

เอกสารประกอบการสอน

รายวิชา ไมโครโพรเซสเซอร์และภาษาแอสเซมบลี
(Microprocessor and Assembly Language)

จุฑาวุฒิ จันทร์มาลี

วท.บ., วท.ม.(วิทยาการคอมพิวเตอร์)

คณะวิทยาศาสตร์และเทคโนโลยี
มหาวิทยาลัยราชภัฏสวนดุสิต

2557

คำนำ

เอกสารประกอบการสอนวิชา ไมโครโพรเซสเซอร์และภาษาแอสเซมบลี รหัสวิชา 4122705 จัดได้ว่าเป็นรายวิชาที่มีความสำคัญต่อการผลิตบัณฑิตด้านคอมพิวเตอร์ ตามหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ นอกจากนั้นเอกสารประกอบการสอนเล่มนี้ได้เรียบเรียงขึ้นเพื่อให้ครูอาจารย์และบุคคลทั่วไปได้ศึกษาและค้นคว้าเพิ่มพูนความรู้และประสบการณ์เกี่ยวกับทักษะและเทคนิครวมทั้งได้ศึกษาการใช้งานไมโครโพรเซสเซอร์และภาษาแอสเซมบลี ทั้งภาคทฤษฎีและปฏิบัติ

สาระสำคัญของเอกสารประกอบการสอนเล่มนี้ ประกอบด้วย 18 บท ได้แก่ ระบบคอมพิวเตอร์ ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ ระบบจำนวน ภาษาสำหรับเขียนโปรแกรมคอมพิวเตอร์ สถาปัตยกรรมคอมพิวเตอร์เบื้องต้น สถาปัตยกรรมคอมพิวเตอร์เบื้องต้น คำสั่งโอนย้ายข้อมูล แฟล็กและคำสั่งคณิตศาสตร์ โปรแกรมภาษาแอสเซมบลีเบื้องต้น การประกาศข้อมูล คำสั่งกระโดดและคำสั่งการกระทำซ้ำ โครงสร้างควบคุม โปรแกรมย่อยเบื้องต้น การกระทำระดับบิต การอ้างแอดเดรส การขัดจังหวะ คำสั่งจัดการกับสายข้อมูล คำสั่งตารางและการสร้างแมคโคร คำสั่งตารางและการสร้างแมคโคร และการฝึกทักษะการเขียนโปรแกรมภาษาแอสเซมบลีเบื้องต้นซึ่งการเรียบเรียงประมวลเอกสารประกอบการสอนตามเนื้อหาดังกล่าวข้างต้นนั้น ผู้เขียนได้คำนึงถึงการจัดการเรียนการสอนที่เน้นผู้เรียนเป็นสำคัญตามพระราชบัญญัติการศึกษาแห่งชาติ รวมทั้งเนื้อหาที่มีความทันสมัยทันเหตุการณ์ ครอบคลุมตามสาระการเรียนรู้ที่พึงมี และนำทักษะและเทคนิคสมัยใหม่ที่เกี่ยวข้องนำมาเสนอประกอบ เพื่อให้เกิดความชัดเจนในองค์ความรู้ของศาสตร์ด้านคอมพิวเตอร์ ผู้เขียนหวังว่าเอกสารประกอบการสอนเล่มนี้คงมีประโยชน์กับท่านผู้อ่านบ้างตามโอกาสอันสมควร

จุฑาวุฒิ จันทร์มาลี
17 มีนาคม 2557

สารบัญ

	หน้า
คำนำ	(1)
สารบัญ	(2)
สารบัญภาพ	(6)
สารบัญตาราง	(8)
แผนบริหารการสอนประจำวิชา	(9)
แผนบริหารการสอนประจำบทที่ 1	1
บทที่ 1 ระบบคอมพิวเตอร์	2
องค์ประกอบของระบบคอมพิวเตอร์	2
ชนิดของคอมพิวเตอร์	3
โครงสร้างของฮาร์ดแวร์ของระบบคอมพิวเตอร์	5
ระบบซอฟต์แวร์ของคอมพิวเตอร์	7
แผนบริหารการสอนประจำบทที่ 2	10
บทที่ 2 ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์	11
ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์	11
ชนิดของไมโครโพรเซสเซอร์	12
โครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์	13
การเกิดอินเทอร์รัพต์	17
รีจิสเตอร์ของซีพียู 8086	18
การระบุตำแหน่งในรีจิสเตอร์	22
การเข้าถึงข้อมูลในหน่วยความจำ	23
แผนบริหารการสอนประจำบทที่ 3	28
บทที่ 3 ระบบจำนวน	29
ความหมายของตัวเลขในหลักต่าง ๆ	29
การแปลงค่าจากเลขฐานสิบเป็นเลขฐานสอง	29
การบวกและการลบเลขฐานสอง	30
การคูณและการหารเลขฐานสอง	31
เลขฐานแปดและเลขฐานสิบหก	32
บิต, ไบต์, เวิร์ด และดับเบิลเวิร์ด	33
ระบบเลข 2' Complement	34
ระบบเลข BCD (Binary Code Decimal)	34
มาตรฐาน IEEE 754	35
แผนบริหารการสอนประจำบทที่ 4	37
บทที่ 4 ภาษาสำหรับเขียนโปรแกรมคอมพิวเตอร์	38
ภาษาต่างๆ สำหรับคอมพิวเตอร์	38
ระดับของภาษาสำหรับเขียนโปรแกรม	38

	หน้า
การแปลภาษาสำหรับคอมพิวเตอร์	38
ภาษาแอสเซมบลี	39
แผนบริหารการสอนประจำบทที่ 5	41
บทที่ 5 สถาปัตยกรรมคอมพิวเตอร์เบื้องต้น	42
หน่วยประมวลผลกลาง	42
หน่วยความจำ	43
การเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ	43
สถาปัตยกรรมของระบบไมโครโปรเซสเซอร์ตระกูล 80x86	44
แผนบริหารการสอนประจำบทที่ 6	52
บทที่ 6 คำสั่งโอนย้ายข้อมูล	53
คำสั่งในการโอนย้ายข้อมูล คำสั่ง MOV	53
เครื่องมือในการทดลองการโปรแกรมภาษาแอสเซมบลี	56
คำสั่งในการโอนย้ายข้อมูล คำสั่งทั่วไป	56
แผนบริหารการสอนประจำบทที่ 7	66
บทที่ 7 แฟล็กและคำสั่งคณิตศาสตร์	67
แฟล็ก (Flags)	67
คำสั่งทางคณิตศาสตร์	70
กลุ่มคำสั่งบวกลบและลบ	70
กลุ่มคำสั่งคูณและหาร	72
กลุ่มคำสั่งแปลงขนาดตัวเลข	73
แผนบริหารการสอนประจำบทที่ 8	79
บทที่ 8 โปรแกรมภาษาแอสเซมบลีเบื้องต้น	80
รูปแบบของโปรแกรมภาษาแอสเซมบลีแบบเก่า	80
รูปแบบของโปรแกรมภาษาแอสเซมบลีแบบใหม่	83
การเรียกใช้บริการของ DOS	84
ขั้นตอนการแปลโปรแกรม	85
ตารางรหัสแอสกี	89
แผนบริหารการสอนประจำบทที่ 9	91
บทที่ 9 การประกาศข้อมูล	92
การประกาศข้อมูล	92
การอ้างใช้ข้อมูลที่ประกาศไว้	94
การอ้างตำแหน่งของข้อมูล	95
การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah	96
การใช้บริการของ DOS หมายเลข 09h : การพิมพ์ข้อความ	96
การใช้บริการของ DOS หมายเลข 0Ah : การอ่านข้อความ	97

	หน้า
แผนบริหารการสอนประจำบทที่ 10	100
บทที่ 10 คำสั่งกระโดดและคำสั่งการกระทำซ้ำ	101
คำสั่งกระโดด	101
คำสั่งวนรอบ	103
แผนบริหารการสอนประจำบทที่ 11	111
บทที่ 11 โครงสร้างควบคุม	112
การสร้างโครงสร้างการตัดสินใจแบบ if-then-else	112
การสร้าง repeat until loop	113
การสร้าง while loop	114
การสร้าง for loop	115
แผนบริหารการสอนประจำบทที่ 12	119
บทที่ 12 โปรแกรมย่อยเบื้องต้น	120
คำสั่งที่รองรับการเรียกโปรแกรมย่อย	120
การประกาศโปรแกรมย่อย	120
คำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแอสตีก	122
ตัวอย่างการใช้งานโปรแกรมย่อย	124
แผนบริหารการสอนประจำบทที่ 13	127
บทที่ 13 การกระทำระดับบิต	128
คำสั่งทางตรรกศาสตร์	108
การประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์	130
คำสั่งเลื่อนบิต	131
ตัวอย่างการใช้งานคำสั่งเลื่อนบิต	131
การประยุกต์ใช้งานคำสั่งเลื่อนบิต	133
คำสั่งหมุนบิต	133
คำสั่งหมุนบิตที่ผ่านแฟล็กทด	134
ตัวอย่างการใช้งานคำสั่งเกี่ยวกับการประมวลผลระดับบิต	135
แผนบริหารการสอนประจำบทที่ 14	139
บทที่ 14 การอ้างแอดเดรส	140
การอ้างแอดเดรส	140
แบบการอ้างแอดเดรส	140
แผนบริหารการสอนประจำบทที่ 15	147
บทที่ 15 การขัดจังหวะ	148
กระบวนการขัดจังหวะใน 8086	148
คำสั่งติดต่อกับอุปกรณ์ใน 8086	151

	หน้า
แผนบริหารการสอนประจำบทที่ 16	159
บทที่ 16 คำสั่งจัดการกับสายข้อมูล	160
คำสั่ง MOVS	161
คำสั่ง REP	161
คำสั่ง STOS	162
คำสั่ง LODS	163
คำสั่ง REPZ	165
คำสั่ง CMPS	165
คำสั่ง REPNZ	166
คำสั่ง SCAS	166
แผนบริหารการสอนประจำบทที่ 17	169
บทที่ 17 คำสั่งตารางและการสร้างแมคโคร	170
คำสั่ง XLAT	170
แมคโคร	170
พารามิเตอร์ของแมคโคร	171
การใช้ LABEL ในแมคโคร	172
แมคโครกับโปรแกรมย่อย	174
แผนบริหารการสอนประจำบทที่ 18	176
บทที่ 18 การฝึกทักษะการเขียนโปรแกรมภาษาแอสเซมบลีเบื้องต้น	177
การดาวน์โหลดและติดตั้งโปรแกรม MASM32	177
การดาวน์โหลดและติดตั้งโปรแกรม EditPlus	197
บรรณานุกรม	213

สารบัญภาพ

ภาพที่		หน้า
1.1	แสดงโครงสร้างของฮาร์ดแวร์ของระบบคอมพิวเตอร์	6
1.2	แสดงแผนภูมิหัวหอม	8
2.1	แสดงบล็อกไดอะแกรมของไมโครโพรเซสเซอร์	11
2.2	แสดงไมโครคอนโทรลเลอร์ (Microcontroller)	12
2.3	แสดง Block Diagram ของ ซีพียู (CPU 8086)	14
2.4	แสดงสถานะแฟล็ก (Flag Status) ที่ใช้ในการตรวจสอบการทำงานของซีพียู (CPU 8086)	15
2.5	แสดงการเกิดอินเตอร์รัพต์ ที่ตำแหน่ง 0000- 03FF ขนาด 1 กิโลไบต์ (1Kbytes)	18
2.6	แสดงรีจิสเตอร์ที่ใช้ในไมโครโพรเซสเซอร์ 8086	19
2.7	แสดงโครงสร้างรีจิสเตอร์ของ 8086 กับการคำนวณหาค่าตำแหน่งในหน่วยความจำ	20
2.8	แสดงการแบ่งกลุ่มของ Segment Register	21
2.9	แสดงตำแหน่งเหมือนกับ Paragraph ใน Segment Register	21
2.10	แสดงการบวกค่าในรีจิสเตอร์ CS และ IP เพื่อคำนวณหา Physical Address	22
2.11	แสดงการบวกค่าในรีจิสเตอร์ DS กับ EA เพื่อหา Physical Address	23
2.12	แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยตรง (Direct Addressing Mode)	24
2.13	แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยอ้อม (Indirect Addressing Mode)	24
2.14	แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้สแต็ก (Stack Addressing Mode)	25
5.1	แสดงขั้นตอนการทำงานของหน่วยประมวลผล	42
5.2	แสดงการเชื่อมต่อของอุปกรณ์ต่าง ๆ ผ่านระบบบัส	43
5.3	แสดงการแปลงแอดเดรสจากแบบเซกเมนต์ : ออฟเซต เป็น แอดเดรสขนาด 20 บิต	45
5.4	แสดงลักษณะการเหลื่อมกันของเซกเมนต์	46
5.5	แสดงการทำงานแบบไปป์ไลน์เทียบกับการทำงานแบบปกติ	49
5.6	แสดงลักษณะของบัสที่มีการใช้หน่วยความจำแคช	50
6.1	แสดงการเรียงไบต์ข้อมูลในหน่วยความจำ	55
7.1	แสดงแฟล็กต่าง ๆ	67
9.1	แสดงการจัดเรียงข้อมูลในหน่วยความจำจากการประกาศในส่วนของโปรแกรมที่ 9.2	93
9.2	แสดงการเปลี่ยนแปลงค่าหลังการทำงานของโปรแกรมที่ 9.5	95
11.1	แสดงโครงสร้างควบคุมแบบ repeat until	113
11.2	แสดงโครงสร้างควบคุมแบบ while	114
11.3	แสดงโครงสร้างควบคุมแบบ for loop	115
12.1	แสดงการทำงานของแอสต์ก	102
12.2	แสดงการเปลี่ยนแปลงของแอสต์กในการเรียกใช้โปรแกรมย่อย	103
13.1	แสดงขั้นตอนการแปลงค่าของ AL	130
13.2	แสดงลักษณะของการเลื่อนบิต	131

	หน้า
13.3 แสดงลักษณะของการเลื่อนบิต และการหมุนบิต	133
13.4 แสดงลักษณะการทำงานของคำสั่งหมุนบิต	133
13.5 แสดงลักษณะการทำงานของคำสั่งหมุนบิตที่ผ่านแฟล็กทด	134
15.1 แสดงการเก็บตำแหน่งที่อยู่ของคำสั่งถัดไป (return address) บน stack	149

สารบัญตาราง

ตารางที่		หน้า
2.1	แสดงชนิดของการอ้างอิงหน่วยความจำในเซกเมนต์รีจิสเตอร์	22
7.1	แสดงคำสั่งสำหรับการกำหนดค่าแฟล็กแฟล็กต่าง ๆ	69
7.2	แสดงผลกระทบของคำสั่งต่าง ๆ ต่อแฟล็ก	75
8.1	แสดงบริการของ DOS ที่สำคัญและพารามิเตอร์	84
8.2	แสดงตารางรหัสแอสกี	89
9.1	แสดงคำสั่งเทียมสำหรับการระบุขนาดข้อมูลในการจองหน่วยความจำ	92
10.1	แสดงคำสั่งกระโดดต่างๆ	101
13.1	แสดงค่าของการกระทำทางตรรกศาสตร์	128
13.2	แสดงผลของการใช้คำสั่งทางตรรกศาสตร์กับข้อมูล	130
13.3	แสดงตัวอย่างผลลัพธ์ของการเลื่อนบิตของข้อมูลต่าง ๆ	132
13.4	แสดงตัวอย่างผลลัพธ์ของการเลื่อนบิตแบบคิดเครื่องหมาย	132

แผนบริหารการสอนประจำวิชา

รายวิชา ไมโครโพรเซสเซอร์และภาษาแอสเซมบลี (Microprocessor and Assembly Language)	รหัสวิชา 4122705
จำนวนหน่วยกิต - ชม.	3 (2-2-5)
เวลาเรียน	45 ชั่วโมง/ภาคเรียน
คำอธิบายรายวิชา	

ศึกษาความรู้ทั่วไปเกี่ยวกับไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ โครงสร้างของไมโครโพรเซสเซอร์ การเกิดอินเทอร์รัพต์ การติดต่อสื่อสารกับหน่วยความจำ การติดต่อรับส่งข้อมูลกับอุปกรณ์ภายนอก การเขียนภาษาแอสเซมบลีเพื่อใช้ในการควบคุมไมโครคอนโทรลเลอร์และไมโครโพรเซสเซอร์

วัตถุประสงค์ทั่วไป

- 1.1 เพื่อให้ผู้เรียนมีความรู้ความเข้าใจ เกี่ยวกับไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์โครงสร้างของไมโครโพรเซสเซอร์ การเกิดอินเทอร์รัพต์ การเกิดอินเตอร์รัพต์ รีจิสเตอร์ของซีพียู 8086 การระบุตำแหน่งในรีจิสเตอร์ การเข้าถึงข้อมูลในหน่วยความจำ
- 1.2 เพื่อให้ผู้เรียนมีความรู้ความเข้าใจ สามารถที่จะใช้โปรแกรมดีบัก (Debug) ในการตรวจสอบการทำงานในระดับบิตคำสั่งและการจัดการกับบิตข้อมูลได้
- 1.3 เพื่อให้ผู้เรียนมีความรู้ความเข้าใจสามารถเขียนภาษาโปรแกรมแอสเซมบลีเบื้องต้นได้

เนื้อหา

- | | |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| บทที่ 1 ระบบคอมพิวเตอร์ | 4 คาบ/สัปดาห์ |
| อธิบายถึงองค์ประกอบของระบบคอมพิวเตอร์ ชนิดของคอมพิวเตอร์ ตลอดจนโครงสร้างของฮาร์ดแวร์และซอฟต์แวร์ที่นำมาใช้ร่วมกับระบบคอมพิวเตอร์ | |
| บทที่ 2 ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ | 4 คาบ/สัปดาห์ |
| อธิบายถึงของแตกต่างระหว่างไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ ชนิดของไมโครโพรเซสเซอร์ โครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์ การเกิดอินเทอร์รัพต์ การระบุตำแหน่งในรีจิสเตอร์และการเข้าถึงข้อมูลใน หน่วยความจำ | |
| บทที่ 3 ระบบจำนวน | 4 คาบ/สัปดาห์ |
| อธิบายความหมายของตัวเลขในหลักต่าง ๆ การแปลงค่าจากเลขฐานสิบเป็นเลขฐานสอง การบวกและการลบเลขฐานสอง การคูณและการหารเลขฐานสอง เลขฐานแปดและเลขฐานสิบหก บิต, ไบต์, เวิร์ด และดับเบิลเวิร์ด ระบบเลข 2' Complement ระบบเลข BCD (Binary Code Decimal) และมาตรฐาน IEEE 754 | |
| บทที่ 4 ภาษาสำหรับเขียนโปรแกรมคอมพิวเตอร์ | 4 คาบ/สัปดาห์ |
| อธิบายภาษาต่าง ๆ สำหรับคอมพิวเตอร์ ระดับของภาษาสำหรับเขียนโปรแกรมกาแปลภาษาสำหรับคอมพิวเตอร์และภาษาแอสเซมบลี | |
| บทที่ 5 สถาปัตยกรรมคอมพิวเตอร์เบื้องต้น | 4 คาบ/สัปดาห์ |
| อธิบายหน่วยประมวลผลกลาง หน่วยความจำ การเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ และสถาปัตยกรรมของระบบไมโครโพรเซสเซอร์ตระกูล 80x86 | |

- บทที่ 6 คำสั่งโอนย้ายข้อมูล** 4 คาบ/สัปดาห์
อธิบายคำสั่งในการโอนย้ายข้อมูล การใช้คำสั่ง MOV เครื่องมือในการทดลองการโปรแกรมภาษาแอสเซมบลี คำสั่งในการโอนย้ายข้อมูล และคำสั่งทั่วไป
- บทที่ 7 แฟล็กและคำสั่งคณิตศาสตร์** 4 คาบ/สัปดาห์
อธิบายการทำงานของแฟล็ก (Flags) คำสั่งทางคณิตศาสตร์ กลุ่มคำสั่งบวกและลบ กลุ่มคำสั่งคูณและหาร กลุ่มคำสั่งแปลงขนาดตัวเลข
- บทที่ 8 โปรแกรมภาษาแอสเซมบลีเบื้องต้น** 4 คาบ/สัปดาห์
อธิบายรูปแบบของโปรแกรมภาษาแอสเซมบลีแบบเก่า เปรียบเทียบกับรูปแบบของโปรแกรมภาษาแอสเซมบลีแบบใหม่ ขั้นตอนการแปลโปรแกรม การเรียกใช้บริการของ DOS ตารางรหัสแอสกี
- บทที่ 9 การประกาศข้อมูล** 4 คาบ/สัปดาห์
อธิบายรูปแบบและวิธีการประกาศข้อมูล การอ้างใช้ข้อมูลที่ประกาศไว้ การอ้างตำแหน่งของข้อมูล การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah การใช้บริการของ DOS หมายเลข 0Ah : การอ่านข้อความ และหมายเลข 09h : การอ่านพิมพ์ข้อความ
- บทที่ 10 คำสั่งกระโดดและคำสั่งการกระทำซ้ำ** 4 คาบ/สัปดาห์
อธิบายคำสั่งกระโดดแบบไม่มีเงื่อนไข คำสั่งกระโดดที่พิจารณาค่าจากแฟล็ก คำสั่งกระโดดที่พิจารณาค่าจากรีจิสเตอร์ มีความรู้และความเข้าใจเกี่ยวกับกลุ่มคำสั่งวนรอบ เช่น LOOP LOOPZ และ LOOPNZ เป็นต้น
- บทที่ 11 โครงสร้างควบคุม** 4 คาบ/สัปดาห์
อธิบายการสร้างโครงสร้างการตัดสินใจแบบ if-then-else การสร้าง repeat until loop การสร้าง while loop และการสร้าง for loop
- บทที่ 12 โปรแกรมย่อยเบื้องต้น** 4 คาบ/สัปดาห์
อธิบายคำสั่งที่รองรับการเรียกโปรแกรมย่อย การประกาศโปรแกรมย่อย คำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแอสตักและตัวอย่างการใช้งานโปรแกรมย่อย
- บทที่ 13 การกระทำระดับบิต** 4 คาบ/สัปดาห์
อธิบายคำสั่งทางตรรกศาสตร์ การประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์ การใช้คำสั่งเลื่อนบิต ตัวอย่างการใช้งานคำสั่งเลื่อนบิต การประยุกต์ใช้งานคำสั่งเลื่อนบิต คำสั่งหมุนบิต คำสั่งหมุนบิตที่ผ่านแฟล็กทด ตัวอย่างการใช้งานคำสั่งเกี่ยวกับการประมวลผลระดับบิต
- บทที่ 14 การอ้างแอดเดรส** 4 คาบ/สัปดาห์
อธิบายเกี่ยวกับการอ้างแอดเดรสในเซกเมนต์ทั้งสี่ คือ CS, DS, SS และ ES การประยุกต์ใช้งานคำสั่งในการอ้างแอดเดรสแบบต่าง ๆ
- บทที่ 15 การขัดจังหวะ** 4 คาบ/สัปดาห์
อธิบายกระบวนการขัดจังหวะและการใช้คำสั่งติดต่อกับอุปกรณ์ใน 8086
- บทที่ 16 การจัดการกับสายข้อมูล** 4 คาบ/สัปดาห์
อธิบายคำสั่งจัดการสายข้อมูลต่างๆ เช่น คำสั่ง MOVS, คำสั่ง REP, คำสั่ง STOS, คำสั่ง LODS, คำสั่ง REPZ, คำสั่ง CMPS, คำสั่ง REPNZ และคำสั่ง SCAS เป็นต้น

- บทที่ 17 คำสั่งตารางและการสร้างแมคโคร** 4 คาบ/สัปดาห์
อธิบายคำสั่ง XLAT ความหมายข้อดีและข้อเสียของการใช้แมคโคร พารามิเตอร์ของแมคโคร การใช้ LABEL ในแมคโครและแมคโครกับโปรแกรมย่อย
- บทที่ 18 การฝึกทักษะการเขียนโปรแกรมภาษาแอสเซมบลีเบื้องต้น** 4 คาบ/สัปดาห์
อธิบายมีวิธีการดาวน์โหลดและติดตั้งโปรแกรม MASM32 รวมถึงวิธีการดาวน์โหลดและติดตั้งโปรแกรม EditPlus การประยุกต์ใช้งานคำสั่งและฝึกทักษะการเขียนโปรแกรมภาษาแอสเซมบลีและตรวจสอบความถูกต้องของโปรแกรมที่เขียนขึ้นมาเบื้องต้นได้

วิธีสอนและกิจกรรม

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล**1. การวัดผล**

1.1 คะแนนระหว่างภาครวม		ร้อยละ 60
1.1.1 คะแนนเข้าชั้นเรียน		ร้อยละ 10
1.1.2 คะแนนจัดทำรายงานกลุ่ม		ร้อยละ 10
1.1.3 คะแนนฝึกปฏิบัติการ		ร้อยละ 10
1.1.4 คะแนนสอนกลางภาค		ร้อยละ 30
1.2 คะแนนสอนปลายภาครวม		ร้อยละ 40
1.2.1 คะแนนเข้าสอนปลายภาค		ร้อยละ 30
1.2.2 คะแนนฝึกปฏิบัติการ		ร้อยละ 10

2. การประเมินผล

ระดับคะแนน	ความหมายของผลการเรียน	ค่าระดับคะแนน	ค่าร้อยละ
A	ดีเยี่ยม	4.0	90 - 100
B+	ดีมาก	3.5	85 - 89
B	ดี	3.0	75 - 84
C+	ดีพอใช้	2.5	70 - 74
C	พอใช้	2.0	60 - 69
D+	อ่อน	1.5	55 - 59
D	อ่อนมาก	1.0	50 - 54
E	ตก	0.0	0 - 49

แผนบริหารการสอนประจำบทที่ 1

หัวข้อเนื้อหา

- องค์ประกอบของระบบคอมพิวเตอร์
- ชนิดของคอมพิวเตอร์
- โครงสร้างของฮาร์ดแวร์ของระบบคอมพิวเตอร์
- ระบบซอฟต์แวร์ของคอมพิวเตอร์

วัตถุประสงค์เชิงพฤติกรรม

- ให้ความหมายองค์ประกอบของระบบคอมพิวเตอร์ได้
- สามารถแยกชนิดของคอมพิวเตอร์ตามรูปแบบและวิธีการนำไปใช้งานได้
- อธิบายโครงสร้างของฮาร์ดแวร์ของระบบคอมพิวเตอร์
- มีความรู้และความเข้าใจเกี่ยวกับระบบซอฟต์แวร์ของคอมพิวเตอร์

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 1 ระบบคอมพิวเตอร์ (Computer System)

คอมพิวเตอร์ คือ อุปกรณ์ที่มนุษย์สร้างขึ้นเพื่ออำนวยความสะดวกและช่วยในการทำงานของมนุษย์ โดยมีการใช้งานที่แตกต่างกันออกไป พัฒนาการของคอมพิวเตอร์มีมาอย่างต่อเนื่องจนในปัจจุบันคอมพิวเตอร์เป็นที่ยอมรับ และราคาถูกลงมากเมื่อเทียบกับสมัยก่อน อีกทั้งความสามารถและประสิทธิภาพของคอมพิวเตอร์ก็เพิ่มขึ้นทั้งในด้านความเร็วในการประมวลผลข้อมูล ความสามารถในการเก็บข้อมูลและปลอดภัยเพิ่มมากขึ้นด้วย

1.1 องค์ประกอบของระบบคอมพิวเตอร์

1.1.1 ฮาร์ดแวร์ (hardware) ซึ่งเป็นอุปกรณ์ที่เราสามารถจับต้องได้ ฮาร์ดแวร์ แบ่งเป็น 5 ประเภท อุปกรณ์รับข้อมูล (input), อุปกรณ์ส่งข้อมูล (output), อุปกรณ์ประมวลผลข้อมูล (system unit), อุปกรณ์เก็บข้อมูล (storage device), และอุปกรณ์ที่ใช้ในการสื่อสารข้อมูล (communication device)

1.1.2 ซอฟต์แวร์ (software) คือ ชุดของคำสั่งที่เป็นตัวกำหนดการทำงานต่างๆ ของคอมพิวเตอร์ สามารถเรียกได้อีกอย่างหนึ่งว่า โปรแกรม เช่น window, winamp, winzip, wordprocessor, powerdvd เป็นต้น

1.1.3 ส่วนบุคคล (peopleware) คือ บุคคลที่เกี่ยวข้องกับคอมพิวเตอร์ เช่น บุคคลทั่วไป, นักเขียนโปรแกรม, นักวิเคราะห์ระบบ เป็นต้น

1.1.4 ข้อมูล (data) คือ ข้อมูลที่เก็บอยู่ในคอมพิวเตอร์เพื่อไว้ใช้งานต่อไป ซึ่งสามารถเป็นได้ ทั้ง รหัสต่างๆ ตัวอักษร ตัวเลข รูปภาพ เสียง และ วิดีโอ เป็นต้น

1.2 ชนิดของคอมพิวเตอร์ เราสามารถแบ่งคอมพิวเตอร์ตามรูปแบบการใช้งานได้เป็น 5 ชนิด ได้แก่

1.2.1 คอมพิวเตอร์ส่วนบุคคล (personal computer) เป็นคอมพิวเตอร์ที่สามารถทำงานได้ทั้ง รับข้อมูลเข้า ประมวลผล ส่งข้อมูลออก และเก็บข้อมูล ด้วยตัวเอง

1.2.2 คอมพิวเตอร์พกพา (mobile computer) และอุปกรณ์พกพา (mobile device) เป็นคอมพิวเตอร์ส่วนบุคคลที่ผู้ใช้สามารถพกพาไปไหนก็ได้ และอุปกรณ์พกพา หมายถึง อุปกรณ์เกี่ยวกับคอมพิวเตอร์ที่มีขนาดเล็กพอที่คุณสามารถถือไว้ในมือได้

1.2.3 เครื่องให้บริการขนาดกลาง (midrange servers) เป็นคอมพิวเตอร์ที่มีขนาดใหญ่ และมีประสิทธิภาพในการคำนวณสูง ซึ่งสามารถรองรับการทำงานของเครื่องคอมพิวเตอร์ที่เชื่อมโยงกันบนเน็ตเวิร์ค (network) มากกว่าพันเครื่องในเวลาเดียวกัน

1.2.4 เมนเฟรม (mainframe) เป็นคอมพิวเตอร์ที่มีขนาดใหญ่ ราคาแพง และมีประสิทธิภาพสูง ที่สามารถรองรับการทำงานของเครื่องคอมพิวเตอร์บนเครือข่ายมากกว่าพันเครื่อง ในเวลาเดียวกัน และสามารถเก็บข้อมูลคำสั่งต่างๆ ได้มหาศาล

1.2.5 ซุปเปอร์คอมพิวเตอร์ (supercomputer) เป็นคอมพิวเตอร์ที่มีการทำงานที่เร็วที่สุดในบรรดาประเภทของคอมพิวเตอร์ที่กล่าวมา มีประสิทธิภาพสูงสุด และราคาแพงที่สุด คอมพิวเตอร์ประเภทนี้ใช้สำหรับการทำงานที่มีการคำนวณที่ซับซ้อนมากๆ

ชนิดของคอมพิวเตอร์ สามารถแบ่งได้ตามขนาดและการใช้งานของคอมพิวเตอร์ได้ดังนี้

- ซุปเปอร์คอมพิวเตอร์ (super computer)



ซุปเปอร์คอมพิวเตอร์เป็นคอมพิวเตอร์ที่มีขนาดใหญ่ มีสมรรถนะสูง สามารถประมวลได้เร็ว และมีความสามารถในการเก็บข้อมูลขนาดใหญ่ เช่น สถิติประชากร การขุดเจาะน้ำมัน คอมพิวเตอร์ชนิดนี้มีราคาแพงที่สุด ส่วนใหญ่จะใช้งานในองค์กรที่มีการทำงานที่ต้องการความเร็วสูง เช่น งานวิเคราะห์ภาพถ่ายจากดาวเทียม อุตุนิยมวิทยา หรือดาวเทียมสำรวจทรัพยากร งานวิเคราะห์พยากรณ์อากาศ งานทำแบบจำลองโมเลกุล ของสารเคมี งานวิเคราะห์โครงสร้างอาคาร ที่ซับซ้อน ปัจจุบันประเทศไทย มีเครื่องซุปเปอร์คอมพิวเตอร์ Cray YMP ใช้ในงานวิจัย อยู่ที่ห้องปฏิบัติการคอมพิวเตอร์สมรรถภาพสูง (HPCC) ศูนย์เทคโนโลยี อิเล็กทรอนิกส์ และคอมพิวเตอร์แห่งชาติ ผู้ใช้เป็นนักวิจัยด้านวิศวกรรม และวิทยาศาสตร์ทั่วประเทศ บริษัทผู้ผลิตที่เด่นๆ ได้แก่ บริษัทแคร์ย์ รีเสิร์ช (Cray Research), บริษัท เอ็นอีซี (NEC) เป็นต้น

- เมนเฟรม (mainframe)



เมนเฟรมเป็นคอมพิวเตอร์ขนาดใหญ่ แต่เล็กกว่า และมีสมรรถนะต่ำกว่าซุปเปอร์คอมพิวเตอร์ มีราคาแพง นิยมใช้งานกับธุรกิจขนาดใหญ่ เช่น ธนาคาร โรงแรม หรือ ใช้เป็นเซิร์ฟเวอร์ขององค์กรขนาดใหญ่ เป็นต้นได้ชื่อว่าเมนเฟรมคอมพิวเตอร์ ก็เพราะครั้งแรกที่สร้างคอมพิวเตอร์ลักษณะนี้ได้สร้างไว้บนฐานรองรับ ที่เรียกว่า คัสซี (Chassis) โดยมีชื่อเรียกฐานรองรับนี้ว่า เมนเฟรม นั่นเอง คอมพิวเตอร์เมนเฟรม ที่มีชื่อเสียงมาก คือ เครื่องของบริษัท IBM

- มินิคอมพิวเตอร์ (minicomputer)



DEC PDP 8 ปี 1965

มินิคอมพิวเตอร์เป็นคอมพิวเตอร์ที่มีสมรรถนะต่ำกว่ารองลงมาจากเมนเฟรม คือทำงานได้ช้ากว่า แต่ราคา ย่อมเยาว์กว่าเมนเฟรม ใช้ในธุรกิจขนาดกลางและเล็กที่ต้องการความสามารถในการประมวลผลสูงและราคาไม่สูง เกินไป เช่น ตามองค์กร และสถานศึกษาระดับอุดมศึกษา ต่างๆ เป็นต้น

- ไมโครคอมพิวเตอร์ (microcomputer) หรือ คอมพิวเตอร์ส่วนบุคคล (personal computer)



คอมพิวเตอร์ส่วนบุคคล หรือ คอมพิวเตอร์แบบตั้งโต๊ะ (Desktop computer) หาซื้อได้ง่าย ราคาไม่แพง มีขนาด เล็กกว่ามินิคอมพิวเตอร์ บุคคลทั่วไปสามารถซื้อไว้ใช้งาน หรือ เพื่อความบันเทิง ได้ เหมาะกับการใช้งานที่ไม่ จำเป็นต้องใช้ความเร็วสูงมาก แต่ในปัจจุบันความสามารถในการทำงานของคอมพิวเตอร์ส่วนบุคคลได้พัฒนา สูงขึ้นมาก และราคาไม่แพง ทำให้เป็นที่นิยมในปัจจุบัน อีกทั้งยังได้รับการพัฒนาไปอย่างรวดเร็วมากบางเครื่องมี ความสามารถมากกว่าเครื่องเมนเฟรมในสมัยแรกๆ เสียอีกด้วยราคาที่ถูกลงกว่าหลายร้อยเท่าทีเดียว และยังได้มีการ พัฒนาคอมพิวเตอร์ส่วนบุคคลในรูปแบบที่พกพาสะดวกได้แก่

- โน้ตบุค (Notebook Computer, Laptop)



เป็นคอมพิวเตอร์ส่วนบุคคลขนาดเล็กประมาณสมุดโน้ต โดยทั่วไปมีราคาสูงกว่าและประหยัดไฟมากกว่า

คอมพิวเตอร์ตั้งโต๊ะ มีแบตเตอรี่ในตัว สามารถพกพาไปที่ใดก็ได้ และเปิดใช้ได้ไม่จำเป็นต้องมีแหล่งจ่ายไฟ ส่วนใหญ่สามารถเปิดใช้ได้ประมาณ 4 ชั่วโมง ปัจจุบันได้พัฒนาให้มีขนาดบางและน้ำหนักเบา อีกทั้งยังมีความสามารถเทียบเท่ากับคอมพิวเตอร์ตั้งโต๊ะอีกด้วย

- เทปเล็ต (tablet PC)



