้นฐาน ในลักษณะที่ความสัมพันธ์เชิงตรรกของหน่วยข้อมูลหน่วยหนึ่งอาจสัมพันธ์กับหน่วยข้อมูลอื่น ๆ มากกว่าหนึ่งหน่วย หรือในทางกลับกันได้ จึงไม่สามารถนำโหนดมาเรียงลำดับตามความสัมพันธ์ให้อยู่ในรูปเส้นตรงเดียวกันได้ ดังตัวอย่างในภาพที่ 2.32 .

ภาพที่ 2.32 (ก) เป็นภาพแสดงความสัมพันธ์ของหน่วยข้อมูลในลักษณะที่ข้อมูลหลายหน่วย มีความสัมพันธ์กับข้อมูลหน่วยหนึ่ง ส่วนภาพที่ 2.32 (ข) เป็นภาพแสดงความสัมพันธ์ของหน่วยข้อมูลในลักษณะที่ข้อมูลหน่วยหนึ่งมีความสัมพันธ์กับข้อมูลหลายหน่วย



ภาพที่ 2.32 (ก) แสดงความสัมพันธ์ของหน่วยข้อมูลในลักษณะที่ข้อมูลหลายหน่วย มีความสัมพันธ์กับข้อมูลหน่วยหนึ่ง

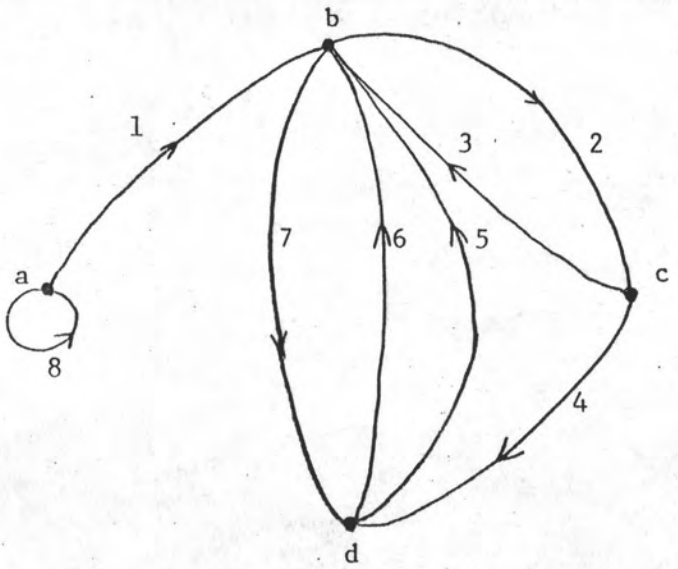


ภาพที่ 2.32(ข) แสดงความสัมพันธ์ของหน่วยข้อมูลในลักษณะที่ข้อมูลหนึ่งหน่วย มีความสัมพันธ์กับข้อมูลหลายหน่วย

โครงสร้างข้อมูลประกอบไม่เชิงเส้นที่นิยมใช้กันทั่ว ๆ ไป ได้แก่ กราฟ ทรี ไบนารีทรี บีทรี บี*ทรี เป็นต้น

4.2.1 กราฟ

กราฟเป็นโครงสร้างข้อมูลประกอบไม่เชิงเส้นที่ประกอบด้วยกลุ่มของโหนดข้อมูล และกลุ่มของลิงค์ ซึ่งเป็นเส้นเชื่อมความสัมพันธ์ระหว่างโหนดข้อมูลที่มีความสัมพันธ์กัน ดังตัวอย่างในภาพที่ 2.33



ภาพที่ 2.33 แสดงตัวอย่างกราฟลักษณะหนึ่ง

จากภาพที่ 2.33 ถ้ากำหนดให้ V_G เป็นเซทของโหนด และให้ E_G เป็นเซทของลิงค์ ของกราฟ G จะได้

$$V_G = \{a,b,c,d\}$$

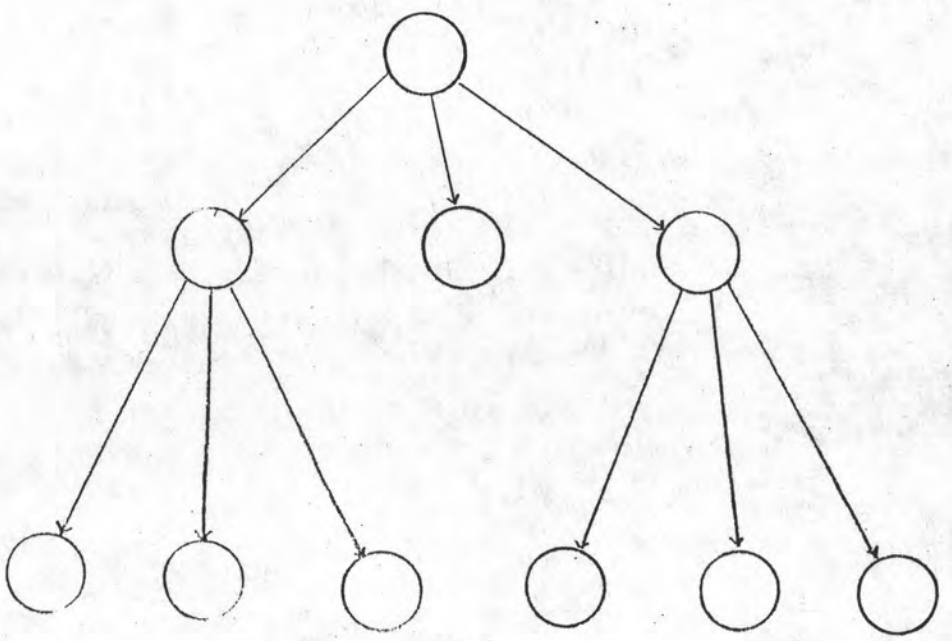
$$E_G = \{1,2,3,4,5,6,7,8\}$$

จำนวนสมาชิกของ V_G เรียกว่า ลำดับของกราฟ G ในตัวอย่างนี้คือ 4 กราฟที่มีลำดับเป็น 0 เรียกว่า กราฟว่าง (Null Graph)

ลิงค์ ถูกกำหนดโดยโหนดที่เป็นจุดเชื่อมโยง ตัวอย่าง เช่น ลิงค์ 4 เชื่อมโยงระหว่างโหนด c และโหนด d เรียกว่า form (c,d) จะเห็นได้ว่า อาจจะมีหลายลิงค์ระหว่าง 2 โหนดใด ๆ เช่น ระหว่างโหนด b กับโหนด d มีลิงค์หมายเลข 5, 6 และ 7 เป็นต้น หรือบางคู่ของโหนดอาจไม่มีความสัมพันธ์กัน เช่นโหนด a กับโหนด d และบางโหนดมีการเชื่อมโยงกับตัวเอง ซึ่งเรียกว่า ลูป (loops) เช่น ลิงค์ 8 เป็นต้น

4.2.2 ทรี

ทรีเป็นโครงสร้างข้อมูลลักษณะเดียวกับกราฟ แต่มีข้อจำกัดเกี่ยวกับลิงค์มากกว่ากราฟ นั่นคือ ทุก ๆ โหนดข้อมูลในทรีจะต้องถูกเชื่อมด้วยลิงค์ ซึ่งลิงค์ระหว่างโหนด 2 โหนดจะมีได้อย่างมากเพียงลิงค์เดียว และลิงค์ระหว่างโหนดจะวกกลับเป็นวงจากโหนด ๆ หนึ่งแล้วกลับมายังโหนดที่ถูกลิงค์แล้วนี้อีกไม่ได้ ตัวอย่างทรี ดังแสดงในภาพที่ 2.34

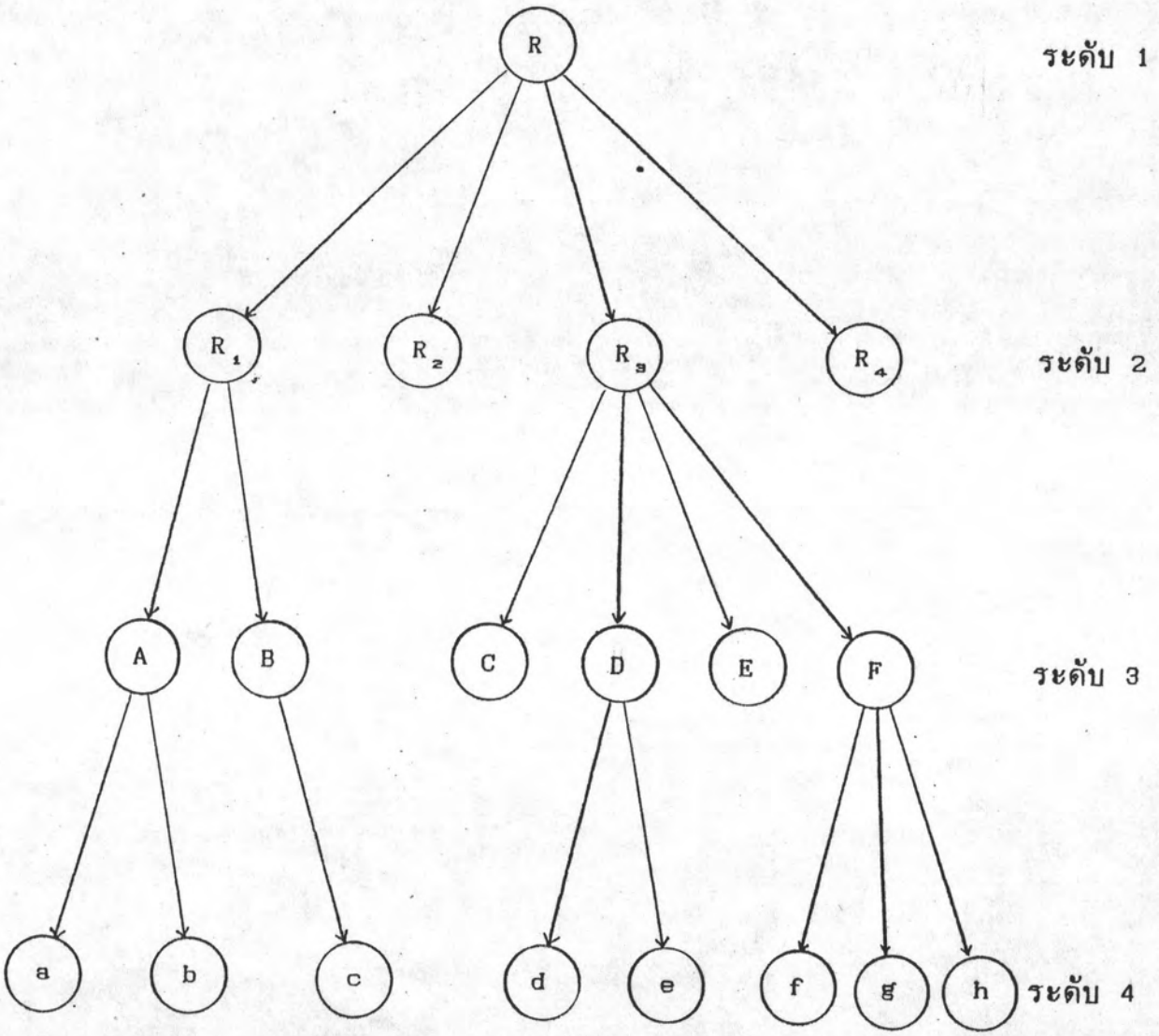


ภาพที่ 2.34 แสดงตัวอย่างของทรีลักษณะหนึ่ง

ทรีเป็นโครงสร้างข้อมูลที่แสดงความสัมพันธ์ของข้อมูลด้วยลำดับขั้นของความสัมพันธ์เป็นระดับ ๆ ซึ่งจะมีกี่ระดับก็ได้ แล้วแต่ลักษณะของข้อมูลโดยทั่วไป แต่ละโหนดในทรีจะมีความสัมพันธ์กับโหนดในระดับต่ำลงมาหนึ่งระดับได้หลาย ๆ โหนด หรืออาจกล่าวได้ว่าแต่ละโหนดของทรีอาจเป็นโหนดแม่ (parent node, mother node) ของโหนดลูก (child node) ในระดับต่ำลงมาหนึ่งระดับได้หลาย ๆ โหนด โหนดต่าง ๆ ที่อยู่ในระดับสูงขึ้นไปของโหนดหนึ่ง ๆ จะเรียกว่าโหนดมาก่อน (predecessor nodes) และโหนดต่าง ๆ ที่อยู่ในระดับต่ำลงมาของโหนดหนึ่ง ๆ จะเรียกว่า โหนดมาทีหลัง (successor node) โหนดต่าง ๆ ในระดับเดียวกันที่มาจากโหนดแม่เดียวกันจะเรียกว่า เป็นพี่น้องกัน (siblings) และจะอยู่ต่ำกว่าโหนดแม่ หนึ่งระดับเสมอ

โหนดทุกโหนดจะต้องมีโหนดแม่เพียงโหนดเดียวเท่านั้น ยกเว้นโหนดแรก หรือโหนดในระดับสูงสุดของทรี ซึ่งไม่มีโหนดแม่ เรียกโหนดนี้ว่า โหนดราก (Root node) โหนดที่มีโหนดลูก เรียกว่า โหนดกิ่ง (Branch node) และโหนดที่ไม่มีโหนดลูก เรียกว่า โหนดใบ (Leaf node) เส้นเชื่อมแสดงความสัมพันธ์ระหว่างโหนดสองโหนดเรียกว่า กิ่ง (branch) หรือลิงค์ (link) ความสูงระหว่างโหนดสองโหนดใด ๆ ในทรี ก็คือ ผลต่างระหว่างระดับทั้งสองของโหนด ภาพที่ 2.35 แสดงส่วนประกอบต่าง ๆ ของทรี และแสดงระดับของโหนด ดังกล่าว จากภาพจะเห็นได้ว่าโหนดรากคือ R ซึ่งเป็นโหนดแม่ของโหนด R_1, R_2, R_3 และ R_4 หรือกล่าวได้ว่า R_1, R_2, R_3 และ R_4 เป็นโหนดลูกของโหนด R นั่นเอง โหนด C,D,E และ F เป็นโหนดพี่น้องกัน โหนด R_2, R_4, C, E, a, b, c, d, e, f, g และ h เป็นโหนดใบ ความยาวระหว่างโหนด R_1 กับโหนด d เท่ากับ 4 ลบ 2 เท่ากับ 2

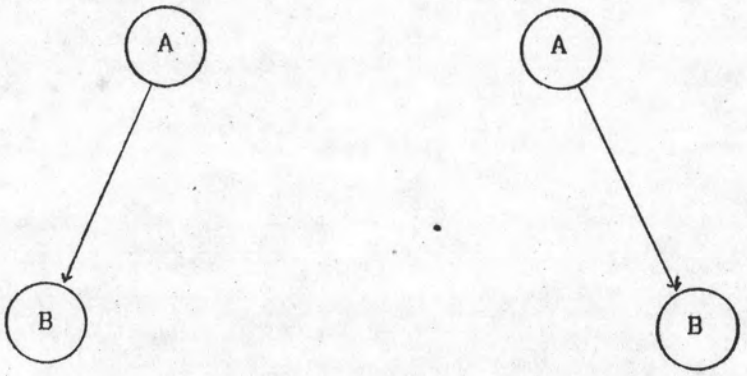
ถ้าตัดแต่ละโหนดใด ๆ ออกจากทรี จะได้ส่วนที่ตัดออกเป็นทรีทุกประการโดยมีโหนดที่ถูกตัดออกเป็นโหนดราก เรียกส่วนที่ถูกตัดออกนี้ว่า สับทรี (subtree) ดังนั้น ในภาพที่ 2.35 โหนดราก R จึงประกอบด้วย 4 สับทรี แต่ละสับทรีมี R_1, R_2, R_3 และ R_4 เป็นโหนดรากตามลำดับ โหนด R_1 มี 2 สับทรี โหนด R_2 ไม่มีสับทรี โหนด R_3 มี 4 สับทรี และโหนด R_4 ไม่มีสับทรี เป็นต้น จำนวนสับทรีของโหนดหนึ่ง ๆ เรียกว่า ดีกรีของโหนดนั้น ๆ เช่น ดีกรีของโหนดราก R คือ 4 หรือ ดีกรีของโหนด R_1 คือ 2 เป็นต้น



ภาพที่ 2.35 แสดงส่วนประกอบต่าง ๆ ของทรี และแสดงระดับของโหนด

4.2.3 ไบนารีทรี

ไบนารีทรีเป็นทรีที่แต่ละโหนดมีสับทรีได้สูงสุด 2 สับทรี คือ สับทรีทางซ้าย และสับทรีทางขวา ดังตัวอย่างในภาพที่ 2.36

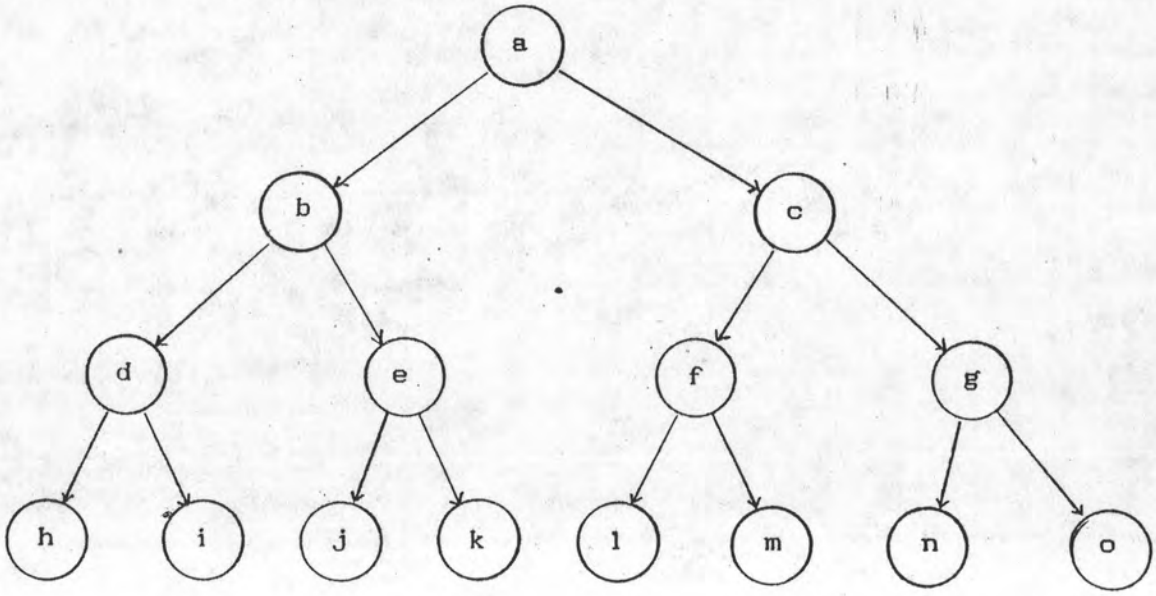


ภาพที่ 2.36 แสดงตัวอย่างไบนารีทรี

ไบนารีทรี 2 ทรี ในภาพที่ 2.36 มีความแตกต่างกัน เนื่องจากมีสับทรีไม่เหมือนกัน คือ รูปแรก มีสับทรีทางซ้าย ส่วนรูปหลังมีสับทรีทางขวา ส่วนทรีในภาพที่ 2.37 ไม่ถือว่าเป็นไบนารีทรี เพราะไม่สามารถระบุได้ว่ามีสับทรีทางซ้ายหรือทางขวา

ภาพที่ 2.37 ทรีธรรมดา

ไบนารีทรีที่ทุก ๆ โหนด (ยกเว้นโหนดใบ) มีทั้งสับทรีทางซ้าย และสับทรีทางขวา และโหนดใบทุก ๆ โหนดอยู่ในระดับเดียวกัน เรียกว่า ไบนารีทรีแบบสมบูรณ์ (complete binary tree) ดังตัวอย่างในภาพที่ 2.38



ภาพที่ 2.38 ไบนารีทรีแบบสมบูรณ์

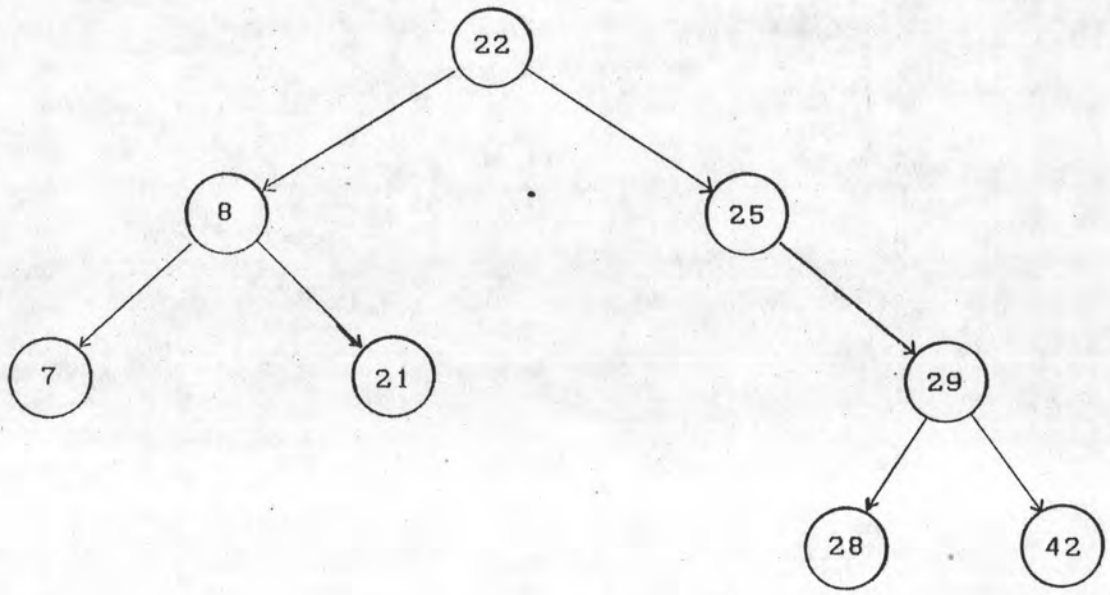
4.2.4 ไบนารีเสิชทรี

ไบนารีเสิชทรีเป็นโครงสร้างข้อมูลแบบไบนารีทรีแบบพิเศษที่เอื้ออำนวยให้สามารถเข้าถึงข้อมูลได้ทั้งแบบโดยตรงและแบบตามลำดับ นั่นคือ ลักษณะการจัดเก็บข้อมูลจะเรียงลำดับตามคีย์ใดคีย์หนึ่ง โดยที่ สำหรับโหนด i ใด ๆ ที่มีค่าคีย์เท่ากับ k_i ถ้าให้ k_l เป็นค่าคีย์ของโหนดลูกด้านซ้าย และให้ k_r เป็นค่าคีย์ของโหนดลูกด้านขวา แล้ว จะมีคุณสมบัติ

$$k_l < k_i < k_r$$

นั่นคือ การสร้างโครงสร้างชนิดนี้ จะสร้างจากชุดคีย์ที่อ่านเข้ามา คีย์ตัวแรกจะเป็นโหนดราก จากนั้นก็ต่อเติมโหนดต่าง ๆ โดยถือหลักว่า ถ้าคีย์ที่อ่านเข้ามาน้อยกว่าโหนดที่มีอยู่แล้ว ณ จุดนั้น ก็ให้ต่อคีย์นั้นเป็นโหนดลูกด้านซ้ายของโหนดตัวนั้น หรือให้เลื่อนไปเปรียบเทียบกับโหนดลูกด้านซ้ายของโหนดที่มีอยู่แล้ว ในกรณีที่คีย์ที่อ่านเข้ามามีค่ามากกว่าโหนดที่มีอยู่แล้วที่จุดนั้น ให้นำคีย์นั้นไปต่อเป็นโหนดลูกด้านขวาของโหนดนั้น หรือเลื่อนไปเปรียบเทียบกับโหนดลูกด้านขวาที่มีอยู่

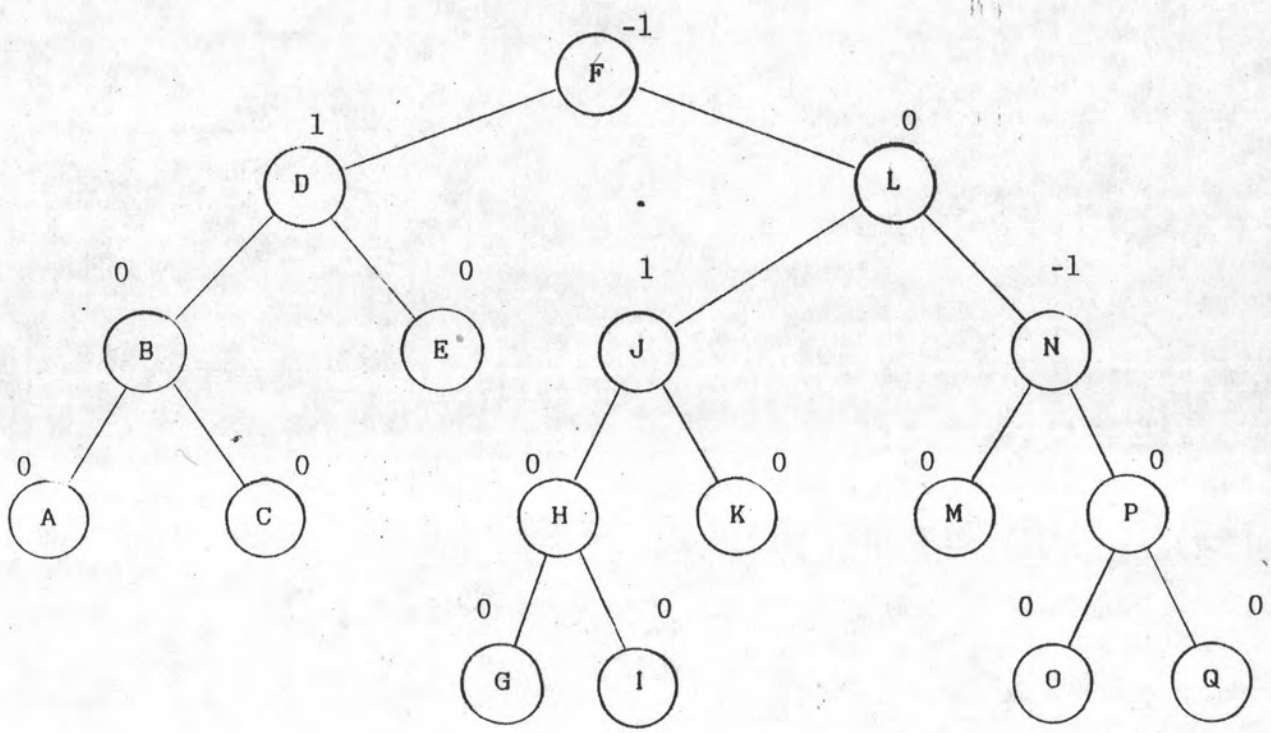
แล้วไปเรื่อย ๆ ตัวอย่างไบนารีเสิชทรี ดังภาพที่ 2.39