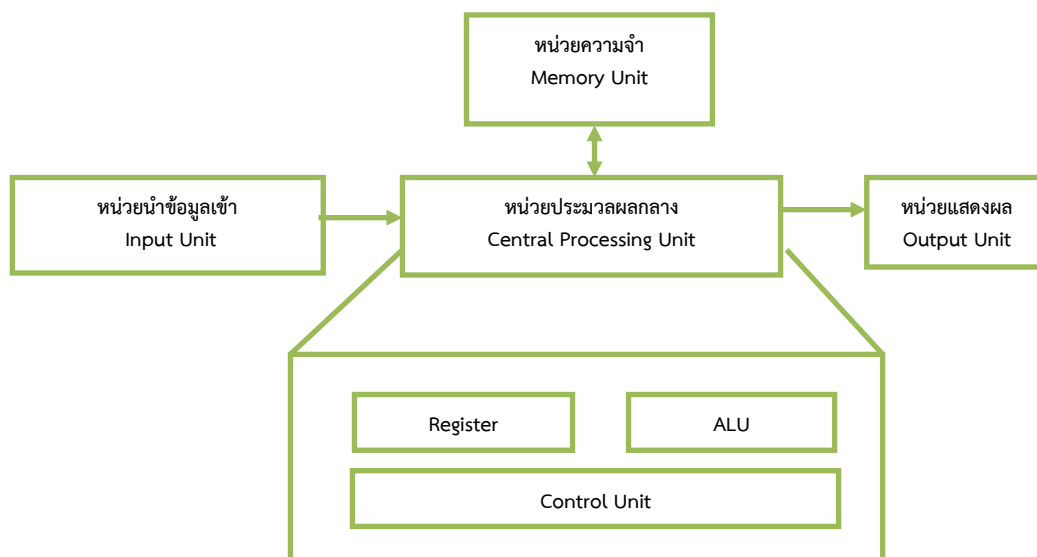
มีลักษณะคล้ายกระดานเขียนตัวหนังสือ สามารถใช้งานได้เหมือนสมุดจดบันทึกหรือสมุดโน้ต โดยคุณสามารถวาดหรือเขียนตัวหนังสือลงไปบนหน้าจอได้เลย ไม่จำเป็นต้องใช้คีย์บอร์ดเหมาะสำหรับคนที่ชอบเขียนมากกว่าชอบพิมพ์ในปัจจุบันเทปเล็ตสามารถบันทึกเสียงได้ด้วย

- พีดีเอ (PDA: Personal Digital Assistant)



เป็นคอมพิวเตอร์ขนาดเล็กเท่าฝ่ามือ ความสามารถในการประมวลน้อยกว่าคอมพิวเตอร์ส่วนบุคคลทั่วไป แต่ยังสามารถดูหนัง ฟังเพลง เล่นอินเทอร์เน็ตได้ มีปฏิทิน และสมุดนัดหมาย บางรุ่นสามารถเป็นโทรศัพท์เคลื่อนที่ได้ด้วย มีอุปกรณ์รับเข้า คือ สไตลัส (stylus) ซึ่งมีลักษณะคล้ายปากกา เวลาใช้จะอาศัยแรงกดลงไปบนหน้าจอ พีดีเอ บางรุ่นสามารถสั่งงานด้วยเสียงได้

1.3 โครงสร้างของฮาร์ดแวร์ของระบบคอมพิวเตอร์โดยทั่วไปแล้วคอมพิวเตอร์แต่ละรุ่นแต่ละยี่ห้อ ก็จะมีการวางส่วนประกอบต่าง ๆ ไม่เหมือนกัน แต่ถ้าเรามองโครงสร้างของคอมพิวเตอร์ในรูปแบบ Module แล้วจะเห็นส่วนประกอบต่าง ๆ ที่มีรูปแบบเหมือนกันดังนี้



รูปที่ 1.1 แสดงโครงสร้างของฮาร์ดแวร์ของระบบคอมพิวเตอร์

1.3.1 หน่วยประมวลผลกลาง (Central Processing Unit : CPU)

หน่วยประมวลผลกลางจัดได้ว่าเป็นส่วนที่สำคัญที่สุดของคอมพิวเตอร์ เปรียบเสมือนเป็นสมองของเครื่องคอมพิวเตอร์ โดยทำหน้าที่ในการคำนวณค่าต่าง ๆ ตามคำสั่งที่ได้รับ และควบคุมการทำงานของส่วนประกอบอื่น ๆ ทั้งหมด ในระบบไมโครคอมพิวเตอร์ หน่วยประมวลผลกลางจะถูกสร้างให้อยู่ในรูป วงจรรวม (Integrated Circuit: IC) เพียงตัวเดียวทำให้ง่ายในการนำไปใช้งาน ภายในหน่วยประมวลผลกลางมีส่วนประกอบย่อย ๆ สามส่วนคือ

- หน่วยคำนวณทางคณิตศาสตร์และตรรกะ (Arithmetic and Logic Unit : ALU) เป็นหน่วยที่ทำหน้าที่ประมวลผลโดยใช้วิธีที่คณิตศาสตร์ เช่น บวก ลบ คูณ หาร หรือ ทำหน้าที่ประมวลผลทางตรรกะ เช่น AND OR NOT COMPLEMENT เป็นต้น รวมทั้งยังทำหน้าที่ในการเปรียบเทียบค่าต่าง ๆ อีกด้วย
- หน่วยเก็บข้อมูลชั่วคราว (Register) เป็นหน่วยความจำขนาดเล็ก ทำหน้าที่เป็นที่พักข้อมูลชั่วคราวก่อนที่จะถูกนำไปประมวลผล โดยปกติแล้วในหน่วยประมวลผลกลางจะมี Register สำหรับเก็บข้อมูลไม่เกิน 64 ตัว การอ้างอิงข้อมูลของ Register จะมีความเร็วเท่ากับความเร็วของหน่วยประมวลผลกลาง เพราะเป็นหน่วยความจำส่วนที่อยู่ภายในตัวหน่วยประมวลผลกลางจึงไม่ต้องไปอ้างอิงถึงภายนอกหน่วยประมวลผล
- หน่วยควบคุม (Control Unit) เป็นเสมือนหน่วยบัญชาการของระบบคอมพิวเตอร์ทั้งหมด ทำหน้าที่กำหนดจังหวะการทำงานต่าง ๆ ของคอมพิวเตอร์ไม่เว้นแม้แต่ส่วนประกอบอื่น ๆ ของ CPU นอกจากนี้ยังทำหน้าที่ควบคุมการส่งข้อมูลระหว่างหน่วยต่าง ๆ ในคอมพิวเตอร์

1.3.2 หน่วยความจำ (Memory Unit)

เป็นหน่วยที่ทำหน้าที่เก็บข้อมูลของระบบคอมพิวเตอร์ ไม่ว่าจะเป็นตัวเลขหรือข้อความแม้กระทั่งคำสั่งต่างๆ ในโปรแกรมที่จะใช้สั่งงานระบบคอมพิวเตอร์ โดยทั่วไปแล้วหน่วยความจำจะถูกสร้างมาจาก IC เพื่อให้มีความจุสูงแต่มีขนาดเล็ก ข้อมูลที่เก็บในหน่วยความจำจะมีสถานะเพียงแค่เปิดวงจร (0) หรือปิดวงจร (1) เท่านั้น หน่วยความจำสามารถแบ่งออกได้เป็นสองประเภทใหญ่ ๆ คือ

- ROM (Read Only Memory) เป็นหน่วยความจำส่วนที่ CPU สามารถอ่านข้อมูลออกมาใช้งานได้ ตามกรรมวิธีปกติ แต่เมื่อต้องการจะเขียนข้อมูลลงไปจะต้องใช้วิธีพิเศษ ทำให้ต้องมีวงจรในการเขียนข้อมูลโดยเฉพาะ ข้อดีของหน่วยความจำแบบนี้ก็คือ ข้อมูลที่เขียนลงไปแล้วจะคงอยู่ไปตลอดแม้ว่าจะปิดเครื่องคอมพิวเตอร์ไปแล้วก็ตาม ดังนั้น ROM จึงมักจะถูกใช้เก็บโปรแกรมสำหรับเริ่มต้นการทำงานของเครื่องคอมพิวเตอร์
- RAM (Random Access Memory) เป็นหน่วยความจำที่ CPU สามารถอ่านเขียนข้อมูลได้ด้วยกรรมวิธีปกติของระบบ ทำให้สามารถเปลี่ยนแปลงข้อมูลได้ตลอดเวลา RAM จึงมักจะถูกนำไปใช้เก็บข้อมูลระหว่างการทำงานของระบบ แต่ RAM ก็มีข้อเสียคือข้อมูลที่เก็บไว้ทั้งหมดจะสูญหายไปทันทีที่หยุดจ่ายไฟให้กับหน่วยความจำ

1.3.3 หน่วยนำข้อมูลเข้าและหน่วยแสดงผล (I/O Unit)

เป็นหน่วยที่เปรียบเสมือนประสาทของเครื่องคอมพิวเตอร์ ทำหน้าที่รับการติดต่อจากภายนอกเข้าสู่ระบบ และแสดงผลที่ได้จากการทำงานของระบบออกสู่ภายนอก เช่น คีย์บอร์ด, จอภาพ, ลำโพง, Disk Drive เป็นต้น โดยปกติแล้วเรามักจะแบ่งความหรรษาของเครื่องคอมพิวเตอร์โดยดูจากหน่วยนำข้อมูลเข้าและหน่วยแสดงผลเป็นส่วนใหญ่ เช่น ขนาดของจอภาพ, ความจุของ Disk เป็นต้น

1.4 ระบบซอฟต์แวร์ของคอมพิวเตอร์

เมื่อคอมพิวเตอร์ไม่มีซอฟต์แวร์ก็ไม่ต่างไปจากเครื่องประดับราคาสูงชิ้นหนึ่งเท่านั้น โดยปกติแล้วราคาของระบบคอมพิวเตอร์ส่วนใหญ่จะเป็นราคาของซอฟต์แวร์ และเป็นส่วนที่จะกำหนดอนาคตของระบบคอมพิวเตอร์ว่าจะอยู่ในตลาดได้นานเท่าใด

ความหมายของซอฟต์แวร์มีได้หลายแบบตามแต่จะใช้คำพูดใด ซึ่งความหมายโดยรวมก็คือ “ซอฟต์แวร์หมายถึง กลุ่มของคำสั่งที่กำหนดการทำงานของคอมพิวเตอร์เพื่อให้บรรลุจุดประสงค์ที่ต้องการ” การพัฒนาระบบซอฟต์แวร์ให้กับคอมพิวเตอร์เครื่องหนึ่ง มักจะมีการกำหนดประเภทของซอฟต์แวร์ออกเป็น 3 ประเภทคือ

1.4.1 ซอฟต์แวร์ระบบ (System Software)

เป็นซอฟต์แวร์ส่วนที่ทำหน้าที่ควบคุมการทำงานของฮาร์ดแวร์โดยตรงเพื่อทำให้ระบบคอมพิวเตอร์โดยรวมมีประสิทธิภาพสูงที่สุด และยังทำหน้าที่ให้บริการการสั่งงานฮาร์ดแวร์ของซอฟต์แวร์ประเภทอื่น ๆ เพื่ออำนวยความสะดวกในเรื่องของการรวมรายละเอียดการสั่งงานที่ซับซ้อนให้เหลือเพียงการสั่งงานง่าย ๆ เช่น การอ่านไฟล์จากดิสก์ก็สามารถสั่งงานด้วยคำสั่งง่าย ๆ เพียงคำสั่งเดียวให้ซอฟต์แวร์ระบบสั่งงานฮาร์ดแวร์ส่วนที่ทำหน้าที่จัดการกับดิสก์ในเรื่องรายละเอียดต่อไป

1.4.2 System Library

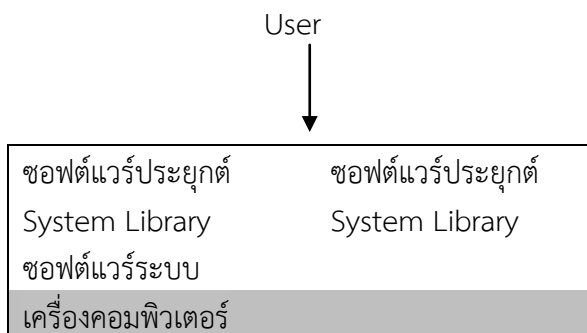
เป็นส่วนที่รวบรวมการคำสั่งการทำงานที่สลับซับซ้อนขึ้นมาจากซอฟต์แวร์ระบบ ทำหน้าที่ให้บริการการทำงานกับซอฟต์แวร์ประยุกต์ต่อไป เช่น ในซอฟต์แวร์ระบบอาจจะมีคำสั่งแค่อ่านเขียนไฟล์ แต่ใน System Library อาจจะมีคำสั่งที่จัดการกับไฟล์หลาย ๆ ไฟล์ในรูปแบบของฐานข้อมูล เป็นต้น

1.4.3 ซอฟต์แวร์ประยุกต์ (Application Software)

เป็นซอฟต์แวร์ที่พัฒนาขึ้นเพื่อใช้กับงานด้านใดด้านหนึ่งโดยเฉพาะ ซอฟต์แวร์ประยุกต์อาจจะเป็นซอฟต์แวร์ที่พัฒนาขึ้นมาเองหรือซื้อมาในรูปแบบโปรแกรมสำเร็จรูปก็ได้ ปัจจุบันโปรแกรมสำเร็จรูปของระบบไมโครคอมพิวเตอร์มีอยู่มากมายในหลายด้าน เช่น Microsoft Office เป็นต้น

ความสัมพันธ์กันของส่วนประกอบต่าง ๆ

ความสัมพันธ์กันระหว่างฮาร์ดแวร์และซอฟต์แวร์ประเภทต่าง ๆ สามารถที่จะแสดงอยู่ในรูปของแผนภูมิ หัวหอมได้ โดยฮาร์ดแวร์จะอยู่ชั้นในสุด ถัดมาจะเป็นส่วนของซอฟต์แวร์ระบบ System Library และซอฟต์แวร์ประยุกต์ตามลำดับ โดยระบบในชั้นนอกจะเรียกใช้การบริการของระบบในชั้นในกว่าลงไปและคอยให้บริการแก่ระบบในชั้นที่อยู่ถัดออกมาจากตัวเอง



รูปที่ 1.2 แสดงแผนภูมิหัวหอม

สรุป

ระบบคอมพิวเตอร์ ประกอบไปด้วยส่วนสำคัญหลายส่วน เช่น ฮาร์ดแวร์ (hardware) ซอฟต์แวร์ (software) ส่วนบุคคล (peopleware) และข้อมูล (data) เป็นต้น ซึ่งแต่ละส่วนมีความสัมพันธ์กันและสามารถทำงานโดยประสานงานกันเพื่อทำให้ระบบคอมพิวเตอร์ทำงานได้อย่างมีประสิทธิภาพสูงสุด ชนิดของคอมพิวเตอร์ นอกจากนี้เรายังสามารถแบ่งคอมพิวเตอร์ตามรูปแบบการใช้งานได้เป็น 5 ชนิด ได้แก่คอมพิวเตอร์ส่วนบุคคล (personal computer) คอมพิวเตอร์พกพา (mobile computer) และอุปกรณ์พกพา (mobile device) เครื่องให้บริการขนาดกลาง (midrange servers) เมนเฟรม (mainframe) และซูเปอร์คอมพิวเตอร์ (supercomputer)

คำถามทบทวน

1. จงอธิบายส่วนประกอบของระบบคอมพิวเตอร์ว่าควรมีส่วนประกอบอย่างน้อยที่สุดอะไรบ้าง
2. ชนิดของคอมพิวเตอร์ สามารถแบ่งได้ตามขนาดและการใช้งานของคอมพิวเตอร์ได้เป็นอย่างไร
3. ถ้าเรามองโครงสร้างของคอมพิวเตอร์ในรูปแบบ Module แล้วจะเห็นส่วนประกอบต่าง ๆ ที่มีรูปแบบเหมือนกันอย่างไร จงอธิบายพร้อมยกตัวอย่างประกอบ
4. การกำหนดประเภทของซอฟต์แวร์ แบ่งเป็นกี่ประเภท อะไรบ้าง

แผนบริหารการสอนประจำบทที่ 2

หัวข้อเนื้อหา

- ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์
- ชนิดของไมโครโพรเซสเซอร์
- โครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์
- การเกิดอินเตอร์รัพต์
- รีจิสเตอร์ของซีพียู 8086
- การระบุตำแหน่งในรีจิสเตอร์
- การเข้าถึงข้อมูลในหน่วยความจำ

วัตถุประสงค์เชิงพฤติกรรม

- เข้าใจความหมายและข้อแตกต่างระหว่างไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์
- ชนิดของไมโครโพรเซสเซอร์และโครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์
- อธิบายและทำความเข้าใจการเกิดอินเตอร์รัพต์ รีจิสเตอร์ของซีพียู 8086 การระบุตำแหน่งในรีจิสเตอร์ และการเข้าถึงข้อมูลในหน่วยความจำ

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

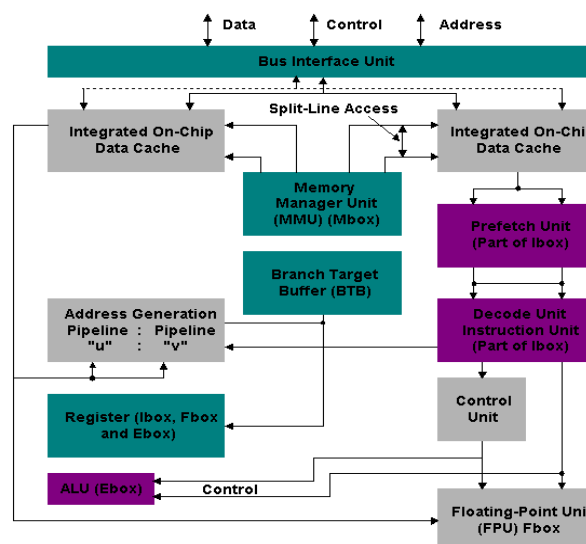
ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 2 ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ (Microprocessor and Microcontroller)

คอมพิวเตอร์มีการพัฒนาอย่างต่อเนื่องเริ่มตั้งแต่ยุคแรกที่มีการใช้หลอดสุญญากาศแทนวงจรรวมในการคำนวณต่อมาในปี ค.ศ. 1954 ได้มีการคิดค้นการใช้ทรานซิสเตอร์แทนหลอดสุญญากาศ โดยบริษัท Texas Instrument และในช่วงทศวรรษที่ 1950 นี้เองก็ได้มีการคิดค้นวงจรรวม (Integrated Circuit) ขึ้นมาใช้ในกับเครื่องคอมพิวเตอร์ จึงทำให้เครื่องคอมพิวเตอร์มีขนาดเล็กลงและใช้พลังงานไฟฟ้าน้อยลง มีความเร็วเพิ่มมากขึ้น และมีการพัฒนางจรรวมขนาดใหญ่ที่เรียกว่า Very Large Scale Integrated Circuit ทำให้สามารถรวมการประมวลผลทั้งหมดไว้ในชิปตัวเดียว เช่น ไมโครโพรเซสเซอร์ตระกูลเพนเทียม (Pentium) ของบริษัทอินเทล (Intel Corporation) และสามารถรวมไมโครโพรเซสเซอร์ หลายๆ ตัวให้สามารถทำงานรวมกันได้ในเวลาเดียวกัน นอกจากนี้จำนวนข้อมูลที่ใช้ในการประมวลผลก็พัฒนาขึ้นอย่างต่อเนื่องจาก 8 บิต มาเป็น 32 บิต และ 64 บิต ตามลำดับ

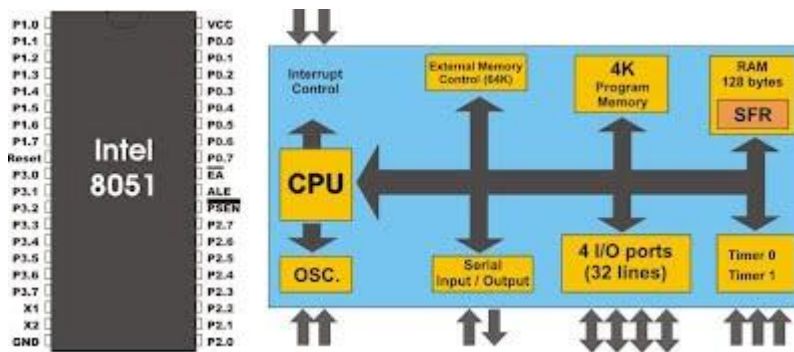
2.1 ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์

ไมโครโพรเซสเซอร์ (Microprocessor) เป็นอุปกรณ์อิเล็กทรอนิกส์ชนิดหนึ่งซึ่งมีลักษณะวงจรรวม (Integrated Circuit) หรือชิป (Chip) โครงสร้างภายในจะมีวงจรรวมขนาดใหญ่ประกอบไปด้วย หน่วยคำนวณทางคณิตศาสตร์ (Arithmetic Unit) รีจิสเตอร์ (Register) บัสข้อมูล (Data Bus) บัสควบคุม (Control Bus) บัสที่อยู่ (Address Bus) รวมกันเป็นหน่วยประมวลผลกลาง (Central Processing Unit) เพื่อทำหน้าที่ในการประมวลผลตามโปรแกรมคำสั่งที่ป้อนเข้ามา แสดงได้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงบล็อกไดอะแกรมของไมโครโพรเซสเซอร์
(Block Diagram of Microprocessor)

ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นชิปประมวลผลประเภทหนึ่งซึ่งทำหน้าที่ตามโปรแกรมหรือชุดคำสั่งที่เขียนขึ้นมา ภายในประกอบด้วยวงจรรวมขนาดใหญ่ (Very Large Scale Integrated Circuit) ประกอบไปด้วย หน่วยคำนวณทางคณิตศาสตร์ (Arithmetic Unit) รีจิสเตอร์ (Register) บัสข้อมูล (Data Bus) บัสควบคุม (Control Bus) บัสที่อยู่ (Address Bus) รวมกันเป็นหน่วยประมวลผลกลาง (Central Processing Unit) หน่วยความจำ (Memory) วงจรนับ (Counter Circuit) วงจรจับเวลา (Timing Circuit) หรือวงจรอื่นๆ รวมอยู่ในชิปไมโครคอนโทรลเลอร์ แสดงได้ดังรูปที่ 2.2



รูปที่ 2.2 แสดงไมโครคอนโทรลเลอร์ (Microcontroller)

2.2 ชนิดของไมโครโพรเซสเซอร์ จัดแบ่งตามลักษณะการใช้งานได้ 3 ประเภท

2.2.1 Dedicated or Embedded Controller

เป็นไมโครโพรเซสเซอร์ที่ใช้ในอุปกรณ์อิเล็กทรอนิกส์โดยเฉพาะที่ เช่น ใช้ในการควบคุมการทำงานของเครื่องซักผ้า เต้าไมโครเวฟ เครื่องคิดเลข และอื่นๆ โดยมากมักจะเรียกไมโครโพรเซสเซอร์ ประเภทนี้ว่า “ไมโครคอนโทรลเลอร์ (Micro Controller)” เช่น TMS-1000 ของบริษัท Texas Instrument สำหรับบริษัท Intel ได้เริ่มพัฒนาจาก ชิพขนาด 4 bits และในปี 1976 บริษัท Intel ได้เปิดตัว 8048 ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาด 8 bits ที่มี RAM, ROM และ I/O port ภายในชิปตัวเดียวกัน ในปัจจุบันมี 8096 ซึ่งมีทั้ง 16-bit CPU, RAM, ROM, UART, Port, Timer และ analog-to-digital converter ขนาด 16-bit ภายในชิปตัวเดียว

2.2.2 Bit-slice Processor

เป็นไมโครโพรเซสเซอร์ที่ได้รับการพัฒนาให้สามารถนำฟังก์ชันบางอย่างที่ไม่มีในไมโครคอนโทรลเลอร์ เช่น ฟังก์ชันการทำงานแบบ Multiplexer และ Sequencer เข้ามาทำงานบนชิปประมวลผลได้ และสามารถควบคุมการทำงานด้วยโปรแกรม โดยโปรแกรมสำหรับฟังก์ชันการทำงานต่างๆ เหล่านั้น จะเรียกว่า “Micro Code”

2.2.3 General-purpose Processor

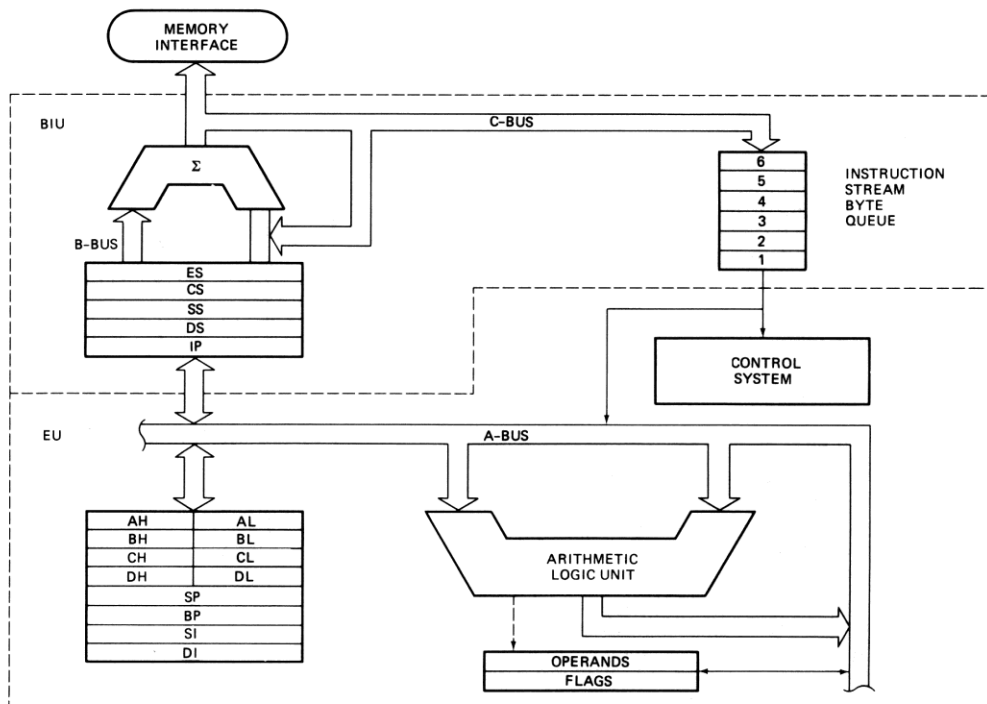
เป็นไมโครโพรเซสเซอร์ที่ใช้ในอุปกรณ์อิเล็กทรอนิกส์โดยทั่วไป ซึ่งควบคุมการทำงานด้วยโปรแกรม แบ่งตามรุ่นที่พัฒนาโดยบริษัทอินเทล ((Intel Corporation) ผู้ผลิต โดยมีการพัฒนามาตั้งแต่รุ่น 4040 ซึ่งเป็นไมโครโพรเซสเซอร์ขนาด 4 บิต และพัฒนาเป็นรุ่น Pentium จำแนกได้ดังนี้

- **8086** มี data bus ขนาด 16 bits ดังนั้นจึงสามารถ อ่าน/เขียนข้อมูลได้ครั้งละ 16 bit 8086 มี data bus ขนาด 16 bits ดังนั้นจึงสามารถ อ่าน/เขียนข้อมูลได้ครั้งละ 16 bits หรือ 8 bits
- **8086** มี address bus ขนาด 20 bits ดังนั้นจึงสามารถระบุตำแหน่งในหน่วยความจำได้สูงสุด 220 หรือ 1,048,576 ตำแหน่ง โดยแอดเดรสแต่ละตำแหน่งจะจัดเก็บข้อมูลขนาด 8 bits ข้อมูลขนาด 16 bit word จะถูกจัดเก็บไว้ในหน่วยความจำ ตำแหน่งที่แอดเดรสอยู่ต่อกัน ดังนั้น ถ้าไบต์แรกของ word ถูกจัดเก็บไว้ใน memory address เลขคู่ 8086 จะสามารถอ่านได้ภายในการปฏิบัติเพียงครั้งเดียว แต่ ถ้าไบต์แรกของ word ถูกจัดเก็บไว้ใน memory address เลขคี่ 8086 จะอ่านข้อมูลนั้นโดยใช้ Bus Operation 2 ครั้ง (อ่านทีละ byte)
- **8088** เป็นไมโครโพรเซสเซอร์ขนาด 16 bit (ALU, รีจิสเตอร์ภายใน และชุดคำสั่ง (Instruction) จะมีขนาด 16 bit word เช่นเดียวกับ 8086)
 - 8088 มี data bus ขนาด 8 bits ดังนั้นจึงสามารถอ่าน หรือเขียนข้อมูลได้ครั้งละ 8 bits เท่านั้น ในการอ่าน/เขียนข้อมูลหรือชุดคำสั่งขนาด 16 bits จะต้องใช้ Bus Operation 2 ครั้ง
 - 8088 มี address bus ขนาด 20 bits ดังนั้นจึงสามารถ address ตำแหน่งในหน่วยความจำได้สูงสุด 220 หรือ 1,048,576 ตำแหน่ง
- **80186 และ 80188** เป็นไมโครโพรเซสเซอร์ที่ปรับปรุงมาจาก 8086 และ 8088 นอกจากทั้งคู่จะเป็น CPU ขนาด 16 bit แล้ว ยังมี programmable peripheral devices บูรณาการรวมอยู่ใน package เดียวกัน ชุดคำสั่งต่างๆ ใน 80186 และ 80188 เป็น superset ของชุดคำสั่งของ 8086 และ 8088 ซึ่งหมายความว่าชุดคำสั่งเดิมใน 8086 และ 8088 ยังคงสามารถทำงานได้ใน 80186 และ 80188 โดยมีชุดคำสั่งเพิ่มเติมไม่มากนัก ดังนั้นโปรแกรมที่เขียนให้สามารถทำงานได้ใน 8086 หรือ 8088 จะยังคงสามารถทำงานได้ใน 80186 และ 80188 (เรียกว่า upward compatible) แต่โปรแกรมที่เขียนให้ทำงานบน 80186 และ 80188 อาจจะไม่สามารถทำงานได้ใน 8086 และ 8088
- **80286** เป็นไมโครโพรเซสเซอร์ ที่ปรับปรุงมาจาก 8086 (มีการปรับปรุงในระดับสูง) ที่ได้รับการออกแบบให้ใช้เป็น CPU ในเครื่องไมโครคอมพิวเตอร์ที่มีความสามารถในการทำงานแบบ Multi-user/ Multitasking การทำงานของ 80286 ใน real address mode จะมีฟังก์ชันการทำงานเหมือน 8086 ความเร็วสูงการทำงานใน virtual address mode จะมีคุณสมบัติในการแยกโปรแกรมประยุกต์ของผู้ใช้ออกจากระบบปฏิบัติการ เพื่อไม่ให้ส่วนของโปรแกรมไปทำลายโปรแกรมระบบได้
- **80386 และ 80486** เป็นไมโครโพรเซสเซอร์ ขนาด 32 bit ซึ่งสามารถระบุตำแหน่งของแอดเดรสในหน่วยความจำได้สูงสุดถึง 4GB และมีการนำเทคโนโลยี RISC (Reduced Instruction Set Computer) มาใช้ ทำให้มีคุณสมบัติที่ซับซ้อนเพิ่มมากขึ้นในเรื่องของการทำงานแบบ Multi-user/ Multitasking

2.3 โครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์

ไมโครโพรเซสเซอร์ตระกูลของ Intel จะมีโครงสร้างสถาปัตยกรรมในลักษณะเดียวกัน สำหรับการเริ่มต้นเราจะศึกษาโครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์จาก 8086 เพื่อให้เข้าใจได้โดยง่าย และไม่ซับซ้อนมากนักและก่อนที่จะเข้าไปถึงการเขียนโปรแกรมสำหรับทำงานบน 8086 จะต้องทราบโครงสร้างการทำงาน

ภายใน ไม่ว่าจะเป็น ALU, Flag, Register, Instruction Byte Queue และ Segment Register 38 CPU 8086 มีแอดเดรส 20 เส้น (Address Bus 20 bits) ทำให้สามารถอ้างถึงแอดเดรสได้ 1,045,576 ไบต์ (หรือ 1 MB - เมกะไบต์) หรือมากเป็น 16 เท่า ของไมโครโปรเซสเซอร์ขนาด 8 บิต และมี data bus ขนาด 16 บิต จึงทำให้ CPU 8086 สามารถเรียกข้อมูลได้ที่ละ 1 เวิร์ด หรือ 2 ไบต์ และยังเตรียมพร้อมสำหรับการทำงานแบบหลาย CPU รวมทั้งการทำงานร่วมกับอุปกรณ์อื่นได้ด้วย แสดงได้ดังรูปที่ 2.3



รูปที่ 2.3 แสดง Block Diagram ของ ซีพียู (CPU 8086)

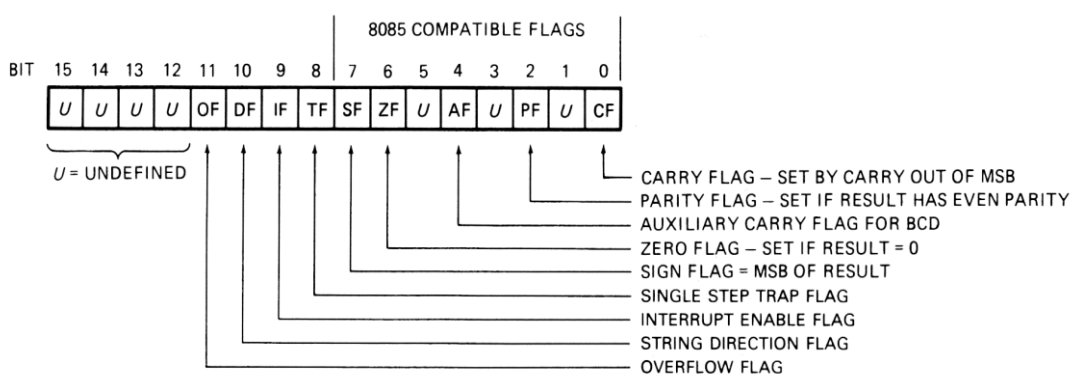
8086 แบ่งส่วนตามฟังก์ชันการทำงานที่เป็นอิสระต่อกันออกเป็น 2 ส่วน คือ BIU (Bus Interface Unit) และ EU (Execution Unit) เพราะจะทำให้ความเร็วในการทำงานสูงขึ้น

- BIU จะทำหน้าที่ส่งค่าแอดเดรสเพื่อไป fetch คำสั่งจากหน่วยความจำ, อ่านข้อมูลจากหน่วยความจำ และพอร์ต, และเขียนข้อมูลไปยังหน่วยความจำ และพอร์ต
- EU ของ 8086 จะบอก location ที่จะไป fetch instruction, decode instruction และ execute instruction

2.3.1 EU (Execution Unit) ประกอบด้วย

- **Control Circuitry** ทำหน้าที่ควบคุม Operation ภายในทั้งหมด
- **Instruction Decoder** ทำหน้าที่แปลความหมายของชุดคำสั่งที่นำมาจาก Memory ให้เป็นลำดับของภาษาเครื่องที่ EU สามารถปฏิบัติได้

- ALU มีขนาด 16 bits ซึ่งมีฟังก์ชันในการ Add (บวก), Subtraction (ลบ), AND, OR, XOR, increment, decrement, complement และ shift
- **Flag Register** เป็น flip-flop ที่แสดงสถานะของการ execute instruction หรือการควบคุมการทำงานของชุดคำสั่ง Flag Register ขนาด 16 บิต ใน EU จะบรรจุ active flag 9 ตัว โดยเป็น flag แสดงสถานะ (Condition flags) จำนวน 6 บิต และ flag ควบคุม (Control flags) จำนวน 3 บิต สำหรับบิตส่วนที่เหลือก็ไม่ได้ใช้ประโยชน์ใดๆ แสดงได้ดังรูปที่ 2.4



รูปที่ 2.4 แสดงสถานะแฟล็ก (Flag Status) ที่ใช้ในการตรวจสอบการทำงานของซีพียู (CPU 8086)

แฟล็กแสดงสถานะ (Conditional Flag) มีทั้งหมด 6 ตัว จะถูกใช้ในการแสดงเงื่อนไขบางอย่างที่สร้างโดยชุดคำสั่งในการ SET หรือ RESET ค่า Flag เหล่านี้ จะดำเนินการโดย EU บนพื้นฐานของผลลัพธ์ของการคำนวณ เช่น carry out flag จะถูก set ให้มีค่าเป็น 1 หากผลลัพธ์ของการบวกเลขฐาน 2 จำนวน 16 bits มีตัวทศหลักเพิ่มเติมจากบิตที่มีความสำคัญสูงสุด หรือ MSB (Most Significant Bit)

- CF (Carry Flag) จะถูก SET เมื่อมี CARRY OUT ของ MSB
- PF (Parity Flag) จะถูก SET เมื่อผลลัพธ์มี parity เป็น EVEN
- AF (Auxiliary Carry Flag) สำหรับ BCD
- ZF (Zero Flag) จะถูก SET เมื่อผลลัพธ์ = 0
- SF (Sign Flag) เป็นค่า MSB ของผลลัพธ์ (0 เป็นค่าบวก และ 1 เป็นค่าลบ)
- OF (Overflow Flag) จะถูก SET เมื่อผลลัพธ์เกิดการ Overflow

แฟล็กควบคุม (Control Flag) มีทั้งหมด 3 ตัว จะถูกใช้ในการควบคุมการปฏิบัติที่แน่นอนของโปรเซสเซอร์ ซึ่งแฟล็กเหล่านี้จะถูก SET หรือ RESET โดยเจตนาของชุดคำสั่งที่ใส่ไว้ในโปรแกรม

- TF (Trap Flag) ถูกนำมาใช้สำหรับโปรแกรมที่ทำงานในลักษณะ Single Step
- IF (Interrupt Flag) ถูกใช้ในการกำหนดให้อนุญาตหรือปฏิเสธการอินเทอร์รัพต์โปรแกรม
- DF (Direction Flag) ถูกนำมาใช้กับชุดคำสั่งที่เป็น String