ภาพที่ 2.39 ไบนารีเสิชทรี

ปรกติแล้ว ไบนารีเสิชทรีจะถูกเปลี่ยนรูปร่างไปเมื่อมีการเพิ่มหรือลบข้อมูล การเพิ่มข้อมูลอาจทำให้ต้องแยกสับทรี ในขณะที่การลบข้อมูลอาจทำให้ต้องรวมสับทรี ซึ่งอาจทำให้เหลือทรีที่เอียงไปข้างใดข้างหนึ่งมากเกินไป ทำให้อาจจะเสียเวลาในการค้นหามาก ทรีที่ดีควรมีความสูงของสับทรีพอ ๆ กัน และต้องเป็นทรีแบบสมบูรณ์จึงจะช่วยให้จำนวนครั้งของการค้นหาน้อยที่สุดได้ จึงมีการปรับไบนารีเสิชทรี ให้เป็นทรีที่มีความสูงสมดุล (Height Balanced Tree หรือ AVL Tree) นั่นคือ ทรีที่มีความสูงสมดุล คือ ไบนารีทรีที่มีความสูงของสับทรีทางซ้ายและของสับทรีทางขวาต่างกันไม่เกิน 1 (ความสูงของทรีหรือสับทรี คือจำนวนระดับของโหนดในทรีหรือสับทรีนั่นเอง) ค่าความแตกต่างนี้ เรียกว่า ค่าสมดุล (balance) จากภาพที่ 2.40 แสดงตัวอย่างทรีที่มีความสูงสมดุล ซึ่งมีตัวเลข -1, 0, และ 1 กำกับแต่ละโหนดแทนค่าสมดุลของโหนดนั้น ๆ เช่น โหนด F มีค่าสมดุลเป็น -1 หมายความว่า ความสูงของสับทรีทางซ้ายน้อยกว่าความสูงของสับทรีทางขวาของโหนด F อยู่ 1 ระดับ โหนด D มีค่าสมดุลเป็น 1 หมายความว่า ความสูงของสับทรีทางซ้ายมากกว่าความสูงของสับทรีทางขวาของโหนด D อยู่ 1 ระดับ และโหนด L มีค่า

สมดุลเป็น 0 หมายความว่าความสูงของสับทรีทางซ้ายและความสูงของสับทรีทางขวาของโหนด L เท่ากัน เป็นต้น



ภาพที่ 2.40 ทรีที่มีความสูงสมดุลพร้อมค่าสมดุลของแต่ละโหนด

4.2.5 ทรีการค้นหาแบบหลายทาง

ทรีการค้นหาแบบหลายทางเป็นโครงสร้างทรีธรรมดาที่ไม่ใช่ไบนารีทรี โดยแต่ละโหนดมีลิงค์ได้มากกว่า 2 ลิงค์ ทรีการค้นหาแบบ m ทางมีคุณสมบัติดังนี้

1. แต่ละโหนดในทรี T ประกอบด้วย

   n,

   S_0,

   $(K_1, A_1, S_1), \ldots, (K_n, A_n, S_n)$

2. n คือ จำนวนคีย์ มีค่ามากกว่าหรือเท่ากับ 1 และน้อยกว่า m

3. K_i $(1 <= i <= n)$ เป็นคีย์ โดยที่ $K_i < K_{i+1}$

สำหรับค่า i ที่มากกว่าหรือเท่ากับ 1 และน้อยกว่า n-1

4. S_0 เป็นตัวชี้ไปยังสับทรีที่เก็บคีย์ที่น้อยกว่า K_1

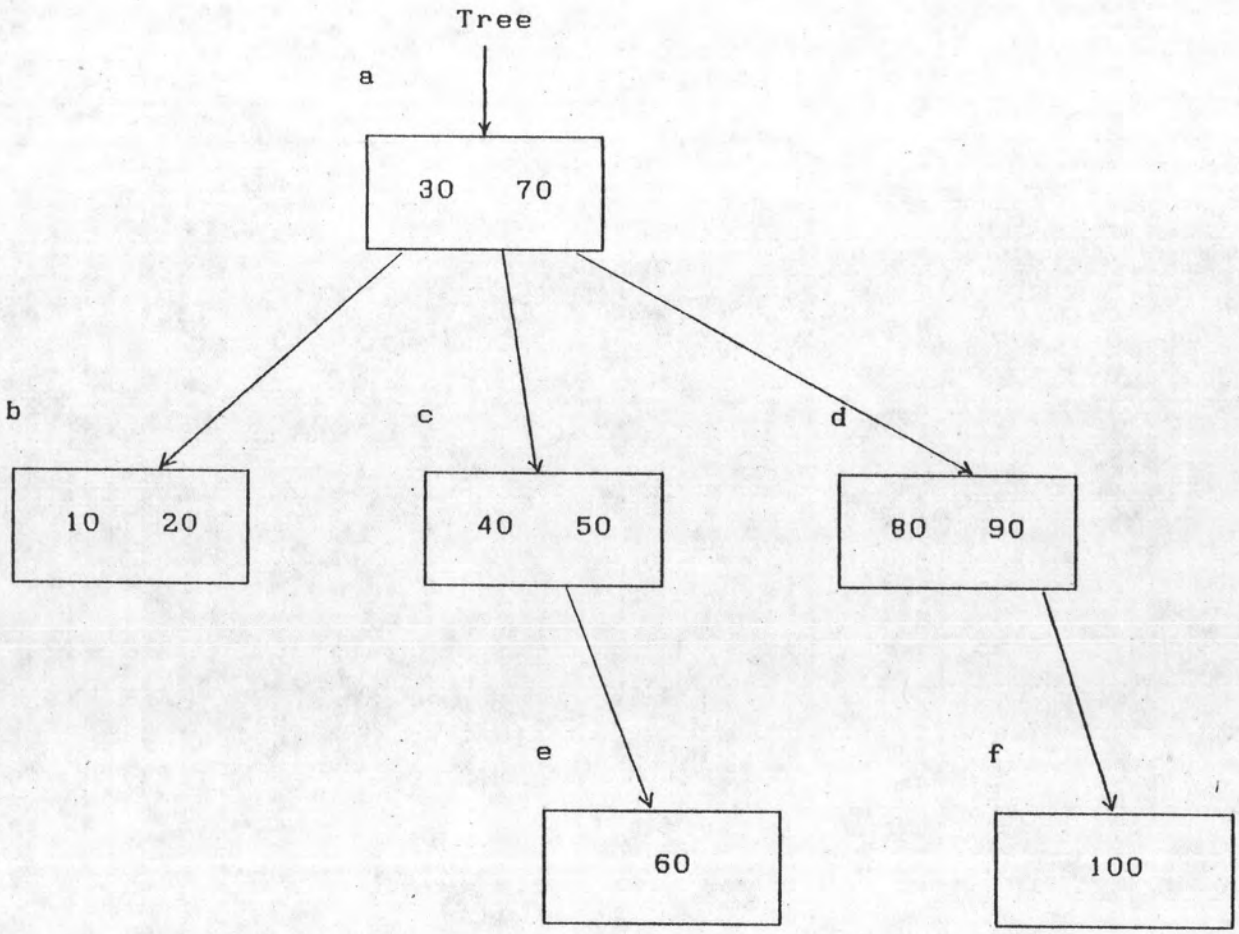
5. S_i $(1<=i<=n)$ เป็นตัวชี้ไปยังสับทรีที่เก็บค่าคีย์ระหว่าง K_i กับ K_{i+1}

6. S_n เป็นตัวชี้ไปยังสับทรีที่เก็บคีย์ที่มีค่ามากกว่า K_n

7. S_i $(0<=i<=n)$ เป็นตัวชี้ไปยังทรีการค้นหาแบบ m ทาง

8. A_i $(1<=i<=n)$ เป็นตำแหน่งของระเบียนที่มีคีย์เป็น K_i ในแฟ้มข้อมูล

ภาพที่ 2.41 แสดงตัวอย่างของทรีการค้นหาแบบ 3 ทาง มีข้อมูลทั้งสิ้น 10 ค่า ได้แก่ 10, 20, ..., 100



ภาพที่ 2.41 แสดงตัวอย่างของทรีการค้นหาแบบ 3 ทาง

จากภาพที่ 2.41 ถ้าเขียนทรีตามคุณสมบัติ 8 ข้อ ข้างต้นจะได้ดังในภาพที่ 2.42

โหนด	รูปแบบ
a	2, b, (30, addr[30], c), (70, addr[70], d)
b	2, 0, (10, addr[10], 0), (20, addr[20], 0)
c	2, 0, (40, addr[40], 0), (50, addr[50], e)
d	2, 0, (80, addr[80], 0), (90, addr[90], f)
e	1, 0, (60, addr[60], 0)
f	1, 0, (100, addr[100], 0)

ภาพที่ 2.42 แสดงคุณสมบัติของทรีการค้นหาแบบ 3 ทาง

4.2.6 <u>บีทรี</u>

บีทรี คือ ทรีการค้นหาหลายทางที่ปรับให้สมดุล เพื่อให้มีประสิทธิภาพสูงสุด บีทรีที่มีระดับ m คือ ทรีการค้นหาแบบ m ทาง ที่มีคุณสมบัติดังนี้

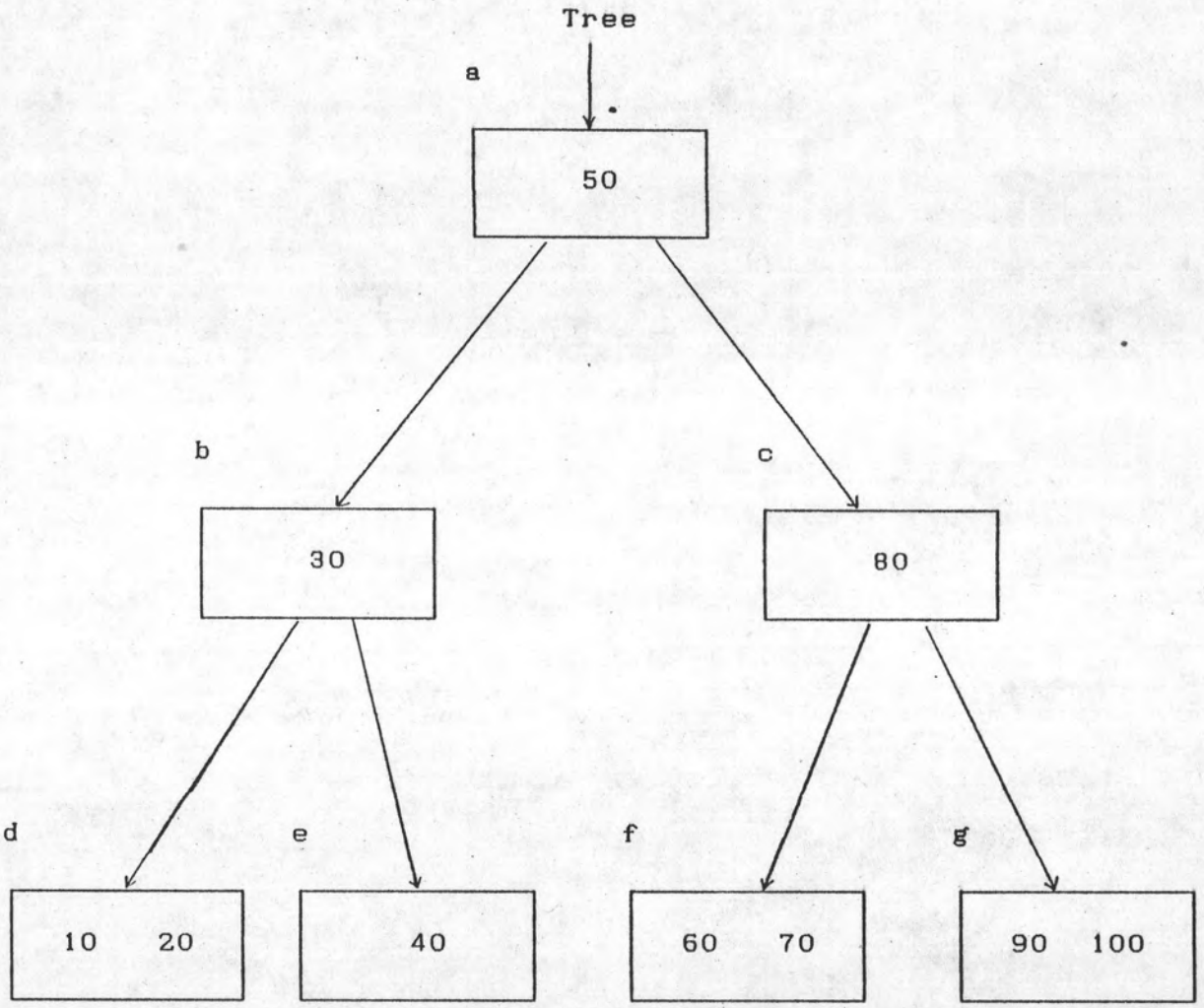
1. แต่ละโหนดในทรี (ยกเว้นโหนดรากและโหนดใบ) จะประกอบด้วยสับทรี อย่างน้อยครึ่งหนึ่งของค่า m และไม่เกิน m

2. โหนดรากของทรีจะมีสับทรีอย่างน้อยที่สุด 2 สับทรี ยกเว้นถ้าโหนดนั้นเป็นโหนดใบด้วย

3. โหนดใบทุกโหนดจะต้องอยู่ในระดับเดียวกัน

คุณสมบัติข้อแรก ทำให้ทุกโหนดจะต้องมีข้อมูลอยู่อย่าง

น้อยครึ่งหนึ่ง คุณสมบัติข้อ 2 ช่วยให้ทรีแผ่ขยายออกไป ไม่มีระดับมากจนเกินควร และคุณสมบัติข้อ 3 ช่วยให้ทรีค่อนข้างที่จะมีความสมดุล ตัวอย่าง บีทรีเป็นดังภาพที่ 2.43