แฟล็ก (flag) จะได้ถูกออกแบบมาเพื่อแสดงผลของการกระทำทางคณิตศาสตร์และลอจิก สถานะที่เป็นไปได้ของ flag จะมีได้แค่สองสถานะ คือ ถูกเซตทำให้มีค่าเป็น 1 หรือถูกรีเซตให้มีค่าเป็น 0 แฟล็กตัวทวด (carry flag) จะแสดงลักษณะบอกการทวดหรือ การขอยืมของบิตสูงสุดของผลลัพธ์ , แฟล็กพาริตี (parity flag) จะใช้แสดงจำนวนคู่หรือคี่ของตัวเลขที่เป็น 1 ในรีจิสเตอร์ ถ้าเป็นคู่ flag นี้จะถูกเซตเป็น 1, แฟล็กตัวทวดช่วย (auxiliary carry flag) จะใช้งานแบบ "decimal adjust" ในการคำนวณ BCD, แฟล็กศูนย์ (zero flag) จะมีค่าเป็น 1 เมื่อผลลัพธ์เป็นศูนย์แฟล็กเครื่องหมาย (sign flag) จะได้รับการเซตเมื่อผลลัพธ์เป็นเลขจำนวนลบ , โอเวอร์โฟลว์แฟล็ก (OF) ใช้เพื่อแสดงว่ามีความผิดพลาดในการคำนวณ เนื่องจากการใส่เครื่องหมายของผลลัพธ์ เช่นถ้ามีการบวกเลข +127 กับ +2 ผลลัพธ์ที่ได้ถ้าคิดคำนวณแบบมีเครื่องหมายจะได้คำตอบเป็น -127 ซึ่งผิดความหมาย เมื่อเกิดเหตุการณ์ในลักษณะนี้ 8086 จะเซตโอเวอร์โฟลว์แฟล็กให้เป็น 1

แฟล็กควบคุมทิศทาง (direction flag) จะกำหนดทิศทางของกลุ่มคำสั่งเกี่ยวกับสตริงว่าจะมีทิศทางการทำงานจากแอดเดรสมากไปน้อยหรือจากแอดเดรสน้อยไปมาก แฟล็กอินเตอร์รัพต์ (IE) ถ้าถูกเซตเป็น 1 แสดงว่าจะยอมรับอินเตอร์รัพต์จากภายนอก โดยหยุดการทำงานตามปกติของโปรแกรมไว้ก่อน และ Tap flag (TF) จะทำให้ 8086 ทำงานแบบทีละคำสั่งเพื่อการแก้ไขโปรแกรม

2.3.2 BIU (Bus Interface Unit) ประกอบด้วย

- **Queue** : BIU จะจัดเก็บชุดคำสั่งที่ fetch มาจากหน่วยความจำได้สูงสุด 6 คำสั่ง โดยจะใช้เทคโนโลยีที่เรียกว่า FIFO (First-In First-Out) เมื่อ EU พร้อมที่จะทำงานกับชุดคำสั่งต่อไปก็เพียงแต่นำข้อมูลและชุดคำสั่งมาจาก Queue โดยไม่ต้องไปยุ่งเกี่ยวกับ Bus จึงสามารถทำงานได้เร็วขึ้นอีกหลายเท่า แทนที่จะต้องรอกการอ่านข้อมูลและชุดคำสั่งมาจากหน่วยความจำภายนอก เว้นแต่ว่าเมื่อเมื่อพบคำสั่ง JMP (Jump) หรือ CALL ที่ต้องเรียกข้อมูลมาจากแอดเดรสที่กำหนดไว้ในโปรแกรม คุณสมบัติในการ Fetch คำสั่งต่อไปมาเก็บไว้ใน Queue ในขณะที่ EU กำลัง execute คำสั่งอื่นอยู่ เรียกว่า Pipelining

- **Segment Register** : BIU จะส่งค่า address ขนาด 20 bits ออกไป ดังนั้นจึงสามารถที่จะระบุตำแหน่งใน Memory ได้ $2^{20} = 1,045,576$ bytes แต่อย่างไรก็ตาม 8086 จะสามารถทำงานกับ segment 4 ส่วน ในหน่วยความจำ (ส่วนละ 65,536 bytes = 64 Kbyte) ดังนั้นภายในหน่วยความจำขนาด 1 Mbyte (1,045,576 bytes) จึงต้องมี segment register 4 ตัว เพื่อจัดเก็บค่าแอดเดรสจุดเริ่มต้นของแต่ละ Segment ในหน่วยความจำ ที่ 8086 ใช้ในการทำงาน

- CS (Code Segment)
- SS (Stack Segment)
- ES (Extra Segment)
- DS (Data Segment)

Segment Register ถูกใช้ในการจัดเก็บค่าแอดเดรสขนาด 16 bits ของจุดเริ่มต้นของแต่ละส่วน ตัวอย่างเช่น Code Segment Register จะจัดเก็บค่าแอดเดรสเริ่มต้นของหน่วยความจำส่วนที่ BIU fetch instruction มา โดย BIU จะใส่ค่า 0 ให้กับ 4 bits สุดท้ายของแอดเดรสขนาด 20 bits เช่นถ้า code segment register มีค่า 348A (Hex) แสดงว่าส่วนในหน่วยความจำ ที่จัดเก็บโปรแกรมจะเริ่มต้นที่ address 348A0 (Hex) จึงหมายความว่าแต่ละส่วนขนาด 64 K สามารถที่จะอยู่ตำแหน่งใดในหน่วยความจำขนาด 1 MB

Stack เป็นส่วนของหน่วยความจำที่แยกไว้ต่างหาก เพื่อจัดเก็บค่าแอดเดรสเริ่มต้นของโปรแกรมน้อยและเก็บข้อมูลที่ใช้ในโปรแกรมน้อย ในขณะที่ทำการ execute Subprogram หรือ Procedure

2.4 การเกิดอินเทอร์รัพต์ (Interrupt Handle)

การอินเทอร์รัพต์ของ 8086 อาจจะมาจกหลาย ๆ แหล่งได้ เช่นอินเทอร์รัพต์ที่มาจากอุปกรณ์ภายนอก อินเทอร์รัพต์ทาง software หรืออินเทอร์รัพต์ที่มาจากตัว CPU 8086 เองก็ได้ อินเทอร์รัพต์จากอุปกรณ์ภายนอก จะเข้ามาทางขา 2 ขา คือ INTR (interrupt request) และ NMI(non-maskable interrupt) โดยที่ INTR จะถูก กำหนดให้ทำงานได้โดยคำสั่ง IE (interrupt enable flag) ที่เป็นส่วนหนึ่งของ 8086 flag ถ้า flag IE ถูกเซตจะ ทำให้ 8086 สามารถจะรับอินเทอร์รัพต์จาก INTR ได้ แต่ถ้า IF ถูกเคลียร์ 8086 จะไม่รับอินเทอร์รัพต์จาก INTR ส่วน NMI จะเป็นอินเทอร์รัพต์ที่ไม่สามารถจะหยุดได้ มักจะใช้ในกรณีที่เกิดเหตุการณ์สำคัญ เช่น เกิดความ ผิดพลาดของพาริตีหน่วยความจำ (memory parity) หรือเกิดไฟตก เป็นต้น

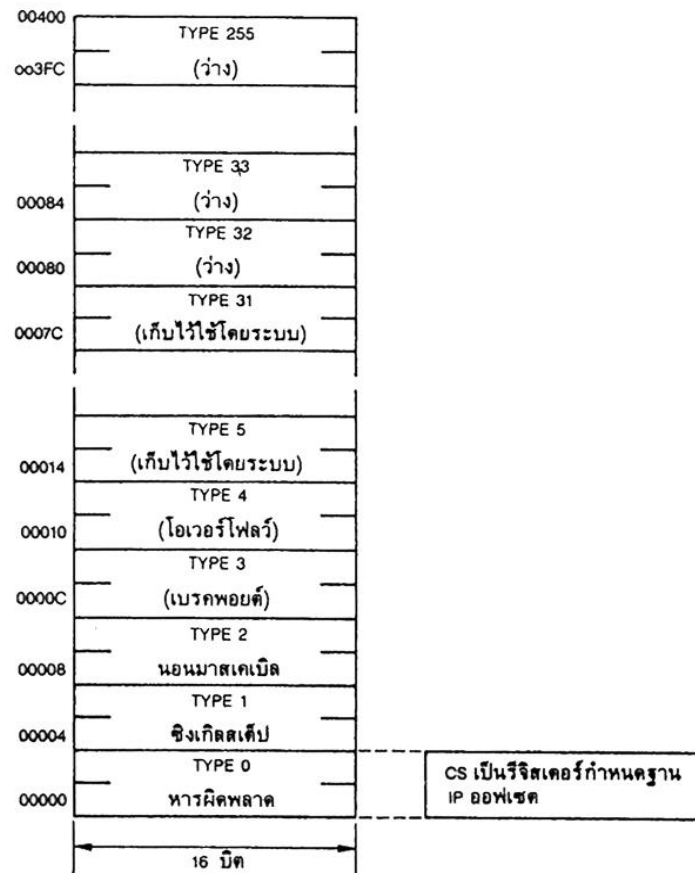
INTO (interrupt overflow) คือ อินเทอร์รัพต์ของ 8086 ที่เกิดจากข้อผิดพลาดของการหารและการทำงานที่ ละชั้น ข้อผิดพลาดจากการหาร (divide error) เกิดขึ้นเนื่องจากผลของการหารอาจจะมีค่ามากกว่าเนื้อที่ซึ่ง เตรียมไว้เป็นที่เก็บผลลัพธ์ หรืออาจจะเกิดจากการหารด้วยเลข 0 ส่วนลักษณะของการทำงานที่ละชั้น (single step) จะเป็นอินเทอร์รัพต์ที่เกิดเนื่องจากการ execute คำสั่งทีละคำสั่ง เหตุการณ์นี้เกิดขึ้นเนื่องจาก TF (trap flag) ในรีจิสเตอร์แฟล็กถูกเซตให้เป็น 1

เมื่อเกิดอินเทอร์รัพต์เหล่านี้ขึ้นมาแล้วก็จะเกิดเหตุการณ์เฉพาะอย่างขึ้นถ้ามีการอินเทอร์รัพต์จากภายนอก คำสั่งชุดสุดท้ายจะถูก execute จนเสร็จสิ้นก่อน (ยกเว้น ถ้าเป็นคำสั่ง MOV หรือ POP จะมีการ execute คำสั่ง ต่อไปก่อนแล้วจึงหยุด) อินเทอร์รัพต์ทุกตัวจะทำการเก็บค่าของแฟล็กรีจิสเตอร์ไว้ในสแตค เสร็จแล้ว 8086 จะทำ การรีเซตแฟล็ก IF และ TF ให้เป็น 0 ทั้งนี้เพื่อป้องกัน INTR อินพุตที่เข้ามาใหม่ และป้องกันการเกิดการทำงานที่ ละชั้นในขณะที่อยู่ในโปรแกรมอินเทอร์รัพต์ เนื่องจากการที่แฟล็กถูกเก็บค่าไว้หลังจากโปรแกรมอินเทอร์รัพต์ สิ้นสุดแล้วค่าต่าง ๆ ที่เก็บไว้จะถูกดึงกลับมาใช้งานได้ จากนั้น 8086 จะทำการเก็บค่าของ CS และ IP ลง บนสแตคเพื่อที่จะเก็บตำแหน่งของแอดเดรสส่งกลับ (return address) ในลักษณะเดียวกับคำสั่ง CALL และใน ขั้นตอนสุดท้าย CS และ IP จะถูกอ่านจากตาราง interrupt vector (ขึ้นกับหมายเลขอินเทอร์รัพต์) ตาราง interrupt vector ก็คือบริเวณของเนื้อที่ของหน่วยความจำเริ่มตั้งแต่แอดเดรส 0000 จำนวน หนึ่งกิโลไบต์ ดัง แสดงได้ดังรูปที่ 2.5

อินเทอร์รัพต์แต่ละตัวจะชี้ไปยังตำแหน่งที่แน่นอนและสามารถที่จะกระโดดไปถึงที่กำหนดได้โดยอัตโนมัติ เมื่อเกิดอินเทอร์รัพต์จากลักษณะของ divide error จะทำให้เกิดอินเทอร์รัพต์ 0, single-step จะเป็น อินเทอร์รัพต์ 1, อินเทอร์รัพต์แบบ Non-maskable เป็นแบบอินเทอร์รัพต์ 2 , bread-point จะเกิดอินเทอร์รัพต์ 3 , และการเกิด over flow จะเป็นอินเทอร์รัพต์ 4 ส่วน อินเทอร์รัพต์ 5 ถึง 31 เตรียมไว้สำหรับ ผู้ใช้จะกำหนด เอาเองภายหลัง ส่วน อินเทอร์รัพต์ที่เหลือเราอาจจะควบคุมโดยใช้คำสั่ง INTR หรือ INT ได้

สำหรับ INTR จะตอบรับสัญญาณอินเทอร์รัพต์จากภายนอก ส่วน INT จะรับได้โดยตัวคำสั่ง นั่นเอง CPU 8086 สามารถที่จะรับรู้ชนิดของการอินเทอร์รัพต์ได้โดยการคูณค่าของคำสั่ง ด้วย 4 ผลลัพธ์ที่ได้จะบอกตำแหน่ง ของตารางอินเทอร์รัพต์ ซึ่งค่าของตารางจะถูกส่งค่าไปยัง CS และ IP ค่าต่าง ๆในตาราง เราจะต้องเป็นผู้กำหนด โดยกำหนดเป็นค่าของเซกเมนต์และค่าของออฟเซตที่แสดงตำแหน่งของอินเทอร์รัพต์รูทีน เช่น อินเทอร์รัพต์ที่ แอดเดรส 0-3 จะถูกต้องกำหนดเป็นค่าบอกแอดเดรสของ CS และ IP เพื่อจะชี้รูทีนของ divide error เป็นต้น

อินเทอร์รัพต์รูทีน จะต้องเริ่มต้นด้วยการเก็บค่าของรีจิสเตอร์ต่าง ๆไว้ก่อน เพราะเราไม่ทราบว่าจะเกิด อินเทอร์รัพต์เมื่อถึงจุดใด และเราไม่ต้องการทำให้ค่า รีจิสเตอร์ต่าง ๆ เสียไปและเราต้อง ปิดท้ายรูทีนด้วยคำสั่ง IRET คำสั่งนี้จะทำงานเหมือนคำสั่ง RET ธรรมดาแต่จะทำการอ่านค่าของแฟล็กออกมาจากสแตคด้วย



รูปที่ 2.5 แสดงการเกิดอินเทอร์รัพต์ ที่ตำแหน่ง 0000- 03FF ขนาด 1 กิโลไบต์ (1Kbytes)

2.6 รีจิสเตอร์ของซีพียู 8086 (CPU 8086 Register)

CPU 8086 ประกอบด้วยรีจิสเตอร์ที่เกี่ยวข้อง ออกเป็น 2 กลุ่ม (แสดงได้ดังรูปที่ 2.6) ดังนี้

2.6.1 กลุ่มข้อมูล (Data Group Register) แบ่งออกเป็น

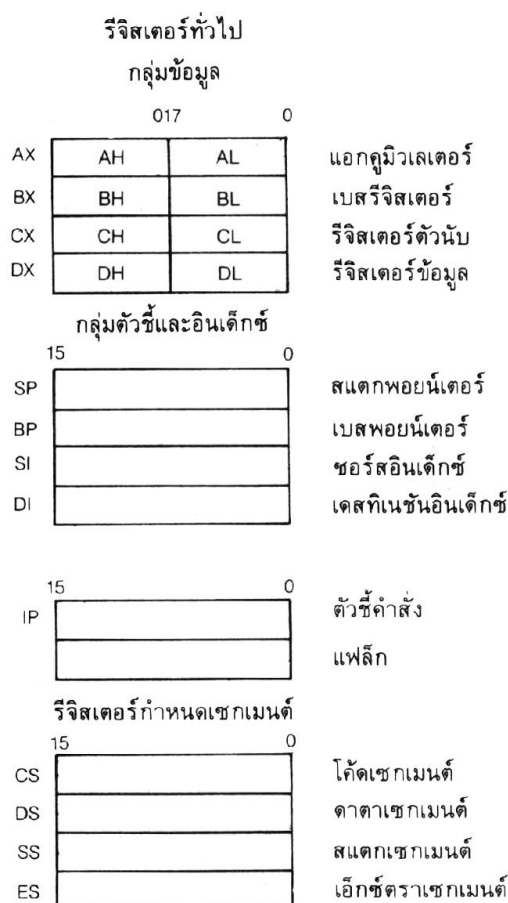
2.6.1.1 รีจิสเตอร์ทั่วไป (General-Purpose Register)

2.6.1.2 รีจิสเตอร์ตัวชี้ (Instruction Pointer Register)

2.6.1.3 แฟล็ก (flag)

2.6.2 กลุ่มกำหนดเซกเมนต์ (Segment Register) แบ่งออกเป็น

2.6.2.1 รีจิสเตอร์กำหนดเซกเมนต์ (Segment Register)



รูปที่ 2.6 แสดงรีจิสเตอร์ที่ใช้ในไมโครโพรเซสเซอร์ 8086

2.6.1.1 รีจิสเตอร์ทั่วไป (General-Purpose Register)

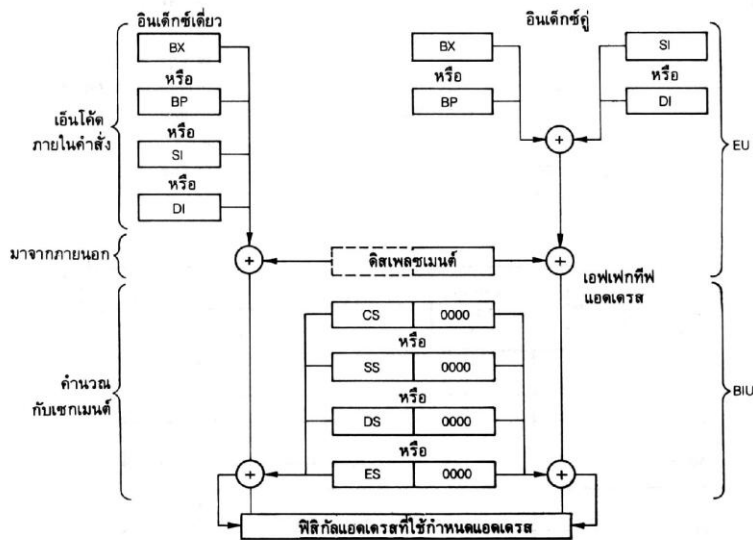
รีจิสเตอร์ที่มีอยู่ทั้งหมดของ 8086 เป็นรีจิสเตอร์ขนาด 16 บิต ซึ่งกลุ่มข้อมูลจะแบ่งย่อยออกเป็น

1. **รีจิสเตอร์ AX (Accumulator)** หรือแอกคูมิวเลเตอร์ จะประกอบด้วย AH และ AL โดย AX ทำหน้าที่ในการใช้คูณหารหรือเป็นรีจิสเตอร์เกี่ยวกับอินพุตเอาต์พุตที่เป็นเวิร์ด ส่วน AL ทำหน้าที่ใช้ในการคำนวณ คูณหาร หรือทำงานเกี่ยวกับอินพุต (Input) เอาต์พุต (Output) แปลง ข้อมูลจัดการคำนวณแบบตัวเลขฐานสิบแบบ 8 บิต และ AH ใช้เป็นรีจิสเตอร์ในการคูณและหารได้
2. **รีจิสเตอร์ BX (Base Register)** หรือเบสรีจิสเตอร์ใช้ในการแปลงข้อมูล
3. **รีจิสเตอร์ CX (Counter Register)** หรือรีจิสเตอร์ตัวนับใช้ในการคำสั่งจัดการเกี่ยวกับสตริง และการทำลูป CL ใช้เป็นตัวแปรสำหรับการเลื่อนบิตหรือหมุนบิต
4. **รีจิสเตอร์ DX (Data Register)** หรือรีจิสเตอร์ข้อมูลใช้ในการคำสั่ง คูณ หารเป็นเวิร์ด (Words) หรือใช้ในรูปแบบอ้างอิงอินพุตเอาต์พุตแบบอ้อม แสดงได้ดังรูปที่ 2.6

2.6.1.2 รีจิสเตอร์ตัวชี้ (Instruction Pointer Register)

รีจิสเตอร์ใช้งานทั่วไปในกลุ่มตัวชี้และอินเด็กซ์ ประกอบด้วย SP หรือสแตคพอยน์เตอร์ใช้กับคำสั่งสแตค สำหรับรีจิสเตอร์ต้นทางหรือ SI (Source Index) รีจิสเตอร์ ปลายทางหรือ DI (Destination Index) จะใช้กับ

คำสั่งที่เกี่ยวข้องกับสตริง และเบสพอยน์เตอร์หรือ BP (base pointer) รีจิสเตอร์เหล่านี้สามารถที่จะนำมาใช้เป็นตัวอ้างถึง Physical address ได้ด้วยวิธีการต่างๆ ดังแสดงได้ดังรูปที่ 2.7



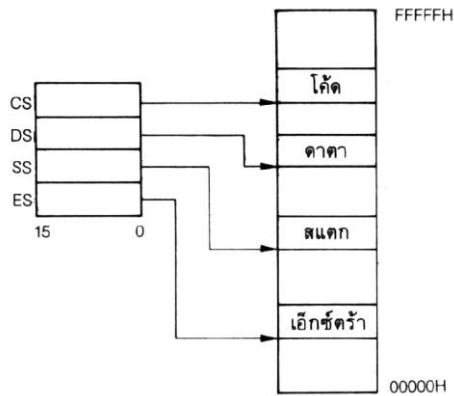
รูปที่ 2.7 แสดงโครงสร้างรีจิสเตอร์ของ 8086 กับการคำนวณหาตำแหน่งในหน่วยความจำ

อธิบายจากรูปที่ 2.7 รีจิสเตอร์ขนาด 16 บิต 4 ตัว ในกลุ่มข้อมูล ได้แก่ AX (Accumulator), BX (Base), CX (Counter) และ DX (Data) ยังถูกแบ่งออกเป็นครึ่งบน (AH, BH, CH และ DH) และครึ่งล่าง (AL, BL, CL และ DL) รีจิสเตอร์เหล่านี้อาจจะทำงานในลักษณะ เป็นเวิร์ดหรือไบต์ก็ได้ ในกลุ่มนี้ถือว่าเป็นรีจิสเตอร์ ข้อมูลรีจิสเตอร์ในกลุ่ม pointer และ index มักจะนำไปใช้ในการอ้าง Physical address และยังใช้ในการปฏิบัติทางคณิตศาสตร์และลอจิกได้รีจิสเตอร์ BP และ SP ใช้ในการชี้ตำแหน่งในสแตก (Stack Address) เพื่อจะทำการเก็บตำแหน่ง address กลับจากโปรแกรมย่อย (Subprogram) โดยใช้หลักการเหมือนสแตก (Stack) ของ CPU 8 บิต ส่วน BP นั้นจะใช้สำหรับเป็น ตัวบวก เพื่อชี้ค่าสแตกอีกทีหนึ่ง เพื่อจะเป็นวิธีที่จะอำนวยความสะดวกในการส่งค่า พารามิเตอร์เข้าสู่โปรแกรมโดยผ่านทางสแตก ซึ่งช่วยในการส่งค่าของตัวแปรระหว่างโปรแกรมได้ดี รีจิสเตอร์ SI และ DI จะทำการติดต่อกับหน่วยความจำได้ ซึ่งจะใช้งานได้ดีในกลุ่มคำสั่งพวก สตริงและการทำงานเป็น block หรือ การเชื่อมโยงข้อมูลในโครงสร้างแบบ array ในการ fetch คำสั่งของ 8086 จะใช้ IP (instruction pointer) ขนาด 16 บิต เป็นตัวชี้ตำแหน่ง address ของคำสั่งถัดไปที่จะถูกทำงาน ซึ่งความจริงแล้วค่าใน IP ไม่ได้กำหนดตำแหน่งโดยตรง แต่จะมีการคำนวณมาก่อน

2.6.2.1 รีจิสเตอร์กำหนดเซกเมนต์ (Segment Register)

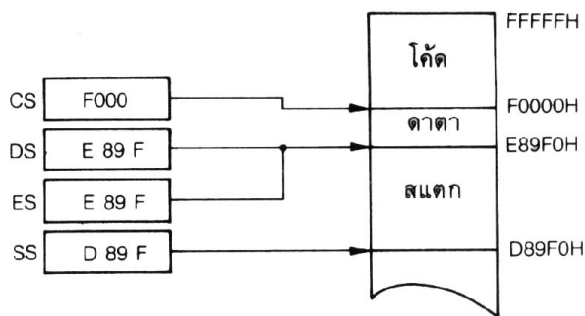
ประกอบด้วยรีจิสเตอร์ย่อยมี 4 ตัว คือ CS (code segment), DS (data segment), SS (stack segment) และ ES (extra segment) ซึ่งจะนำมาใช้ในหลักการแบบบล็อกล็อกแอดเดรสของ 8086 หน่วยความจำ (Memory) จะได้รับการจัดสรรเป็นส่วนใหญ่ ๆ คือ code (หรือชุดคำสั่ง), ข้อมูล (ตัวเลข หรือตัวอักษร) และสแตกสำหรับการเก็บค่าแอดเดรสกลับคืนจากโปรแกรมย่อยเพื่อที่จะป้องกันการสับสน จึงกำหนดค่าให้แยกกันได้ว่า 8086 จะมองลักษณะของหน่วยความจำ โดยแบ่งหน่วยความจำเป็นกลุ่ม ๆ ในรูปแบบของ เซกเมนต์ ในหนึ่งเซกเมนต์จะชี้ได้ถึง 64 กิโลไบต์ โดยเซกเมนต์รีจิสเตอร์ทั้งสี่ตัว จะแสดงแอดเดรสเริ่มต้นของหน่วยความจำที่จะติดต่อกับ สังกัดได้จากรูปที่ 2.8 รีจิสเตอร์ CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรม, DS จะเก็บค่า

Data segment ในขณะนั้น,SS จะเก็บค่า Stack segment ในขณะนั้น และ ES จะกำหนด Segment ของข้อมูลรวมที่เรียกว่า Global Data Segment แสดงได้ดังรูปที่ 2.8



รูปที่ 2.8 แสดงการแบ่งกลุ่มของ Segment Register

เซกเมนต์จะแสดงตำแหน่งเหมือนกับ Paragraph โดยจะเลื่อนไปทางซ้าย 4 บิต เพื่อที่จะกำหนด หรือ อ่างแอดเดรสให้ครบ 20 เส้น โดยจุดเริ่มต้นของ Paragraph จะต้องมีการมี 4 บิต หลังสุดเป็น 0 เช่น เป็น 00000H, 00010H , 00020H เป็นต้น จากรูป ค่าของ SS จะชี้ตำแหน่งของเซกเมนต์ค่า D89F0 สังเกตเห็นว่าส่วนพื้นที่ของหน่วยความจำอาจจะสลับหรือใช้บริเวณเดียวกันของหน่วยความจำได้แสดงได้ดังรูปที่ 2.8



รูปที่ 2.9 แสดงตำแหน่งเหมือนกับ Paragraph ใน Segment Register

เพื่อที่จะทำการติดต่อกับข้อมูลหนึ่งไบต์หรือหนึ่งเวิร์ดนั้น 8086 ได้เตรียมค่าออฟเซต เพื่อใช้อ้างตำแหน่ง ตั้งแต่จุดเริ่มต้นของเซกเมนต์แอดเดรส ตำแหน่งใด ๆ จะได้มาจากการบวกค่า เซกเมนต์ รีจิสเตอร์กับค่าของออฟเซต 16 บิต เช่นถ้าเซกเมนต์มีค่า E89F จะให้ออฟเซตมีค่า 0003H จะทำให้การอ่างแอดเดรสไปที่ 89F3 การจะใช้เซกเมนต์รีจิสเตอร์และค่าออฟเซตตัวใดนั้นจะขึ้นอยู่กับชนิดของคำสั่งด้วย ดังแสดงในตารางที่ 2.1

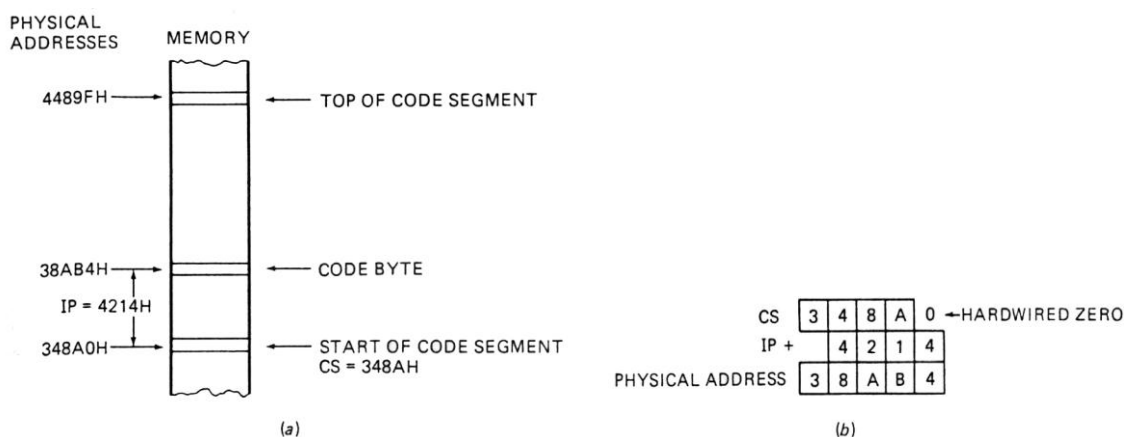
ตารางที่ 2.1 แสดงชนิดของการอ้างอิงหน่วยความจำในเซกเมนต์รีจิสเตอร์

ชนิดของการอ้างอิงหน่วยความจำ	ค่าเซกเมนต์ปกติ	ค่าเซกเมนต์ที่เลือกได้	ค่าออฟเซต
ค่า Fetch คำสั่ง	CS	-	IP
การทำ Stack	SS	-	SP
กำหนดตัวแปร	DS	CS, ES, SS	ค่า Address ที่กำหนด
สตริงต้นทาง	DS	CS, ES, SS	SI
สตริงปลายทาง	ES	-	DI
Base Register (BP)	SS	CS, ES, SS	ค่า Address ที่กำหนด

2.7 การระบุตำแหน่งในรีจิสเตอร์ (Addressing Data in Register)

การเข้าถึงข้อมูลในรีจิสเตอร์นับว่าเป็นหัวใจสำคัญในการเขียนโปรแกรมควบคุมการทำงานของไมโครโพรเซสเซอร์ 8086 สามารถ access เข้าไปในส่วนของโปรแกรม โดยใช้ค่าในรีจิสเตอร์ โดย

CS + IP -> physical address (20 bits) แสดงได้ดังรูปที่ 2.10



รูปที่ 2.10 แสดงการบวกค่าในรีจิสเตอร์ CS และ IP เพื่อคำนวณหา Physical Address

ในการ Access เข้าไปยังข้อมูล เพื่อนำมาใช้ในการทำงาน เรียกว่า “addressing mode” ซึ่งในภาษาแอสเซมบลี ใช้คำสั่ง MOV: format -- MOV destination, source เมื่อชุดคำสั่งนี้ถูก execute 8086 จะทำการคัดลอก word หรือ byte จากตำแหน่งต้นทางไปยังจุดหมายปลายทาง Addressing Mode ใช้ในการระบุตำแหน่งของ operand ใน Memory ซึ่งสามารถแบ่งวิธีการเข้าถึงหน่วยความจำได้ดังนี้ คือ

2.7.1 Immediate Addressing Mode

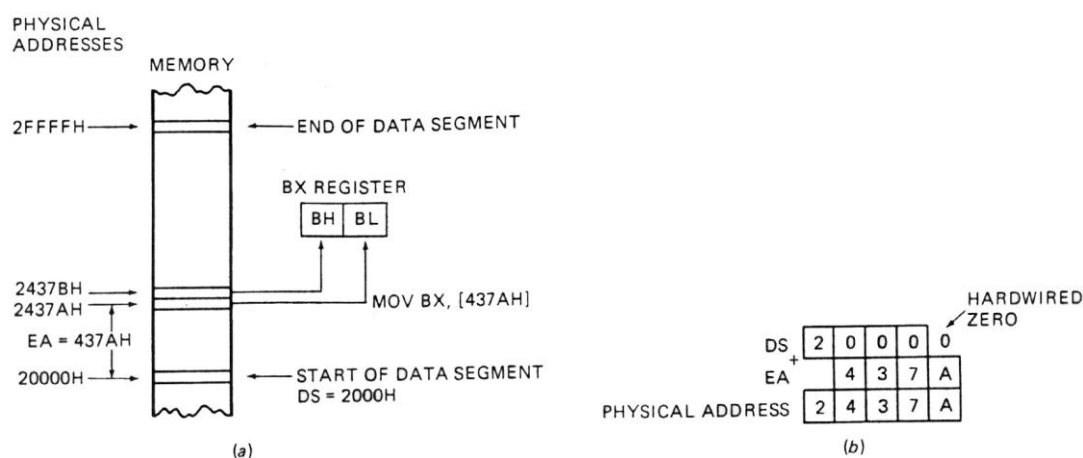
เป็นวิธีการอย่างง่ายในการกำหนดแอดเดรสในรีจิสเตอร์ กล่าวคือสามารถกำหนด operand ซึ่งเป็นค่าแอดเดรสให้กับชุดคำสั่งภาษาแอสเซมบลีได้โดยตรง ตัวอย่างเช่น ถ้าโปรแกรมต้องการที่จะใส่ค่า 437B (Hex) ลงใน CX Register ก็สามารถใช้คำสั่ง MOV CX, 437B (Hex) ได้โดยตรง

2.7.2 Register Addressing Mode

ในกรณีที่ใช้ค่าในรีจิสเตอร์เป็น operand ของชุดคำสั่ง เช่น MOV CX, AX คือการคัดลอก เนื้อหาหรือข้อมูล(Content or Data) ที่อยู่ใน AX Register มาไว้ใน CX Register ใช้ในกรณีที่จำนวน bit เท่ากันเท่านั้น ในกรณีที่รีจิสเตอร์มีขนาดไม่เท่ากันจะไม่สามารถใช้คำสั่ง MOV CX, AL ได้ ทั้งนี้เนื่องจากหากใช้คำสั่งนี้ 8086 จะพยายามที่จะทำสำเนาข้อมูล (Copy Data) ในรูปแบบ byte-type (AL) เข้าไปเก็บในรีจิสเตอร์ซึ่งเป็นรูปแบบ word-type (CX) ซึ่งเป็นไปไม่ได้ อย่างไรก็ตามหากใช้คำสั่ง MOV AL, CX ก็ยังเป็นไปไม่ได้ เนื่องจากถึงแม้ว่าขนาดของรีจิสเตอร์จะสามารถรองรับข้อมูลได้ แต่ 8086 จะไม่ทราบว่าผู้เขียนโปรแกรมต้องการที่จะนำข้อมูลสำเนาไปเก็บไว้ในครั้งใดของรีจิสเตอร์ CX ในกรณีเช่นนี้ โดยทั่วไป Assembler จะตรวจสอบพบ และจะบอกให้ทราบว่า Type error ดังนั้นการที่จะคัดลอกไบต์จาก AL ไปเก็บยังไบต์สูงของ CX จะใช้คำสั่ง MOV CH, AL และหากต้องการคัดลอกไบต์จาก AL ไปเก็บยังไบต์ต่ำของ CX จะใช้คำสั่ง MOV CL, AL

2.8 การเข้าถึงข้อมูลในหน่วยความจำ (Addressing Data in Memory)

ที่ผ่านมาได้กล่าวถึง Addressing Mode ที่จะระบุตำแหน่งของ operand ในหน่วยความจำ ในกรณีที่ต้องการที่จะเข้าถึงข้อมูลในหน่วยความจำ 8086 จะต้องสร้าง physical address (20 bits) ซึ่งสามารถทำได้โดยการรวม Effective address (16 bits) เข้ากับ Segment Base Address (16 bits) แสดงได้ดังรูปที่ 2.11

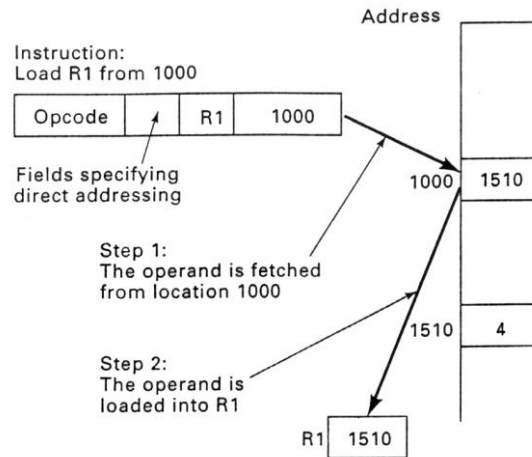


รูปที่ 2.11 แสดงการบวกค่าในรีจิสเตอร์ DS กับ EA เพื่อหา Physical Address

วิธีการเข้าถึงข้อมูลในหน่วยความจำในไมโครโพรเซสเซอร์ 8086 แบ่งได้เป็น 4 วิธี คือ

2.8.1 วิธีการเข้าถึงข้อมูลในหน่วยความจำโดยตรง (Direct Addressing Mode)