ภาพที่ 2.43 แสดงบีทรีที่มีลำดับเป็น 3

4.2.7 บี*ทรี

บี*ทรี เป็นโครงสร้างข้อมูลที่ปรับจากบีทรี ให้มีคุณสมบัติพิเศษเพิ่มเติม เรียกว่า เทคนิคโอเวอร์โฟลว์ (Overflow Technique) เพื่อ

พัฒนาขั้นตอนวิธีการเพิ่มข้อมูล โดยนำข้อมูลที่เต็มจากโหนดหนึ่ง ไปยังโหนดพี่น้องซึ่งยังไม่เต็ม แทนที่จะสร้างโหนดใหม่ ความแตกต่างจากบีทรีก็คือ ทุก ๆ โหนดจะมีข้อมูลอยู่สูงสุด 2 ใน 3 แทนที่จะเป็นครึ่งหนึ่ง ทำให้ประหยัดเนื้อที่กว่าบีทรี และลดจำนวนโหนดลง ทำให้การค้นหารวดเร็วขึ้น มีคุณสมบัติสรุปได้ดังนี้

1. เป็นทรีการค้นหาแบบ m ทาง ซึ่งอาจจะว่างหรือมีความสูง มากกว่าหรือเท่ากับ 1

2. โหนดรากจะต้องมีโหนดลูกอย่างน้อย 2 โหนด (นั่นคือ มีอย่างน้อย 1 ค่า) และค่าสูงสุดคือ $2\lfloor (2m-2)/3 \rfloor + 1$ เมื่อ $\lfloor x \rfloor$ หมายถึงค่าต่ำสุดของ x

3. โหนดที่ไม่ใช่โหนดรากและโหนดใบ จะต้องมีโหนดลูกจำนวนอย่างน้อย $\lceil (2m-1)/3 \rceil$ โหนด (ดังนั้น จะต้องมีอย่างน้อย $\lceil (2m-1)/3 \rceil - 1$ ค่า เมื่อ $\lceil x \rceil$ หมายถึงค่าสูงสุดของ x

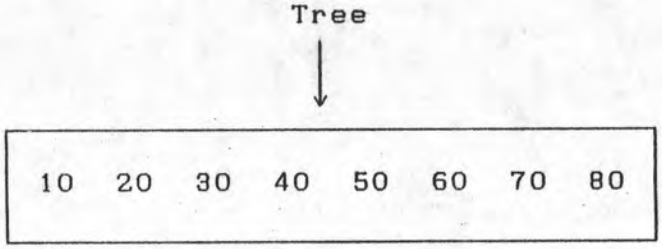
4. โหนดใบทุกโหนดจะต้องอยู่ในระดับเดียวกัน

5. โหนดที่ไม่ใช่โหนดรากและโหนดใบ ที่ kS_s จะมีจำนวนคีย์เป็น k-1

ดังนั้น บี*ทรีแบบ 7 ทาง จะมีค่าต่ำสุดและสูงสุดของข้อมูลและคีย์ตามคุณสมบัติข้างต้นดังนี้

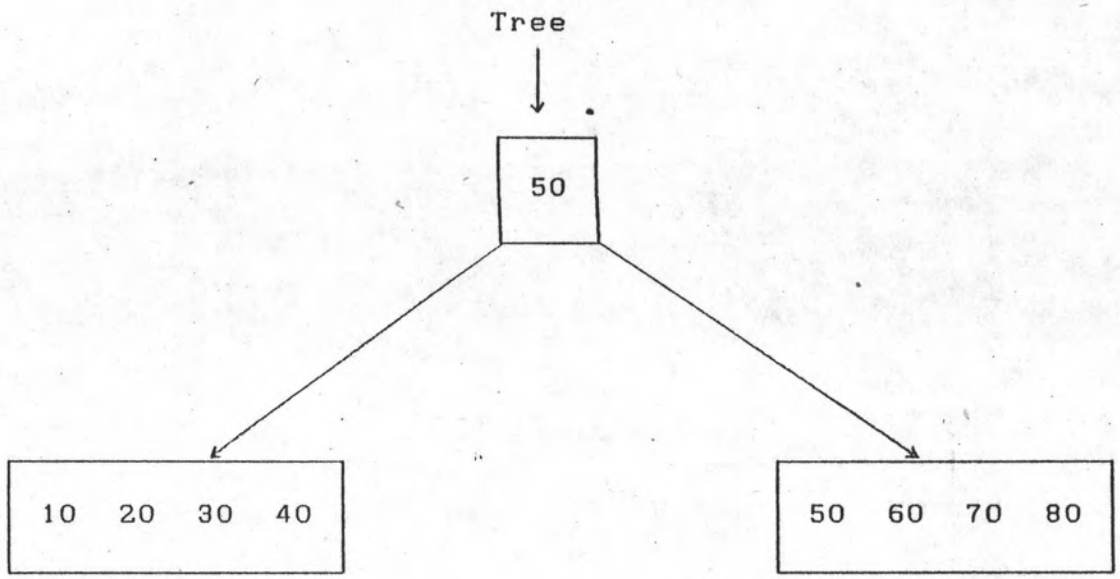
โหนดราก จะมีโหนดลูกได้ตั้งแต่ 2 ถึง 7 โหนด และจำนวนคีย์ ตั้งแต่ 1 ถึง 6 คีย์

โหนดที่ไม่ใช่โหนดรากหรือโหนดใบ จะมีโหนดลูกได้ตั้งแต่ 5 ถึง 7 โหนด และจำนวนคีย์ตั้งแต่ 4 ถึง 6

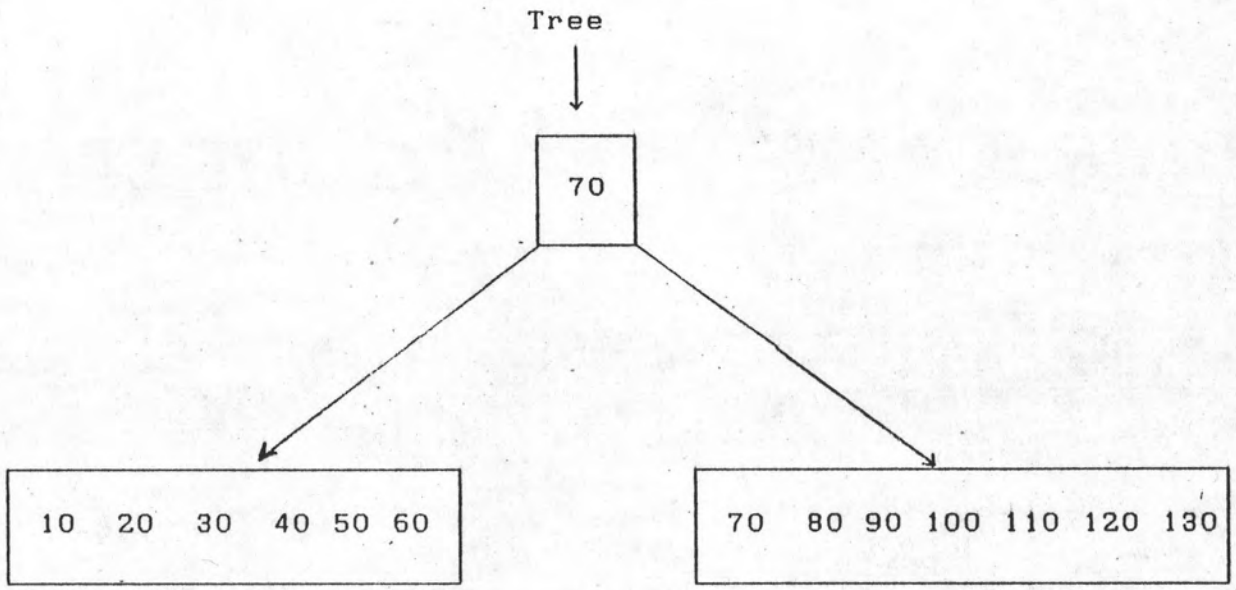


ภาพที่ 2.44 แสดงบี*ทรีแบบ 7 ทาง ที่มีคีย์เต็มรากโหนด จำนวน 8 ค่า

ตัวอย่างบี*ทรี ในภาพที่ 2.44 เป็นแบบ 7 ทาง ที่มีคีย์ 8 ค่า ซึ่งเต็มรากโหนดแล้ว เมื่อมีการเพิ่มคีย์ เช่น 90 เข้าไป จึงต้องมีการแบ่งเป็นโหนดลูกดังภาพที่ 2.45



ภาพที่ 2.45 แสดงบี*ทรีแบบ 7 ทาง เมื่อเพิ่มคีย์ 90



ภาพที่ 2.45 แสดงบี*ทรีแบบ 7 ทาง เมื่อเพิ่มคีย์ 100, 110 120 และ 130

การจัดองค์การแฟ้มข้อมูล

การจัดองค์การแฟ้มข้อมูล เป็นการจัดระบบโครงสร้างของข้อมูลที่จัดสำหรับข้อมูลที่เก็บในหน่วยความจำภายนอก ได้แก่ พวกอุปกรณ์ที่เป็นสื่อบันทึกข้อมูลที่มีการผนวกเอาความเหมาะสมในการใช้เนื้อที่บนอุปกรณ์ เวลาที่เสียไปในการประมวลผลด้วยโปรแกรม ภาษาคอมพิวเตอร์ต่าง ๆ สามารถใช้เทคนิคในการจัดองค์การของแฟ้มข้อมูลและการเข้าถึงข้อมูลแตกต่างกัน ถ้าจะพิจารณาระบบแฟ้มข้อมูลในรูปคำศัพท์ทั่ว ๆ ไป สามารถแบ่งออกได้เป็น 7 ประเภท คือ แฟ้มข้อมูลพื้นฐาน (Piles File) แฟ้มข้อมูลแบบเรียงลำดับ (Sequential File) แฟ้มข้อมูลแบบเข้าถึงโดยตรง (Direct File) แฟ้มข้อมูลแบบมีดัชนี (Index File) แฟ้มข้อมูลเรียงลำดับดัชนี (Indexed Sequential File) แฟ้มข้อมูลหกกลับ (Inverted File) และแฟ้มข้อมูลแบบมีตัวชี้ (Pointer File)

1. แฟ้มข้อมูลพื้นฐาน

แฟ้มข้อมูลพื้นฐาน คือ กลุ่มของระเบียนที่ไม่มีการจำกัดในแต่ละระเบียนแน่นอน ยกเว้นในกรณีที่ระเบียนนั้นประกอบด้วยเขตข้อมูลที่มีความสัมพันธ์กัน โครงสร้างโดยทั่วไปของแฟ้มข้อมูลพื้นฐาน คือ แฟ้มข้อมูลที่ประกอบด้วยระเบียนที่มีกฎเกณฑ์เพียงหลวม ๆ บนระเบียนหนึ่ง ๆ จะประกอบด้วยเขตข้อมูลหนึ่ง ๆ ที่ไม่แน่นอน โดยแต่ละเขตข้อมูลจะมีการใส่บทความเกี่ยวกับข้อมูลในแต่ละเขตเข้าไปด้วย ทำให้ข้อมูลของแต่ละระเบียนจะประกอบด้วย บทความและข้อมูล ซึ่งสามารถพิจารณาแฟ้มข้อมูลแบบพื้นฐานในลักษณะเรียงลำดับโดยไม่คำนึงถึงรูปแบบของระเบียนว่าแต่ละระเบียนจะเก็บข้อมูลอะไรบ้าง และอยากทราบว่าระเบียนนั้นมีข้อมูลอะไรก็หาเอาจากระเบียนนั้น

2. แฟ้มข้อมูลแบบเรียงลำดับ

แฟ้มข้อมูลแบบเรียงลำดับประกอบด้วยระเบียนที่เรียงลำดับก่อนหลังของข้อมูลที่บันทึกเข้ามา วิธีการเข้าถึงข้อมูลนั้น โปรแกรมสามารถทราบเพียงตำแหน่งของระเบียนถัดไปว่าอยู่ ณ ตำแหน่งใดเท่านั้น การจัดแฟ้มข้อมูลประเภทนี้

เหมาะสำหรับงานประเภทที่มีการปรับปรุงข้อมูลครั้งละมากกว่า 30% ของแฟ้มข้อมูลนั้น เพราะว่า เมื่อมีการปรับปรุงข้อมูลมาก ๆ การทำงานแบบเรียงลำดับนี้จะปรับปรุงได้เร็วกว่าแบบอื่น ๆ เนื่องจากคุณลักษณะการทำงานของมันทำตามลำดับก่อนหลังของข้อมูลที่ต้องการปรับปรุง ข้อดีของแฟ้มข้อมูลแบบนี้ คือ สามารถใช้เนื้อที่บนอุปกรณ์อย่างมีประสิทธิภาพ และถ้าเป็นการเข้าถึงข้อมูลแบบเรียงลำดับต่อเนื่องแล้วจะกระทำได้รวดเร็วมาก เพราะแฟ้มข้อมูลแบบนี้มีตำแหน่งข้อมูลแน่นอน โอกาสที่ข้อมูลจะหายไปจากระบบไม่มี ค่าใช้จ่ายต่ำ เพราะสามารถใช้กับอุปกรณ์ประเภทเทปแม่เหล็กที่มีราคาถูกได้ ข้อเสียคือ มีการเข้าถึงข้อมูลแบบเรียงลำดับ ไม่สามารถเข้าถึงข้อมูลแบบสุ่มเลือกได้