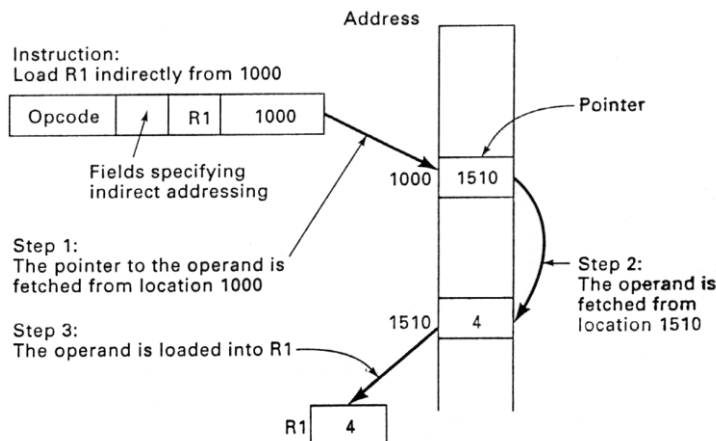
เป็นวิธีการเข้าถึงข้อมูลในหน่วยความจำที่ง่ายที่สุด ซึ่งสามารถทำได้โดยการนำค่า effective address (16 bits) ใส่เข้าไปในชุดคำสั่งโดยตรง ตัวอย่างเช่นคำสั่ง MOV BL, [437A (Hex)] - 8086 จะคัดลอก Low byte -> Low address (437A) และ High byte -> High address (437B) ดังนั้นในกรณีที่ต้องการ Load รีจิสเตอร์ R1 ด้วยค่าที่อยู่ในแอดเดรส 1000 จะมีการทำงานแสดงได้ดังรูปที่ 2.12



รูปที่ 2.12 แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยตรง (Direct Addressing Mode)

2.8.2 วิธีการเข้าถึงข้อมูลในหน่วยความจำโดยอ้อม (Indirect Addressing Mode)

เป็นวิธีการเข้าถึงข้อมูลในหน่วยความจำที่แตกต่างจาก Direct Addressing ตรงที่แทนที่จะใส่ค่า effective address (16 bits) เข้าไปในชุดคำสั่งโดยตรง แต่จะใส่ค่า Address ของหน่วยความจำในตำแหน่งที่จัดเก็บ effective address เข้าไปในชุดคำสั่ง จะเห็นได้ว่าวิธีการนี้ ค่าในแอดเดรสที่ระบุจะเป็นแอดเดรสที่บรรจุค่าที่ต้องการไว้ เราเรียกแอดเดรสภายในแอดเดรสที่ระบุว่าเป็น “ตัวชี้ (Pointer)” เช่นถ้าต้องการ Load รีจิสเตอร์ R1 ด้วยค่าในแอดเดรสที่แอดเดรส 1000 กำหนดไว้ จะมีการทำงานแสดงได้ดังรูปที่ 2.13



รูปที่ 2.13 แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยอ้อม (Indirect Addressing Mode)

2.8.3 วิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้ตัวชี้ (Indexing Addressing Mode)

บางครั้งในการทำงานซึ่งต้องการกระทำบางอย่างกับชุดข้อมูลเป็นจำนวนมากที่มีโครงสร้างการจัดเก็บเป็นลำดับ อาจจะสามารถทำได้โดยใช้วิธี Indirect Addressing แต่อย่างไรก็ตามหากต้องการ Load ค่า 2 ค่าจาก 2 แอดเดรส เข้ามาในรีจิสเตอร์ตัวเดียวกัน จำเป็นต้องใช้ Pointer 2 ตัว และทุกครั้งที่ Pointer เลื่อนไปชี้ที่แอดเดรสหนึ่ง Pointer จะถูกเพิ่มค่าขึ้น 1 เพื่อชี้ไปยังแอดเดรสถัดไป จึงสามารถทำงานกับชุดข้อมูลที่มีโครงสร้างการจัดเก็บเป็นลำดับได้ แต่อย่างไรก็ตามเนื่องจาก Pointer เป็นส่วนหนึ่งของข้อมูล ไม่ได้เป็นส่วนของโปรแกรม

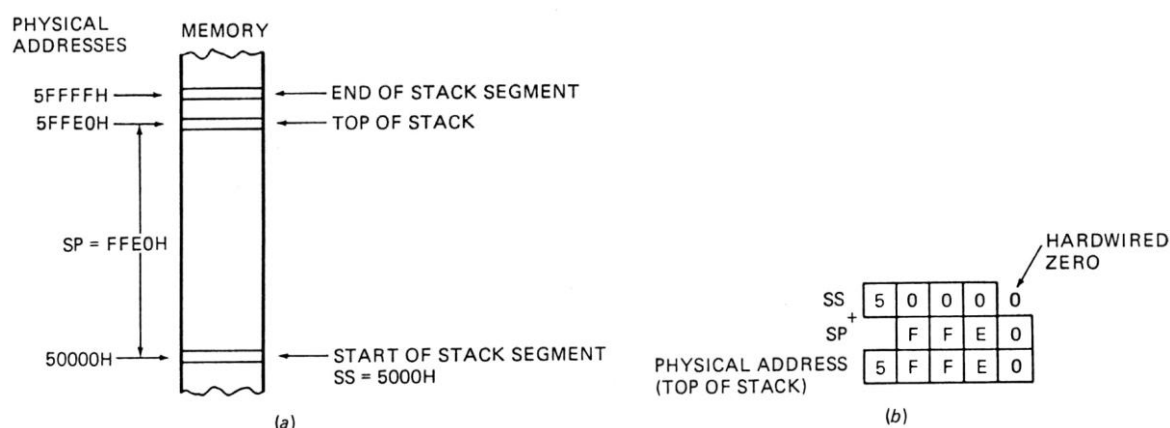
ดังนั้นผู้อื่นจะไม่สามารถทราบได้ว่าโปรแกรมนั้นต้องการใช้ทำอะไรแน่ การแก้ปัญหาอย่างหนึ่งคือการใช้รีจิสเตอร์เพิ่มขึ้นมาจากเดิมอีก 1 ตัว เรียกว่า “Index Register” โดยจะทำหน้าที่ในการจัดเก็บค่าดัชนีแอดเดรสของชุดข้อมูลเพื่อให้สะดวกต่อการเรียกใช้ ในปัจจุบันได้มีการพัฒนารีจิสเตอร์ชนิดนี้ให้มีขีดความสามารถในการเพิ่ม หรือลดค่าลงครั้งละ 1 เพื่อให้สามารถจัดการกับชุดข้อมูลที่มีโครงสร้างการจัดเก็บเรียงเป็นลำดับได้โดยอัตโนมัติ จึงทำให้มีความสามารถในการ Address โดยอัตโนมัติเรียกว่า “autoindexing”

2.8.4 วิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้สแต็ก (Stack Addressing Mode)

เป็นวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้สแต็ก (Stack Addressing) ซึ่งสแต็กประกอบด้วยข้อมูลที่ถูกจัดเก็บเรียงเป็นลำดับไว้ในหน่วยความจำ โดยข้อมูลแรกจะถูกใส่ (Push) เข้าไปในสแต็ก และจะลงไปอยู่ที่ส่วนล่างสุดของสแต็ก ข้อมูลต่อไปก็จะถูกนำไปจัดเก็บในแอดเดรสถัดมา การที่จะเข้าไปเรียกข้อมูลในสแต็กจึงจำเป็นต้องมีแอดเดรสที่ระบุว่าเป็นส่วนบนสุดของสแต็ก ซึ่งเรียกว่า “Stack Pointer” กระบวนการเข้าไปยังสแต็กสามารถทำได้โดยนำค่า

$$SS (\text{Stack Segment}) + SP (\text{Stack Pointer}) \rightarrow \text{Physical address (Stack's top address)}$$

แสดงได้ดังรูปที่ 2.14



รูปที่ 2.14 แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้สแต็ก (Stack Addressing Mode)

สรุป

ไมโครโพรเซสเซอร์ (microprocessor) เป็นอุปกรณ์อิเล็กทรอนิกส์ชนิดหนึ่งซึ่งมีลักษณะวงจรรวม (Integrated Circuit) หรือชิพ (Chip) โครงสร้างภายในจะมีวงจรรวมขนาดใหญ่ ส่วนไมโครคอนโทรลเลอร์ (microcontroller) เป็นชิพประมวลผลประเภทหนึ่งซึ่งทำหน้าที่ตามโปรแกรมหรือชุดคำสั่งที่เขียนขึ้นมา ภายในประกอบด้วยวงจรรวมขนาดใหญ่ (Very Large Scale Integrated Circuit) การนำไมโครโพรเซสเซอร์ หลายๆ ตัวให้สามารถทำงานรวมกันได้ในเวลาเดียวกัน ทำให้ประสิทธิภาพการทำงานและความเร็วนั้นเพิ่มมากขึ้น นอกจากนี้จำนวนข้อมูลที่ใช้ในการประมวลผลได้ถูกพัฒนาเพิ่มมากขึ้นจาก 8 บิต มาเป็น 32 บิต และ 64 บิต ตามลำดับและจะถูกพัฒนาเพิ่มมากขึ้นอย่างต่อเนื่องในอนาคต

ชนิดของไมโครโพรเซสเซอร์ จัดแบ่งตามลักษณะการใช้งานได้ 3 ประเภท คือ

1. Dedicated or Embedded Controller
2. Bit-slice Processor
3. General-purpose Processor

CPU 8086 ประกอบด้วยรีจิสเตอร์ที่เกี่ยวข้อง ออกเป็น 2 กลุ่ม คือ

1. กลุ่มข้อมูล (Data Group Register) แบ่งออกเป็น
 - 1.1 รีจิสเตอร์ทั่วไป (General-Purpose Register)
 - 1.2 รีจิสเตอร์ตัวชี้ (Instruction Pointer Register)
 - 1.3 แฟล็ก (flag)
2. กลุ่มกำหนดเซกเมนต์ (Segment Register) แบ่งออกเป็น
 - 2.1 รีจิสเตอร์กำหนดเซกเมนต์ (Segment Register)

คำถามทบทวน

1. จงอธิบายข้อแตกต่างที่ระหว่างไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์
2. ไมโครโพรเซสเซอร์ จัดแบ่งออกตามลักษณะการใช้งานได้กี่ประเภทอะไรบ้าง
3. จงอธิบายโครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์ 8086 ว่าประกอบด้วยอะไรบ้างพร้อมวาดรูปประกอบอธิบาย
4. จงอธิบายข้อแตกต่างที่สำคัญของไมโครโพรเซสเซอร์ 8086 และ 8088
5. ถ้า Code Segment ของโปรแกรมใน 8086 เริ่มต้นที่ 90400H ค่าที่ถูกจัดเก็บในรีจิสเตอร์ CS คือค่าใด
6. ถ้า Code Segment ของโปรแกรมใน 8086 เริ่มต้นที่ 90400H จงหา Physical Address ของหน่วยความจำที่จะถูก Fetch เข้ามาใช้งาน ถ้าในขณะนั้นรีจิสเตอร์ IP เก็บค่า 0345CH
7. ถ้า Stack Segment Register มีค่า 0500H และ Stack Pointer Register มีค่า 8225H จงหา Physical Address ส่วนบนสุดของสแต็ก (Top of Stack)
8. จงอธิบายข้อได้เปรียบของการใช้รีจิสเตอร์ในหน่วยประมวลผลกลาง (CPU) เป็นที่เก็บข้อมูลชั่วคราว แทนที่จะเก็บข้อมูลไว้ในหน่วยความจำ
9. จงอธิบายการปฏิบัติการของไมโครโพรเซสเซอร์ 8086 เมื่อ Execute คำสั่งต่างๆ ต่อไปนี้
 - 9.1 MOV BX, 08FF
 - 9.2 MOV AX, 0ABH
 - 9.3 MOV DX, CL
 - 9.4 MOV BX, AX
10. จงอธิบายความแตกต่างระหว่างคำสั่ง MOV AX, 1234H กับ MOV AX, [1234H]

แผนบริหารการสอนประจำบทที่ 3

หัวข้อเนื้อหา

- ความหมายของตัวเลขในหลักต่าง ๆ
- การแปลงค่าจากเลขฐานสิบเป็นเลขฐานสอง
- การบวกและการลบเลขฐานสอง
- การคูณและการหารเลขฐานสอง
- เลขฐานแปดและเลขฐานสิบหก
- บิต, ไบต์, เวิร์ด และดับเบิลเวิร์ด
- ระบบเลข 2' Complement
- ระบบเลข BCD (Binary Code Decimal)
- มาตรฐาน IEEE 754

วัตถุประสงค์เชิงพฤติกรรม

- เข้าใจความหมายของตัวเลขในหลักต่าง ๆ
- สามารถแยกบิต, ไบต์, เวิร์ด, ดับเบิลเวิร์ด และทำความเข้าใจได้
- อธิบายและทำความเข้าใจระบบเลข 2' Complement ระบบเลข BCD และมาตรฐาน IEEE 754 ได้

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 3 ระบบเลขจำนวน (Numeric System)

คอมพิวเตอร์เป็นเครื่องจักรที่กลไกการทำงานพื้นฐานเป็นสองสถานะ (Binary) คือเปิดวงจรกับปิดวงจร ซึ่งสามารถแทนสถานะดังกล่าวได้ด้วยตัวเลขโดดสองตัวคือ 0 กับ 1 ข้อมูลแบบอื่นของคอมพิวเตอร์จะเกิดจากการประกอบรวมกันของเลข 0 กับ 1 เท่านั้น เราเรียกระบบเลขจำนวนที่ประกอบด้วยตัวเลข 0 กับ 1 เท่านั้นว่า “เลขฐาน 2”

ส่วนการนับของมนุษย์โดยปกติแล้ว เราจะมีตัวเลขโดดอยู่สิบตัวคือ 0, 1, 2, 3, 4, 5, 6, 7, 8, และ 9 ซึ่งจะประกอบรวมกันเป็นระบบเลขจำนวนที่เรียกกันว่า “เลขฐาน 10” จะเห็นว่าระบบเลขจำนวนที่ใช้ในคอมพิวเตอร์มีความแตกต่างจากระบบเลขจำนวนที่มนุษย์ใช้กันโดยปกติ ดังนั้นเราจะต้องเรียนรู้ถึงทักษะในการคำนวณของระบบเลขจำนวนทั้งสองแบบรวมถึงวิธีการเปลี่ยนระบบเลขจำนวนไปมา

3.1 ความหมายของตัวเลขในหลักต่าง ๆ

ในระบบเลขฐานสิบนั้น ค่าของเลขโดด ณ ตำแหน่งใด ก็คือค่าของเลขโดดนั้นคูณด้วยสิบกกำลังของตำแหน่งนั้น เช่น 12345 หมายความว่า ค่า 5 อยู่ในตำแหน่งหลักหน่วยซึ่งค่าของสิบกกำลังของหลักหน่วยคือ 10^0 ค่า 4 อยู่ในตำแหน่งของหลักสิบ (10^1) ค่า 3 อยู่ในตำแหน่งของหลักร้อย (10^2) ค่า 2 อยู่ในตำแหน่งของหลักพัน (10^3) และค่า 1 อยู่ในตำแหน่งของหลักหมื่น (10^4) ซึ่ง 12345 สามารถเขียนอยู่ในรูปผลบวกทางคณิตศาสตร์ได้ดังนี้

$$\begin{aligned} 12345 &= (1 \times 10^4) + (2 \times 10^3) + (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0) \\ &= 10000 + 2000 + 300 + 40 + 5 \end{aligned}$$

จะเห็นว่าเลขกำลังของสิบจะเริ่มต้นจากศูนย์ที่หลักหน่วย แล้วเพิ่มขึ้นหนึ่งทุกครั้งในหลักถัดมาทางด้านซ้ายมือ ในกรณีที่เลขเป็นจำนวนทศนิยม ให้เริ่มกำลังศูนย์ที่หลักหน่วย แล้วลดกำลังลงหนึ่งทุกครั้งในหลักถัดไปทางด้านขวามือ ส่วนทางด้านซ้ายมือก็จะเป็นไปในรูปแบบเดิม เช่น 12.34 จะสามารถเขียนได้เป็น

$$\begin{aligned} 12.34 &= (1 \times 10^1) + (2 \times 10^0) + (3 \times 10^{-1}) + (4 \times 10^{-2}) \\ &= 10 + 2 + 0.3 + 0.04 \end{aligned}$$

เราสามารถใช้หลักการเดียวกันนี้กับเลขฐานสองเพื่อหาค่าของจำนวนดังกล่าวในรูปของเลขฐานสิบ (ในความเป็นจริงแล้วสามารถที่จะนำไปใช้ได้กับเลขทุกฐาน) เช่น 1011.01_2 จะเขียนได้เป็น

$$\begin{aligned} 1011.01_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) \\ &= 8 + 0 + 2 + 1 + 0 + 0.25 \\ &= 11.25 \end{aligned}$$

3.2 การแปลงค่าจากเลขฐานสิบเป็นเลขฐานสอง

การแปลงเลขฐานสิบเป็นฐานสองจะมีขั้นตอนอยู่สองขั้นตอนคือ การแปลงเลขส่วนที่อยู่หน้าทศนิยมและการแปลงเลขส่วนที่อยู่หลังทศนิยม

การแปลงเลขในส่วนที่อยู่หน้าทศนิยม ให้นำเลขดังกล่าวมาหารด้วยสองไปเรื่อย ๆ จนกว่าจะได้ผลลัพธ์เป็นศูนย์ โดยการหารแต่ละครั้งจะได้เศษเป็น 0 หรือ 1 ลำดับของเศษที่เกิดขึ้นก็คือกำลังของเลขสอง กล่าวคือเศษที่ได้จากการหารครั้งแรกจะเป็นเลขในหลัก 2^0 , เศษที่เกิดจากการหารครั้งที่สองจะเป็นเลขในหลัก 2^1 เรื่อยไป

ตัวอย่าง จงเปลี่ยนค่า 13_{10} ให้เป็นเลขฐานสอง

$$\begin{array}{r} 2 \overline{) 13} \\ \underline{6} \\ 3 \\ \underline{1} \\ 0 \end{array} \quad \begin{array}{l} \text{เศษ } 1 \\ \text{เศษ } 0 \\ \text{เศษ } 1 \\ \text{เศษ } 1 \end{array}$$

$$\therefore 13_{10} = 1101_2$$

ส่วนการแปลงเลขหลังจุดทศนิยมนั้น จะใช้วิธีคูณตัวเลขนั้นด้วยสองไปเรื่อย ๆ จนกว่าจะมีเลขหลังจุดทศนิยมเป็นศูนย์ ซึ่งในการคูณแต่ละครั้งอาจจะมีการทดค่าหลังจุดทศนิยมขึ้นมาเป็นตัวเลข 1 หน้าจุดทศนิยมหรือไม่ก็ได้ ในการคูณแต่ละครั้งก็เท่ากับว่าเราเลื่อนการคำนวณจากหลักแรกหลังจุดทศนิยม (2^{-1}) ไปยังหลักต่อไป

ตัวอย่าง จงเปลี่ยนค่า 0.25_{10} ให้เป็นเลขฐานสอง

$$\begin{array}{r} 0.25 \\ \times \quad \underline{2} \\ \hline 0.50 \\ \times \quad \underline{2} \\ \hline 1.00 \\ \hline 0.01 \end{array}$$

$\therefore 0.25_{10} = 0.01_2$

ดังนั้นจากตัวอย่างข้างต้นสามารถสรุปได้ว่า $13.25_{10} = 1101.01_2$

3.3 การบวกและการลบเลขฐานสอง

การบวกเลขฐานสองมีหลักการเหมือนกับการบวกเลขฐานสิบ การบวกเลขในฐานสิบนั้นเมื่อผลบวกในหลักใดมีค่ามากกว่า 9 ก็จะต้องมีการทดเลข 1 ไปยังหลักถัดไป ซึ่งหลักเกณฑ์การทดเลขนี้ยังสามารถใช้ได้กับเลขฐานสอง เพียงแต่ว่าเลขโดดที่สูงที่สุดของเลขฐานสองคือ 1 ดังนั้นถ้าผลบวกมีค่าเกิน 1 ก็จะมีการทดไปยังหลักถัดไปทางซ้าย รูปแบบการบวกเป็นดังนี้

$$\begin{aligned}
 0 + 0 &= 0 \\
 0 + 1 &= 1 \\
 1 + 0 &= 1 \\
 1 + 1 &= 0 \quad \text{ทด } 1
 \end{aligned}$$

ตัวอย่าง จงบวกเลข 1011.101_2 กับ 110.011_2

$$\begin{array}{r}
 1011.101 \\
 + 110.011 \\
 \hline
 10010.000
 \end{array}$$

การลบเลขเป็นการดำเนินการที่ผกผันกับการบวก ในการลบ ถ้ามีการลบเลขที่มากกว่าจากเลขที่น้อยกว่า ต้องมีการขอยืมจากเลขในหลักถัดไปทางซ้ายมา 1 รูปแบบการลบเป็นดังนี้

$$\begin{aligned}
 0 - 0 &= 0 \\
 0 - 1 &= 1 \quad \text{ขอยืม } 1 \\
 1 - 0 &= 1 \\
 1 - 1 &= 0
 \end{aligned}$$

ตัวอย่าง จงลบเลข 1001.11 กับ 101.1

$$\begin{array}{r}
 1001.11 \\
 - 101.10 \\
 \hline
 100.01
 \end{array}$$

3.4 การคูณและการหารเลขฐานสอง

การคูณและการหารของเลขฐานสอง ก็มีหลักการเช่นเดียวกับเลขฐานสิบ เพียงแต่มีสูตรคูณแค่แม่ 0 กับ 1 เท่านั้น อีกทั้งการหารด้วยศูนย์ก็ไม่มี ความหมายเช่นเดียวกับการหารในระบบเลขฐานสิบ ตารางการคูณและการหารของระบบเลขฐานสองคือ

$$\begin{aligned}
 0 \times 0 &= 0 \\
 0 \times 1 &= 0 \\
 1 \times 0 &= 0 \\
 1 \times 1 &= 1 \\
 0 \div 1 &= 0 \\
 1 \div 1 &= 1
 \end{aligned}$$

ตัวอย่าง จงคูณเลขฐานสอง 1.01×10.1

$$\begin{array}{r} 1.01 \\ \times 10.10 \\ \hline 101 \\ 000 \\ 101 \\ \hline 11.001 \end{array}$$

จงหารเลขฐานสอง $11001 \div 101$

$$\begin{array}{r} 101 \overline{)11001} \\ \underline{101} \\ 101 \\ \underline{101} \\ 101 \\ \underline{101} \\ 0000 \end{array}$$

3.5 เลขฐานแปดและเลขฐานสิบหก

ถึงแม้ว่าระบบคอมพิวเตอร์จะเข้าใจแต่ระบบเลขฐานสองเพียงอย่างเดียว แต่ในทางปฏิบัติจะเกิดปัญหาให้กับผู้ใช้งานคอมพิวเตอร์เป็นอย่างมาก เนื่องจากระบบเลขฐานสองมีจำนวนเลขโดดน้อยจึงต้องมีจำนวนหลักมากขึ้นเพื่อแทนค่าตัวเลขต่าง ๆ ทำให้จดจำได้ยาก จึงมีความพยายามในการรวมหลักของเลขฐานสองหลาย ๆ หลักเข้าด้วยกันเป็นเลขฐานที่ใหญ่ขึ้น เพื่อให้ง่ายต่อการจดจำ ซึ่งการแปลงเลขฐานที่ได้นี้กลับเป็นเลขฐานสองจะทำได้อย่างง่ายดายเนื่องจากแต่ละหลักของเลขฐานดังกล่าวแทนเลขฐานสองที่มีจำนวนหลักแน่นอน

โดยปกติแล้วเรามักจะรวมเลขฐานสองจำนวนสามหรือสี่หลักเป็นเลขฐานใหม่ เมื่อเรารวมเลขฐานสอง 3 หลักจะได้เลขที่มี 8 รูปแบบแตกต่างกันคือ 000, 001, 010, 011, 100, 101, 110, และ 111 ซึ่งเราสามารถใช้ตัวเลขโดด 0 ถึง 7 แทนเลขฐานสองในแต่ละแบบได้ ซึ่งระบบเลขจำนวนที่มีตัวเลขโดด 8 ตัวก็คือเลขฐานแปดนั่นเอง โดยที่เลขโดดแต่ละเลขของระบบเลขฐานแปดจะแทนรูปแบบของเลขฐานสองจำนวนสามหลักที่มีค่าเท่ากัน

ในการทำงานเดียวกัน ถ้าเรามีเลขฐานสองจำนวนสี่หลัก ก็จะได้เลขฐานสองที่มี 16 รูปแบบแตกต่างกัน ซึ่งสามารถใช้เลขฐานสิบหกแทนแต่ละรูปแบบของเลขฐานสองได้ แต่เนื่องจากเรามีเลขโดดใช้งานกันแค่สิบตัว ดังนั้นจึงมีการนำตัว A – F มาแทนค่าเลขโดดที่มีค่า 10 – 15 แทน ทำให้เลขฐานสิบหกมีเลขโดดคือ 0 – 9 และ A – F

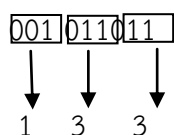
เลขฐานสิบ	เลขฐานสอง	เลขฐานแปด	เลขฐานสิบหกๆ
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4

เลขฐานสิบ	เลขฐานสอง	เลขฐานแปด	เลขฐานสิบหก
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

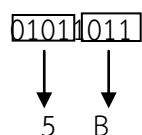
ในการแปลงเลขฐานสองให้เป็นเลขฐานแปดหรือฐานสิบหกนั้น เราจะต้องจับกลุ่มเลขฐานสองให้ได้จำนวนหลักตามที่เลขโดดของฐานที่เราจะแปลงไป เช่น จับกลุ่มสามหลักสำหรับการแปลงเป็นฐานแปด เป็นต้น การจับกลุ่มจะเริ่มจับจากหลักทางด้านขวามือสุดก่อน ในกรณีที่เหลือเลขไม่ครบจำนวนหลักที่ต้องการให้เติมศูนย์ไปทางด้านซ้ายมือเรื่อย ๆ จนกว่าจะได้จำนวนหลักที่ต้องใช้ จากนั้นจึงแปลงฐานเลขโดยแปลงทีละกลุ่มก็จะได้เลขฐานแปดหรือฐานสิบหกตามต้องการ

ตัวอย่าง จงแปลงเลขฐานสอง 1011011 ให้เป็นเลขฐานแปดและฐานสิบหก

แปลงเป็นเลขฐานแปด



แปลงเป็นเลขฐานสิบหก



$$\therefore 1011011_2 = 113_8 = 5B_{16}$$

3.6 บิต, ไบต์, เวิร์ด และดับเบิลเวิร์ด

ข้อมูลต่าง ๆ ในระบบคอมพิวเตอร์มักจะได้ไม่มาจกเลขฐานสองเพียงแค่หลักเดียว เพื่อความสะดวกในการเรียกและความกะทัดรัดของจำนวนที่จะต้องใช้เรียก (คงไม่สะดวกนักถ้าต้องมีการเรียกกันว่า เลขฐานสอง 32 หลัก) ดังนั้นจึงได้มีการตั้งชื่อเฉพาะเรียกกลุ่มของเลขฐานสองที่มีจำนวนหลักตั้งแต่หนึ่งหลักขึ้นไป ดังนี้

เลขฐานสองหนึ่งหลักเรียก	หนึ่ง	“บิต” (bit)
4 บิต เท่ากับ	หนึ่ง	“นิบเบิล” (nibble) [ไม่ค่อยนิยมใช้กันนัก]
2 นิบเบิล เท่ากับ	หนึ่ง	“ไบต์” (byte)
2 ไบต์ เท่ากับ	หนึ่ง	“เวิร์ด” (word)
2 เวิร์ด เท่ากับ	หนึ่ง	“ดับเบิลเวิร์ด” (double word)

3.7 ระบบเลข 2' Complement

การคำนวณเลขฐานสองที่เราคุ้นเคยกัน มักจะเป็นเลขฐานสองที่มีค่าเป็นบวกอยู่เสมอ แต่ในความเป็นจริงเรามีการใช้งานจำนวนทั้งที่เป็นบวกและลบ ดังนั้นระบบคอมพิวเตอร์จึงต้องมีกลไกบางอย่างเพื่อระบุเครื่องหมายของตัวเลขต่าง ๆ ที่อยู่ในระบบ หนึ่งในวิธีการระบุเครื่องหมายของตัวเลขในระบบคอมพิวเตอร์ที่ได้รับความนิยมมากที่สุดก็คือ วิธี 2' Complement

วิธี 2' Complement จะเริ่มจากการกำหนดจำนวนหลักสูงสุดของตัวเลข (จำนวนบิตสูงสุด) กระบวนการทางคณิตศาสตร์ใด ๆ ก็ตาม ถ้าทำให้เกิดการทดเลขเลยบิตซ้ายสุดที่กำหนด เลขทดดังกล่าวจะหายไป เช่น ถ้ากำหนดให้ตัวเลขมีทั้งหมด 4 บิต 1111 บวกกับ 0001 จะเท่ากับ 0000 ไม่ใช่ 10000 เป็นต้น แต่ถ้ามีการขอยืมจากหลักหน้าสุดที่เป็นศูนย์ ให้ถือเสมือนว่ามีหลักที่เป็นค่า 1 อยู่ถัดออกไปจากบิตสูงสุดที่กำหนดไว้ แล้วทำการขอยืมตามปกติ เช่น ถ้าต้องการลบตัวเลขขนาด 4 บิต 0100 ด้วย 0111 จะถือว่าเลข 0100 เสมือนเป็น (1)0100 แล้วจึงลบกันตามปกติ เป็นต้น

ข้อกำหนดข้อที่สองของ 2' Complement คือบิตที่ถูกกำหนดให้เป็นบิตนัยสำคัญสูงสุดจะเป็นบิตที่บอกเครื่องหมายของตัวเลข เช่น กำหนดให้ใช้ตัวเลขขนาด 6 บิต ดังนั้นบิตที่ 5 (เริ่มนับทางขวาสุดเป็นบิตที่ 0) จะเป็นบิตที่ระบุเครื่องหมายของตัวเลขนั้น เป็นต้น โดยการระบุเครื่องหมายจะใช้มาตรฐานว่า “**ถ้าบิตหน้าสุดเป็น 0 หมายถึงเป็นเลขบวก ถ้าบิตหน้าสุดเป็นหนึ่งหมายถึงเป็นเลขลบ**” ดังนั้นจึงไม่สามารถใช้บิตทุกบิตที่กำหนดให้ในการเก็บค่าตัวเลขได้

ข้อกำหนดข้อที่สามของ 2' Complement คือ ค่าตัวเลขที่เก็บอยู่เมื่อบวกกับค่าที่มีเครื่องหมายตรงกันข้ามกันตามแบบวิธีการบวกเลขฐานสองปกติ จะต้องได้ผลลัพธ์เป็นศูนย์ (แต่จะมีบิตทดที่หลุดหายทางซ้ายมือสุด) เช่น ถ้ากำหนดให้เป็นตัวเลขขนาด 4 บิต 1011 จะเท่ากับค่า -5 เพราะถ้าบวกกับ 0101 แล้วจะได้ (1)0000 โดยเลขทดด้านหน้าสุดจะหายไป เป็นต้น

จากข้อกำหนดทั้งหมดข้างต้น ถ้ากำหนดให้ตัวเลขที่ใช้มีขนาด 1 ไบต์ จะสามารถเก็บค่าบวกระหว่าง 0 (00000000) ถึง 127 (01111111) และค่าลบระหว่าง -1 (11111111) ถึง -128 (10000000)

การนำเลข 2' Complement ไปบวกหรือลบกัน จะสามารถทำได้ตามวิธีการปกติ แต่จะผลลัพธ์สุดท้ายจะต้องเป็นไปตามข้อกำหนดของ 2' Complement ทั้งสามข้อ สำหรับการคูณและการหารนั้นจะต้องแยกค่าสัมบูรณ์ (Absolute) ออกมาคูณหารกัน แล้วจึงนำเครื่องหมายมาคิดภายหลัง แต่มีข้อควรระวังคือ ถ้าเป็นการคูณกันระหว่างตัวเลขขนาด 1 ไบต์ด้วยกัน ผลลัพธ์ที่ได้จะต้องใช้ที่เก็บขนาด 2 ไบต์ (ดูตัวอย่างในเรื่องการคูณเลขฐานสอง) ซึ่งมากเกินกว่าที่จะเก็บได้ ดังนั้นถ้าต้องการเก็บผลลัพธ์ไว้เป็นขนาด 1 ไบต์ ก็จะต้องให้ตัวตั้งและตัวคูณมีขนาดไม่เกิน 1 นิบเปิล (4 บิต)

3.8 ระบบเลข BCD (Binary Code Decimal)

ระบบเลข BCD มีชื่อเรียกอีกชื่อหนึ่งว่า Packed Decimal เป็นวิธีการหนึ่งที่จะทำให้การแปลงเลขระหว่างฐานสองกับฐานสิบง่ายขึ้นโดยใช้วิธีเดียวกับการสร้างเลขฐานสิบหก โดยจัดกลุ่มให้เลขฐานสอง 4 บิต เป็นเลขฐานสิบ 1 หลัก แต่จำนวนเลขโดดของเลขฐานสิบมีค่าน้อยกว่ารูปแบบของเลขฐานสองที่เป็นไปได้ใน 4 บิต ดังนั้นจึงมีเลขฐานสองบางรูปแบบที่ไม่ได้ถูกใช้งาน สาเหตุที่ต้องจัดกลุ่มเลขฐานสองให้เป็น 4 บิตแทนที่จะเป็นค่าอื่น ๆ เพราะจำนวนเลขโดดของเลขฐานสองมีค่าเท่ากับ 10 ซึ่งอยู่ระหว่างเลขฐานสอง 3 บิตกับ 4 บิต นั่นเอง

ปกติแล้วเรามักจะไม่นำเลข BCD ไปคำนวณแบบซับซ้อน มักจะนำไปบวกลบกันเท่านั้นซึ่งการบวกลบกันของเลข BCD จะต้องทำทีละ 1 นิบเบิ้ล และต้องปรับค่าผลลัพธ์เพื่อเลี้ยงเลขฐานสองที่ไม่ได้ใช้งานด้วย

3.9 มาตรฐาน IEEE 754

มาตรฐาน IEEE754 เป็นมาตรฐานที่ถูกกำหนดขึ้นโดยหน่วยงานชื่อ IEEE (...) เพื่อเป็นมาตรฐานในการเก็บเลขทศนิยม (Floating Point) ในระบบคอมพิวเตอร์ โดยมาตรฐาน IEEE754 จะเก็บเลขทศนิยมฐานสิบในรูป $(-1 \times S)1.M \times 2^{E-B}$

โดยที่	S	เป็นตัวระบุเครื่องหมายของตัวเลข ถ้าเป็น 0 จะเป็นบวก ถ้าเป็น 1 จะเป็นลบ
	1.M	เป็นฐานของเลขยกกำลัง อยู่ในรูป 1.XXXX
	E	เป็นตัวยกกำลังของสอง ไว้สำหรับระบุตำแหน่งของทวินิยม
	B	เป็นค่า BIAS เพื่อให้ค่าของ E ไม่ติดลบ จะมีค่าคงที่สำหรับทุกหมายเลข (ค่าของ B จะถูกกำหนดมาแล้วคือ 127 สำหรับ IEEE754 ขนาด 32 bit)

เช่น 2.5 จะสามารถเขียนได้เป็น $(-1 \times 0)1.010 \times 2^{128-B}$

ในการเก็บค่าต่าง ๆ จะเก็บอยู่ในรูป S-M-E โดย S จะมีขนาด 1 บิต ส่วน M และ E จะมีขนาดแตกต่างกันไปหลายขนาด ขึ้นอยู่กับความละเอียดที่ต้องการ

มาตรฐาน IEEE754 นอกจากจะสามารถแทนค่าเลขทศนิยมปกติแล้ว ยังมีข้อกำหนดที่สามารถใช้แทนสิ่งที่เรียกกันว่า NaN (Not a Number) เช่น ค่าอินฟินิตี้ ด้วย

การคำนวณเลขตามมาตรฐาน IEEE754 มีความซับซ้อนมาก ดังนั้นจึงไม่ขอกล่าวในที่นี้

สรุป

ระบบจำนวนเลขที่ใช้กับคอมพิวเตอร์ จะทำงานบนพื้นฐานของสองสถานะ (Binary) คือเปิดวงจรกับปิดวงจร ซึ่งสามารถแทนสถานะดังกล่าวได้ด้วยตัวเลขโดดสองตัวคือ 0 กับ 1 เราเรียกระบบเลขจำนวนที่ประกอบด้วยตัวเลข 0 กับ 1 นี้ว่า “เลขฐาน 2” ข้อมูลต่าง ๆ ในระบบคอมพิวเตอร์มักจะได้มาจากเลขฐานสองเพียงแค่หลักเดียว เพื่อความสะดวกในการเรียกและความกระชับรัดของจำนวนที่จะต้องใช้เรียก (คงไม่สะดวกนักถ้าต้องมีการเรียกกันว่า เลขฐานสอง 32 หลัก) ดังนั้นจึงได้มีการตั้งชื่อเฉพาะเรียกกลุ่มของเลขฐานสองที่มีจำนวนหลักตั้งแต่หนึ่งหลักขึ้นไป โดยชื่อต่าง ๆ มีดังนี้คือ บิต, ไบต์, เวิร์ด และดับเบิลเวิร์ด