3. <u>แฟ้มข้อมูลแบบเข้าถึงโดยตรง</u>

แฟ้มข้อมูลแบบเข้าถึงโดยตรงเป็นแฟ้มข้อมูลที่มีการจัดตำแหน่งของแต่ละระเบียนบนสื่อบันทึกข้อมูลในลักษณะที่ไม่คำนึงถึงตำแหน่งที่อยู่สัมพันธ์กับระเบียนอื่น ๆ ในแฟ้มข้อมูล แต่จัดตำแหน่งของระเบียนโดยการใช้ค่าคีย์เป็นตัวกำหนดตำแหน่ง หรือได้จากการคำนวณหาเพื่อให้สามารถเข้าถึงระเบียนได้โดยตรง มักจะใช้เมื่อต้องการความรวดเร็วในการทำงาน และข้อมูลไม่จำเป็นต้องเรียงลำดับ การใช้จะมีประสิทธิภาพดีที่สุดเมื่อแฟ้มข้อมูลมีลักษณะค่อนข้างจะแน่นอน ข้อดี คือ มีความรวดเร็วในการเข้าถึงข้อมูล ไม่ว่าจะใช้วิธีการใช้คีย์เป็นตัวแสดงตำแหน่งที่อยู่ หรือใช้วิธีการคำนวนหาตำแหน่งก็ตาม ข้อเสียคือยากแก่การปรับปรุงแฟ้มข้อมูลใหม่ หรือเพิ่มขนาดของแฟ้มข้อมูล เพราะตำแหน่งของข้อมูลถูกกำหนดไว้คงที่ นอกจากนี้ยังเปลืองเนื้อที่ว่างมากกว่าการจัดแฟ้มข้อมูลแบบเรียงลำดับ

4. <u>แฟ้มข้อมูลแบบมีดัชนี</u>

แฟ้มข้อมูลแบบมีดัชนีใช้วิธีการเก็บคีย์ของระเบียนและตำแหน่งที่อยู่ไว้เป็นตารางดัชนี การเข้าถึงข้อมูลจะต้องค้นหาจากตารางดัชนีจนกระทั่งได้คีย์ที่ต้องการจึงจะทราบตำแหน่งของระเบียน การเพิ่มระเบียนจะต้องมีการเพิ่มคีย์ในตารางดัชนี การลบระเบียนกระทำโดยการลบคีย์นั้นออกจากตาราง หรือทำสัญญลักษณ์พิเศษเพื่อบอกว่า ระเบียนนี้เลิกใช้แล้ว การค้นหาระเบียนบนแฟ้มข้อมูลแบบนี้จะเร็วหรือช้าขึ้นอยู่กับระดับของดัชนี ข้อดีคือ การเข้าถึงข้อมูลทำได้รวดเร็ว มีการใช้เนื้อที่บนอุปกรณ์หรือสื่อบันทึกข้อมูลอย่างมีประสิทธิภาพ ง่ายต่อการจัดองค์กรแฟ้มข้อมูลใหม่

นอกจากนี้ยังสามารถเข้าถึงข้อมูลได้ด้วยคีย์มากกว่า 1 คีย์ ข้อเสีย คือ ถ้าแฟ้มข้อมูลใหญ่มาก ๆ ตารางดัชนีก็จะใหญ่มากด้วย ทำให้สิ้นเปลืองเนื้อที่ในการเก็บดัชนี การเพิ่มระเบียนในแฟ้มข้อมูล ถ้าเพิ่มมากจะตกลงในพื้นที่ต่างหากที่จัดเตรียมไว้ ทำให้ประสิทธิภาพการค้นหาข้อมูลด้อยลง ต้องทำการจัดองค์การใหม่

5. <u>แฟ้มข้อมูลหกกลับ</u>

แฟ้มข้อมูลหกกลับถูกพัฒนาขึ้นเพื่อให้การเข้าถึงข้อมูลในลักษณะที่มีการดึงข้อมูลเป็นกลุ่มระเบียนที่เป็นเรื่องเดียวกันสามารถทำได้รวดเร็ว นั่นคือ สร้างตารางดัชนีที่ใช้คีย์เป็นเขตข้อมูลร่วมที่เป็นที่ต้องการ ส่วนเขตข้อมูลอื่น ๆ ในระเบียนก็ประกอบด้วย เขตข้อมูลที่เป็นตัวบอกว่า มีระเบียนใดบ้างในแฟ้มข้อมูลที่ตรงตามคีย์ที่ระบุ ทำให้การค้นหาเป็นไปได้อย่างรวดเร็ว ข้อดีคือ สามารถเข้าถึงข้อมูลแบบสุ่ม โดยใช้คีย์หลาย ๆ อย่าง ข้อเสียคือ จะต้องเสียเนื้อที่เก็บมากขึ้นอีก 1 แฟ้ม และการบำรุงรักษาแฟ้มข้อมูลนี้จะต้องทำเสมอหลังจากมีการปรับปรุงแฟ้มข้อมูลหลัก

6. <u>แฟ้มข้อมูลแบบมีตัวชี้</u>

แฟ้มข้อมูลแบบมีตัวชี้ มีการเชื่อมแต่ละระเบียนในแฟ้มข้อมูลด้วยตัวชี้ ซึ่งมาจากแนวความคิดพื้นฐานของ ลิงค์ลิสต์ โดยอาจจะมีการจัดแบบทางเดียว หรือแบบหลายทางก็ได้ ข้อดีคือมีโครงสร้างที่ค่อนข้างอิสระ สามารถจัดได้ทั้งระเบียนที่มีความยาวคงที่ และไม่คงที่ ข้อเสียคือ เสียเนื้อที่มากและเสียเวลาในการประมวลผลมากทั้งในการอ่านหรือค้นหาระเบียน และเสียเวลาในการประมวลผลเกี่ยวกับตัวชี้

<u>การจัดการระเบียนข้อมูลและแฟ้มข้อมูลของระบบปฏิบัติการเอ็มเอส-ดอส</u> (MS-DOS File and Record Manipulation)

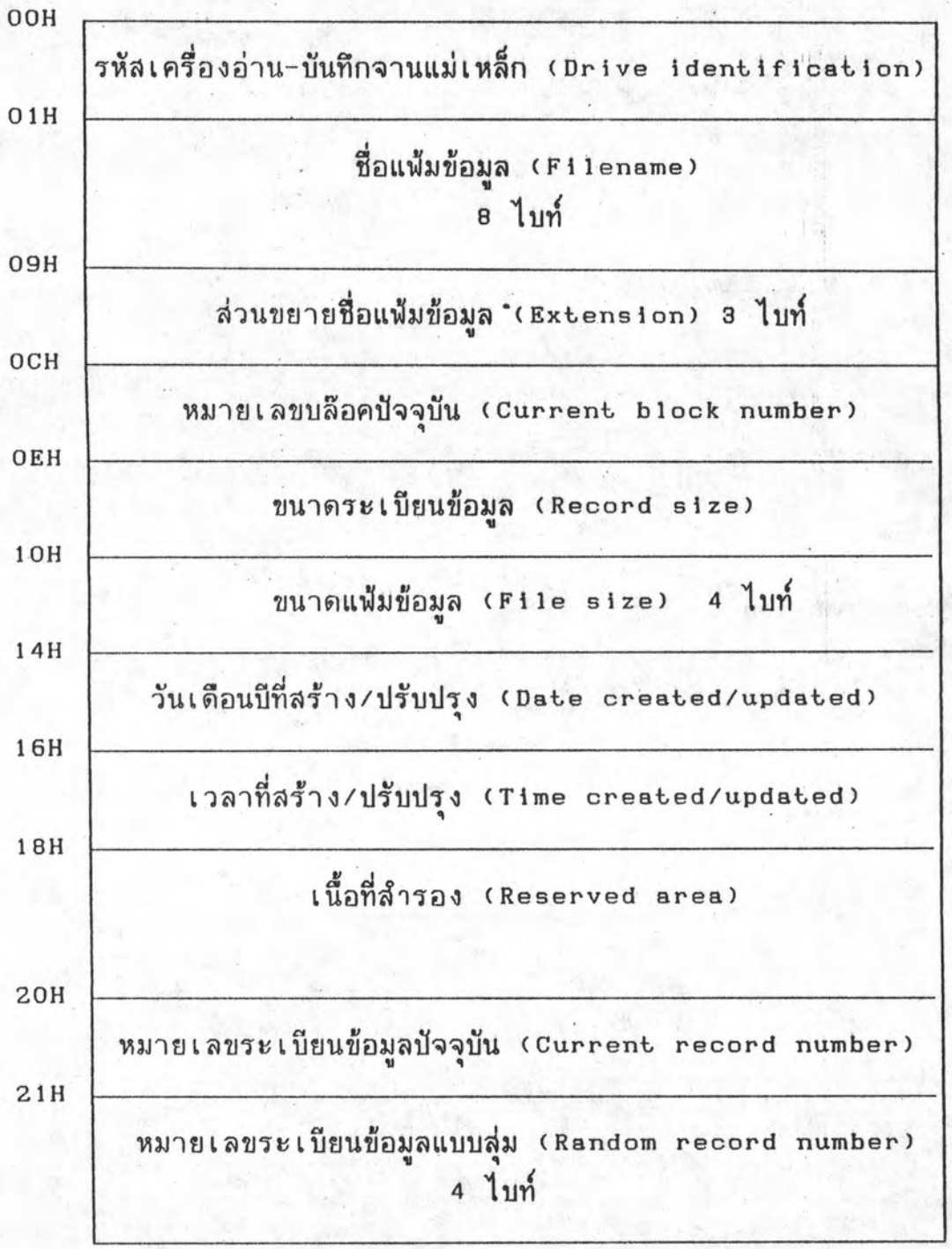
ระบบปฏิบัติการเอ็มเอส-ดอส ได้ถูกออกแบบให้มีความคล้ายคลึงกับ ทั้งระบบปฏิบัติการยูนิกซ์/ซีนิกซ์ (UNIX/XENIX) และระบบปฏิบัติการซีพี/เอ็ม (CP/M) ทั้งนี้ เพื่อประโยชน์ในการถ่ายโปรแกรมจากระบบเดิมเป็นระบบใหม่ เช่น ถ่ายโปร

แกรมจากระบบซีพี/เอ็มไปยังระบบเอ็มเอส/ดอส หรือถ่ายโปรแกรมจากเอ็มเอส-ดอสไปยังระบบยูนิกซ์ หรือในทางกลับกัน เป็นต้น แต่เนื่องจากโครงสร้างของการจัดการระเบียนข้อมูลและแฟ้มข้อมูลของระบบซีพีเอ็ม ต่างกับของระบบยูนิกซ์หรือซีนิกซ์มาก เอ็มเอส-ดอสจึงได้จัดเตรียมชุดคำสั่งที่ใช้ในการจัดการระเบียนข้อมูลและแฟ้มข้อมูลให้ผู้ใช้เลือกใช้ได้ตามต้องการ โดยแบ่งเป็น 2 กลุ่ม ได้แก่ ฟังก์ชันเอฟซีบี และฟังก์ชันแฮนเดิล

1. <u>ฟังก์ชันเอฟซีบี</u> (FCB Function หรือ File Control Block Function)

ฟังก์ชันเอฟซีบี หรือ ฟังก์ชันไฟล์คอนโทรลบล๊อค คือ ชุดคำสั่งของเอ็มเอส/ดอส ที่ใช้ในการจัดการระเบียนข้อมูลและแฟ้มข้อมูล ที่เลียนแบบมาจากของระบบปฏิบัติการซีพี/เอ็ม ลักษณะโครงสร้างระเบียนข้อมูลประกอบด้วยเขตข้อมูลต่าง ๆ ความยาว 37 ไบท์ เก็บอยู่ในหน่วยความจำหลักในส่วนของโปรแกรมใช้งาน ฟังก์ชันเอฟซีบีมีชุดคำสั่งที่เอื้ออำนวยให้ผู้ใช้ทำการสร้าง เปิด ปิด หรือ ลบแฟ้มข้อมูล ตลอดจนอ่านหรือบันทึกระเบียนข้อมูลขนาดต่าง ๆ ณ ตำแหน่งใด ๆ ในแฟ้มข้อมูลได้ แต่ชุดคำสั่งเหล่านี้ไม่ได้ถูกออกแบบให้ใช้กับโครงสร้างแฟ้มข้อมูลแบบ hierarchical ซึ่งเริ่มใช้กับระบบปฏิบัติการเอ็มเอส-ดอส ตั้งแต่รุ่น 2.0 เป็นต้นมาได้ ดังนั้น การเข้าถึงแฟ้มข้อมูลจึงสามารถกระทำได้เพียงภายในไดเรคตอรีย่อย (subdirectory) ของหน่วยอ่าน-บันทึกจานแม่เหล็กในขณะนั้น ๆ เท่านั้น ไม่สามารถเข้าถึงข้อมูลที่อยู่ในไดเรคตอรีอื่น ๆ ได้ ผู้เขียนโปรแกรมรุ่นหลัง ๆ จึงไม่นิยมใช้วิธีนี้เนื่องจากข้อจำกัดดังกล่าว

ตัวอย่างโครงสร้างระเบียนข้อมูลของเอฟซีบี ดังแสดงในภาพ 2.46 เมื่อเริ่มประมวลผล เอฟซีบีจะเริ่มถูกกำหนดโดยโปรแกรมด้วยข้อมูล รหัสเครื่องอ่าน-บันทึกจานแม่เหล็ก ชื่อแฟ้มข้อมูล และส่วนขยายชื่อแฟ้มข้อมูล จากนั้น ตำแหน่งของเอฟซีบีจะถูกส่งไปให้เอ็มเอส-ดอส เพื่อเปิดหรือสร้างแฟ้มข้อมูล ถ้าเปิดหรือสร้างแฟ้มข้อมูลได้แล้ว เอ็มเอส-ดอส จะนำรายละเอียดเกี่ยวกับแฟ้มข้อมูลที่เปิดได้มาจากไดเรคตอรีของจานแม่เหล็กมาเก็บในเอฟซีบี ได้แก่ ขนาดของแฟ้มข้อมูลที่เปิด (หน่วยเป็นไบท์) วันเดือนปีที่สร้างหรือบรรณาธิกรแฟ้มข้อมูลครั้งหลังสุด เวลาที่สร้างหรือบรรณาธิกรครั้งหลังสุด เป็นต้น นอกจากนี้ยังมีข้อมูลอื่น ๆ ถูกจัดเก็บไว้ในเนื้อที่สำรอง ซึ่งมีขนาดแตกต่างกันในเอ็มเอส-ดอสแต่ละรุ่น ผู้เขียนโปรแกรมต้องไม่เปลี่ยนแปลงข้อมูลในเนื้อที่สำรองนี้



ภาพ 2.46 แสดงโครงสร้างของเอฟซีบี ขนาด 37 ไบท์

เขตข้อมูลรหัสเครื่องอ่าน-บันทึกจานแม่เหล็ก เก็บค่าเลขฐานสอง จำนวน 2 หลัก ระบุรหัสเครื่องอ่าน-บันทึกจานแม่เหล็กที่แฟ้มข้อมูลนั้น ๆ อยู่ ถ้าเป็น 00

หมายถึงเครื่องอ่าน-บันทึกจานแม่เหล็กที่ระบบติดต่ออยู่ขณะนั้น ถ้าเป็น 01 หมายถึงเครื่องอ่าน-บันทึก A ถ้าเป็น 02 หมายถึงเครื่องอ่าน-บันทึก B ถ้าเป็น 03 หมายถึงเครื่องอ่าน-บันทึก C เป็นต้น ในกรณีที่โปรแกรมระบุให้รหัสมีค่าเป็น 0 รหัสจริงของเครื่องอ่านจะถูกเติมโดยระบบเมื่อระบบสามารถเปิดหรือสร้างแฟ้มข้อมูลได้แล้ว

สำหรับชื่อแฟ้มข้อมูลนั้น ความยาวสูงสุดไม่เกิน 8 ไบท์ เก็บชิดซ้ายเขตข้อมูล และที่เหลือทางขวาจะเป็นช่องว่าง ส่วนขยายชื่อ ความยาวสูงสุดไม่เกิน 3 ไบท์ เก็บชิดซ้ายเขตข้อมูลเช่นกัน ที่เหลือทางขวาจะเป็นช่องว่าง

ข้อมูลในไบท์ 0000H ถึงไบท์ 000FH และจาก 0020H ถึงไบท์ 0024H นั้น จะถูกกำหนดโดยผู้ใช้ระบบ ส่วน ไบท์ 0010H ถึงไบท์ 001FH นั้น จะถูกกำหนดโดยเอ็มเอส-ดอส และไม่สามารถถูกแก้ไขโดยโปรแกรมงานได้

เพื่อให้คล้ายกับของระบบซีพี/เอ็ม เอ็มเอส-ดอสจะกำหนดให้เขตข้อมูลขนาดระเบียนข้อมูล (Record-size field) มีค่า 128 ไบท์โดยอัตโนมัติ ถ้าผู้ใช้ต้องการเปลี่ยนแปลงก็สามารถกระทำได้โดยมาแก้ไขข้อมูลในเขตข้อมูลนี้

ในขณะประมวลผล ถ้าโปรแกรมต้องการอ่านหรือบันทึกระเบียนข้อมูล จะเริ่มต้นจากการส่งตำแหน่งของเอฟซีบี ไปยังเอ็มเอส-ดอส จากนั้น เอ็มเอส-ดอสจะปรับปรุงข้อมูลของตัวชี้แฟ้มข้อมูล (file pointer) และขนาดของแฟ้มข้อมูล (file size) ในเอฟซีบีให้ทันสมัย ถ้าโปรแกรมมีการเข้าถึงระเบียนข้อมูลแบบสุ่ม (Random record access) จะต้องทำการกำหนดหมายเลขระเบียนข้อมูล (record number) ในเอฟซีบีก่อนที่จะเรียกใช้คำสั่งในการเข้าถึงระเบียนข้อมูล

โดยทั่วไปแล้ว คำสั่งของเอ็มเอส-ดอสที่ใช้เอฟซีบี จะรับตำแหน่งของเอฟซีบีไว้ที่รีจิสเตอร์ DS:DX และส่งรหัสตอบกลับ (return code) มากับรีจิสเตอร์ AL สำหรับคำสั่งในการจัดการแฟ้มข้อมูล (เปิด ปิด สร้าง และลบทิ้ง) นั้น จะส่งรหัสตอบกลับเป็นศูนย์ ถ้าทำคำสั่งสำเร็จ และจะส่งรหัสตอบกลับเป็น 0FFH (255) ถ้าทำ

คำสั่งไม่สำเร็จ แต่สำหรับคำสั่งในการจัดการระเบียนข้อมูลแบบเอฟซีบี ถ้าทำคำสั่งสำเร็จ จะส่งรหัสตอบกลับเป็นศูนย์เช่นกัน แต่ถ้าไม่สำเร็จ จะส่งรหัสตอบกลับมาหลายค่า แล้วแต่ความผิดพลาดที่เกิดขึ้น

ฟังก์ชันเอฟซีบีทั้งหมดที่เกี่ยวกับการจัดการระเบียนข้อมูลและแฟ้มข้อมูลมี 15 ฟังก์ชัน ดังในตาราง 2.2

ฟังก์ชัน	การประมวลผล
0FH	เปิดแฟ้มข้อมูล (Open file)
10H	ปิดแฟ้มข้อมูล (Close file)
16H	สร้างแฟ้มข้อมูล (Create file)
14H	อ่านระเบียนข้อมูลแบบตามลำดับ (Sequential read)
15H	บันทึกระเบียนข้อมูลแบบตามลำดับ (Sequential write)
21H	อ่านระเบียนข้อมูลแบบสุ่ม (Random read)
22H	บันทึกระเบียนข้อมูลแบบสุ่ม (Random write)
27H	อ่านบล๊อคของระเบียนข้อมูลที่กำหนด (Random block read)
28H	บันทึกบล๊อคของระเบียนข้อมูลที่กำหนด (Random block write)
1AH	กำหนดตำแหน่งบัฟเฟอร์ในการย้ายข้อมูล (Set disk transfer address)
29H	ส่งชื่อแฟ้มข้อมูล (Parse filename)
13H	ลบแฟ้มข้อมูล (Delete file)
17H	เปลี่ยนชื่อแฟ้มข้อมูล (Rename file)
23H	คำนวณขนาดของแฟ้มข้อมูล (Obtain file size)
24H	กำหนดหมายเลขระเบียนข้อมูลแบบสุ่ม (Set random record number)

ตารางที่ 2.2 แสดงฟังก์ชันเอฟซีบี ที่เกี่ยวกับการจัดการระเบียนข้อมูลและแฟ้มข้อมูล

มีหลายฟังก์ชันในตารางที่มีคุณสมบัติพิเศษ ตัวอย่างเช่น ฟังก์ชัน 27H และฟังก์ชัน 28H สามารถอ่านและบันทึกครั้งละหลายระเบียนโดยไม่จำกัดขนาด ซึ่งจะทำการปรับปรุงเขตข้อมูลระเบียนข้อมูลแบบสุ่มให้โดยอัตโนมัติ นอกจากนี้ ฟังก์ชัน 28H สามารถยุบ (truncate) แฟ้มข้อมูลได้ตามต้องการ เป็นต้น

2. ฟังก์ชันแฮนเดิล (Handle File and Record Function)

ฟังก์ชันแฮนเดิล คือ ชุดคำสั่งของเอ็มเอส/ดอส ที่ใช้ในการจัดการระเบียนข้อมูลและแฟ้มข้อมูล ที่เลียนแบบมาจากของระบบปฏิบัติการยูนิกซ์/ซีนิกซ์ แฟ้มข้อมูลจะถูกกำหนดโดยสตริงค์ ASCIIZ (นั่นคือ อักขระแอสกีใด ๆ ที่ปิดท้ายสตริงค์ด้วยค่า 00) ซึ่งทำให้ผู้ใช้ระบบสามารถกำหนดเครื่องอ่าน-บันทึกจานแม่เหล็ก ไดเรคตอรีย่อย(ถ้ามี) ชื่อแฟ้มข้อมูล และส่วนขยายชื่อแฟ้มข้อมูล นั่นคือสามารถใช้กับโครงสร้างแฟ้มข้อมูลแบบ hierarchical ได้ซึ่งเริ่มใช้กับระบบปฏิบัติการเอ็มเอส-ดอสตั้งแต่รุ่น 2.0 เป็นต้นมาได้ ดังนั้น การเข้าถึงแฟ้มข้อมูลจึงกระทำได้ตลอด ไม่ว่าจะอยู่ในไดเรคตอรีย่อยใด ๆ ตัวอย่างเช่น ข้อกำหนดของแฟ้มข้อมูล

```
C:\SYSTEM\COMMAND.COM
```

จะปรากฏในหน่วยความจำในลักษณะเรียงลำดับไบท์ได้ดังนี้

```
43 3A 5C 53 59 53 54 45 4D 5C 43 4F 4D 4D 41 4E 44 2E 43 4F 4D 00
```

เมื่อโปรแกรมต้องการสร้างหรือเปิดแฟ้มข้อมูล จะต้องทำการส่งค่าตำแหน่งของสตริงค์ ASCIIZ ที่ระบุชื่อแฟ้มข้อมูลไปให้เอ็มเอส-ดอสทราบโดยเก็บในรีจิสเตอร์ DS:DX ซึ่งถ้าการประมวลผลทำได้สำเร็จ ระบบจะส่งค่าแฮนเดิล ซึ่งเป็นตัวเลข ขนาด 16 บิทกลับมาให้โปรแกรม โดยเก็บไว้ในรีจิสเตอร์ AX เมื่อจะมีการประมวลผลแฟ้มข้อมูล แฮนเดิลจะถูกเก็บไว้ในรีจิสเตอร์ BX ก่อนที่จะใช้คำสั่งเรียกฟังก์ชันของเอ็มเอส-ดอสที่เกี่ยวกับแฮนเดิล หลังจากที่ระบบทำงานตามฟังก์ชันที่เรียกใช้แล้ว ถ้าการปฏิบัติงานเป็นผลสำเร็จ ค่า carry flag จะถูกลบ แต่ถ้าไม่สำเร็จ carry flag จะถูกกำหนด โดยรหัสแสดงข้อผิดพลาดจะถูกเก็บอยู่ในรีจิสเตอร์ AX

จำนวนของแฮนเดิลที่สามารถใช้ได้ในการประมวลผลครั้งหนึ่ง ๆ นั้นก็

คือจำนวนของแฟ้มข้อมูลและอุปกรณ์ (devices) ที่สามารถเปิดใช้ได้พร้อม ๆ กันนั้น ปกติจะมีได้ 8 แฮนเดิลเท่านั้น แต่ผู้ใช้สามารถเปลี่ยนแปลงจำนวนของแฮนเดิลได้ โดยระบุข้อความ

FILES = nn เมื่อ nn เป็นตัวเลขระบุจำนวนแฮนเดิลสูงสุด

ซึ่งข้อความนี้อยู่ในแฟ้มข้อมูล CONFIG.SYS โดย nn เป็นจำนวนสูงสุดของแฟ้มข้อมูลที่จะกำหนดในตารางการเปิดแฟ้มข้อมูลระบบ (System Open-file Table) ซึ่งภายใต้ระบบปฏิบัติการเอ็มเอส-ดอส ตั้งแต่รุ่นที่ 3 นั้น ค่ากำหนดล่วงหน้าของ nn คือ 8 และค่าสูงสุดคือ 255

ฟังก์ชันแฮนเดิลทั้งหมดที่เกี่ยวกับการจัดการระเบียนข้อมูลและแฟ้มข้อมูลมี 12 ฟังก์ชัน ดังในตาราง 2.3

ฟังก์ชัน	การประมวลผล
3CH	สร้างแฟ้มข้อมูล (Create file)
3DH	เปิดแฟ้มข้อมูล (Open file)
3EH	ปิดแฟ้มข้อมูล (Close file)
42H	กำหนดตำแหน่งตัวชี้แฟ้มข้อมูล (Set file pointer)
3FH	อ่านระเบียนข้อมูล (Read record)
40H	บันทึกระเบียนข้อมูล (Write record)
41H	ลบแฟ้มข้อมูล (Delete file)
43H	รับหรือปรับปรุงข้อกำหนดแฟ้มข้อมูล (Get or modify attributes)
56H	เปลี่ยนชื่อแฟ้มข้อมูล (Rename file)
57H	รับหรือกำหนดวันที่และเวลาของแฟ้มข้อมูล (Get or set file date and time)
5AH	สร้างแฟ้มข้อมูลชั่วคราว (Create temporary file)
5BH	สร้างแฟ้มข้อมูลใหม่ไม่ซ้ำชื่อเดิม (Create file (fails if file already exists; version 3 only))

ตารางที่ 2.3 แสดงฟังก์ชันแฮนเดิล ที่เกี่ยวกับการจัดการระเบียนข้อมูลและแฟ้มข้อมูล

เนื่องจากโปรแกรมมินิ-ไมโคร ซีดีเอส/ไอซิส ใช้เทคนิคการจัดการเกี่ยวกับแฟ้มข้อมูลและระเบียนข้อมูลแบบฟังก์ชันแฮนเดิล จึงขอแสดงรายละเอียดเกี่ยวกับฟังชั่นต่าง ๆ ดังนี้

1. <u>การสร้าง/ยุบแฟ้มข้อมูล (Create/truncate file) โดยใช้อินเทอร์รัพท์ 21H ฟังก์ชัน 3CH</u>