คำถามทบทวน

1. จงเปลี่ยนค่า 123_{10} ให้เป็นเลขฐานสอง
2. จงเปลี่ยนค่า 0.125_{10} ให้เป็นเลขฐานสอง
3. จงคูณเลขฐานสอง 1.101×10.11
4. จงเปลี่ยนค่า CAT_{16} ให้เป็นเลขฐานแปด
5. จงแปลงเลขฐานสอง 110111011 ให้เป็นเลขฐานแปดและฐานสิบหก

แผนบริหารการสอนประจำบทที่ 4

หัวข้อเนื้อหา

- ภาษาต่าง ๆ สำหรับคอมพิวเตอร์
- ระดับของภาษาสำหรับเขียนโปรแกรม
- การแปลภาษาสำหรับคอมพิวเตอร์
- ภาษาแอสเซมบลี

วัตถุประสงค์เชิงพฤติกรรม

- เข้าใจความหมายและประวัติความเป็นมาของภาษาต่าง ๆ สำหรับคอมพิวเตอร์
- รู้และเข้าใจขั้นตอนต่าง ๆ ในการแปลสัญลักษณ์ต่าง ๆ ให้กลายเป็นภาษาเครื่อง

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 4 ภาษาสำหรับเขียนโปรแกรมคอมพิวเตอร์ (Language for Computer Programming)

ดังที่ได้กล่าวมาแล้วว่า คอมพิวเตอร์สามารถทำงานได้กับระบบเลขฐานสองเพียงอย่างเดียว ซึ่งเลขฐานสองดังกล่าว ไม่ได้หมายถึงข้อมูลเท่านั้น แต่ยังหมายถึงคำสั่งต่าง ๆ ของเครื่องคอมพิวเตอร์ด้วย ดังนั้นถ้าเราต้องการเขียนโปรแกรมขึ้นมาสักหนึ่งโปรแกรม เราจะต้องเรียบเรียงคำสั่งต่าง ๆ ให้อยู่ในรูปของลำดับของเลขฐานสองต่าง ๆ ตามแต่ผู้ที่ผลิตเครื่องคอมพิวเตอร์นั้นจะกำหนดไว้ จะเห็นว่าการเขียนโปรแกรมแบบนี้ไม่มีความสะดวกเลย จึงมีผู้คิดค้นภาษาสำหรับเขียนโปรแกรมขึ้นมาเพื่อให้ง่ายในการเขียนโปรแกรม

4.1 ภาษาต่าง ๆ สำหรับคอมพิวเตอร์

ในบรรดาภาษาสำหรับเขียนโปรแกรมนั้น ภาษาสำหรับการเขียนโปรแกรมด้วยเลขฐานสองหรือฐานสิบหก เป็นภาษาที่เก่าแก่ที่สุดซึ่งเครื่องคอมพิวเตอร์สามารถจะเข้าใจและปฏิบัติตามได้โดยทันที เราเรียกภาษาแบบนี้ว่า “ภาษาเครื่อง” ซึ่งภาษาดังกล่าวไม่สื่อความหมายที่ชัดเจนกับมนุษย์ซึ่งเป็นผู้ที่เขียนโปรแกรม จึงมีการคิดค้นสัญลักษณ์ต่าง ๆ ขึ้นมาแทนที่ภาษาเครื่อง โดยสัญลักษณ์ที่คิดขึ้นในตอนแรกจะเป็นการแทนค่าภาษาเครื่องหนึ่งคำสั่งต่อสัญลักษณ์หนึ่งตัว เช่น 00111110 แทนว่า LD A เป็นต้น ซึ่งเรียกภาษาสัญลักษณ์แบบนี้ว่า “ภาษานีโมนิค (Mnemonic)” หรือ “ภาษาแอสเซมบลีพื้นฐาน” แต่ทว่าสัญลักษณ์ที่ได้นี้ ก็ยังยากที่จะใช้เขียนโปรแกรม จึงได้มีการสร้างสัญลักษณ์คำสั่งใหม่ให้มีความง่ายขึ้น โดยเทียบเคียงสัญลักษณ์กับภาษาอังกฤษเพื่อให้สื่อความหมายมากที่สุด จึงเกิดเป็นภาษาสำหรับเขียนโปรแกรมขึ้นมาหลายภาษาขึ้นอยู่กับสัญลักษณ์และรูปแบบที่ใช้ เช่น Fortran, C, Pascal, และ Basic เป็นต้น

4.2 ระดับของภาษาสำหรับเขียนโปรแกรม

เมื่อมีการสร้างสัญลักษณ์แทนภาษาเครื่อง ทำให้มีการแบ่งระดับของภาษาสำหรับเขียนโปรแกรมออกเป็นระดับต่าง ๆ สองระดับคือ ภาษาระดับสูง และภาษาระดับต่ำ

ภาษาระดับสูงเป็นภาษาที่เน้นการสื่อความหมายกับผู้เขียนโปรแกรมเป็นหลัก ทำให้ภาษาในระดับนี้มีความง่ายในการเขียนและทำความเข้าใจ ส่วนภาษาระดับต่ำเน้นที่ความง่ายที่เครื่องคอมพิวเตอร์จะนำคำสั่งต่าง ๆ ไปปฏิบัติซึ่งมีอยู่ด้วยกันแค่สองภาษาคือ ภาษาเครื่องกับภาษาแอสเซมบลี

ในตำราบางเล่มได้กล่าวถึงภาษาระดับกลาง ซึ่งเป็นระดับพิเศษอีกระดับโดยเป็นระดับที่มีความสมดุลกันระหว่างความง่ายในการเขียนกับความง่ายในการนำไปให้เครื่องคอมพิวเตอร์ปฏิบัติ แต่ในทางปฏิบัติจริงเราไม่สามารถหาจุดสมดุลกันจริง ๆ ของคุณสมบัติทั้งสองได้ ดังนั้นจึงไม่สามารถแบ่งแยกระดับกลางออกมาได้อย่างเด็ดขาด ซึ่งภาษาที่ได้รับการกล่าวถึงว่าเป็นภาษาระดับกลางก็คือภาษา C

4.3 การแปลภาษาสำหรับคอมพิวเตอร์

ภาษาสำหรับเขียนโปรแกรมที่อยู่ในรูปของสัญลักษณ์ต่าง ๆ นั้น ไม่สามารถนำไปให้เครื่องคอมพิวเตอร์ปฏิบัติได้ทันที เพราะเครื่องคอมพิวเตอร์เข้าใจแต่ภาษาเครื่องเท่านั้น ดังนั้นเราจะต้องมีขั้นตอนต่าง ๆ ในการแปลสัญลักษณ์ต่าง ๆ ให้กลายเป็นภาษาเครื่อง ซึ่งขั้นตอนการแปลจะมีสองแบบคือ

- **Interpret** เป็นการแปลแบบคำสั่งต่อคำสั่ง โดยดึงสัญลักษณ์ของภาษาระดับสูงมาทีละคำสั่งแล้วแปลคำสั่งนั้น จากนั้นจึงส่งคำสั่งที่แปลได้ไปให้เครื่องคอมพิวเตอร์ทำงานทันที เมื่อคอมพิวเตอร์ทำงานเสร็จแล้วจึง จะเริ่มแปลคำสั่งถัดไปเรื่อย ๆ โดยไม่มีการเก็บภาษาเครื่องของคำสั่งที่แปลไปแล้ว ดังนั้นถ้ามีการวนที่คำสั่งใด ซ้ำแล้วซ้ำอีก ก็จะต้องแปลคำสั่งนั้นใหม่ทุกครั้งที่ทำงานผ่านคำสั่งนั้น ข้อดีของการแปลแบบนี้คือคำสั่งต่าง ๆ จะผ่านขั้นตอนต่าง ๆ อย่างสมบูรณ์ในขั้นตอนการแปลภาษา จึงสามารถพบข้อผิดพลาดได้ง่าย
- **Compile** เป็นกระบวนการแปลสัญลักษณ์ของภาษาระดับสูงเป็นภาษาเครื่องทีเดียวทั้งโปรแกรมแล้วจึงนำผลลัพธ์ที่ได้ทั้งหมดไปให้เครื่องคอมพิวเตอร์ปฏิบัติต่อเนื่องกันไปโดยไม่มีการแปลสัญลักษณ์ใด ๆ อีก ซึ่งภาษาเครื่องที่ได้อาจจะถูกเก็บไว้เพื่อนำกลับมาให้เครื่องคอมพิวเตอร์ทำงานได้อีกโดยที่ไม่ต้องผ่านกระบวนการแปลภาษาอีกครั้ง สัญลักษณ์ของภาษาระดับสูงแต่ละตัวจะถูกแปลเป็นภาษาเครื่องเพียงครั้งเดียว ดังนั้นเมื่อเครื่องคอมพิวเตอร์ทำงานตามโปรแกรมที่เขียนไว้ จึงทำงานด้วยความเร็วสูงกว่าการแปลแบบ Interpret แต่วิธีนี้ก็ยังมีข้อเสียคือ ข้อผิดพลาดบางประการของโปรแกรมจะตรวจพบเมื่อเครื่องคอมพิวเตอร์ทำงานตามโปรแกรมแล้วเท่านั้น ซึ่งเมื่อมีการแก้ไขให้ถูกต้อง ก็จะต้องมีการแปลสัญลักษณ์ทั้งหมดใหม่อีกครั้ง
- **Assembling** จริง ๆ แล้ว Assembling เป็นการ Compile โปรแกรมแบบหนึ่ง แต่ถูกตั้งชื่อใหม่ตามชื่อของภาษาที่ถูกนำมาแปลซึ่งก็คือภาษาแอสเซมบลี การทำงานของ Assembling มักจะอยู่ในรูปการแทนค่าสัญลักษณ์ต่าง ๆ โดยเทียบกับตาราง เพราะภาษาแอสเซมบลีเป็นการแทนค่าภาษาเครื่องในแบบคำสั่งต่อคำสั่งนั่นเอง

4.4 ภาษาแอสเซมบลี

ภาษาแอสเซมบลีจริง ๆ แล้วก็คือภาษานีโนนิคที่มีการเพิ่มเติมคำสั่งพิเศษในการจัดการเรื่องการอ้างอิงหน่วยความจำนั่นเอง ภาษาแอสเซมบลีจัดเป็นภาษาระดับต่ำภาษาหนึ่ง ซึ่งสัญลักษณ์ที่ใช้มักจะถูกกำหนดโดยผู้ผลิต IC หน่วยประมวลผลกลาง ดังนั้นเมื่อมีการเปลี่ยนหน่วยประมวลผลกลาง ภาษาแอสเซมบลีที่ใช้ ก็จะต้องเปลี่ยนตามไปด้วย เมื่อเราพัฒนาโปรแกรมเป็นภาษาแอสเซมบลีแล้ว ก็ต้องนำไป Assembling ด้วยตัวแปลภาษาชื่อ Assembler ซึ่งจะแปลภาษาแอสเซมบลีให้เป็นภาษาเครื่องอีกทีหนึ่ง

ข้อดี ของการใช้ภาษาแอสเซมบลีคือ

- สามารถสั่งงานระบบคอมพิวเตอร์ได้โดยตรง
- สามารถเขียนให้ได้โปรแกรมภาษาเครื่องที่มีขนาดเล็กที่สุดได้
- โปรแกรมภาษาเครื่องที่ได้จะทำงานได้รวดเร็ว

ข้อเสีย ของการใช้ภาษาแอสเซมบลีคือ

- เป็นภาษาระดับต่ำทำให้พัฒนาโปรแกรมได้ยาก
- เหมาะกับการพัฒนาโปรแกรมที่ไม่ใหญ่มาก
- ไม่มีโครงสร้างข้อมูลในระดับสูง
- โปรแกรมที่พัฒนาขึ้นจะไม่สามารถนำไปใช้กับหน่วยประมวลผลกลางแบบอื่นได้

สรุป

ภาษาสำหรับการเขียนโปรแกรมด้วยเลขฐานสองหรือฐานสิบหก เป็นภาษาที่เก่าแก่ที่สุดซึ่งเครื่องคอมพิวเตอร์สามารถจะเข้าใจและปฏิบัติตามได้เลยทันที เราเรียกภาษาแบบนี้ว่า “ภาษาเครื่อง” ซึ่งภาษาดังกล่าวไม่มีสื่อความหมายที่ชัดเจนกับมนุษย์ซึ่งเป็นผู้ที่เขียนโปรแกรม จึงมีการคิดค้นสัญลักษณ์ต่าง ๆ ขึ้นมาแทนที่ภาษาเครื่อง โดยสัญลักษณ์ที่คิดขึ้นในตอนแรกจะเป็นการแทนค่าภาษาเครื่องหนึ่งคำสั่งต่อสัญลักษณ์หนึ่งตัว เช่น 00111110 แทนว่า LD A เป็นต้น ซึ่งเรียกภาษาสัญลักษณ์แบบนี้ว่า “ภาษานี้มนิก (Mnemonic)” หรือ “ภาษาแอสเซมบลีพื้นฐาน” แต่ทว่าสัญลักษณ์ที่ได้นี้ ก็ยังยากที่จะใช้เขียนโปรแกรม จึงได้มีการสร้างสัญลักษณ์คำสั่งใหม่ให้มีความง่ายขึ้น โดยเทียบเคียงสัญลักษณ์กับภาษาอังกฤษเพื่อให้สื่อความหมายมากที่สุด จึงเกิดเป็นภาษาสำหรับเขียนโปรแกรมขึ้นมาหลายภาษาขึ้นอยู่กับสัญลักษณ์และรูปแบบที่ใช้ เช่น Fortran, C, Pascal, และ Basic เป็นต้น และในปัจจุบันก็มีการพัฒนาภาษาโปรแกรมขึ้นมาอีกมากมายเพื่อให้เหมาะสมกับการใช้งานในด้านต่างๆ เช่น Java, C++, PHP, Perl, Visual Basic, C#, Python, Delphi/Kylix เป็นต้น

คำถามทบทวน

1. ภาษานี้มนิก (Mnemonic) เรียกอีกอย่างหนึ่งว่าภาษาอะไรและมีความสำคัญอย่างไรในการเขียนภาษาโปรแกรมในยุคแรกๆ
2. การสร้างสัญลักษณ์แทนภาษาเครื่อง ทำให้มีการแบ่งระดับของภาษาสำหรับเขียนโปรแกรมออกเป็นกี่ระดับ อะไรบ้าง
3. Interpret และ Compile แตกต่างกันอย่างใดจงอธิบาย
4. จงอธิบายข้อดีและข้อเสียข้อดีของการใช้ภาษาแอสเซมบลี

แผนบริหารการสอนประจำบทที่ 5

หัวข้อเนื้อหา

- หน่วยประมวลผลกลาง
- หน่วยความจำ
- การเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ
- สถาปัตยกรรมของระบบไมโครโปรเซสเซอร์ตระกูล 80x86

วัตถุประสงค์เชิงพฤติกรรม

- เข้าใจความหมายและขั้นตอนการทำงานของหน่วยประมวลผลกลาง
- รู้และเข้าใจการเก็บข้อมูลในหน่วยความจำ
- รู้และเข้าใจความหมายของการเชื่อมต่อระหว่างอุปกรณ์ต่างๆ เช่น Address Bus, Data Bus และ Control Bus เป็นต้น
- สามารถอธิบายสถาปัตยกรรมของระบบไมโครโปรเซสเซอร์ตระกูล 80x86 ได้

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

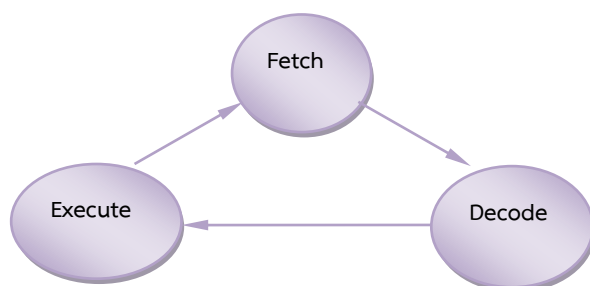
ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 5 สถาปัตยกรรมคอมพิวเตอร์เบื้องต้น (Introduction to computer architecture)

ระบบคอมพิวเตอร์ประกอบด้วยหน่วยประมวลผลกลาง (CPU) หน่วยความจำ (Memory) อุปกรณ์รับข้อมูล (Input Units) และอุปกรณ์แสดงผล (Output Unit) ในบทนี้เราจะเริ่มพิจารณาการเชื่อมโยงของอุปกรณ์ต่าง ๆ และโครงสร้างภายในขององค์ประกอบเหล่านั้น

5.1 หน่วยประมวลผลกลาง

หน่วยประมวลผลกลางมีหน้าที่ประมวลผลข้อมูลต่าง ๆ ในระบบคอมพิวเตอร์ โดยหน่วยประมวลผลกลางจะทำงานตามโปรแกรมที่ระบุโดยผู้ใช้ ขั้นตอนการทำงานของหน่วยประมวลผลกลางมีลักษณะเป็นวงรอบ โดยขั้นแรกหน่วยประมวลผลกลางจะอ่านคำสั่งจากหน่วยความจำ (fetch) จากนั้นหน่วยประมวลผลกลางจะตีความคำสั่งนั้น (decode) และในขั้นตอนสุดท้ายหน่วยประมวลผลกลางก็จะประมวลผลตามคำสั่งที่อ่านเข้ามา (execute) เมื่อทำงานเสร็จหน่วยประมวลผลก็จะเริ่มอ่านคำสั่งเข้ามาอีกครั้ง ขั้นตอนดังกล่าวมีลักษณะแสดงได้ดังรูปที่ 5.1



รูป 5.1 แสดงขั้นตอนการทำงานของหน่วยประมวลผล

หน่วยประมวลผลกลางจะทำงานตามชุดคำสั่ง (instructions) ที่อ่านขึ้นมาจากหน่วยความจำหลักเท่านั้น เราจะเรียกสถาปัตยกรรมของระบบคอมพิวเตอร์ที่มีการเก็บโปรแกรมและข้อมูลไว้ในหน่วยความจำหลัก โดยที่หน่วยประมวลผลจะทำงานกับหน่วยความจำเท่านั้นว่า Stored Program Architecture หรือคอมพิวเตอร์แบบวอนนอยแมน (von Neumann Computer) โดยตั้งเป็นเกียรติให้กับ John von Neumann¹

ชุดคำสั่งของคอมพิวเตอร์ โดยทั่วไปจะประกอบด้วยส่วนย่อย ๆ 2 ส่วนคือ ส่วน Opcode ซึ่งเป็นส่วนที่ระบุประเภทของการประมวลผล และส่วน Operand ซึ่งเป็นส่วนที่ระบุข้อมูลสำหรับการประมวลผลตามที่ระบุใน opcode

โดยปกติแล้ว เรานิยมใช้ไมโครโปรเซสเซอร์ทำหน้าที่เป็นหน่วยประมวลผลกลางในระบบคอมพิวเตอร์ ดังนั้นการที่เราอ้างถึงไมโครโปรเซสเซอร์เราจะอ้างถึงในหน้าที่ที่เป็นหน่วยประมวลผลกลาง โดยคำสองคำนี้อาจใช้แทนกันได้

¹ ผู้บุกเบิกเทคโนโลยีคอมพิวเตอร์เชื่อว่าชื่อของคอมพิวเตอร์แบบวอนนอยแมน ให้เกียรติกับ von Neumann ผู้เขียนแนวคิดเกี่ยวกับคอมพิวเตอร์รูปแบบนี้มากไป โดยให้ความสำคัญกับ J.Presper Eckert และ John Mauchly วิศวกรผู้สร้างเครื่องคอมพิวเตอร์ลักษณะนี้ขึ้นมาโดยเร็ว

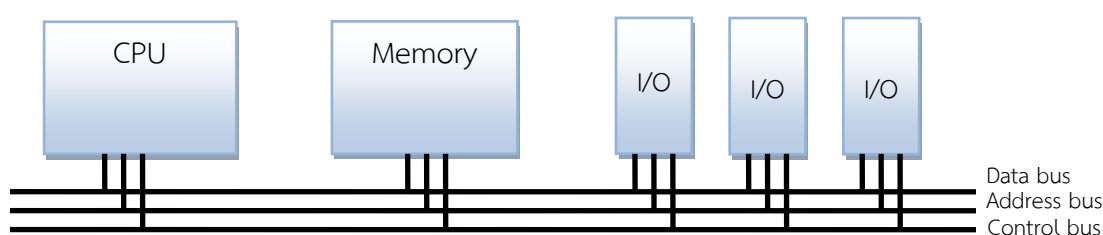
5.2 หน่วยความจำ

การเก็บข้อมูลในหน่วยความจำมีหน่วยที่เล็กที่สุดในการเก็บข้อมูลคือบิต แต่ในการเรียกข้อมูลจากหน่วยความจำนั้นจะกระทำในรูปของข้อมูลที่มีขนาดใหญ่กว่า คือจะมีขนาด 8 บิต หรือ 1 ไบท์ ภายในหน่วยความจำของระบบคอมพิวเตอร์หนึ่ง ๆ จะประกอบด้วยหน่วยย่อย ๆ ขนาด 8 บิตเหล่านี้อยู่มากมาย หน่วยย่อย ๆ เหล่านี้จะมีหมายเลขเฉพาะตัว เพื่อให้หน่วยประมวลผลสามารถใช้อ้างถึงเมื่อจะอ่าน หรือเขียนข้อมูลลงไปหน่วยย่อยหน่วยนั้น หมายเลขนี้เราเรียกว่า **แอดเดรส (Address)** ดังนั้นการที่หน่วยประมวลผลจะอ้างถึงข้อมูลข้อมูลหนึ่งที่เก็บอยู่ในหน่วยความจำได้นั้น หน่วยประมวลผลจะต้องระบุแอดเดรสของข้อมูลชิ้นนั้นให้ได้ด้วย

5.3 การเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ

บัส (Bus) : ช่องทางสื่อสาร

อุปกรณ์ต่าง ๆ จะเชื่อมต่อกับโดยผ่านทางกลุ่มของสายสัญญาณ ที่เราเรียกว่า “บัส” อุปกรณ์ต่าง ๆ จะส่งและรับสัญญาณผ่านทางกลุ่มสายสัญญาณชุดเดียวกัน แสดงดังรูปที่ 5.2



รูปที่ 5.2 แสดงการเชื่อมต่อของอุปกรณ์ต่าง ๆ ผ่านระบบบัส

เราสามารถแบ่งกลุ่มของบัสออกเป็น 3 กลุ่ม คือ

- บัสข้อมูล (Data bus) ใช้สำหรับส่งรับข้อมูล
- บัสตำแหน่ง หรือ แอดเดรสบัส (Address bus) ใช้สำหรับส่งรับตำแหน่งสำหรับอ้างอิงข้อมูลจากหน่วยความจำ หรือ จากอุปกรณ์ I/O
- บัสควบคุม (Control bus) ใช้สำหรับส่งสัญญาณควบคุม

แอดเดรสบัส (Address bus)

ในระบบคอมพิวเตอร์ที่หน่วยประมวลผลเชื่อมต่อกับอุปกรณ์อื่น ๆ ผ่านทางบัส ข้อมูลต่าง ๆ ที่ส่ง/รับกันระหว่างอุปกรณ์ต่าง ๆ นั้นจะส่งผ่านทางบัสข้อมูล ดังนั้นการที่หน่วยประมวลผลจะติดต่อกับหน่วยความจำ หรืออุปกรณ์รับและแสดงผลข้อมูลที่ต้องการได้นั้น ทั้งหน่วยความจำ และ อุปกรณ์รับหรือแสดงผลข้อมูลทุกอุปกรณ์ จะต้องหมายเลขเฉพาะ หมายเลขนี้สำหรับหน่วยความจำก็คือแอดเดรส ส่วนอุปกรณ์อินพุตและอุปกรณ์เอาต์พุตก็มีหมายเลขเฉพาะสำหรับอุปกรณ์หนึ่ง ๆ เช่นเดียวกัน โดยเรียกว่า หมายเลข I/O (I/O

address) เมื่อหน่วยประมวลผลต้องการติดต่อกับหน่วยความจำที่ตำแหน่งใด หรือติดต่อกับอุปกรณ์ใดก็จะส่งแอดเดรสของหน่วยความจำนั้น หรือของอุปกรณ์นั้นมา ในการเลือกว่าหมายเลขแอดเดรสที่ส่งมาเป็นของหน่วยความจำหรือของอุปกรณ์อินพุตเอาต์พุต หน่วยประมวลผลจะส่งสัญญาณระบุมาทางสัญญาณในบัสควบคุม

5.4 สถาปัตยกรรมของระบบไมโครโปรเซสเซอร์ตระกูล 80x86

5.5.1 ความเป็นมา

ไมโครโปรเซสเซอร์ตระกูล 80x86 เป็นไมโครโปรเซสเซอร์ที่พัฒนาขึ้นโดยบริษัท Intel โดยมีการพัฒนา มาตั้งแต่รุ่น 4040 ซึ่งเป็นไมโครโปรเซสเซอร์ขนาด 4 บิต จนกระทั่งในปัจจุบันได้พัฒนาเป็นไมโครโปรเซสเซอร์รุ่น Pentium รายละเอียดคร่าว ๆ ของไมโครโปรเซสเซอร์รุ่นต่าง ๆ เป็นดังต่อไปนี้

- 8086 เป็นไมโครโปรเซสเซอร์ขนาด 16 บิต รุ่นแรกที่ Intel ผลิตขึ้น สามารถอ้างหน่วยความจำได้ 1 MB มีรีจิสเตอร์ภายในขนาด 16 บิต
- 8088 เนื่องจากในขณะนั้นอุปกรณ์ต่าง ๆ โดยมากเป็นอุปกรณ์ขนาด 8 บิต บริษัท Intel จึงได้ผลิต ไมโครโปรเซสเซอร์ 8088 ซึ่งมีสถาปัตยกรรมภายในเหมือน 8086 แต่มีการติดต่อกับระบบภายนอก เป็นแบบ 8 บิต
- 80186 เป็นหน่วยประมวลผลซึ่งเพิ่มการจัดการเกี่ยวกับอุปกรณ์รอบข้างเข้าไป เพื่อให้เป็นหน่วยประมวลผลสำหรับงานควบคุมต่าง ๆ
- 80286 เป็นหน่วยประมวลผลขนาด 16 บิตที่มีความเร็วในการทำงานสูงขึ้น ขยายขอบเขตการอ้าง หน่วยความจำเป็น 24 เมกะไบต์ และมีการเพิ่มความสามารถในการจัดการหน่วยความจำ
- 80386 เป็นหน่วยประมวลผลขนาด 32 บิต อ้างหน่วยความจำได้ถึง 4 กิกะไบต์ มีความสามารถในการจัดการหน่วยความจำที่ซับซ้อน และสามารถใช้หน่วยความจำแบบเสมือนได้ มาตรฐานของ ชุดคำสั่งและกรรมวิธีในการจัดการหน่วยความจำของหน่วยประมวลผลรุ่นนี้ยังคงใช้เป็นมาตรฐานอยู่ จนถึงปัจจุบันนี้ ยกตัวอย่างเช่น แม้แต่ระบบปฏิบัติการ Windows 95 ยังสามารถนำมาทำงานได้บน หน่วยประมวลผลรุ่นนี้ แต่จะทำงานได้ช้ามากเท่านั้นเอง
- 80386SX เป็นหน่วยประมวลผลซึ่งมีสถาปัตยกรรมภายในเหมือน 80386 แต่มีระบบบัสภายนอก เป็น 16 บิต หน่วยประมวลผลรุ่นนี้ได้รับการออกแบบขึ้นด้วยสาเหตุคล้ายกับ 8088
- 80486 ได้รับการพัฒนาต่อจากรุ่น 80386 โดยทางบริษัท Intel ได้มีการเพิ่มหน่วยประมวลผลเลข ทศนิยมเข้าไป
- 80486SX เป็นหน่วยประมวลผลรุ่น 80486 ซึ่งตัดความสามารถในส่วนของการประมวลผลเลข ทศนิยมออก
- Pentium, Pentium Pro, Pentium II เป็นหน่วยประมวลผลรุ่นล่าสุดของบริษัท Intel มีการเพิ่ม ความสามารถในการประมวลผลให้สูงขึ้น โดยใช้เทคโนโลยีต่าง ๆ เช่นการประมวลผลแบบไปป์ไลน์ การประมวลผลแบบซูเปอร์สเกลาร์ เป็นต้น

นอกจากไมโครโปรเซสเซอร์ตระกูล 80x86 แล้ว ยังมีหน่วยประมวลผลกลางที่ผลิตและพัฒนาโดยบริษัท ต่าง ๆ อีกหลายตระกูลเช่น ตระกูล 68000 และ PowerPC ของบริษัทโมโตโรล่าตระกูล Alpha ของบริษัท DEC และตระกูล Sparc ของบริษัท Sun Microsystem เป็นต้น

ในการศึกษาบทนี้จะศึกษาเกี่ยวกับสถาปัตยกรรมของระบบไมโครโปรเซสเซอร์ 8086 เท่านั้น

5.5.2 ลักษณะทั่วไปของไมโครโปรเซสเซอร์ 8086

ระบบบัส

ไมโครโปรเซสเซอร์ 8086 มีแอดเดรสบัสขนาด 20 บิต ทำให้สามารถอ้างแอดเดรสได้ 1 เมกะไบต์ และมีบัสข้อมูลขนาด 16 บิต ซึ่งทำให้การอ่านและเขียนข้อมูลทำได้ครั้งละ 2 ไบต์ หน่วยประมวลผลทางคณิตศาสตร์และตรรกศาสตร์ภายใน 8086 สามารถประมวลผลได้กับข้อมูลขนาด 16 บิต รีจิสเตอร์ภายในไมโครโปรเซสเซอร์ 8086 มีขนาด 16 บิต

การจัดการหน่วยความจำ

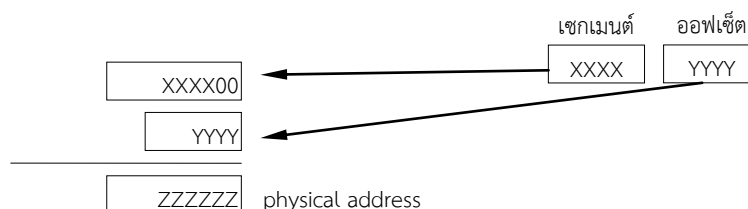
ภายในหน่วยประมวลผล 8086 มีรีจิสเตอร์ขนาด 16 บิต แต่มีแอดเดรสบัสขนาด 20 บิต ด้วยสาเหตุดังกล่าวหน่วยประมวลผลจะไม่สามารถเก็บตำแหน่งข้อมูลภายในหน่วยความจำได้ภายในรีจิสเตอร์เพียงตัวเดียว ดังนั้นการจัดเก็บตำแหน่งของข้อมูลภายในหน่วยความจำใน 8086 จึงต้องเก็บด้วยรีจิสเตอร์ 2 ตัว โดยมีวิธีการจัดเก็บแบบ **เซกเมนต์ : ออฟเซต (segment : offset)** แอดเดรสที่แท้จริง (physical address) ขนาด 20 บิต จะถูกจัดเก็บด้วยรีจิสเตอร์ขนาด 16 บิต 2 ตัว ค่าที่เก็บในรีจิสเตอร์ตัวแรกเรียกว่าเซกเมนต์ (segment) ส่วนค่าที่เก็บในรีจิสเตอร์อีกตัวเรียกว่าออฟเซต (offset)

การอ้างถึงตำแหน่งภายในหน่วยความจำแบบเซกเมนต์ : ออฟเซตนั้น อาจเปรียบได้เสมือนกับการที่เราแบ่งหน่วยความจำเป็นส่วนย่อย ๆ โดยส่วนย่อยนี้เราจะเรียกว่า เซกเมนต์ การที่เราจะอ้างถึงตำแหน่งใด ๆ เราจะทำอย่างไรถึงเซกเมนต์ที่ตำแหน่งข้อมูลนั้นอยู่ และระยะห่างของหน่วยความจำที่คิดเทียบกับจุดเริ่มต้นของเซกเมนต์ที่เราระบุไป

การแปลงแอดเดรสที่เก็บอยู่ในรูปของ เซกเมนต์ : ออฟเซต เป็นแอดเดรสขนาด 20 บิต มีขั้นตอนดังนี้

1. เลื่อนบิตของค่าเซกเมนต์ไปทางซ้าย 4 บิต ดังนั้นจากข้อมูล 16 บิต เราจะได้ข้อมูล 20 บิต ที่มี 4 บิตทางขวาเป็น 0 ในทุกหลัก
2. นำค่าออฟเซตมาบวกเข้ากับค่าเซกเมนต์ที่เลื่อนบิตแล้ว จะได้แอดเดรสขนาด 20 บิต ที่จะนำไปอ้างตำแหน่งที่แท้จริงของข้อมูล

ขั้นตอนทั้งสองแสดงได้ดังรูปที่ 5.3



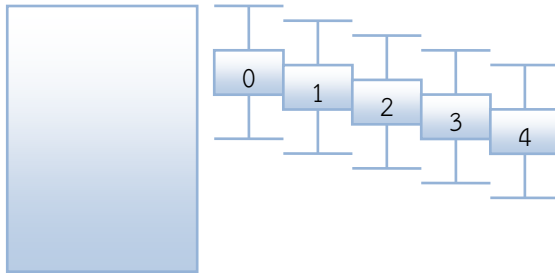
รูปที่ 5.3 แสดงการแปลงแอดเดรสจากแบบเซกเมนต์ : ออฟเซต เป็น แอดเดรสขนาด 20 บิต

ตัวอย่าง เช่น 123Ah : 22B6h แปลงเป็นแอดเดรสขนาด 20 บิตได้ดังนี้

เลื่อนบิตของ 123Ah ไป 4 บิต ได้	123A0h
นำ 22B6h มาบวก	22B6h
จะได้แอดเดรสขนาด 20 บิตคือ	14656h

ในทางกลับกันที่ตำแหน่ง แอดเดรส 14656h ก็จะสามารถอ้างแบบ เซกเมนต์ : ออฟเซต ได้เป็น 123Ah : 22B6h เช่นเดียวกัน แต่ ตำแหน่งแอดเดรส 14656h สามารถอ้างแบบ เซกเมนต์ : ออฟเซต ค่าอื่นก็ได้เช่น 1465h : 0006h หรือ 1460h : 0056h และยังอ้างโดยใช้คู่เซกเมนต์ : ออฟเซต คู่อื่น ๆ ได้อีกหลายคู่

ผลจากการใช้การอ้างแอดเดรสแบบ เซกเมนต์ : ออฟเซต ทำให้ลักษณะของหน่วยความจำที่ 8086 มองเห็นจะมีลักษณะเป็นส่วน ๆ ที่อ้างอิงตามค่าของเซกเมนต์ โดยแต่ละส่วนนี้จะมีขนาดส่วนละ 64 กิโลไบต์ ส่วนของหน่วยความจำขนาด 64 กิโลไบต์นี้เรียกว่า เซกเมนต์ การจัดเรียงตัวของเซกเมนต์ต่าง ๆ ในหน่วยความจำจะจัดเรียงเป็นส่วน ๆ ที่มีการเหลื่อมกันแสดงได้ดังรูปที่ 5.4



รูปที่ 5.4 แสดงลักษณะการเหลื่อมกันของเซกเมนต์

เนื่องจากการอ้างถึงข้อมูลใด ๆ ในหน่วยความจำของไมโครโปรเซสเซอร์ 8086 จะต้องอ้างตำแหน่งเป็นคู่เซกเมนต์ : ออฟเซต การอ้างถึงข้อมูลในตำแหน่งต่าง ๆ อาจทำได้ยุ่งยากเพราะต้องมีการระบุทั้งเซกเมนต์และออฟเซต ในไมโครโปรเซสเซอร์ 8086 จึงได้ออกแบบรีจิสเตอร์พิเศษขึ้น 4 ตัวเพื่อใช้เก็บค่าของเซกเมนต์ต่าง ๆ ที่กำลังใช้งานอยู่ในขณะนั้น กลุ่มของรีจิสเตอร์นั้นเรียกว่า **เซกเมนต์รีจิสเตอร์** ซึ่งได้แก่ CS (Code segment) DS (Data segment) ES (Extra segment) และ SS (Stack segment) เซกเมนต์รีจิสเตอร์ทั้ง 4 ตัวนี้จะใช้ประกอบกับค่าออฟเซตต่าง ๆ เพื่อระบุตำแหน่งของ โปรแกรม (code) ข้อมูล (data) และแอสตัก (stack) ส่วนรีจิสเตอร์ ES มีหน้าที่เก็บเซกเมนต์ของข้อมูลที่ใช้ในการส่งงานคำสั่งพิเศษบางประเภท เช่น คำสั่งเกี่ยวกับข้อความ

ดังนั้นเราจะเห็นได้ว่า ถึงแม้ไมโครโปรเซสเซอร์ 8086 จะสามารถมองหน่วยความจำได้รวมถึง 1 เมกะไบต์ แต่โปรแกรมที่ทำงานอยู่จะมองเห็นหน่วยความจำได้พร้อม ๆ กันแค่เพียง 4 เซกเมนต์เท่านั้น นั่นคือ เซกเมนต์ของโปรแกรม เซกเมนต์ของข้อมูล 2 เซกเมนต์ และ เซกเมนต์ของแอสตัก

แอสตัก (Stack)

ภายในหน่วยความจำของระบบไมโครโปรเซสเซอร์ 8086 จะมีหน่วยความจำส่วนหนึ่งที่ถูกกันเนื้อที่ไว้สำหรับเป็น แอสตักโดยเซกเมนต์ของแอสตักจะถูกชี้โดยรีจิสเตอร์ SS ลักษณะเฉพาะของหน่วยความจำแบบแอสตักคือการที่ระบบจะเก็บข้อมูลและอ่านข้อมูลออกไปแบบ เข้าก่อน ออกทีหลัง (First In Last Out : FILO) โดยอาจมองลักษณะเหมือนการวางซ้อนข้อมูลเหมือนซ้อนจาน ข้อมูลที่ถูกนำมาเก็บก่อนจะอยู่ทางด้านล่าง ข้อมูลถัดไป

จะวางซ้อนอยู่ด้านบน ข้อมูลที่อยู่ทางด้านล่างจะไม่สามารถอ่านออกไปได้ถ้ามีข้อมูลอื่นที่เก็บที่หลังและยังไม่ได้
อ่านออกไป ระบบจะใช้แอสติกในการเรียกโปรแกรมย่อย และเราจะศึกษาเกี่ยวกับแอสติกโดยละเอียดอีกครั้งใน
ส่วนของโปรแกรมย่อย

5.5.3 รายละเอียดของส่วนประกอบภายในไมโครโปรเซสเซอร์

หน่วยประมวลผลทางคณิตศาสตร์และตรรกศาสตร์ (ALU)

ไมโครโปรเซสเซอร์ 8086 มี ALU ที่สามารถประมวลผลได้ครั้งละ 16 บิต ด้วยความสามารถในการ
ประมวลผลที่ละ 16 บิตนี้ ทำให้เราเรียกไมโครโปรเซสเซอร์ 8086 ว่าเป็นไมโครโปรเซสเซอร์ขนาด 16 บิต

หน่วยความจำชั่วคราว (รีจิสเตอร์ --- Register)

รีจิสเตอร์ใน 8086 มีทั้งขนาด 16 บิต และ 8 บิต โดยจะแบ่งเป็นกลุ่ม ๆ ได้ดังนี้

1. รีจิสเตอร์สำหรับใช้งานทั่วไป (General-Purpose Registers)

รีจิสเตอร์ในกลุ่มนี้ ผู้เขียนโปรแกรมสามารถนำไปใช้งานได้ตามความต้องการ โดยในกลุ่มนี้จะมี
รีจิสเตอร์ขนาด 16 บิต อยู่ 4 ตัว โดยรีจิสเตอร์ขนาด 16 บิต ทั้ง 4 ตัวจะแบ่งได้เป็นรีจิสเตอร์ขนาด 8 บิตอีก
8 ตัว รีจิสเตอร์ขนาด 16 บิตและคูรีจิสเตอร์ขนาด 8 บิต มีดังต่อไปนี้

1.1. รีจิสเตอร์ AX (Accumulator Register)

AX	
AH	AL

1.2. รีจิสเตอร์ BX (Base Register)

BX	
BH	BL

1.3. รีจิสเตอร์ CX (Counter Register)

CX	
CH	CL

1.4. รีจิสเตอร์ DX (Data Register)

DX	
DH	DL

สำหรับรีจิสเตอร์ทั้ง 4 ตัวนี้ นอกจากจะมีไว้สำหรับใช้งานทั่วไปได้แล้ว ยังมีหน้าที่เฉพาะอื่น ๆ
อีก ซึ่งจะกล่าวถึงในบทต่อ ๆ ไป

2. รีจิสเตอร์สำหรับอ้างอิง (Index Register)

ไมโครโปรเซสเซอร์ 8086 มีรีจิสเตอร์สำหรับอ้างอิง 2 ตัว คือ SI (Source Index) และ DI
(Destination Index) รีจิสเตอร์ในกลุ่มนี้ใช้สำหรับการอ้างอิงตำแหน่งแบบอ้างอิง และใช้ในคำสั่งที่
เกี่ยวกับข้อความ แต่ผู้ใช้สามารถนำไปใช้งานทั่วไปได้ด้วยเช่นกัน

รีจิสเตอร์สำหรับกรชี้ (Pointer Register) รีจิสเตอร์กลุ่มนี้คือ SP และ BP รีจิสเตอร์ SP ใช้ประกอบกับรีจิสเตอร์ SS มีหน้าที่ชี้ตำแหน่งปัจจุบันของแอสต์ก รีจิสเตอร์ BP ส่วนใหญ่จะใช้เพื่อชี้ตำแหน่งของแอสต์กเช่นเดียวกัน แต่นิยมใช้ในส่วนของการส่งพารามิเตอร์ในโปรแกรมย่อย

3. เซกเมนต์รีจิสเตอร์ (segment register)

เซกเมนต์รีจิสเตอร์ทั้ง 4 ตัวคือ CS DS ES และ SS ใช้ประกอบกับค่าของออฟเซต เพื่อชี้ตำแหน่งของ โปรแกรม ข้อมูลปกติ ข้อมูลพิเศษ และ แอสต์ก ตามลำดับ

4. แฟล็ก (flag)

ไมโครโปรเซสเซอร์ 8086 จะเก็บลักษณะของผลลัพธ์ของการคำนวณทางคณิตศาสตร์ไว้ใน แฟล็ก

5. รีจิสเตอร์อื่น ๆ ของระบบ

นอกเหนือจากรีจิสเตอร์ต่าง ๆ ที่ผู้ใช้สามารถกำหนดและใช้งานได้แล้ว ยังมีรีจิสเตอร์อีกกลุ่มหนึ่งซึ่งผู้เขียนโปรแกรมไม่สามารถเรียกใช้ได้ รีจิสเตอร์ในกลุ่มนี้ เช่น IP ซึ่งเป็นรีจิสเตอร์ที่ใช้ประกอบกับ CS เพื่อชี้ตำแหน่งของคำสั่งที่จะทำงานต่อไปหรือ IR ซึ่งเป็นรีจิสเตอร์ที่เก็บคำสั่งปัจจุบันที่ไมโครโปรเซสเซอร์อ่าน (fetch) ขึ้นมาจากหน่วยความจำ

5.5.4 โหมดการอ้างแอดเดรส

ไมโครโปรเซสเซอร์ 8086 สามารถอ้างถึงข้อมูลได้หลายแบบ โดยวิธีการต่าง ๆ ที่ 8086 อ้างถึงข้อมูลนั้น เรารวมเรียกว่า**โหมดการอ้างแอดเดรส (Addressing mode)** ซึ่งรูปแบบที่ 8086 อ้างถึงข้อมูลนั้นแบ่งเป็นกลุ่ม ๆ ได้ 3 กลุ่มใหญ่ ๆ คือ กลุ่มที่อ้างถึงข้อมูลในรีจิสเตอร์ กลุ่มที่อ้างถึงข้อมูลที่ระบุในคำสั่ง และกลุ่มที่อ้างถึงข้อมูลในหน่วยความจำ เราจะศึกษาเรื่องของโหมดการอ้างแอดเดรสอย่างละเอียดอีกครั้งในบทถัด ๆ ไป

5.5.5 การอินเตอร์รัพท์ (Interrupt)

การอินเตอร์รัพท์ หรือการขัดจังหวะ คือการสั่งให้หน่วยประมวลผลหยุดการทำงานชั่วคราว แล้วกระโดดไปทำงานบางอย่างเพื่อตอบสนองการขัดจังหวะนั้น ตัวอย่างของการขัดจังหวะ เช่น อุปกรณ์บางชิ้นได้รับข้อมูล หรือ ข้อมูลได้รับเขียนเก็บลงในฮาร์ดดิสก์เรียบร้อยแล้ว เป็นต้น เมื่อหน่วยประมวลผลตอบสนองการขัดจังหวะเรียบร้อยแล้ว ก็จะคืนสู่สถานะเดิมและกลับไปประมวลผลงานเก่าที่ประมวลผลค้างไว้ เสมือนไม่มีอะไรเกิดขึ้น การขัดจังหวะนี้มีสองประเภทคือ ซอฟต์แวร์อินเตอร์รัพท์ และ ฮาร์ดแวร์อินเตอร์รัพท์ เรานิยมใช้ซอฟต์แวร์อินเตอร์รัพท์ในการเรียกใช้การบริการต่าง ๆ ของระบบ ส่วนฮาร์ดแวร์อินเตอร์รัพท์จะนิยมใช้ในการแจ้งการเปลี่ยนสถานะของอุปกรณ์อินพุตเอาต์พุต ต่าง ๆ

5.5.6 สถาปัตยกรรมของระบบคอมพิวเตอร์สมัยใหม่

เทคโนโลยีของสถาปัตยกรรมคอมพิวเตอร์สมัยใหม่ได้พัฒนาไปอย่างรวดเร็วมาก เครื่องคอมพิวเตอร์ที่เราใช้ในปัจจุบันมีประสิทธิภาพมากกว่าเครื่องคอมพิวเตอร์ทั่วไปเมื่อ 2-3 ปีก่อนหลายเท่า ทั้งนี้เนื่องจากการวิจัยและสร้างหน่วยประมวลผลกลางและระบบคอมพิวเตอร์ที่มีประสิทธิภาพสูงขึ้นมาก

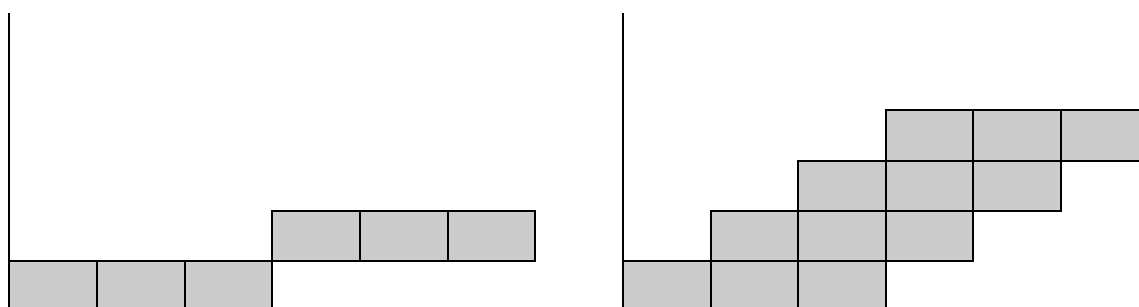
เทคโนโลยีของหน่วยประมวลผลกลาง

● หน่วยประมวลผลแบบ RISC

ชุดคำสั่งของหน่วยประมวลผลยุคเก่ามีลักษณะเป็นแบบ CISC : Complex Instruction Set Computer นั่นคือชุดคำสั่งจะหนึ่ง ๆ จะมีความซับซ้อนมาก การที่ชุดคำสั่งซับซ้อนทำให้การออกแบบส่วนควบคุมภายในหน่วยประมวลผลทำได้ยาก ในปัจจุบันหน่วยประมวลผลต่าง ๆ ได้เปลี่ยนแนวทางในการพัฒนาไปเป็นแบบ RISC : Reduced Instruction Set Computer โดยเน้นชุดคำสั่งที่มีความซับซ้อนน้อยลง แต่มีความเร็วในการทำงานสูงขึ้น การทำให้ชุดคำสั่งมีรูปแบบที่ง่ายขึ้นทำให้การออกแบบส่วนควบคุมทำได้ง่ายขึ้น และยังทำให้สามารถใช้วิธีการแบบไปป์ไลน์ (Pipeline) และซูเปอร์สเกลาร์ (Superscalar) ในการเพิ่มประสิทธิภาพของหน่วยประมวลผลได้ง่ายขึ้นด้วย

● ไปป์ไลน์ (Pipeline)

หน่วยประมวลผลรุ่นใหม่จะมีการประมวลผลแบบไปป์ไลน์ กล่าวคือจะมีการ fetch decode และ execute คำสั่งเหลื่อมกันแสดงได้ดังรูปที่ 5.5 การประมวลผลเหลื่อมกันนี้ทำให้ประสิทธิภาพของการประมวลผลสูงขึ้นมาก



ลำดับการทำงานเมื่อทำงานแบบปกติ

ลำดับการทำงานเมื่อทำงานแบบไปป์ไลน์

รูปที่ 5.5 แสดงการทำงานแบบไปป์ไลน์เทียบกับการทำงานแบบปกติ

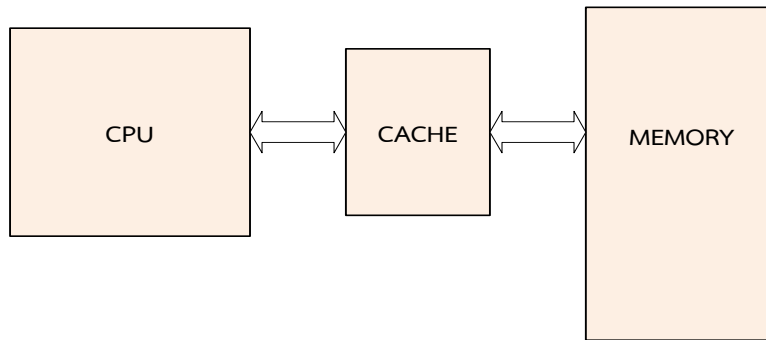
● ซูเปอร์สเกลาร์ (Superscalar)

ในหน่วยประมวลผลที่มีประสิทธิภาพสูงบางรุ่น จะประมวลผลชุดคำสั่งหลายชุดคำสั่งได้พร้อมกัน การที่หน่วยประมวลผลประมวลผลคำสั่งได้หลายชุดพร้อมกันนี้เรียกว่า ซูเปอร์สเกลาร์

ระบบบัตลneck ใหม่

ระบบคอมพิวเตอร์สมัยก่อน หน่วยประมวลผลมีความเร็วในการประมวลผลไม่มากนักทำให้การโอนย้ายข้อมูลระหว่างหน่วยประมวลผลกับหน่วยความจำกระทำได้โดยไม่ก่อให้เกิดการเสียเวลา แต่การพัฒนาของหน่วยประมวลผลเป็นไปอย่างรวดเร็วกว่าการพัฒนาของหน่วยความจำมากทำให้ปัจจุบันอัตราการประมวลผลของหน่วยประมวลผลสูงกว่าอัตราการโอนย้ายข้อมูลระหว่างหน่วยประมวลผลกับหน่วยความจำ สถานการณ์เช่นนี้ก่อให้เกิดปัญหาในรูปแบบคอขวด (Bottleneck problem) ขึ้น นั่นคือจุดเชื่อมต่อระหว่างหน่วยความจำกับหน่วยประมวลผลทำให้ประสิทธิภาพของระบบลดลง เราเรียกคอขวดระหว่างหน่วยประมวลผลกับหน่วยความจำว่า คอขวดของวอนนอยแมน (von Neumann Bottleneck)

วิธีการแก้ปัญหานี้คือการใช้หน่วยความจำที่มีความเร็วสูงมากขนาดเล็กลงมาเป็นบัฟเฟอร์ (ที่พักข้อมูลชั่วคราว) ระหว่างหน่วยความจำและหน่วยประมวลผล หน่วยความจำที่มีความเร็วสูงนี้เรียกว่า **หน่วยความจำแคช (cache memory)** ในหน่วยประมวลผลปัจจุบันหลายรุ่น ได้มีการบรรจุหน่วยความจำแคชลงไปภายในไมโครโปรเซสเซอร์ด้วย ลักษณะของบัสที่มีการใช้หน่วยความจำแคชเป็นแสดงได้ดังรูปที่ 5.6



รูปที่ 5.6 แสดงลักษณะของบัสที่มีการใช้หน่วยความจำแคช

สรุป

สถาปัตยกรรมไมโคร (Micro Architecture) เป็นลักษณะของ เครื่องคอมพิวเตอร์ส่วนใหญ่ที่มีการใช้งานมาตั้งแต่ยุคเริ่มต้น จนถึงยุคปัจจุบัน ได้รับการออกแบบโครงสร้างและการทำงานโดยจอห์นนิวแมน(John Von Neumann) ซึ่งเป็นผู้นำในการออกแบบเครื่องคอมพิวเตอร์ โดยเครื่องคอมพิวเตอร์ ที่เขาได้ออกแบบ มีส่วนประกอบที่สำคัญ 3 อย่างด้วยกันคือ หน่วยประมวลผลกลาง (Central Processing Unit), หน่วยความจำ (Main Memory) และ หน่วยเชื่อมต่ออุปกรณ์ภายนอก (Input/Output) สำหรับไมโครโปรเซสเซอร์ ในตระกูล x86 ออกแบบโดยใช้หลักการเดียวกันกับที่ จอห์นนิวแมนได้กำหนดไว้ กล่าวคือ การประมวลผลทั้งหมด ที่หน่วยประมวลผลกลาง สำหรับข้อมูล ไม่ว่าจะป็นข้อมูลที่ใช้สำหรับประมวลผล หรือข้อมูลที่เป็นคำสั่งก็ตาม จะถูกเก็บไว้ในหน่วยความจำจนกว่าจะมีการเรียกใช้งานจากหน่วยประมวลผลกลาง สำหรับส่วนต่อเชื่อมกับอุปกรณ์ภายนอกนั้น การทำงานของหน่วยประมวลผลกลางก็จะคล้ายกับการทำงานกับหน่วยความจำ ทั้งนี้เพราะทั้งหน่วยความจำและอุปกรณ์ภายนอก เมื่อมีการทำงานกับหน่วยประมวลผลกลางก็มีลักษณะของการเรียกใช้ข้อมูล และนำ ข้อมูลไปเก็บ ต่างกันเพียงสถานที่ และ วิธีการเข้าถึงเท่านั้น บัสของระบบ หน่วยงานต่าง ๆ ภายในเครื่องคอมพิวเตอร์นั้น ต่อเชื่อมเข้าด้วยกันด้วยระบบบัส สำหรับโปรเซสเซอร์ในตระกูล 80x86 มีด้วยกัน 3 กลุ่มอันได้แก่ ดาตาบัส (Data Bus) แอดเดรสบัส (Address Bus) และคอนโทรลบัส (Control Bus) เป็นต้น

คำถามทบทวน

1. ขั้นตอนการทำงานของหน่วยประมวลผลมีทั้งหมดกี่ขั้นตอนอะไรบ้าง
2. เราสามารถแบ่งกลุ่มของบัสออกเป็นกี่กลุ่มอะไรบ้าง จงยกตัวอย่างพร้อมวาดรูปประกอบ
คำอธิบาย
3. จงแสดงวิธีการแปลงแอดเดรส 234Ah : 44B6h เป็นแอดเดรสขนาด 20 บิต
4. โปรเซสเซอร์ในตระกูล 80x86 ประกอบไปด้วยกลุ่มข้อมูลอะไรบ้าง จงยกตัวอย่างพร้อมวาดรูปประกอบ
คำอธิบาย
5. รีจิสเตอร์ใน 8086 มีทั้งขนาด 16 บิต และ 8 บิต แบ่งเป็นกลุ่ม ๆ ได้อย่างไรบ้าง
6. จงอธิบายความแตกต่างระหว่างหน่วยประมวลผลแบบ RISC กับแบบ CISC
7. วิธีการทำงานแบบไปป์ไลน์ (Pipeline) และซูเปอร์สเกลาร์ (Superscalar) แตกต่างกันอย่างไรร

แผนบริหารการสอนประจำบทที่ 6

หัวข้อเนื้อหา

- คำสั่งในการโยนย้ายข้อมูล คำสั่ง MOV
- เครื่องมือในการทดลองการโปรแกรมภาษาแอสเซมบลี
- คำสั่งในการโยนย้ายข้อมูล คำสั่งทั่วไป

วัตถุประสงค์เชิงพฤติกรรม

- เข้าใจความหมายและขั้นตอนการทำงานของคำสั่งในการโยนย้ายข้อมูล
- รู้และเข้าใจ สามารถใช้งานโปรแกรม DEBUG ภาษาแอสเซมบลีได้
- รู้และเข้าใจการใช้งานคำสั่งทั่วไปและการ DEBUG เพื่อตรวจสอบความถูกต้องของคำสั่งโปรแกรมภาษาแอสเซมบลีได้

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 6 คำสั่งโอนย้ายข้อมูล (Command Transfer)

ในบทนี้จะเริ่มศึกษาเกี่ยวกับคำสั่งภาษาแอสเซมบลีเบื้องต้น โดยจะเป็นคำสั่งเกี่ยวกับการโอนย้ายข้อมูล ทั้งจากรีจิสเตอร์และหน่วยความจำ ในตอนท้ายของบทนี้จะเป็นการแนะนำโปรแกรม DEBUG ซึ่งจะเป็นเครื่องมือที่เราจะใช้ทดลองการโปรแกรมภาษาแอสเซมบลีเบื้องต้น

6.1 คำสั่งในการโอนย้ายข้อมูล

คำสั่ง MOV

คำสั่ง MOV เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลจากแหล่งข้อมูลต้นทางไปยังแหล่งข้อมูลปลายทาง โดยมีรูปแบบดังนี้

MOV	reg, reg	reg : register
MOV	reg, mem	mem : memory
MOV	mem, reg	imm : immediate (ค่าคงที่)
MOV	reg, imm	
MOV	mem, imm	

ในการคัดลอกข้อมูลจะกระทำจากโอเปอเรนด์ (operand) ตัวหลังไปยังโอเปอเรนด์ตัวหน้า สังเกตว่า เราไม่สามารถคัดลอกข้อมูลจากหน่วยความจำไปยังหน่วยความจำได้

ข้อจำกัดของคำสั่ง MOV

- * โอเปอเรนด์ทั้งสองตัวต้องมีขนาดเท่ากัน
- * ไม่สามารถคัดลอกค่าคงที่ (immediate) ไปยังเซกเมนต์รีจิสเตอร์ได้โดยตรง ต้องกระทำผ่านทางรีจิสเตอร์อื่น ๆ เช่น AX เป็นต้น
- * ในการคัดลอกค่าคงที่ไปยังหน่วยความจำจะต้องระบุขนาดของหน่วยความจำด้วย

ตัวอย่าง

MOV	AX,100h	กำหนดค่ารีจิสเตอร์ AX ให้เป็น 100h
MOV	BX,AX	จากนั้นคัดลอกค่าจาก AX ไปยัง BX
MOV	DX,BX	และคัดลอกจาก BX ไปยัง DX ตามลำดับ
MOV	AX,1234h	กำหนดค่า 1234h ให้กับ AX
MOV	DX,5678h	และ 5678h ให้กับ DX
MOV	AL,DL	จากนั้น คัดลอกค่าในรีจิสเตอร์ DL (มีค่าเท่ากับ 78h เพราะอะไร?) ไปให้กับรีจิสเตอร์ AL และคัดลอกค่าจาก

MOV	BH,DH	DH (มีค่าเท่ากับ 56h เพราะอะไร?) ไปยังรีจิสเตอร์ DH
MOV	AX,1000h	กำหนดค่า 1000h ให้กับ AX จากนั้นคัดลอกข้อมูลจาก AX ไปยังหน่วยความจำตำแหน่ง offset ที่ 100h และคัดลอกข้อมูลจากหน่วยความจำตำแหน่งนั้นมายัง BX
MOV	[100h],AX	
MOV	BX,[100h]	
MOV	BYTE PTR [200h],10h	กำหนดค่า 10h ไปยังตำแหน่งในหน่วยความจำที่ offset 200h โดยโอนย้ายข้อมูลแบบขนาด 1 Byte และกำหนดค่า 10h ที่มีขนาด 1 word (0010h) ไปยังตำแหน่งในหน่วยความจำที่ offset 300h
MOV	WORD PTR [300h],10h	
MOV	AX,2300h	กำหนดค่า 2300h ให้กับรีจิสเตอร์ DS โดยต้องกำหนดผ่านรีจิสเตอร์ขนาด 16 บิตตัวอื่น (ในที่นี้คือ AX)
MOV	DS,AX	

การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์

การโอนย้ายข้อมูลระหว่างรีจิสเตอร์สามารถทำได้ถ้าขนาดของรีจิสเตอร์ทั้งคู่เท่ากัน แต่ยังมีเซกเมนต์รีจิสเตอร์บางตัวซึ่งไม่ควรเข้าไปกำหนดค่าโดยตรง เช่น CS หรือ SS

คูรีจิสเตอร์ 16 บิต กับ 8 บิต

ในการใช้งานรีจิสเตอร์ทั่วไปเราจะต้องคำนึงถึงเรื่องของคูรีจิสเตอร์ 16 บิต กับ 8 บิตด้วย จากที่เราได้ทราบมาแล้วว่ารีจิสเตอร์ 8 บิตที่เราใช้ได้นั้น เป็นส่วนหนึ่งของรีจิสเตอร์ทั่วไปขนาด 16 บิต เช่น รีจิสเตอร์ AH เป็นไบต์สูงรีจิสเตอร์ AX (บิตที่ 8-15) เป็นต้น ดังนั้นถ้าเรากำหนดค่าให้กับรีจิสเตอร์ 8 บิตก็จะมีผลเปลี่ยนแปลงกระทบถึงรีจิสเตอร์ 16 บิตที่รีจิสเตอร์นั้นประกอบอยู่ด้วย และในทางกลับกัน การเปลี่ยนแปลงค่าในรีจิสเตอร์ 16 บิตก็มีผลกระทบมาถึงรีจิสเตอร์ 8 บิตด้วย

ตัวอย่าง

คำสั่ง	ค่าในรีจิสเตอร์หลังการทำงานของคำสั่ง
MOV AX,1000h	AX = 1000h AH = 10h AX = 00h
MOV AL,3Ah	AX = 103Ah AH = 10h AX = 3Ah
MOV AH,AL	AX = 3A3Ah AH = 3Ah AX = 3Ah
MOV AX,234h	AX = 234h AH = 02h AX = 34h

การโอนย้ายข้อมูลกับหน่วยความจำ

โดยปกติการโอนย้ายข้อมูลกับหน่วยความจำนั้น เราจะระบุเฉพาะออฟเซตของหน่วยความจำที่เราต้องการจะโอนย้ายข้อมูลด้วย ออฟเซตนั้นจะถูกนำมาประกอบกับเซกเมนต์รีจิสเตอร์ DS เพื่อเป็นตำแหน่งในหน่วยความจำที่แท้จริงที่จะอ่านเขียนข้อมูลด้วย

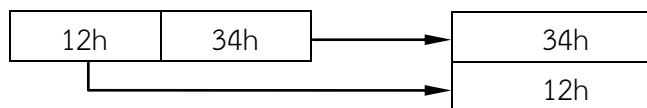
ตัวอย่าง

โปรแกรม	ตำแหน่ง	ค่าในหน่วยความจำหลังการทำงานของคำสั่งที่							
		1,2	3	4	5	6	7	8,9	
MOV AX,1234h	DS:								
MOV BX,6789h	100h	?	12h	12h	12h	12h	12h	12h	12h
MOV [100h],AH	101h	?	?	89h	89h	89h	89h	89h	89h
MOV [101h],BL	102h	?	?	?	89h	89h	89h	89h	89h
MOV [102h],BX	103h	?	?	?	67h	67h	67h	67h	67h
MOV [104h],AX	104h	?	?	?	?	34h	34h	34h	34h
MOV [105h],BH	105h	?	?	?	?	12h	67h	67h	67h
MOV AX,[104h]	106h	?	?	?	?	?	?	?	?
MOV BL,[103h]									

หลังการทำงานของ รีจิสเตอร์ AX มีค่าเท่ากับ 6734h และ BX มีค่าเท่ากับ 3467h

ข้อสังเกตในการจัดเรียงไบต์เมื่อเก็บข้อมูลในหน่วยความจำ

สังเกตว่าในการเก็บค่าในหน่วยความจำเมื่อเราเก็บค่าเป็น 16 บิต การเรียงไบต์ในหน่วยความจำจะเก็บค่าในไบต์ที่มีนัยสำคัญสูงไว้ในไบต์ที่มีแอดเดรสสูงกว่า และไบต์ที่มีนัยสำคัญต่ำไว้ในแอดเดรสที่มีแอดเดรสต่ำกว่า แสดงได้ดังรูปที่ 6.1



รูปที่ 6.1 แสดงการเรียงไบต์ข้อมูลในหน่วยความจำ

ลักษณะของการเก็บข้อมูลโดยเรียงไบต์ที่มีนัยสำคัญต่ำไว้ที่แอดเดรสต่ำและไบต์ที่มีนัยสำคัญสูงไว้ที่แอดเดรสสูงเราเรียกว่าเป็นการเรียงแบบ C

ในการกำหนดออฟเซตของหน่วยความจำที่จะอ่านและเขียนข้อมูลนั้น จากตัวอย่างข้างต้นเราระบุโดยใช้หมายเลขออฟเซตโดยตรง แต่เราสามารถระบุออฟเซตโดยผ่านทางค่าในรีจิสเตอร์ BX (base register) ได้อีกทางหนึ่ง

ตัวอย่าง

```
MOV BX,100h
MOV AX,[BX]
MOV BX,102h
MOV [BX],DX
```

การกำหนดค่าคงที่ให้กับหน่วยความจำ

การกำหนดค่าคงที่ให้กับหน่วยความจำสามารถกระทำได้ แต่จะต้องมีการระบุขนาดของข้อมูลที่จะคัดลอกสู่หน่วยความจำ เพื่อที่จะป้องกันการสับสนดังตัวอย่างเช่น คำสั่ง MOV [100h],10h จากคำสั่งดังกล่าว แอสเซมเบลอร์จะไม่สามารถทราบได้เลยว่าการคัดลอกข้อมูลจะเป็นในลักษณะของ 8 บิต หรือ 16 บิต

ในการระบุขนาดของการคัดลอกข้อมูลเราจะใช้ keyword ว่า BYTE PTR (Byte Pointer) หรือ WORD PTR (Word Pointer) สำหรับระบุว่าคำสั่งอ้างอิงถึงหน่วยความจำในตำแหน่งนั้นเป็นแบบไบต์ หรือ เวิร์ด

ตัวอย่าง

```
MOV  BYTE PTR [100h],10h
MOV  WORD PTR [102h],10h
MOV  BX,104h
MOV  WORD PTR [BX],2345h
```

6.2 เครื่องมือในการทดลองการโปรแกรมภาษาแอสเซมบลี : โปรแกรม DEBUG

โปรแกรม DEBUG เป็นโปรแกรมสารพัดประโยชน์สำหรับการทดลองเกี่ยวกับการเขียนโปรแกรมภาษาแอสเซมบลี โปรแกรมนี้มีอยู่ทั้งในระบบปฏิบัติการ DOS และ Windows

เมื่อเราเรียกโปรแกรม DEBUG โดยพิมพ์ DEBUG ที่ DOS prompt เราจะเป็นเครื่องหมายเตรียมพร้อมของโปรแกรม DEBUG เป็นเครื่องหมายขีด

```
-
```

เราสามารถจะพิมพ์คำสั่งต่าง ๆ ลงไปได้ที่ prompt นี้

6.3 คำสั่งในการโยนย้ายข้อมูล

คำสั่งทั่วไป

คำสั่งแสดงความช่วยเหลือ : ?

เราสามารถสั่งให้โปรแกรม DEBUG แสดงรายการคำสั่งต่าง ๆ ได้โดยใช้คำสั่ง ?

คำสั่งจัดการกับรีจิสเตอร์ : R (register)

คำสั่ง R คือคำสั่งให้โปรแกรม DEBUG แสดงค่าในรีจิสเตอร์รวมถึงกำหนดค่าให้กับรีจิสเตอร์ต่าง ๆ ด้วย ถ้าเราใช้คำสั่ง R โดยไม่ระบุชื่อรีจิสเตอร์ โปรแกรม DEBUG จะแสดงค่าในรีจิสเตอร์ออกมาให้

```
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=12AF ES=12AF SS=12AF CS=12AF IP=0100 NV UP EI PL NZ NA PO NC
12AF:0100 5F          POP     DI
-
```

โปรแกรม DEBUG จะแสดงทั้งค่าในรีจิสเตอร์ คำสั่งถัดไปที่จะทำงาน รวมถึงแฟล็กต่าง ๆ ด้วย เราสามารถระบุชื่อรีจิสเตอร์ต่อท้ายคำสั่ง R เพื่อกำหนดค่าให้กับรีจิสเตอร์นั้น ดังตัวอย่าง

```

-RCX
CX 0000
:100
-R
AX=0000 BX=0000 CX=0100 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=12AF ES=12AF SS=12AF CS=12AF IP=0100 NV UP EI PL NZ NA PO NC
12AF:0100 5F          POP     DI
-

```

คำสั่งแสดงค่าในหน่วยความจำ : D (dump)

เราสามารถสั่งให้โปรแกรม DEBUG แสดงค่าข้อมูลในหน่วยความจำได้ โดยใช้คำสั่ง D (dump) เราสามารถระบุตำแหน่งของหน่วยความจำที่เริ่มต้นแสดงข้อมูลได้ ถ้าเราไม่กำหนดโปรแกรม DEBUG จะแสดงค่าในหน่วยความจำถัดจากตำแหน่งเก่าที่แสดงไว้

```

-D100
12AF:0100  C7 06 16 00 6E 30 26 C7-06 18 00 C7 40 26 8B 06  ....n0&.....@&..
12AF:0110  08 00 26 03 06 0C 00 26-3B 06 06 00 34 00 9E 12  ..&....&;...4...
          ...
12AF:0170  C2 02 00 90 C8 04 00 00-57 56 8B 7E 0A 57 E8 57  .....WV~.W.W
-

```

การทดลองโปรแกรมภาษาแอสเซมบลีใน DEBUG

เราสามารถป้อนโปรแกรมภาษาแอสเซมบลีเบื้องต้นได้ในโปรแกรม DEBUG และสามารถสั่งให้โปรแกรมนั้นทำงานเพื่อสังเกตผลการทำงานได้ เรายังสามารถสั่งให้โปรแกรมทำงานที่ละบรรทัดได้ด้วย

คำสั่งสำหรับแปลโปรแกรมภาษาแอสเซมบลี : A (assemble)

เราใช้คำสั่ง A (assemble) เพื่อสั่งให้โปรแกรม DEBUG รับคำสั่งภาษาแอสเซมบลีแล้วแปลคำสั่งนั้นเป็นภาษาเครื่องเก็บไว้ในหน่วยความจำได้ ในการสั่งคำสั่ง assemble เราสามารถระบุตำแหน่งที่โปรแกรมภาษาเครื่องที่แปลแล้วจะถูกเขียนลงไปได้ ถ้าเราไม่ระบุตำแหน่งลงไปโปรแกรมที่เขียนจะถูกเก็บไว้ที่ตำแหน่งถัดจากการแปลครั้งสุดท้าย ในการป้อนโปรแกรมเราสามารถป้อนโปรแกรมต่อเนื่องไปได้เรื่อย ๆ เมื่อป้อนเสร็จแล้วให้กดปุ่ม Enter วางไปหนึ่งบรรทัดโปรแกรม DEBUG จะแสดง prompt เพื่อรับคำสั่งใหม่ต่อไป

```
A100
```

```

12AF:0100 mov ax,10
12AF:0103 mov bx,20
12AF:0106 mov [200],ax
12AF:0109 mov [202],bx
12AF:010D mov bx,204
12AF:0110 mov cx,1234
12AF:0113 mov [bx],cx
12AF:0115 int 20
12AF:0117
-

```

ข้อสังเกต ตัวเลขต่าง ๆ ในโปรแกรม DEBUG จะถือว่าเป็นเลขฐานสิบหกทั้งหมด ถ้าเราป้อนโปรแกรมผิด โปรแกรม DEBUG จะรายงานว่าโปรแกรมผิดให้เราทราบและป้อนเข้าไปใหม่

คำสั่งสำหรับแสดงโปรแกรมภาษาแอสเซมบลีจากภาษาเครื่อง : U (unassemble)

เราสามารถสั่งให้โปรแกรม DEBUG แสดงคำสั่งภาษาเครื่องที่อยู่ในหน่วยความจำเป็นรหัสคำสั่งภาษาแอสเซมบลีได้โดยใช้คำสั่ง unassemble เราสามารถระบุตำแหน่งที่จะเริ่มแสดงได้

```

-U100
12AF:0100 B81000 MOV AX,0010
12AF:0103 BB2000 MOV BX,0020
12AF:0106 A30002 MOV [0200],AX
12AF:0109 891E0202 MOV [0202],BX
12AF:010D BB0402 MOV BX,0204
12AF:0110 B93412 MOV CX,1234
12AF:0113 890F MOV [BX],CX
12AF:0115 CD20 INT 20
12AF:0117 26 ES:
12AF:0118 3B060600 CMP AX,[0006]
12AF:011C 3400 XOR AL,00
12AF:011E 9E SAHF
12AF:011F 12FE ADC BH,DH
-

```

โปรแกรม DEBUG จะแสดงทั้งรหัสภาษาเครื่องและรหัสนิมอิก (รหัสภาษา assembly) ด้วย

การสั่งให้โปรแกรมทำงาน

เราสามารถสั่งให้โปรแกรมทำงานได้ทั้งสิ้น 3 รูปแบบโดยโปรแกรมจะเริ่มทำงานที่ตำแหน่ง CS:IP เท่านั้น
คำสั่งเริ่มทำงาน : G (go)

เมื่อเราสั่งให้โปรแกรมทำงานโดยใช้คำสั่ง g โปรแกรมจะเริ่มทำงานทันที และโปรแกรมจะทำงานจนกระทั่งจบ

```
-G
Program terminated normally
-
```