ฟังก์ชัน 3CH ทำหน้าที่ในการสร้างแฟ้มข้อมูล หรือ ยุบแฟ้มข้อมูล โดยป้อนสตริงค์ ASCIIZ เพื่อกำหนดแฟ้มข้อมูล ระบบจะสร้างแฟ้มข้อมูลให้ใหม่เก็บไว้ที่ไดเรคตอรี่ที่ระบุที่เครื่องอ่าน-บันทึกจานแม่เหล็กที่ระบุ ถ้าชื่อแฟ้มข้อมูลที่ระบุซ้ำกับชื่อแฟ้มข้อมูลที่มีอยู่แล้ว ระบบจะลบข้อมูลในแฟ้มข้อมูลที่มีอยู่แล้วนั้นทั้งหมด เมื่อระบบทำงานเสร็จแล้วจะส่งค่าแฮนเดิลประจำแฟ้มข้อมูลนั้น ๆ กลับมาไว้ใช้งานต่อไป การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 3CH

CX = คุณลักษณะของแฟ้มข้อมูล

- 00H ถ้าเป็นแฟ้มข้อมูลปกติ
- 01H ถ้าเป็นแฟ้มข้อมูลที่อ่านได้เท่านั้น
- 02H ถ้าเป็นแฟ้มข้อมูลที่ไม่แสดงชื่อ
- 04H ถ้าเป็นแฟ้มข้อมูลระบบ

DS:DX = เซกเมนต์:ออฟเซตของสตริงค์ ASCIIZ ที่กำหนดแฟ้ม

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ

Carry flag = ลบ

AX = แฮนเดิล

ถ้าการปฏิบัติงานไม่สำเร็จ

Carry flag = กำหนด

AX = รหัสแสดงข้อผิดพลาด

1	ถ้าหมายเลขฟังก์ชันไม่ถูกต้อง
2	ถ้าไม่พบแฟ้มข้อมูล
3	ถ้าค้นหาไดเรคตอรี่ไม่พบ
4	ถ้าแฮนเดิลไม่พอ
5	ถ้าเข้าถึงแฟ้มข้อมูลไม่ได้
OCH	ถ้ารหัสการเข้าถึงแฟ้มข้อมูลไม่ถูกต้อง

ตัวอย่างการใช้ฟังก์ชัน 3DH ในการเปิดแฟ้มข้อมูลแบบอ่าน/บันทึก โดยระบุชื่อแฟ้มข้อมูลในสตริงค์ fname (ได้แก่ แฟ้มข้อมูล ในเครื่องอ่าน-บันทึกจานแม่เหล็ก C ในไดเรคตอรี่ย่อยชื่อ MYDIR ชื่อแฟ้มข้อมูล MYFILE.DAT) และเก็บค่าแฮนเดิลไว้ใน ตัวแปร handle เพื่อใช้งานต่อไปเป็นดังภาพที่ 2.47

```
            mov   ah,3ch                ;function number
            xor   cx,cx                 ;file attribute = normal
            mov   dx,seg fname          ;address of file
            mov   ds,dx                 ;specification
            mov   dx,offset fname
            int   21h                   ;transfer to DOS.
            jc    failure               ;jump, create failed
            mov   handle,ax             ;create successful,
            .                           ;save file handle
failure:    .
            .
fname       db    'C:\MYDIR\MYFILE.DAT',0
handle      dw    0
```

ภาพที่ 2.47 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 3CH ในการสร้างแฟ้มข้อมูล

2. การเปิดแฟ้มข้อมูล (Open file) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 3DH

ฟังก์ชัน 3DH ทำหน้าที่ในการเปิดแฟ้มข้อมูลที่มีอยู่แล้วในจานแม่เหล็ก (ต่างจากการสร้าง/ยุบ เพราะถ้าจะสร้างหรือยุบ แสดงว่าผู้ใช้ไม่ต้องการข้อมูลที่มีอยู่เดิมในแฟ้มข้อมูล แต่การเปิดแฟ้มข้อมูลแสดงว่าผู้ใช้แน่ใจว่าเป็นแฟ้มข้อมูลที่มีอยู่แล้วและต้องการใช้ข้อมูลที่มีอยู่ในแฟ้มข้อมูลนั้น ๆ) โดยระบุสตริงค์ ASCIIZ ที่กำหนดแฟ้มข้อมูล ระบบจะไปทำการค้นหาแฟ้มข้อมูลที่ระบุแล้วเปิดแฟ้มข้อมูลดังกล่าว เมื่อระบบทำงานเสร็จแล้วจะส่งค่าแฮนเดิลประจำแฟ้มข้อมูลนั้น ๆ กลับมาไว้ใช้งานต่อไป การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 3DH
AL = ลักษณะการเข้าถึงแฟ้มข้อมูล
บิท 0-2 จะเป็น
000 ถ้าเป็นการเข้าถึงแบบอ่าน
001 ถ้าเป็นการเข้าถึงแบบบันทึก
010 ถ้าเป็นการเข้าถึงแบบอ่าน/บันทึก
DS:DX = เซกเมนต์:ออฟเซตของสตริงค์ ASCIIZ ที่กำหนดแฟ้ม

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ
Carry flag = ลบ
AX = แฮนเดิล

ถ้าการปฏิบัติงานไม่สำเร็จ
Carry flag = กำหนด
AX = รหัสแสดงข้อผิดพลาด
3 ถ้าค้นหาไดเรคตอรี่ไม่พบ
4 ถ้าแฮนเดิลไม่พอ
5 ถ้าเข้าถึงแฟ้มข้อมูลไม่ได้

ตัวอย่างการใช้ฟังก์ชัน 3DH ในการเปิดแฟ้มข้อมูล โดยระบุชื่อแฟ้มข้อมูลในสตริงค์ fname (ได้แก่ แฟ้มข้อมูล ในเครื่องอ่าน-บันทึกจานแม่เหล็ก C ในไดเรคตอรี่ย่อยชื่อ MYDIR ชื่อแฟ้มข้อมูล MYFILE.DAT) และเก็บค่าแฮนเดิลไว้ในตัวแปร handle เพื่อใช้งานต่อไปเป็นดังภาพที่ 2.48

```
            mov   ah,3dh                ;function number
            mov   al,2                  ;access mode=read/write
            mov   dx,seg fname          ;address of ASCIIZ file
            mov   ds,dx                 ;specification
            mov   dx,offset fname
            int   21h                   ;transfer to DOS.
            jc    failure               ;jump if open failed
            mov   handle,ax             ;open was successful,
            .                           ;save file handle
failure:    .
            .
fname       db    'C:\MYDIR\MYFILE.DAT',0
handle      dw    0
```

ภาพที่ 2.48 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 3DH ในการเปิดแฟ้มข้อมูล

3. <u>การปิดแฟ้มข้อมูล (Close file) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 3EH</u>

ฟังก์ชัน 3EH ทำหน้าที่ในการปิดแฟ้มข้อมูลที่ถูกเปิดโดยการใช้ฟังก์ชัน 3DH, 3CH, 5AH หรือ 5BH โดยการระบุหมายเลขแฮนเดิลของแฟ้มข้อมูลที่ต้องการจะปิดแล้วเรียกฟังก์ชันนี้ ระบบจะทำการย้ายข้อมูลที่ตกค้างอยู่ในบัฟเฟอร์ (ถ้ามี) เก็บลงจานแม่เหล็ก ปิดแฟ้มข้อมูล แล้วคืนหมายเลขแฮนเดิลให้กับระบบเพื่อนำไปใช้กับแฟ้มข้อมูลอื่น ๆ ที่จะเปิดใหม่ ถ้าแฟ้มข้อมูลถูกปรับปรุง เวลาและวันที่ กับ

ขนาดของแฟ้มข้อมูลใหม่ก็จะถูกปรับปรุงด้วยในรายการไดเรคตอรี่ การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 3EH
BX = แฮนเดิล

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ
Carry flag = ลบ

ถ้าการปฏิบัติงานไม่สำเร็จ
Carry flag = กำหนด
AX = รหัสแสดงข้อผิดพลาด
6 ถ้าแฮนเดิลไม่ถูกต้องหรือไม่ได้เปิด

ตัวอย่างการใช้ฟังก์ชัน 3EH ในการปิดแฟ้มข้อมูล โดยระบุหมายเลขแฮนเดิลที่ต้องการปิดในตัวแปร handle เป็นดังภาพที่ 2.49

```
        mov   ah,3eh              ;function number
        mov   bx,handle           ;handle of previously
                                  ;opened file
        int   21h                 ;transfer to DOS.
        jc    failure             ;jump, close failed
        .
failure:.
        .
handle  dw    0
```

ภาพที่ 2.49 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 3EH ในการปิดแฟ้มข้อมูล

4. การกำหนดตำแหน่งตัวชี้แฟ้มข้อมูล (Move file pointer) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 42H

ฟังก์ชัน 42H ทำหน้าที่ในการกำหนดตำแหน่งตัวชี้แฟ้มข้อมูลในการอ่านหรือบันทึกข้อมูล โดยสามารถกำหนดได้ 3 วิธี คือ สัมพัทธ์กับต้นแฟ้มข้อมูล, สัมพัทธ์กับท้ายแฟ้มข้อมูล หรือสัมพัทธ์กับตำแหน่งปัจจุบันของแฟ้มข้อมูล การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 42H

AL = รหัสเลือกวิธีกำหนด

- 0 ถ้าใช้วิธีสัมพัทธ์กับต้นแฟ้มข้อมูล (absolute byte offset from beginning of file) always positive double integer
- 1 ถ้าใช้วิธีสัมพัทธ์กับตำแหน่งปัจจุบัน (byte offset from current location) positive or negative double integer
- 2 ถ้าใช้วิธีสัมพัทธ์กับท้ายแฟ้มข้อมูล (byte offset from end of file) positive or negative double integer

BX = แฮนเดิล

CX = ค่า offset ที่มีนัยสำคัญสูงสุด (most significant half of offset)

DX = ค่า offset ที่มีนัยสำคัญต่ำสุด (least significant half of offset)

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ

Carry flag = ลบ

DX = ตำแหน่งตัวชี้ใหม่ในส่วนที่มีนัยสำคัญสูงสุด (most significant part of new pointer location)

AX = ตำแหน่งตัวชี้ใหม่ในส่วนที่มีนัยสำคัญต่ำสุด (least significant part of new pointer location)

ถ้าการปฏิบัติงานไม่สำเร็จ

Carry flag = กำหนด

AX = รหัสแสดงข้อผิดพลาด

1 ถ้าหมายเลขฟังก์ชันไม่ถูกต้อง

2 ถ้าหมายเลขแฮนเดิลไม่ถูกต้องหรือไม่ได้เปิด

ตัวอย่างการใช้ฟังก์ชัน 42H ในการกำหนดตำแหน่งตัวชี้แฟ้มข้อมูลของแฟ้มข้อมูลที่ค่าแฮนเดิลเก็บอยู่ในตัวแปร fhandle และต้องการชี้ไปที่ตำแหน่งที่อยู่ถัดจากต้นแฟ้มข้อมูล 1,024 ไบท์ จะเป็นดังภาพที่ 2.50

```
        mov   ah,42h            ;function number
        mov   al,0              ;method=absolute offset
        mov   bx,fhandle        ;handle of previously
                                ;opened file
        mov   cx,0              ;upper part of offset
        mov   dx,1024           ;lower part of offset
        int   21h               ;transfer to DOS.
        jc    error             ;jump, function failed
        .
error:  .
fhandle dw    0                 ;file handle
```

ภาพที่ 2.50 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 42H ในการกำหนดตำแหน่งตัวชี้แฟ้มข้อมูล

5. การอ่านข้อมูล (Read file or device) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 3FH

ฟังก์ชัน 3FH ทำหน้าที่ในการอ่านข้อมูลจากแฮนเดิลที่ระบุใส่บัฟเฟอร์ในหน่วยความจำ แล้วปรับปรุงตำแหน่งของตัวชี้แฟ้มข้อมูล การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 3FH
BX = แฮนเดิลของแฟ้มข้อมูลที่จะอ่าน
CX = จำนวนไบท์ที่ต้องการอ่านข้อมูล
DS:DX = ค่า เซกเมนต์:ออฟเซต ของบัฟเฟอร์

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ
Carry flag = ลบ
AX = จำนวนไบท์ที่อ่านได้
0 ถ้าจบแฟ้มข้อมูล

ถ้าการปฏิบัติงานไม่สำเร็จ
Carry flag = กำหนด
AX = รหัสแสดงข้อผิดพลาด
5 ถ้าไม่สามารถเข้าถึงแฟ้มข้อมูล
6 ถ้าหมายเลขแฮนเดิลไม่ถูกต้องหรือไม่ได้เปิด

ตัวอย่างการใช้ฟังก์ชัน 3FH ในการอ่านข้อมูลจากตำแหน่งตัวชี้แฟ้มข้อมูลของแฟ้มข้อมูลที่ค่าแฮนเดิลเก็บอยู่ในตัวแปร file1 จำนวน 1,024 ไบท์ แล้วนำไปจัดเก็บไว้ที่บัฟเฟอร์ชื่อ databuf จะเป็นดังภาพที่ 2.51

```
            mov   ah,3Fh                ;function number
            mov   bx,file1              ;handle of previously
                                        ;opened file
            mov   cx,1024               ;length to read
            mov   dx,seg databuf        ;address of read buffer
            mov   ds,dx
            mov   dx,offset databuf
            int   21h                   ;transfer to DOS.
            jc    failure               ;jump, read failed
            .
failure:    .
            .
databuf:    db     1024 dup (?)         ;buffer for read
file1:      dw     0                    ;contains file handle
```

ภาพที่ 2.51 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 3FH ในการอ่านข้อมูลจากแฟ้มข้อมูล

6. <u>การบันทึกข้อมูล (Write to file or device) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 40H</u>