ข้อสังเกต โปรแกรมตัวอย่างจะมีคำสั่ง INT 20h ระบุอยู่ตอนท้าย คำสั่งนี้เป็นคำสั่งเรียกใช้บริการของระบบเพื่อที่จะจบโปรแกรม ถ้าไม่มีคำสั่งนี้ปิดท้าย โปรแกรมจะทำงานตามคำสั่งต่อไปเรื่อย ๆ จนกว่าจะพบคำสั่งที่สั่งให้โปรแกรมหยุดการทำงาน ดังนั้นเวลาเราทดลองโปรแกรมใน DEBUG เราควรใส่คำสั่ง INT 20h ปิดท้ายไว้ทุกครั้ง

คำสั่งตามรอยการทำงาน (คำสั่งให้ทำงานทีละบรรทัดแบบที่หนึ่ง) : T (trace)

ถ้าเราสั่งคำสั่งนี้โปรแกรมจะทำงานไปหนึ่งคำสั่งแล้วจะกลับมาที่โปรแกรม DEBUG และแสดงค่าของรีจิสเตอร์ต่าง ๆ รวมถึงผลจากคำสั่งนั้นทันที ทำให้เราสามารถตรวจสอบสถานะของโปรแกรมได้ตลอดขั้นตอนการทำงาน

```
-T
AX=0010 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=12AF ES=12AF SS=12AF CS=12AF IP=0103 NV UP EI PL NZ NA PO NC
12AF:0103 BB2000 MOV BX,0020
-T
AX=0010 BX=0020 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=12AF ES=12AF SS=12AF CS=12AF IP=0106 NV UP EI PL NZ NA PO NC
12AF:0106 A30002 MOV [0200],AX DS:0200=0010
```

ในการทำงานโดยคำสั่ง T โปรแกรมที่ทำงานจะหยุดการทำงานกลับมาที่โปรแกรม DEBUG ภายหลังจากทำงานของทุก ๆ คำสั่ง รวมถึงในกรณีที่เรารู้จักโปรแกรมย่อย หรือเรียกใช้บริการของระบบด้วย เราต้องการให้โปรแกรมทำงานไปหยุดที่บรรทัดถัดไปเลยโดยไม่ต้องคำนึงถึงการเรียกโปรแกรมย่อยต่าง ๆ เราสามารถใช้คำสั่ง P (proceed) แทนคำสั่ง trace ได้

คำสั่งทำงานต่อจนถึงบรรทัดถัดไป (คำสั่งให้ทำงานทีละบรรทัดแบบที่สอง) : P (proceed)

ผลโดยทั่ว ๆ ไปของคำสั่งนี้เหมือนกับการใช้คำสั่ง T แต่การใช้คำสั่งนี้จะสะดวกกว่าเมื่อโปรแกรมที่ทำงานมีการเรียกโปรแกรมย่อย หรือมีการเรียกใช้บริการจากระบบ โดยโปรแกรมจะกลับมาที่ DEBUG เมื่อทำงานถึงบรรทัดถัดไป

การตั้งค่าให้กับรีจิสเตอร์ IP

เนื่องจากโปรแกรม DEBUG จะประมวลผลคำสั่งที่ตำแหน่ง CS:IP เสมอ แต่เมื่อโปรแกรมทำงานเสร็จ รีจิสเตอร์ IP จะชี้ที่ออฟเซตของคำสั่งถัดไปซึ่งเป็นคำสั่งที่ไม่ใช่ส่วนของโปรแกรมของเรา ดังนั้นภายหลังจากทำงานของโปรแกรม เราควรตรวจสอบว่ารีจิสเตอร์ IP ชี้ไปยังจุดเริ่มต้นของโปรแกรมที่เราเขียนไว้หรือไม่ ซึ่งโดยปกติแล้วโปรแกรมจะเริ่มต้นที่ออฟเซต 100h ถ้า รีจิสเตอร์ IP ชี้ไปที่อื่น เราสามารถตั้งค่าให้รีจิสเตอร์ IP ได้โดยใช้คำสั่ง R

```
-RIP
IP 0106
:100
-
```

ตัวอย่างการทดลองโปรแกรมภาษาแอสเซมบลี

เราจะยกตัวอย่างการทดลองโปรแกรมภาษาแอสเซมบลีต่อไปนี้ด้วยโปรแกรม DEBUG

ตัวอย่างโปรแกรม

```
MOV AX,5678h
MOV BX,204h
MOV CX,1234h
MOV [200h],CX
MOV [202h],AH
MOV [203h],AL
MOV [BX],AX
MOV DX,[202h]
MOV [204h],10h
```

ป้อนโปรแกรม

เราสามารถที่จะป้อนโปรแกรมและสั่งให้ DEBUG แปลโปรแกรมลงในหน่วยความจำโดยใช้คำสั่ง A ถ้าเราไม่ระบุแอดเดรสเริ่มต้นในครั้งแรกโปรแกรมจะถูกแปลลงในตำแหน่ง CS:100h และถ้าเป็นการสั่งครั้งถัดไป โปรแกรมจะแปลลงไปต่อเนื่องกับการสั่งครั้งก่อน

```
-A100
14FF:0100 mov ax,5678
14FF:0103 mov bx,204
14FF:0106 mov cx,1234
```

```

14FF:0109 mov [200],cx
14FF:010D mov [202],ah
14FF:0111 mov [203],al
14FF:0114 mov [bx],ax
14FF:0116 mov dx,[202]
14FF:011A mov [204],10
          ^ Error

```

ในระหว่างการป้อนโปรแกรมอาจมีข้อผิดพลาดขึ้นโปรแกรม DEBUG จะแสดงข้อความขึ้นดังตัวอย่างจากโปรแกรมดังกล่าวเราจะต้องระบุขนาดของการคัดลอกข้อมูลด้วย เราสามารถใส่คำสั่งที่แก้ไขแล้วลงไปใหม่ได้ทันที

```

14FF:011A mov word ptr [204],10
14FF:0120

```

เมื่อใส่โปรแกรมเสร็จจะต้องกดปุ่ม Enter ว่าง ๆ ไปเพื่อบอกโปรแกรม DEBUG ว่าโปรแกรมสิ้นสุดแล้ว เราสามารถเรียกดูโปรแกรมที่เราใส่ลงไปได้ด้วยคำสั่ง U

```

-U
14FF:0100 B87856      MOV     AX,5678
14FF:0103 BB0402      MOV     BX,0204
14FF:0106 B93412      MOV     CX,1234
14FF:0109 890E0002    MOV     [0200],CX
14FF:010D 88260202    MOV     [0202],AH
14FF:0111 A20302      MOV     [0203],AL
14FF:0114 8907      MOV     [BX],AX
14FF:0116 8B160202    MOV     DX,[0202]
14FF:011A C70604021000    MOV     WORD PTR [0204],0010
-

```

เราจะเห็นทั้งโปรแกรมภาษาแอสเซมบลีและโปรแกรมภาษาเครื่องที่แปลแล้ว

ติดตามการทำงานของโปรแกรม

เราสามารถสั่งให้โปรแกรมทำงานโดยใช้คำสั่ง G (go) และตรวจสอบค่าในหน่วยความจำได้ แต่โปรแกรมที่จะทดลองนั้นต้องมีการสั่งให้โปรแกรมจบการทำงาน มิฉะนั้นโปรแกรมจะทำงานเลยไปถึงโปรแกรมอื่น ๆ ที่อยู่ในหน่วยความจำ ดังนั้นเราอาจจะใส่คำสั่ง INT 20h ปิดท้ายโปรแกรมไว้เพื่อสั่งให้โปรแกรมจบการทำงาน จากตัวอย่างแอดเดรสถัดไปของคำสั่งคือออฟเซตที่ 0120h (สามารถหาได้โดยใช้คำสั่ง U แล้วสั่งเกออฟเซตของตคำสั่งอื่น ๆ ถัดจากโปรแกรมที่เราป้อน) ดังนั้นเราจะป้อนคำสั่ง INT 20h ลงไปที่แอดเดรสนี้

```
-A120
```

```

14FF:0120 int 20
14FF:0122
-U100
14FF:0100 B87856      MOV    AX,5678
14FF:0103 BB0402      MOV    BX,0204
14FF:011A C70604021000    MOV    WORD PTR [0204],0010
-U11A
14FF:011A C70604021000    MOV    WORD PTR [0204],0010
14FF:0120 CD20        INT    20
...

```

เมื่อเราใส่คำสั่งให้โปรแกรมจบการทำงานแล้ว เราสามารถที่ใช้คำสั่ง G (go) เพื่อสั่งให้โปรแกรมทำงานได้ และใช้คำสั่ง D (dump) เพื่อสังเกตค่าในหน่วยความจำ หรือ คำสั่ง R (register) เพื่อสังเกตค่าในรีจิสเตอร์ได้

```

-G
Program terminated normally
-D200
14FF:0200 34 12 56 78 10 00 DA EB-04 9D F8 EB 02 9D F9 89 4.Vx.....
14FF:0210 36 8A DB 5E 5F 5A 59 C3-53 51 52 57 56 9C E8 6E 6..^_ZY.SQRWW..n
14FF:0220 00 83 3E 7A DB 40 7D 57-8A F7 8B 1E 7A DB FF 06 ..>z.@}W....z...
14FF:0230 7A DB B8 BA D8 E8 95 00-C7 47 07 00 00 F6 C6 01 z.....G.....
14FF:0240 74 03 89 6F 07 89 4F 05-88 77 02 8B 36 84 DB 89 t..o..O..w..6...
14FF:0250 37 03 36 5C D8 2B F7 89-77 03 8B 36 8A DB 89 77 7.6\..+..w..6...w
14FF:0260 09 8B F7 8B 3E 84 DB 03-F9 3B 3E 80 DB 7D 15 2B ....>....;>..}.+
14FF:0270 F9 FC F3 A4 B0 00 AA 89-3E 84 DB 9D F8 EB 0A B8 .....>.....
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=14FF IP=0100 NV UP EI PL NZ NA PO NC
14FF:0100 B87856      MOV    AX,5678

```

สังเกตว่าเมื่อเรากลับมายังโปรแกรม DEBUG หลังจากโปรแกรมที่เราป้อนทำงานเรียบร้อยแล้ว รีจิสเตอร์ต่าง ๆ จะถูกกำหนดค่าเริ่มต้นใหม่รวมทั้งรีจิสเตอร์ IP ด้วย ดังนั้นเราสามารถสังเกตผลของการทำงานได้จากค่าหน่วยความจำเท่านั้น เราสามารถสั่งให้โปรแกรมทำงานที่ละบรรทัดโดยใช้คำสั่ง T หรือ P เราจะต้องระวังในเรื่องของการตั้งค่าเริ่มต้นของรีจิสเตอร์ IP ให้มาชี้ที่ตำแหน่งเริ่มต้นของโปรแกรมด้วย

-T

```

AX=5678 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=14FF IP=0103 NV UP EI PL NZ NA PO NC
14FF:0103 BB0402      MOV    BX,0204
-T
AX=5678 BX=0204 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=14FF IP=0106 NV UP EI PL NZ NA PO NC
14FF:0106 B193412     MOV    CX,1234
-T
AX=5678 BX=0204 CX=1234 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=14FF IP=0109 NV UP EI PL NZ NA PO NC
14FF:0109 890E0002    MOV    [0200],CX          DS:0200=1234
-T
AX=5678 BX=0204 CX=1234 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=14FF IP=010D NV UP EI PL NZ NA PO NC
14FF:010D 88260202    MOV    [0202],AH          DS:0202=56
-T
AX=5678 BX=0204 CX=1234 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=14FF IP=0111 NV UP EI PL NZ NA PO NC
14FF:0111 A20302      MOV    [0203],AL          DS:0203=78
-T
AX=5678 BX=0204 CX=1234 DX=7856 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=14FF IP=0120 NV UP EI PL NZ NA PO NC
14FF:0120 CD20        INT    20

```

-P

Program terminated normally

-

สังเกตว่าในคำสั่งสุดท้ายเราใช้คำสั่ง P เพราะคำสั่ง INT 20h เป็นการเรียกใช้บริการของระบบ และโปรแกรมจะกระโดดไปทำงานที่โปรแกรมน้อยของระบบซึ่งอาจจะยาวมาก ถ้าเราใช้คำสั่ง T เราจะได้เห็นการกระโดดไปทำงานนี้ และเราสามารถใช้คำสั่ง G เพื่อให้โปรแกรมทำงานจนจบต่อจากนั้นก็ได้อีก

-T

```

AX=5678 BX=0204 CX=1234 DX=7856 SP=FFE8 BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=14FF IP=0120 NV UP DI PL NZ NA PO NC
14FF:0120 CD20 INT 20
-T
AX=5678 BX=0204 CX=1234 DX=7856 SP=FFE2 BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=00C9 IP=0FA8 NV UP DI PL NZ NA PO NC
00C9:0FA8 90 NOP
-T
AX=5678 BX=0204 CX=1234 DX=7856 SP=FFE2 BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=00C9 IP=0FA9 NV UP DI PL NZ NA PO NC
00C9:0FA9 90 NOP
-T
AX=5678 BX=0204 CX=1234 DX=7856 SP=FFE2 BP=0000 SI=0000 DI=0000
DS=14FF ES=14FF SS=14FF CS=00C9 IP=0FAA NV UP DI PL NZ NA PO NC
00C9:0FAA E8DB00 CALL 1088
-G
Program terminated normally
-

```

ข้อสังเกต เราควรจะพิจารณาค่าของรีจิสเตอร์ CS และ IP ก่อนที่จะเริ่มสั่งให้โปรแกรมทำงานใหม่เสมอ โดยปกติในโปรแกรม DEBUG ค่าของรีจิสเตอร์ CS จะมีค่าเท่ากับเซกเมนต์รีจิสเตอร์ตัวอื่น ๆ (DS ES และ SS) และเรานิยมป้อนโปรแกรมที่จะทดลองลงในตำแหน่ง CS:100h

สรุป

คำสั่งภาษาแอสเซมบลีเบื้องต้น โดยจะเป็นคำสั่งเกี่ยวกับการโอนย้ายข้อมูล ซึ่งมีอยู่ด้วยกัน 2 แบบ คือ

1. การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ เป็นการโอนย้ายข้อมูลระหว่างรีจิสเตอร์สามารถทำได้ถ้าขนาดของรีจิสเตอร์ของทั้งคู่เท่ากัน แต่ยังมีเซกเมนต์รีจิสเตอร์บางตัวซึ่งไม่ควรเข้าไปกำหนดค่าโดยตรง เช่น CS หรือ SS

2. การโอนย้ายข้อมูลกับหน่วยความจำ โดยปกติการโอนย้ายข้อมูลกับหน่วยความจำนั้น เราจะระบุเฉพาะออฟเซตของหน่วยความจำที่เราต้องการจะโอนย้ายข้อมูลด้วย ออฟเซตนั้นจะถูกนำมาประกอบกับเซกเมนต์รีจิสเตอร์ DS เพื่อเป็นตำแหน่งในหน่วยความจำที่แท้จริงที่จะอ่านเขียนข้อมูลด้วย

ดีบั๊ก (Debug) คือ โปรแกรมที่พัฒนาเพื่อแก้ไขปัญหาค้นหาพื้นฐานในระบบปฏิบัติการดอส (DOS = Disk Operation System) เป็นโปรแกรมสำหรับแก้ไขแฟ้มอย่างง่าย เป็นคำสั่งภายนอก (External Command) ของดอส (DOS) ที่นิยมใช้งานในกลุ่มนักพัฒนามาตั้งแต่ยุคระบบปฏิบัติการดอส เครื่องมือในการทดลองการโปรแกรมภาษาแอสเซมบลีก็คือโปรแกรม DEBUG เป็นโปรแกรมสารพัดประโยชน์สำหรับการทดลองเกี่ยวกับการเขียนโปรแกรมภาษาแอสเซมบลี โปรแกรมนี้มีอยู่ทั้งในระบบปฏิบัติการ DOS และ Windows

คำถามทบทวน

1. จงให้คำจำกัดความของค่าว่าโอเปอเรนด์ (operand)
2. โปรแกรม DEBUG คืออะไรและมีหน้าที่อย่างไรในภาษาโปรแกรมภาษาแอสเซมบลี
3. รีจิสเตอร์ AX มีหน้าที่อย่างไรในคำสั่ง MOV
4. คำสั่งต่อไปนี้ทำงานอย่างไร
 - คำสั่ง G (go)
 - คำสั่ง D (dump)
 - คำสั่ง R (register)
5. คำสั่ง A (assemble) มีหน้าที่และทำงานอย่างไร
6. คำสั่งภาษาแอสเซมบลีที่ใช้สำหรับการโอนย้ายข้อมูลมีกี่แบบอะไรบ้าง

แผนบริหารการสอนประจำบทที่ 7

หัวข้อเนื้อหา

- แฟล็ก (Flags)
- คำสั่งทางคณิตศาสตร์
- กลุ่มคำสั่งบวกและลบ
- กลุ่มคำสั่งคูณและหาร
- กลุ่มคำสั่งแปลงขนาดตัวเลข

วัตถุประสงค์เชิงพฤติกรรม

- เข้าใจความหมายของแฟล็กแต่ละบิตใน 8086
- รู้และเข้าใจ สามารถใช้งานคำสั่งทางคณิตศาสตร์ได้
- รู้และเข้าใจการใช้งานกลุ่มคำสั่งต่างๆ ในภาษาโปรแกรมภาษาแอสเซมบลี เช่น กลุ่มคำสั่งบวก และลบ กลุ่มคำสั่งคูณและหารและกลุ่มคำสั่งแปลงขนาดตัวเลข เป็นต้น

วิธีสอนและกิจกรรมการเรียนรู้การสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

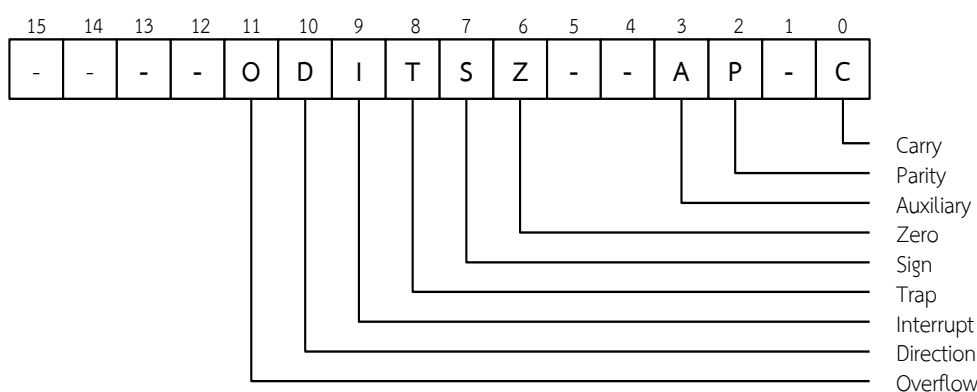
ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 7 แฟล็กและคำสั่งคณิตศาสตร์ (Flags and Mathematics Instruction)

ในบทนี้จะศึกษาเกี่ยวกับการใช้คำสั่งคำนวณทางคณิตศาสตร์และการแสดงสถานะของผลลัพธ์ของการคำนวณนั้นในแฟล็ก สถานะที่เก็บในแฟล็กจะใช้สำหรับการจัดการกับผลการคำนวณนั้น ๆ รวมถึงใช้ในคำสั่งเกี่ยวกับการกระโดดแบบมีเงื่อนไขด้วย

7.1 แฟล็ก (Flags)

แฟล็กเปรียบเสมือนรีจิสเตอร์ตัวหนึ่ง แต่แทนที่จะใช้เก็บค่าต่าง ๆ แฟล็กจะเก็บสถานะของการคำนวณทางคณิตศาสตร์ที่ผ่านมา ตัวอย่างของสถานะของการคำนวณ เช่น การมีบิตทด มีการเก็บค่าล้นหลักหรือผลลัพธ์มีค่าเป็นศูนย์ เป็นต้น ใน 8086 แฟล็กจะมีขนาด 16 บิต โดยในแต่ละบิตจะเก็บค่าของสถานะการคำนวณแบบหนึ่ง ๆ แสดงได้ดังรูปที่ 7.1



รูปที่ 7.1 แสดงแฟล็กต่าง ๆ

ค่าในบิตของแฟล็กนั้น ๆ จะมีค่าเป็น 1 เมื่อสถานะนั้นเป็นจริง เราจะเรียกสถานะที่แฟล็กเป็น 1 ว่า แฟล็กเซต (flag set) และถ้าแฟล็กมีค่าเป็นศูนย์เราจะเรียกว่าแฟล็กเคลียร์ (flag cleared) โดยทั่วไปแล้วคำสั่งที่จะมีผลกับแฟล็กจะเป็นคำสั่งเกี่ยวกับการคำนวณทางคณิตศาสตร์ ส่วนคำสั่งในกลุ่มของการโอนย้ายข้อมูล เช่น คำสั่ง MOV จะไม่เปลี่ยนแปลงค่าในแฟล็ก

ความหมายของแฟล็กแต่ละบิตเป็นดังต่อไปนี้

1. แฟล็กศูนย์ (Zero flag)

แฟล็กศูนย์จะมีค่าเป็นหนึ่ง (flag set) เมื่อผลการคำนวณมีค่าเท่ากับศูนย์

ตัวอย่างคำสั่ง

MOV	AX,100h	Z-flag = ?
MOV	BX,80h	Z-flag = ?

SUB	AX,BX	Z-flag = 0	{AX=80h}
SUB	AX,BX	Z-flag = 1	{AX=0h}
MOV	CX,80h	Z-flag = 1	
ADD	AX,CX	Z-flag = 0	{AX=80h}
SUB	AX,BX	Z-flag = 1	{AX=0h}

2. พาริตีแฟล็ก (Parity flag)

พาริตีแฟล็กจะมีค่าเป็นหนึ่งเมื่อในผลลัพธ์มีจำนวนบิตที่มีค่าเป็น 1 เป็นจำนวนคู่
ตัวอย่างคำสั่ง

MOV	AL,34h	P-flag = ?	
MOV	BL,11h	P-flag = ?	
ADD	AL,BL	P-flag = 0	{AL=45h[0100 0101b]}
ADD	AL,6	P-flag = 1	{AL=4Bh[0100 1011b]}
SUB	AL,BL	P-flag = 1	{AL=3Ah[0011 1010b]}
MOV	CL,10h	p-flag = 1	
ADD	AL,CL	p-flag = 0	{AL=4Ah[0100 1010b]}

3. แฟล็กเครื่องหมาย

แฟล็กเครื่องหมายจะเซตเมื่อผลลัพธ์มีค่าเป็นลบ (เมื่อคิดว่าผลลัพธ์นั้นเก็บตัวเลขแบบคิดเครื่องหมายแบบ

2' Complement)

ตัวอย่างคำสั่ง

MOV	AL,50h	S-flag = ?	
ADD	AL,50h	S-flag = 1	{AL=0A0h[1010 0000b]}
ADD	AL,0A0h	S-flag = 0	{AL=40h[0100 0000b]}
SUB	AL,10h	S-flag = 0	{AL=30h[0011 0000b]}
ADD	AL,0B0h	S-flag = 1	{AL=0E0h[1110 0000b]}

4. แฟล็กทด (Carry flag)

แฟล็กทดจะเซตเมื่อการคำนวณมีการทดหรือมีการยืม ในการพิจารณาเราจะพิจารณาค่าของข้อมูลแบบไม่คิด
เครื่องหมาย แฟล็กทดยังใช้ในการเก็บบิตข้อมูลในคำสั่งเกี่ยวกับการเลื่อนบิตด้วย

ตัวอย่างคำสั่ง

MOV	AL,60h	C-flag = ?	
ADD	AL,60h	C-flag = 0	{AL=0C0h}
ADD	AL,60h	C-flag = 1	{AL=20h,0C0+60=120h}
SUB	AL,15h	C-flag = 0	{AL=0Ah}
SUB	AL,15h	C-flag = 1	{AL=F5h,0Ah-15h=0F5h}

5. โอเวอร์โฟลล์แฟล็ก (Overflow flag)

ในการพิจารณาโอเวอร์โฟลล์แฟล็กเราจะพิจารณาค่าของข้อมูลเป็นแบบคิดเครื่องหมาย โดยโอเวอร์โฟลล์แฟล็กจะมีค่าเป็นหนึ่งเมื่อผลลัพธ์มีความผิดพลาด เช่น การบวกค่าที่มากกว่าขอบเขตทำให้ผลลัพธ์ที่ได้ มีเครื่องหมายที่ผิดเป็นต้น

ตัวอย่างคำสั่ง

MOV	AL,0E3h	O-flag = ?	
ADD	AL,79h	O-flag = 0	{AL=5Ch,-29+121 = 92[5Ch]}
ADD	AL,60h	O-flag = 1	{AL=0BCh,0BCh=-68 <> 92+96=188}
SUB	AL,20h	O-flag = 0	{AL=9Ch,9Ch=-100 = -68-32}
SUB	AL,20h	O-flag = 1	{AL=7Ch,7Ch=124 <> -100-32=-132}
ADD	AL,9Ah	O-flag = 0	{AL=16h,16h=22 = 124-102}

6. แฟล็กเสริม (Auxiliary Flag)

แฟล็กเสริมจะเป็นแฟล็กที่ใช้ในการปรับค่าของการคำนวณเลขแบบ BCD

7. แฟล็กทิศทาง (Direction Flag)

แฟล็กทิศทางเป็นแฟล็กที่ใช้ในการระบุทิศทางของการปรับค่ารีจิสเตอร์ดัชนีในการประมวลผลคำสั่งเกี่ยวกับสายข้อมูล

8. แทรปแฟล็ก (Trap Flag)

แทรปแฟล็กเป็นแฟล็กที่ใช้ระบุให้หน่วยประมวลผลสร้างสัญญาณขัดจังหวะเมื่อประมวลผลคำสั่งเสร็จสิ้นหนึ่งคำสั่ง โดยแฟล็กนี้จะใช้ในการตรวจสอบการทำงานของโปรแกรม

9. อินเทอร์รัพท์แฟล็ก (Interrupt Flag)

แฟล็กนี้ใช้ระบุว่าหน่วยประมวลผลจะตอบสนองการขัดจังหวะจากอุปกรณ์ฮาร์ดแวร์หรือไม่

คำสั่งเกี่ยวกับการกำหนดค่าของแฟล็ก

เราสามารถกำหนดค่าของแฟล็กทด แฟล็กทิศทาง และอินเทอร์รัพท์แฟล็กได้โดยใช้คำสั่งต่อไปนี้

ตารางที่ 7.1 คำสั่งสำหรับการกำหนดค่าแฟล็ก

แฟล็ก	คำสั่งในการเซต	คำสั่งในการเคลียร์
carry flag	STC	CLC
direction flag	STD	CLD
interrupt flag	STI	CLI

7.2 คำสั่งทางคณิตศาสตร์

คำสั่งทางคณิตศาสตร์ใน 8086 ที่เราจะศึกษาในวิชานี้แบ่งได้เป็นกลุ่มใหญ่ ๆ 3 กลุ่มดังนี้

7.2.1 กลุ่มคำสั่งบวกและลบ

คำสั่งเพิ่มและลดค่า : INC [Increment] และ DEC [Decrement]

คำสั่ง INC จะเพิ่มค่าของโอเพอร์แรนด์ขึ้นหนึ่ง ส่วนคำสั่ง DEC จะลดค่าของโอเพอร์แรนด์ลงหนึ่ง คำสั่งนี้มีผลกระทบกับแฟล็กทั้งหมดยกเว้นแฟล็กทด.

รูปแบบของทั่วไปของคำสั่ง INC และ DEC เป็นดังนี้

INC	<i>register</i>	DEC	<i>register</i>
INC	<i>memory</i>	DEC	<i>memory</i>

ตัวอย่างคำสั่ง

MOV	AL,10h	
INC	AL	รีจิสเตอร์ AL = 11h
MOV	BX,200h	
MOV	WORD PTR [BX],10h	
DEC	WORD PTR [BX]	ข้อมูลแบบเวิร์ดที่ตำแหน่ง [DS:BX] = 0Fh

คำสั่งบวกและบวกรวมตัวทด : ADD [Addition] และ ADC [Add with carry]

คำสั่งบวกจะนำค่าของโอเพอร์แรนด์ตัวที่สองมาบวกกับค่าของโอเพอร์แรนด์ตัวที่หนึ่ง แล้วนำค่าที่ได้เก็บในโอเพอร์แรนด์ตัวที่หนึ่ง. คำสั่ง ADD และ ADC มีผลกระทบกับแฟล็กทางคณิตศาสตร์ทุกแฟล็ก เรานิยมใช้คำสั่ง ADC ในการบวกข้อมูลที่ต้องนำบิตที่ทดจากการบวกครั้งก่อนมารวมด้วย เช่น การบวกข้อมูลที่เก็บอยู่ในหลายรีจิสเตอร์ต่อเนื่องกันเป็นต้น

รูปแบบของทั่วไปของคำสั่ง ADD และ ADC เป็นดังนี้

ADD	<i>register,number</i>	ADC	<i>register,number</i>
ADD	<i>memory,number</i>	ADC	<i>memory,number</i>
ADD	<i>register,register</i>	ADC	<i>register,register</i>
ADD	<i>register,memory</i>	ADC	<i>register,memory</i>
ADD	<i>memory,register</i>	ADC	<i>memory,register</i>

ตัวอย่างคำสั่ง

MOV	AL,10h	รีจิสเตอร์ AL = 30h
ADD	AL,20h	
MOV	BX,200h	
MOV	WORD PTR [BX],10h	ข้อมูลแบบเวิร์ดที่ตำแหน่ง [DS:BX] = 80h
ADD	WORD PTR [BX],70h	

MOV AX,5678h
MOV DX,1234h
ADD AX,4321h
ADC DX,8765h

เราจะบวก 1234 5678h เข้ากับ 8765 4321h โดยผลลัพธ์ที่ได้ 16 บิตบนจะเก็บที่รีจิสเตอร์ DX ส่วน 16 บิตล่างจะเก็บที่รีจิสเตอร์ AX สังเกตว่าเราจะใช้คำสั่ง ADC ในการบวกครั้งที่สองเพื่อนำตัวทดจากการบวกครั้งแรกมารวมด้วย

คำสั่งลบและลบรวมตัวยืม : SUB [Substraction] และ SBB [Subtract with borrow]

คำสั่ง SUB และ SBB จะทำงานคล้ายกับคำสั่ง ADD และ ADC เพียงแต่เป็นการนำค่าในโอเปอร์แรนด์ตัวที่สองไปลบออกจากโอเปอร์แรนด์ตัวที่หนึ่ง ลักษณะของการใช้งานคำสั่ง SBB จะคล้ายคลึงกับการใช้คำสั่ง ADC นั่นคือจะนิยมใช้ในกรณีที่มีการลบเลขที่เก็บอยู่ในรีจิสเตอร์หลายตัวต่อเนื่องกัน

รูปแบบของคำสั่ง SUB และ SBB จะมีลักษณะเช่นเดียวกับคำสั่ง ADD และคำสั่ง ADC โดยมีรูปแบบทั้งหมดดังนี้

SUB	<i>register,number</i>	SBB	<i>register,number</i>
SUB	<i>memory,number</i>	SBB	<i>memory,number</i>
SUB	<i>register,register</i>	SBB	<i>register,register</i>
SUB	<i>register,memory</i>	SBB	<i>register,memory</i>
SUB	<i>memory,register</i>	SBB	<i>memory,register</i>

ตัวอย่างคำสั่ง

MOV	CX,10h	รีจิสเตอร์ CX จะถูกลดค่าลงเท่ากับข้อมูลแบบเวิร์ดในแอดเดรส [DS:200h].
SUB	CX,[200h]	
MOV	BX,202h	
MOV	DX,345h	
SUB	WORD PTR [BX],DX	ข้อมูลแบบเวิร์ดที่ตำแหน่ง [DS:BX] มีค่าลดลง 345h
MOV	AX,0AAFFh	
MOV	DX,0BBCCh	เราจะลบค่าของข้อมูลขนาด 32 บิตที่เก็บที่ CX BX ด้วยค่าขนาด 32 บิตในรีจิสเตอร์ DX และ AX
SUB	BX,AX	
SBB	CX,DX	

คำสั่งเปรียบเทียบ : CMP [Compare]

คำสั่ง CMP จะมีการทำงานเหมือนกับคำสั่ง SUB ทุกประการ แต่จะไม่มีการเปลี่ยนค่าในโอเปอร์แรนด์ใด ๆ โดยผลลัพธ์ที่แท้จริงของคำสั่งนี้คือการเปลี่ยนค่าในแฟล็กต่าง ๆ เพื่อแสดงผลลัพธ์ของการลบ เราจะใช้คำสั่งนี้ในการเปรียบเทียบค่าต่าง ๆ และนำผลที่ได้ในแฟล็กไปใช้ในการกำหนดเงื่อนไขของการกระโดด รูปแบบของคำสั่ง CMP จะเหมือนคำสั่ง SUB โดยมีรูปแบบทั่วไปเป็นดังนี้

CMP	<i>register,number</i>
CMP	<i>memory,number</i>
CMP	<i>register,register</i>
CMP	<i>register,memory</i>
CMP	<i>memory,register</i>

ตัวอย่างคำสั่ง

MOV	CX,10h	CX จะมีไม่มีการเปลี่ยนแปลง แต่จะมีการเปลี่ยนแปลงของแฟล็ก โดย S-flag=1, C-flag=1, O-flag=0
CMP	CX,20h	
MOV	BX,40h	
CMP	BX,40h	Z-flag=1

```
MOV    AX,30h
CMP    AX,20h           S-flag=0, C-flag=0, O-flag=0.
```

คำสั่งเปลี่ยนเครื่องหมาย : NEG [Negation]

คำสั่งเปลี่ยนเครื่องหมายจะเปลี่ยนค่าในโอเปอร์แรนด์ซึ่งจะพิจารณาเป็นตัวเลขแบบคิดเครื่องหมายเป็นค่าลบของค่านั้น. โดยการเปลี่ยนค่านั้นจะเปลี่ยนแบบ 2's complement ผลจากคำสั่งนี้จะทำให้ **แฟล็กทดมีค่าเป็น 1 เสมอ**

รูปแบบของคำสั่ง NEG

```
NEG    register
```

```
NEG    memory
```

ตัวอย่างคำสั่ง

```
MOV    CX,10h
NEG    CX           CX = 0FFF0h
MOV    AX,0FFFFh
NEG    AX           AX = 1
MOV    BX,1h
NEG    BX           BX = 0FFFFh
```

7.2.2 กลุ่มคำสั่งคูณและหาร

คำสั่งคูณแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย : IMUL [Integer multiplication] และ MUL [Multiplication]

การคูณใน 8086 นั้นค่าของตัวตั้งของการคูณ และค่าผลลัพธ์ของการคูณนั้นจะต้องเก็บในรีจิสเตอร์ที่กำหนดไว้ โดยขึ้นกับขนาดของการคูณ รีจิสเตอร์ที่กำหนดไว้เป็นดังนี้

การคูณข้อมูล 8 บิต : ตัวตั้ง AL ผลลัพธ์ AX.