ฟังก์ชัน 40H ทำหน้าที่ในการบันทึกข้อมูลจากบัฟเฟอร์ในหน่วยความจำเก็บในแฟ้มข้อมูลที่ระบุโดยหมายเลขแฮนเดิล แล้วปรับปรุงตำแหน่งของตัวชี้แฟ้มข้อมูล การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 40H
BX = แฮนเดิลของแฟ้มข้อมูลที่จะบันทึก
CX = จำนวนไบท์ที่ต้องการบันทึกข้อมูล
DS:DX = ค่า เซกเมนต์:ออฟเซต ของบัฟเฟอร์

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ

Carry flag = ลบ

AX = จำนวนไบท์ที่บันทึกได้

0 ถ้าจานแม่เหล็กเต็ม

ถ้าการปฏิบัติงานไม่สำเร็จ

Carry flag = กำหนด

AX = รหัสแสดงข้อผิดพลาด

5 ถ้าไม่สามารถเข้าถึงแฟ้มข้อมูล

6 ถ้าหมายเลขแฮนเดิลไม่ถูกต้องหรือไม่ได้เปิด

ตัวอย่างการใช้ฟังก์ชัน 40H ในการบันทึกข้อมูลจากบัฟเฟอร์ในหน่วยความจำ databuf จำนวน 1,024 ไบท์ เก็บในแฟ้มข้อมูลที่ระบุโดยหมายเลขแฮนเดิลซึ่งอยู่ในตัวแปร file1 โดยบันทึก ณ ตำแหน่งปัจจุบันของตัวชี้แฟ้มข้อมูล จะเป็นดังภาพที่ 2.52

```
mov   ah,40h                ;function number
mov   bx,file1              ;handle of previously
                            ;opened file
mov   cx,1024               ;length to write
mov   dx,seg databuf        ;address of buffer for
mov   ds,dx                 ;record to be written
mov   dx,offset databuf
int   21h                   ;transfer to DOS.
jc    failure               ;jump, write failed
```

ภาพที่ 2.52 ตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 40H ในการบันทึกข้อมูลเก็บแฟ้มข้อมูล

```
            cmp   ax,1024              ;entire record written ?
            jne   diskfull             ;no, jump.
            .
diskfull:   .
            .
failure:    .
            .
            .
databuf:    db     1024 dup (?)        ;buffer for write
file1:      dw     0                   ;contains file handle
```

ภาพที่ 2.52 ตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 40H ในการบันทึกข้อมูลเก็บในแฟ้มข้อมูล (ต่อ)

7. การลบแฟ้มข้อมูล (Delete file) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 41H

ฟังก์ชัน 41H ทำหน้าที่ในการลบแฟ้มข้อมูลที่ระบุ โดยระบุสตริงค์ ASCIIZ ที่กำหนดแฟ้มข้อมูล ระบบจะไปทำการค้นหาแฟ้มข้อมูลที่ระบุแล้วลบแฟ้มข้อมูลดังกล่าว การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 41H

DS:DX = เซกเมนต์:ออฟเซตของสตริงค์ ASCIIZ ที่กำหนดแฟ้มข้อมูล

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ

Carry flag = ลบ

ถ้าการปฏิบัติงานไม่สำเร็จ

Carry flag = กำหนด

AX = รหัสแสดงข้อผิดพลาด

2 ถ้าไม่มีแฟ้มข้อมูลที่ระบุ

5 ถ้าเข้าถึงแฟ้มข้อมูลไม่ได้

ตัวอย่างการใช้ฟังก์ชัน 41H ในการลบแฟ้มข้อมูล MYFILE.DAT ซึ่งอยู่ในไดเรคตอรี่ย่อยชื่อ \MYDIR ในเครื่องอ่าน-บันทึกจานแม่เหล็ก C โดยได้ทำการระบุสตริงค์ ASCIIZ ที่กำหนดแฟ้มข้อมูลไว้ที่ตัวแปร fname เป็นดังภาพที่ 2.53

```
        mov   ah,41h                ;function number
        mov   dx,seg fname          ;address of ASCIIZ file
        mov   ds,dx                 ;specification
        mov   dx,offset fname
        int   21h                   ;transfer to DOS.
        jc    failure               ;jump delete failed
        .
failure: .
        .
fname   db    'C:\MYDIR\MYFILE.DAT',0
```

ภาพที่ 2.53 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 41H ในการลบแฟ้มข้อมูล

8. การรับหรือกำหนดคุณลักษณะของแฟ้มข้อมูล (Get of set file attributes) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 43H

ฟังก์ชัน 43H ทำหน้าที่ในการรับหรือกำหนดคุณลักษณะของแฟ้มข้อมูล โดยสามารถกำหนดได้ 4 วิธี คือ อ่านได้อย่างเดียว, ไม่แสดงชื่อแฟ้มข้อมูล,

แฟ้มข้อมูลระบบ และแฟ้มข้อมูลปรกติ การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 43H

AL = 00H ถ้าเป็นการรับคุณลักษณะของแฟ้มข้อมูล

01H ถ้าเป็นการกำหนดคุณลักษณะของแฟ้มข้อมูล

CX = คุณลักษณะใหม่ ถ้า AL = 01

บิท 5 = แฟ้มข้อมูลปรกติ

บิท 2 = แฟ้มข้อมูลระบบ

บิท 1 = แฟ้มข้อมูลที่ไม่แสดงชื่อ

บิท 0 = แฟ้มข้อมูลที่อ่านได้เท่านั้น

DS:DX = เซกเมนต์:ออฟเซตของสตริงค์ ASCIIZ ที่กำหนดแฟ้มข้อมูล

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ

Carry flag = ลบ

AX = 00 on call

CX = คุณลักษณะ

ถ้าการปฏิบัติงานไม่สำเร็จ

Carry flag = กำหนด

AX = รหัสแสดงข้อผิดพลาด

1 ถ้าหมายเลขฟังก์ชันไม่ถูกต้อง

2 ถ้าไม่มีแฟ้มข้อมูลที่ระบุ

3 ถ้าค้นหาไดเรคตอรี่ไม่พบ

5 ถ้าเปลี่ยนคุณลักษณะแฟ้มข้อมูลไม่ได้

ตัวอย่างการใช้ฟังก์ชัน 43H ในการรับหรือกำหนดคุณลักษณะของแฟ้มข้อมูล โดยเปลี่ยนคุณลักษณะของแฟ้มข้อมูล D:\MYDIR\MYFILE.DAT เป็นชนิดอ่านได้เท่านั้น จะเป็นดังภาพที่ 2.54

```
read_only equ  01h
hidden    equ  02h
system    equ  04h
volume    equ  08h
subdir    equ  10h
archive   equ  20h
          .
          .
          .
          .
          mov  ah,43h              ;function number call
          mov  al,01               ;is modify attribute
          mov  cx,read_only        ;load desire attribute
          mov  dx,seg fname        ;address of ASCIIZ
          mov  ds,dx               ;file specification
          mov  dx,offset fname
          int  21h                 ;transfer to DOS.
          jc   failure             ;jump, function failed
          .
failure:  .
          .
fname:    db   'C:\MYDIR\MYFILE.DAT',0
          .
          .
          .
```

ภาพที่ 2.54 ตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 43H ในการรับ/กำหนดคุณลักษณะแฟ้มข้อมูล

9. การเปลี่ยนชื่อแฟ้มข้อมูล (Rename file) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 56H

ฟังก์ชัน 56H ทำหน้าที่ในการเปลี่ยนชื่อแฟ้มข้อมูลรวมทั้งย้ายแฟ้มข้อมูลไปยังไดเรคตอรี่อื่น ๆ ภายในเครื่องอ่าน-บันทึกจานแม่เหล็กนั้น ๆ การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

AH = 56H
DS:DX = เซกเมนต์:ออฟเซตของสตริงค์ ASCIIZ กำหนดชื่อแฟ้มข้อมูลเดิม
ES:DI = เซกเมนต์:ออฟเซตของสตริงค์ ASCIIZ กำหนดชื่อแฟ้มข้อมูลใหม่

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ
Carry flag = ลบ

ถ้าการปฏิบัติงานไม่สำเร็จ
Carry flag = กำหนด
AX = รหัสแสดงข้อผิดพลาด
2 ถ้าไม่มีแฟ้มข้อมูลที่ระบุ
3 ถ้าค้นหาไดเรคตอรี่ไม่พบ
5 ถ้าเข้าถึงแฟ้มข้อมูลไม่ได้
11H ถ้าไม่ใช่อุปกรณ์เดียวกัน

ตัวอย่างการใช้ฟังก์ชัน 56H ในการเปลี่ยนชื่อแฟ้มข้อมูลจากแฟ้มข้อมูล MYFILE.DAT ซึ่งอยู่ในไดเรคตอรี่ย่อย \MYDIR ในเครื่องอ่าน-บันทึกจานแม่เหล็ก C ให้เป็นชื่อ MYTEXT.DAT ซึ่งอยู่ในไดเรคตอรี่ย่อย \SYSTEM ในเครื่องอ่าน-บันทึกจานแม่เหล็ก C เช่นกัน จะเป็นดังภาพที่ 2.55

```
            mov   ah,56h                ;function number call
            mov   dx,set old_name       ;address of ASCIIZ
            mov   ds,dx                 ;specification for
            mov   dx,off old_name       ;old file name
            mov   di,seg new_name       ;address of ASCIIZ
            mov   es,di                 ;specification for
            mov   di,off new_name       ;new file name
            int   21h                   ;transfer to DOS.
            jc    failure               ;jump, rename failed
            .
failure:    .
            .
old_name    db    'C:\MYDIR\MYFILE.DAT',0
new_name    db    'C:\SYSTEM\MYTEXT.DAT',0

 .
```

ภาพที่ 2.55 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 56H ในการเปลี่ยนชื่อแฟ้มข้อมูล

10. การอ่านหรือกำหนดวันเดือนปีและเวลาของระบบ (Get or set file date and time) โดยใช้ อินเทอร์รัพท์ 21H ฟังก์ชัน 57H

ฟังก์ชัน 57H ทำหน้าที่ในการอ่านหรือกำหนดวันเดือนปีและเวลาที่บันทึกอยู่ในหน่วยไดเรคตอรี่ของแฟ้มข้อมูลแต่ละแฟ้มข้อมูล การเรียกฟังก์ชันนี้ให้ทำงานจะต้องกำหนดค่าต่าง ๆ ดังนี้

ถ้าเป็นการอ่านวันเดือนปีและเวลาจากหน่วยไดเรคตอรี่ของแฟ้มข้อมูล

AH = 57H

AL = 00

BX = แฮนเดิล

ถ้าเป็นการกำหนดวันเดือนปีและเวลาไปเก็บที่หน่วยไดเรคตอรี่ของแฟ้ม

AH = 57H

AL = 01

BX = แฮนเดิล

CX = เวลา

บิท 0BH-0FH = ชั่วโมง (0 ถึง 23)

บิท 05H-0AH = นาที (0 ถึง 59)

บิท 00H-04H = จำนวนครั้ง ๆ ละ 2 วินาที (0 ถึง 29)

DX = วันเดือนปี

บิท 09-0FH = ปี (สัมพัทธ์กับปี ค.ศ. 1980)

บิท 05-08H = เดือน (0 ถึง 12)

บิท 09-04H = วันที่ (0 ถึง 12)

ผลลัพธ์จากการทำงานของฟังก์ชันจะเป็นดังนี้

ถ้าการปฏิบัติงานสำเร็จ

Carry flag = ลบ

ถ้าเป็นการอ่านวันเดือนปีและเวลา

CX = เวลา

DX = วันเดือนปี

ถ้าการปฏิบัติงานไม่สำเร็จ

Carry flag = กำหนด

AX = รหัสแสดงข้อผิดพลาด

1 ถ้าหมายเลขฟังก์ชันไม่ถูกต้อง

6 ถ้าหมายเลขแฮนเดิลไม่ถูกต้องหรือไม่ได้เปิด

ตัวอย่างการใช้ฟังก์ชัน 57H ในการอ่านวันเดือนปีและเวลาที่ได้ปรับปรุงแฟ้มข้อมูล MYFILE.DAT ไว้หลังสุด แล้วแยกเดือนเก็บไว้ที่ตัวแปร month แยกวันที่เก็บไว้ที่ตัวแปร day และเก็บปีไว้ที่ตัวแปร year จะเป็นดังภาพที่ 2.56

```
        mov   ah,3Dh                ;function number
        mov   al,0                  ;mode = read-only
        mov   dx,seg fname          ;DS:DX = filename
        mov   ds,dx
        mov   dx,offset fname
        int   21h                   ;transfer to DOS.
        jc    op_err                ;jump if open failed
        mov   bx,ax                 ;BX = file handle
        mov   ah,57h                ;function get date/time
        mov   al,0                  ;AL = 0 to get date/time
        int   21h                   ;transfer to DOS.
        jc    get_err               ;jump if get failed
        mov   day,dx                ;decompose date
        and   day,01fh              ;day of month
        mov   cl,5
        shr   dx,cl
        mov   month,dx              ;month of year
        and   month,0fh
        mov   cl,4
        shr   dx,cl
        and   dx,03fh               ;year relative to 1980
        add   dx,1980               ;correct to real year
        mov   year,dx
        mov   ah,3eh                ;close file
```

ภาพที่ 2.56 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 57H ในการอ่านวันเดือนปีและเวลา

```
            int   21h                   ;transfer to DOS.
            jc    cl_err                ;jump if close failed
            .
op_err:     .
            .
get_err:    .
            .
cl_err:     .
            .
month       dw    0
day         dw    0
year        dw    0
fname       db    'MYFILE.DAT',0
```

ภาพที่ 2.56 แสดงตัวอย่างโปรแกรมที่ใช้ฟังก์ชัน 57H ในการอ่านวันเดือนปีและเวลา (ต่อ)