การคูณข้อมูล 16 บิต : ตัวตั้ง AX ผลลัพธ์ DX, AX [16 บิตบนที่ DX 16 บิตล่างที่ AX]

เราจะระบุตัวคูณและขนาดของการคูณที่โอเปอร์แรนด์ของคำสั่ง IMUL หรือ MUL ทั้งสองคำสั่งมีรูปแบบดังนี้

```
IMUL    register      MUL    register
```

```
IMUL    memory       MUL    memory
```

ตัวอย่างคำสั่ง

```
MOV    AL,17h        เราจะคูณ 17h ด้วย 13h แบบ 8 บิต
MOV    CL,13h
IMUL   CL            ผลลัพธ์ที่ได้จะมีขนาด 16 บิต เก็บที่รีจิสเตอร์ AX
MOV    BX,1234h      เราจะนำผลลัพธ์ที่ได้จาก 17h คูณ 13h มาคูณด้วย 1234h
IMUL   BX            ค่าใน DX,AX มีค่าเท่ากับ (17h x 13h) x 1234h
```

คำสั่ง MUL และ IMUL จะมีผลกระทบกับแฟล็กทดและโอเวอร์โฟลล์แฟล็กเท่านั้น

คำสั่งหารแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย : IDIV [Integer division] และ DIV [Division]

เช่นเดียวกับคำสั่งคูณ ตัวตั้งและผลลัพธ์สำหรับการประมวลผลคำสั่งหารใน 8086 จะต้องเก็บรีจิสเตอร์ซึ่งกำหนดไว้ โดยจะขึ้นกับขนาดของการหารตัวเลขเช่นเดียวกัน

การหารข้อมูล 8 บิต : ตัวตั้ง AX ผลลัพธ์ AL เศษ AH

การหารข้อมูล 16 บิต : ตัวตั้ง DX, AX ผลลัพธ์ AX เศษ DX

คำสั่ง IDIV และ DIV มีรูปแบบดังนี้

IDIV	register	DIV	register
IDIV	memory	DIV	memory

ตัวอย่างคำสั่ง

MOV	AX,4022h	เราจะหาร 4022h ด้วย 1000h การหารดังกล่าวเป็นการหารแบบ 16 บิตดังนั้นเราจึงต้องกำหนดค่าตัวตั้งใน DX,AX
MOV	DX,0000h	
MOV	CX,1000h	
DIV	CX	ผลลัพธ์จากการหารจะมีขนาด 16 บิต เก็บที่รีจิสเตอร์ AX โดยเศษจะเก็บที่ DX. AX = 4 และ DX = 22h
MOV	BL,3h	เราจะนำผลลัพธ์ที่ได้จากมาหารด้วย 3h
DIV	BL	ค่าใน AL จะเก็บผลหารซึ่งมีค่าเท่ากับ 1 และ AH จะเก็บเศษโดยจามีค่าเท่ากับ 1

คำสั่ง DIV และ IDIV จะไม่มีผลกระทบต่อแฟล็กใด ๆ แต่ถ้าในการหารมีการหารด้วยศูนย์ หรือเกิดการหารที่ไม่สามารถเก็บผลลัพธ์ลงในรีจิสเตอร์ที่ต้องการได้ เช่นการหาร 1234 5678h ด้วย 2h หน่วยประมวลผลจะสร้างสัญญาณการขัดจังหวะขึ้นเพื่อแจ้งกับโปรแกรมที่จัดการระบบต่อไป การเกิดการขัดจังหวะในลักษณะนี้เราเรียกว่าเกิด Exception

7.3 กลุ่มคำสั่งแปลงขนาดตัวเลข

จากข้อจำกัดของการใช้คำสั่งหารที่ตัวตั้งจะต้องมีขนาดมากกว่าตัวหาร ทำให้ในบางครั้งที่เราต้องการหารข้อมูลที่มีขนาดเท่ากัน เราจะต้องแปลงตัวตั้งให้มีขนาดที่เหมาะสมเสียก่อน สำหรับการแปลงขนาดของเลขไม่คิดเครื่องหมายนั้น เราสามารถกระทำได้โดยกำหนดให้ข้อมูลนัยสำคัญสูงที่ขยายเพิ่มมานั้นมีค่าเป็นศูนย์ ตัวอย่างเช่นการขยาย AL ที่เป็นตัวเลขแบบไม่คิดเครื่องหมายให้เป็นข้อมูล 16 บิตใน AX สามารถกระทำได้โดยกำหนดค่าศูนย์ให้กับ AH แต่ในกรณีของตัวเลขแบบคิดเครื่องหมายนั้นถ้าตัวเลขมีค่าเป็นลบการกำหนดค่าศูนย์ให้กับข้อมูลนัยสำคัญสูงที่ขยายเพิ่มมานั้น จะทำให้ค่าของเลขที่ได้มีความผิดพลาดได้ ในการขยายขนาดของเลขคิดเครื่องหมายเราจึงต้องใช้คำสั่งที่เหมาะสม

คำสั่งแปลงจากไบต์เป็นเวิร์ด : CBW [Convert byte to word]

คำสั่งนี้จะแปลงเลขแบบคิดเครื่องหมายขนาด 8 บิตใน AL เป็นเลขคิดเครื่องหมายขนาด 16 บิตใน AX รูปแบบของคำสั่ง

CBW

ตัวอย่างคำสั่ง

MOV	AL,22h	AL = 22h
CBW		เมื่อแปลงแล้ว AX=0022h
MOV	AL,F0h	AL = F0h = -16
CBW		เมื่อแปลงแล้ว AX=FFF0h = -16

คำสั่งแปลงจากเวิร์ดเป็นดับเบิลเวิร์ด : CWD [Convert word to doubleword]

คำสั่งนี้จะแปลงเลขแบบคิตเครื่องหมายขนาด 16 บิตใน AX เป็นเลขคิตเครื่องหมายขนาด 32 บิตใน DX,AX

รูปแบบของคำสั่ง

CWD

ตัวอย่างคำสั่ง

MOV	AX,3422h	AX = 3422h
CWD		เมื่อแปลงแล้ว DX = 0000h , AX=3422h
MOV	AX,FFF0h	AX = FFF0h = -16
CWD		เมื่อแปลงแล้ว DX = FFFFh , AX=FFF0h

ตัวอย่างการใช้คำสั่งทางคณิตศาสตร์

ตัวอย่างที่ 1

คำนวณค่า $AL^2 + BL^2$ คิตตัวเลขแบบไม่คิตเครื่องหมายโดยให้ผลลัพธ์เป็นเลข 16 บิต เก็บที่รีจิสเตอร์ AX

MUL	AL	หาค่า $AL*AL$ เสียก่อน
MOV	CX, AX	
MOV	AL, BL	จากนั้นนำไปเก็บไว้ที่ CX เนื่องจากรีจิสเตอร์ AL ต้องใช้ในการคำนวณ $BL*BL$.
MUL	BL	
ADD	AX, CX	ได้ผลลัพธ์แล้วนำค่า $AL*AL$ ที่เก็บใน CX มาบวก

ตัวอย่างที่ 2

คำนวณค่า $(CL*BL+DX)/SI$

MOV	AL,CL	โดยคิตเป็นเลขคิตเครื่องหมาย
MUL	BL	หาค่าของ $CL*BL$ โดยผลลัพธ์ที่ได้เก็บใน AX.
ADD	AX,DX	จากนั้นนำ DX มาบวกเข้ากับ AX
CWD		ขยายข้อมูลใน AX เป็น 32 บิตใน DX,AX เพื่อที่จะได้นำ SI
IDIV	SI	มาหาร ผลลัพธ์ที่ได้อยู่ที่ AX เศษของการหารเก็บที่ DX

ตัวอย่างที่ 3

คำนวณค่า $(DX+AX*BX)/(DI-CX)$	โดยคิดเป็นเลขคิดเครื่องหมาย
MOV SI,DX	นำค่าของ DX ไปเก็บที่ SI เสียก่อนเนื่องจาก DX จะถูก ใช้
MUL BX	ในการคำนวณ $AX*BX$. เมื่อคำนวณ $AX*BX$ แล้วนำ SI มา
ADD AX,SI	บวกโดยต้องนำตัวทศของการบวก SI กับ AX มาทศยัง DX
ADC DX,0	ด้วย
SUB DI,CX	ลบ DI ด้วย CX
IDIV DI	นำไปหาร

ผลกระทบของคำสั่งทางคณิตศาสตร์ต่อแฟล็ก

ตารางที่ 7.2 ผลกระทบของคำสั่งต่าง ๆ ต่อแฟล็ก

Instruction	Flag affected				
	Z-flag	C-flag	S-flag	O-flag	A-flag
ADD	yes	yes	yes	yes	yes
ADC	yes	yes	yes	yes	yes
SUB	yes	yes	yes	yes	yes
SBB	yes	yes	yes	yes	yes
INC	yes	no	yes	yes	yes
DEC	yes	no	yes	yes	yes
NEG	yes	yes	yes	yes	yes
CMP	yes	yes	yes	yes	yes
MUL	no	yes	no	yes	no
IMUL	no	yes	no	yes	no
DIV	no	no	no	no	no
IDIV	no	no	no	no	no
CBW	no	no	no	no	no
CWD	No	no	no	no	no

ตัวอย่างของการเปลี่ยนแปลงของแฟล็กในการทำงานของคำสั่งต่าง ๆ

ตัวอย่างที่ 1

Instruction	Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV AX,7100h	?	?	?	?	?	
MOV BX,4000h	?	?	?	?	?	
ADD AX,BX	0	0	1	1	1	AX=0B100h
ADD AX,7700h	0	1	0	0	1	AX=2800h
SUB AX,2000h	0	0	0	0	0	AX=0800h
SUB AX,1000h	0	1	0	1	0	AX=F800h
ADD AX,0800h	1	1	0	0	1	AX=0000h

ตัวอย่างที่ 2

Instruction	Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV AL,10	?	?	?	?	?	
ADD AL,F0h	0	0	0	1	1	AL=0FAh
ADD AL,6	1	1	0	0	1	AL=0
SUB AL,5	0	1	0	1	0	AL=0FBh
INC AL	0	0	0	1	1	AL=0FCh
ADD AL,10	0	1	0	0	1	AL=6h
ADD AL,FBh	0	1	0	0	0	AL=1h
DEC AL	1	0	0	0	1	AL=0h
DEC AL	0	0	0	1	1	AL=0FFh
INC AL	1	0	0	0	1	AL=0

หมายเหตุ คำสั่ง DEC และ INC ไม่กระทบแฟล็กทด

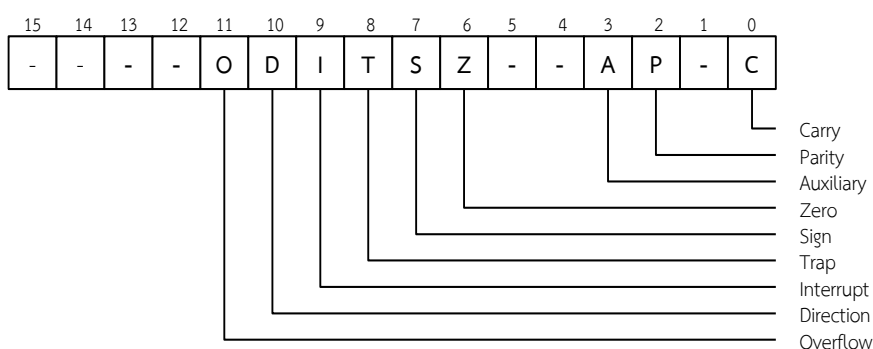
ตัวอย่างที่ 3

Instruction	Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV AL,120	?	?	?	?	?	
ADD AL,15	0	0	1	1	1	AL=87h=-121
NEG AL	0	1	0	0	0	AL=79h
SUB AL,130	0	1	1	1	0	AL=0F7h

หมายเหตุ คำสั่ง NEG ทำให้แฟล็กทดมีค่าเป็น 1 เสมอ

สรุป

แฟล็ก (Flags) เป็นรีจิสเตอร์ขนาด 16 บิตที่ใช้บ่งบอกผลลัพธ์ที่ได้จากการคำนวณ คำสั่งที่เกี่ยวข้องกับการคำนวณจะส่งผลกระทบต่อแฟล็กเปลี่ยนแปลงค่าไป การคัดลอกข้อมูลจะไม่ทำให้แฟล็กเปลี่ยนแปลง ค่าแฟล็กเปรียบเสมือนรีจิสเตอร์ตัวหนึ่ง แต่แทนที่จะใช้เก็บค่าต่าง ๆ แฟล็กจะเก็บสถานะของการคำนวณทางคณิตศาสตร์ที่ผ่านมา ตัวอย่างของสถานะของการคำนวณ เช่น การมีบิตทด มีการเก็บค่าล้นหลัก หรือผลลัพธ์มีค่าเป็นศูนย์ เป็นต้น ใน 8086 แฟล็กจะมีขนาด 16 บิต โดยในแต่ละบิตจะเก็บค่าของสถานะการคำนวณแบบหนึ่ง ๆ ดังรูป



คำสั่งทางคณิตศาสตร์ใน 8086 ที่เราจะศึกษาในวิชานี้แบ่งได้เป็นกลุ่มใหญ่ ๆ 3 กลุ่ม คือ

1. กลุ่มคำสั่งบวกและลบ ประกอบไปด้วย
 - 1.1 คำสั่งเพิ่มและลดค่า : INC [Increment] และ DEC [Decrement]
 - 1.2 คำสั่งบวกและบวกรวมตัวทด : ADD[Addition] และ ADC [Add with carry]
 - 1.3 คำสั่งลบและลบรวมตัวยืม : SUB [Substraction] และ SBB [Subtract with borrow]
 - 1.4 คำสั่งเปรียบเทียบ: CMP [Compare]
 - 1.5 คำสั่งเปลี่ยนเครื่องหมาย : NEG [Negation]
2. กลุ่มคำสั่งคูณและหาร ประกอบไปด้วย
 - 2.1 คำสั่งคูณแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย:
IMUL [Integer multiplication] และ MUL [Multiplication]
 - 2.2 คำสั่งหารแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย: IDIV [Interger division] และ DIV [Division]
3. กลุ่มคำสั่งแปลงขนาดตัวเลข ประกอบไปด้วย
 - 3.1 คำสั่งแปลงจากไบต์เป็นเวิร์ด : CBW [Convert byte to word]
 - 3.2 คำสั่งแปลงจากเวิร์ดเป็นดับเบิลเวิร์ด : CWD [Convert word to doubleword]

คำถามทบทวน

1. จงอธิบายความหมายและหน้าที่ของแฟล็ก (Flags) แต่ละบิตว่าต่อไปนี้
 - 1.1 แฟล็กศูนย์ (Zero flag)
 - 1.2 พาริตีแฟล็ก (Parity flag)
 - 1.3 แฟล็กทด (Carry flag)
 - 1.4 แฟล็กเครื่องหมาย (Sign-flag)
 - 1.5 โอเวอร์โฟลล์แฟล็ก (Overflow flag)
 - 1.6 แฟล็กเสริม (Auxiliary Flag)
 - 1.7 แฟล็กทิศทาง (Direction Flag)
 - 1.8 แทรปแฟล็ก (Trap Flag)
 - 1.9 อินเตอร์รัพท์แฟล็ก (Interrupt Flag)
2. จงอธิบายความแตกต่างระหว่างคำสั่งลบ SUB และลบรวมตัวยืม SBB
3. จงอธิบายความแตกต่างระหว่างคำสั่งคูณแบบคิดเครื่องหมาย IMUL และไม่คิดเครื่องหมาย MUL
4. จงอธิบายความหมายของคำสั่งเปรียบเทียบ CMP และคำสั่งแปลงจากไบต์เป็นเวิร์ด CBW
5. จากตัวอย่างคำสั่งข้างล่างจงอธิบายความหมายของคำสั่งแต่ละบรรทัด

ตัวอย่างคำสั่ง

```
MOV AL,25h
MOV CL,13h
IMUL CL
MOV BX,3456h
IMUL BX
```
6. จากตัวอย่างคำสั่งข้างล่างจงอธิบายความหมายของคำสั่งแต่ละบรรทัด

ตัวอย่างคำสั่ง

```
MOV AX,2513h
MOV DX,0000h
MOV CX,1000H
DIV CX
MOV BL,3h
DIV BL
```
7. จากตัวอย่างคำสั่งข้างล่างจงอธิบายความหมายของคำสั่งแต่ละบรรทัด

ตัวอย่างคำสั่ง

```
MOV AL,11h
CBW
MOV AL,A0h
CBW
```

แผนบริหารการสอนประจำบทที่ 8

หัวข้อเนื้อหา

- รูปแบบของโปรแกรมภาษาแอสเซมบลีแบบเก่า
- รูปแบบของโปรแกรมภาษาแอสเซมบลีแบบใหม่
- การเรียกใช้บริการของ DOS
- ขั้นตอนการแปลโปรแกรม
- ตารางรหัสแอสกี

วัตถุประสงค์เชิงพฤติกรรม

- เข้าใจรูปแบบของการเขียนโปรแกรมภาษาแอสเซมบลี การประกาศเซกเมนต์แบบต่างๆ เช่น เซกเมนต์ cseg dseg และ sseg เป็นต้น
- สามารถและเข้าใจการประกาศข้อมูล การใส่หมายเหตุ และการสั่งให้โปรแกรมจบการทำงาน
- สามารถและเข้าใจการเรียกใช้บริการของ DOS และอ่านตารางรหัสแอสกีได้

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 8 โปรแกรมภาษาแอสเซมบลีเบื้องต้น (Assembly Language Programming)

ในบทนี้จะศึกษาการเขียนโปรแกรมภาษาแอสเซมบลีแบบเต็มรูปแบบ นั่นคือจะเขียนโปรแกรมภาษาแอสเซมบลีที่เป็นโปรแกรมที่ทำงานได้จริง มีการกำหนดรูปแบบต่างๆ ครบถ้วน โปรแกรมภาษาแอสเซมบลีที่จะเขียนต่อไปนี้ไม่ได้ทำงานบนโปรแกรม DEBUG เท่านั้น จะต้องใช้ assembler แปลโปรแกรมที่เขียนขึ้นให้อยู่ในรูปแบบที่คอมพิวเตอร์สามารถนำไปประมวลผลได้เสียก่อน

8.1 รูปแบบของโปรแกรมภาษาแอสเซมบลี

โปรแกรมที่ทำงานในเครื่องคอมพิวเตอร์ซึ่งใช้หน่วยประมวลผลตระกูล 80x86 นั้นจะมีการแบ่งโปรแกรมทั้งหมดเป็นเซกเมนต์ย่อย ๆ เช่น Code segment Data segment หรือ Stack segment ดังนั้นในโปรแกรมภาษาแอสเซมบลีที่เขียนจะประกอบไปด้วยเซกเมนต์ต่าง ๆ เช่นเดียวกัน ภายในเซกเมนต์ต่าง ๆ ที่ประกาศจะระบุข้อมูลและโปรแกรมที่จะอยู่ในเซกเมนต์นั้น

ตัวอย่างโปรแกรม

```

;
; This program prints the message "Hello world"
;
dseg  segment
msg1  db    'Hello world',10h,13h,'$'
dseg  ends

sseg  segment stack
      db    100 dup (?)
sseg  ends

cseg  segment
      assume cs:cseg,ds:dseg,ss:sseg
start:
      mov  ax,dseg
      mov  ds,ax
      mov  ah,9h
      mov  dx,offset msg1
      int  21h
      mov  ax,4c00h

```

```

int    21h
cseg   ends
end    start

```

จากตัวอย่าง จะสังเกตได้ว่าโปรแกรมได้ประกาศเซกเมนต์ทั้งหมด 3 เซกเมนต์ คือ cseg dseg และ sseg เซกเมนต์ดังกล่าวนี้ถูกประกาศด้วยคำสั่งเทียม **segment** การที่เรียกคำสั่ง segment ว่าคำสั่งเทียมเพราะคำสั่งนี้เป็นคำสั่งที่ผู้เขียนโปรแกรมระบุให้โปรแกรม assembler แปลโปรแกรมตามลักษณะที่กำหนด โดยจะไม่มีคำสั่งภาษาเครื่องถูกสร้างจากคำสั่งกลุ่มนี้ ตัวอย่างอื่น ๆ ของคำสั่งเทียมคือ db assume และ org เป็นต้น

การประกาศเซกเมนต์

การประกาศเซกเมนต์ในโปรแกรมภาษาแอสเซมบลี ใช้คู่คำสั่งเทียม segment และ ends โดยมีลักษณะการประกาศดังนี้.

```

segment_name    segment
...
segment_name    ends

```

จากตัวอย่างได้ประกาศเซกเมนต์ cseg dseg และ sseg คำสั่งเทียม **stack** ระบุให้ระบบใช้เซกเมนต์ sseg เป็นแอสต์กของโปรแกรม คำสั่งเทียม **assume** เป็นการระบุให้ assembler ได้ทราบว่าเซกเมนต์ที่ประกาศนั้นจะให้ระบบพิจารณาว่าทำหน้าที่อะไรและมีเซกเมนต์รีจิสเตอร์ใดเป็นตัวเก็บค่าเซกเมนต์ จากตัวอย่างประกาศให้ assembler ทราบว่าเซกเมนต์ cseg จะชี้โดยรีจิสเตอร์ CS เซกเมนต์ dseg จะชี้โดย รีจิสเตอร์ DS และเซกเมนต์ sseg จะชี้โดยรีจิสเตอร์ SS คำสั่งเทียม assume นี้จะเป็นการบอก assembler ให้พิจารณาตามที่ระบุเท่านั้น ไม่ได้เป็นการสั่งให้ assembler กำหนดค่าต่าง ๆ ให้โดยอัตโนมัติ สังเกตได้จากในตอนต้นของโปรแกรมมีชุดคำสั่งเพื่อปรับค่าของรีจิสเตอร์ DS ดังนี้

```

mov    ax,dseg
mov    ds,ax

```

ชุดคำสั่งนี้จะปรับค่าของรีจิสเตอร์ DS ให้ชี้ไปที่ dseg สำหรับรีจิสเตอร์ SS ระบบจะปรับค่าให้ชี้ไปที่เซกเมนต์ที่ระบุไว้โดยคำสั่งเทียม stack ส่วนกรณีของรีจิสเตอร์ CS นั้นระบบจะตั้งค่าให้ตรงกับเซกเมนต์ที่เริ่มต้นโปรแกรม

โปรแกรมภาษาแอสเซมบลีจะประกอบไปด้วยการประกาศเซกเมนต์ต่าง ๆ และจะสิ้นสุดโปรแกรมที่คำสั่งเทียม **end** หลังคำสั่งเทียม end จะระบุจุดเริ่มต้นของโปรแกรม ในโปรแกรมตัวอย่างระบุจุดเริ่มต้นของโปรแกรมที่เลเบล start ที่ประกาศไว้ที่ตอนต้นของโปรแกรม การประกาศเลเบลสามารถทำได้ดังนี้

```

label_name:

```

ระบบจะจดจำตำแหน่งของเลเบลที่ประกาศไว้และจะนำแอดเดรสของเลเบลไปแทนที่ที่ให้โดยอัตโนมัติ. การที่โปรแกรม assembler จัดการเรื่องเกี่ยวกับเลเบลในโปรแกรมภาษาแอสเซมบลีนั้น นับเป็นการเพิ่มความสะดวกให้กับผู้เขียนโปรแกรมเป็นอย่างมาก

การประกาศข้อมูล

ภายในเซกเมนต์ข้อมูลสามารถประกาศข้อมูลต่าง ๆ ได้ จากโปรแกรมตัวอย่างประกาศข้อมูลเป็นข้อความที่จะให้โปรแกรมพิมพ์ออกมา จะศึกษารูปแบบการประกาศข้อมูลในบทต่อไป

การใส่หมายเหตุ

หลังเครื่องหมาย ';' assembler จะตีความว่าเป็นหมายเหตุ การใส่หมายเหตุจะช่วยทำให้โปรแกรมอ่านง่ายขึ้น จากตัวอย่างโปรแกรมข้างต้น 3 บรรทัดแรกจะเป็นหมายเหตุ

การสั่งให้โปรแกรมจบการทำงาน

โปรแกรมจะจบการทำงานเมื่อสั่งให้โปรแกรมจบการทำงานเท่านั้น ถ้าไม่ได้สั่งให้จบการทำงานเมื่อจบโปรแกรมแล้ว หน่วยประมวลผลจะทำงานคำสั่งอื่น ๆ ที่อยู่ในหน่วยความจำต่อจากโปรแกรมของไปเรื่อยๆ ในโปรแกรม DEBUG เรียกใช้คำสั่ง INT 20h เพื่อให้โปรแกรมจบการทำงาน แต่ในโปรแกรมภาษาแอสเซมบลีทั่วไปจะเรียกใช้บริการหมายเลข 4Ch ของระบบปฏิบัติการ DOS โดยจากโปรแกรมตัวอย่างใช้คำสั่งดังนี้

```
mov ax,4C00h
int 21h
```

ในโปรแกรมตัวอย่างนี้ ได้เรียกใช้บริการของ DOS ในการพิมพ์ข้อความด้วย เรียกใช้บริการหมายเลข 9 โดยใช้คำสั่ง

```
mov ah,9h
mov dx,offset msg1
int 21h
```

สำหรับการเรียกใช้บริการของ DOS จะกล่าวถึงในหัวข้อถัดไป

ตัวอย่างโครงสร้างของโปรแกรมภาษาแอสเซมบลี

```
dseg segment
; ประกาศข้อมูล
dseg ends
sseg segment stack
db 100 dup (?)
sseg ends
cseg segment
assume cs:cseg,ds:dseg,ss:sseg
```

```

start:
    mov  ax,dseg;ตั้งค่า DS
    mov  ds,ax
;   ตัวโปรแกรม
    mov  ax,4c00h    ;จบโปรแกรม
    int  21h
cseg  ends
end   start

```

8.2 รูปแบบของโปรแกรมภาษาแอสเซมบลีแบบใหม่

รูปแบบของโปรแกรมภาษาแอสเซมบลีที่ใช้ในตอนต้นนั้นเป็นรูปแบบเก่า ในปัจจุบันโปรแกรม assembler ส่วนใหญ่มีรูปแบบในการประกาศเซกเมนต์ต่าง ๆ ให้ง่ายขึ้น โดยใช้คุณสมบัติของ MACRO ต่าง ๆ โปรแกรมตัวอย่างแรกของเมื่อนำมาเขียนในรูปแบบใหม่จะได้เป็น

```

;
; This program prints the message "Hello world"
;
.model small
.dosseg
.data
msg1  db    'Hello world',10h,13h,'$'
.stack 100h
.code
start:
    mov  ax,@data
    mov  ds,ax
    mov  ah,9h
    mov  dx,offset msg1
    int  21h
    mov  ax,4c00h
    int  21h
end   start

```

จะสังเกตได้ว่าโปรแกรมกระชับรัดขึ้นมาก ข้อแตกต่างของโปรแกรมที่เขียนในรูปแบบใหม่คือชื่อของเซกเมนต์ต่าง ๆ จะถูกกำหนดให้โดยอัตโนมัติ จะสังเกตได้ว่าในส่วนของกำหนัดค่า DS ใช้ชื่อของเซกเมนต์ข้อมูลว่า @data เป็นต้น

8.3 การเรียกใช้บริการของ DOS

สามารถเรียกใช้บริการต่าง ๆ ของ DOS ได้โดยผ่านทางคำสั่งจังหวะหมายเลข 21h DOS ได้จัดสรรบริการ (function) ต่าง ๆ มากมายให้กับผู้เขียนโปรแกรม. เมื่อเรียกใช้บริการจะต้องระบุว่าการบริการใด. ระบุโดยกำหนดค่าหมายเลขของบริการลงในรีจิสเตอร์ AH พร้อมทั้งข้อมูลต่าง ๆ ของการเรียกใช้บริการนั้น (พารามิเตอร์ต่าง ๆ) รูปแบบคร่าว ๆ ของการเรียกใช้บริการของ DOS เป็นดังนี้

```
mov ah,function_number ;(set function parameters)
int 21h
```

บริการต่าง ๆ ของ DOS ที่สำคัญ และพารามิเตอร์ของบริการต่าง ๆ มีดังต่อไปนี้

ตาราง 8.1 แสดงบริการของ DOS ที่สำคัญและพารามิเตอร์

หมายเลข	หน้าที่	พารามิเตอร์	หมายเหตุ
01h	รับค่าจากแป้นพิมพ์	Input : AH = 01h Output :AL = รหัสแอสกีของปุ่มที่กด	
02h	แสดงตัวอักษร	Input : AH = 02h DL = รหัสแอสกีของอักขรที่จะแสดง	
05h	พิมพ์ตัวอักษรทางเครื่องพิมพ์	Input : AH = 05h DL = รหัสแอสกีของอักขรที่จะพิมพ์	
07h	อ่านตัวอักษรจากแป้นพิมพ์ แต่ไม่แสดงผล (ไม่ตรวจสอบ Ctrl-Break)	Input : AH = 07h Output :AL = รหัสแอสกีของอักขรที่อ่านได้	
08h	อ่านตัวอักษรจากแป้นพิมพ์ แต่ไม่แสดงผล (ตรวจสอบ Ctrl-Break)	Input : AH = 07h Output :AL = รหัสแอสกีของอักขรที่อ่านได้	
09h	พิมพ์ข้อความ	Input : AH = 09h DS:DX = ตำแหน่งของข้อความที่ต้องการพิมพ์ ข้อความจบด้วยอักขร '\$'	การประกาศข้อมูลในหน่วยความจำจะอธิบายในบทถัดไป
0Ah	อ่านข้อความ	Input : AH = 0Ah DS:DX = ตำแหน่งของบัฟเฟอร์เก็บข้อมูล.	รูปแบบของบัฟเฟอร์และการใช้บริการนี้จะอธิบายในบทถัดไป
4Ch	จบโปรแกรม	Input : AH = 4Ch AL = รหัสที่จะส่งคือสู่ระบบ	

8.4 ขั้นตอนการแปลโปรแกรม

จะต้องแปลโปรแกรมที่เขียนขึ้นให้อยู่ในรูปแบบที่สามารถทำงานได้ โดยขั้นตอนการแปลโปรแกรมเป็นดังนี้

1. แปลโปรแกรมเป็นแฟ้มเป้าหมาย (object file) นามสกุล OBJ โดยใช้โปรแกรม assembler ต่าง ๆ เช่น Macro Assembler (MASM) หรือ Turbo Assembler (TASM)
2. นำมาแฟ้มเป้าหมายแฟ้มเดียวหรือหลายแฟ้มมาเชื่อมโยงเข้าด้วยกันโดยใช้โปรแกรม LINK.

ตัวอย่างการแปลโปรแกรม

จากโปรแกรมตัวอย่าง สมมติว่าเก็บในแฟ้มชื่อ EX1.ASM สามารถสั่งแปลโปรแกรมโดยใช้ Macro Assembler ได้ดังนี้

```
A:\>masm ex1;
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1991. All rights reserved.

Invoking: ML.EXE /I. /Zm /c /Ta ex1.asm

Microsoft (R) Macro Assembler Version 6.00
Copyright (C) Microsoft Corp 1981-1991. All rights reserved.

Assembling: ex1.asm
```

ถ้าโปรแกรมมีข้อผิดพลาด assembler จะแจ้งข้อผิดพลาดกลับมาให้ทราบ สามารถแก้ไขและแปลโปรแกรมใหม่ได้ เมื่อแปลโปรแกรมภาษาแอสเซมบลีเรียบร้อยแล้ว จะได้แฟ้มเป้าหมายที่มีนามสกุลเป็น OBJ เช่นจากตัวอย่างจะได้ EX1.OBJ จะให้โปรแกรม LINK เพื่อแปลแฟ้มเป้าหมาย (Object file) ให้เป็นโปรแกรมที่สามารถทำงานได้ ดังนี้

```
A:\>link ex1;
Microsoft (R) Segmented-Executable Linker Version 5.13
Copyright (C) Microsoft Corp 1984-1991. All rights reserved.
```

จะได้แฟ้มที่มีนามสกุล EXE ซึ่งสามารถเรียกใช้ได้จาก DOS prompt

ตัวอย่างโปรแกรม

ตัวอย่างที่ 1

โปรแกรมนี้รับการกดปุ่มจากผู้ใช้โดยใช้บริการหมายเลข 01h แล้วแสดงอักขระที่อ่านได้โดยใช้บริการของ DOS หมายเลข 02h สังเกตว่าโปรแกรมนี้ไม่มีการใช้ข้อมูลในหน่วยความจำ ดังนั้นจึงไม่ต้องประกาศเซกเมนต์ข้อมูล

```

;Ex1
.model small
.dosseg
.stack 100h
.code
start:
    mov     ah,01h        ;read character (Function 01h)
    int     21h
    mov     dl,al        ;copy character to DL
    mov     ah,02h        ;display it (Function 02h)
    int     21h
    mov     ax,4C00h     ;Exit (Function 4Ch)
    int     21h
end start

```

ตัวอย่างที่ 2

โปรแกรมนี้รับการกดปุ่มจากผู้ใช้โดยใช้บริการหมายเลข 01h แล้วแสดงอักขระที่มีรหัสแอสกีถัดจากอักขระที่อ่านได้ การแสดงตัวอักษรใช้บริการของ DOS หมายเลข 02h เช่นเดียวกับตัวอย่างที่ 1 โปรแกรมนี้ไม่มีการใช้ข้อมูลในหน่วยความจำจึงไม่มีการประกาศเซกเมนต์ข้อมูล โปรแกรมนี้เขียนโดยใช้รูปแบบในการเขียนแบบเก่า

```

;Ex2
sseg segment stack
    db    100 dup (?)
sseg ends

cseg segment
    assume cs:cseg,ss:sseg
start:
    mov     ah,01h        ;read character (Function 01h)
    int     21h

```

```

        mov     dl,al           ;copy to DL
        inc     dl             ;increase DL (next char.)
        mov     ah,02h        ;display it (Function 02h)
        int     21h
        mov     ax,4C00h      ;Exit
        int     21h
cseg ends
        end     start

```

ตัวอย่างที่ 3

โปรแกรมนี้รับตัวอักษรจากผู้ใช้นั้นแปลงตัวอักษรเล็กให้เป็นตัวอักษรใหญ่โดยการลบค่ารหัสแอสกีด้วย 32 แล้วแสดงอักษรนั้นกับผู้ใช้นั้น

```

.model small
.dosseg

.stack 100h

.code
start:
        mov     ah,01h        ;read char.
        int     21h
        mov     dl,al
        sub     dl,32         ;change char. case
        mov     ah,02h        ;display it
        int     21h
        mov     ax,4C00h      ;exit
        int     21h
end     start

```

ตัวอย่างที่ 4

โปรแกรมนี้ทำงานเหมือนโปรแกรมในตัวอย่างที่ 3 แต่ไม่แสดงอักขรที่ผู้ใช้ป้อนให้ผู้ใช้เห็น โดยใช้บริการหมายเลข 08h แทนบริการหมายเลข 01h ในตัวอย่างที่ 3

```
sseg segment stack
    db 100 dup (?)
sseg ends

cseg segment
    assume cs:cseg,ss:sseg
start:
    mov ah,08h      ; read char (Function 08h)
    int 21h
    mov dl,al
    sub dl,32      ; Change case
    mov ah,02h
    int 21h
    mov ax,4C00h   ; exit
    int 21h
cseg ends
end start
```

8.5 ตารางรหัสแอสกี

ตารางที่ 8.5 แสดงตารางรหัสแอสกี (ASCII: American Standard Code for Information Interchange)

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	'	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	

รหัสหมายเลข 0 - 31 เป็นรหัสควบคุม รหัส 32 คือช่องว่าง

สรุป

ภาษาแอสเซมบลี (Assembly Language) เป็นภาษาที่ใช้สัญลักษณ์ในการสื่อสารความหมาย ภาษาแอสเซมบลีมีลักษณะคำสั่ง ที่ขึ้นกับเครื่องคอมพิวเตอร์ที่ใช้งานและมีการแปลคำสั่งให้เป็นภาษาเครื่อง นอกจากภาษาเครื่อง และ ภาษาแอสเซมบลีแล้ว ก็ยังมีภาษาระดับสูง เช่น Basic Cobol Fortran ซึ่งเป็นภาษาที่มีคำสั่งใกล้เคียงกับภาษาอังกฤษมากทำให้ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมได้สะดวกและรวดเร็ว แต่ว่าโปรแกรมที่เขียนด้วยภาษาระดับสูงต้องใช้เนื้อที่เก็บในหน่วยความจำเป็นจำนวนมาก อีกทั้งทำงานได้ช้ากว่า ภาษาแอสเซมบลี ดังนั้นภาษาระดับสูงจึงไม่นิยมนำมาประยุกต์ใช้กับการทำงานที่ระบบการควบคุมที่มีความสำคัญมาก

ภาษาแอสเซมบลี เหมาะกับโปรแกรมที่ใช้เนื้อที่ในหน่วยความจำไม่มากนัก ทั้งทำงานได้รวดเร็ว และในการควบคุมการทำงานของเครื่องคอมพิวเตอร์ได้โดยตรง

คำสั่งปฏิบัติการของภาษาแอสเซมบลี แบ่งออกเป็น 4 ชนิดคือ

1. Machine instruction เป็นคำสั่งที่ทำให้เกิดการปฏิบัติการ (execution) ชุดของคำสั่งอยู่ใน assembler's instruction
2. Assembler instruction เป็นคำสั่งที่บอกแอสเซมเบลร์ให้ทำการระหว่าง Assembly source program
3. Macro instruction เป็นคำสั่งที่บอกแอสเซมเบลร์ให้ดำเนินการกับชุดของคำสั่งที่ได้บอกไว้ก่อนแล้ว ซึ่งจากชุดของคำสั่งนี้ แอสเซมเบลร์จะผลิตชุดของคำสั่งซึ่งต่อไปจะดำเนินการเหมือนหนึ่งว่าชุดของคำสั่งนี้เป็นส่วนหนึ่งของ source program แต่เริ่มแรก
4. Pseudo instruction เป็นคำสั่งที่บอกให้แอสเซมเบลร์รู้ว่า ควรปฏิบัติการเช่นไรกับข้อมูลการ branch อย่างมีข้อแม้ แมคโคและ listing ซึ่งปกติแล้วคำสั่งเหล่านี้จะไม่ผลิตคำสั่งภาษาเครื่องให้

คำถามทบทวน

1. โปรแกรมที่ทำงานในเครื่องคอมพิวเตอร์ซึ่งใช้หน่วยประมวลผลตระกูล 80x86 นั้นจะมีการแบ่งโปรแกรมทั้งหมดเป็นเซกเมนต์ย่อย ๆ อะไรบ้าง
2. จงอธิบายขั้นตอนการแปลโปรแกรมในภาษาแอสเซมบลี
3. จงแสดงรูปแบบวิธีการเขียนโปรแกรมภาษาแอสเซมบลีแบบใหม่
4. แฟ้มเป้าหมาย (Object file) คืออะไรและมีความสัมพันธ์กันอย่างไรกับการเขียนโปรแกรมภาษาแอสเซมบลี
5. จงเขียนโปรแกรมนี้รับตัวอักษรจากผู้ใช้นั้นแปลงตัวอักษรใหญ่ให้เป็นตัวอักษรเล็กออกทางหน้าจอคอมพิวเตอร์ (Display on Screen)

แผนบริหารการสอนประจำบทที่ 9

หัวข้อเนื้อหา

- การประกาศข้อมูล
- การอ้างใช้ข้อมูลที่ประกาศไว้
- การอ้างตำแหน่งของข้อมูล
- การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah
- การใช้บริการของ DOS หมายเลข 09h : การอ่านพิมพ์ข้อความ
- การใช้บริการของ DOS หมายเลข 0Ah : การอ่านข้อความ

วัตถุประสงค์เชิงพฤติกรรม

- เข้าใจรูปแบบการประกาศข้อมูลหรือตัวแปรในโปรแกรมภาษาแอสเซมบลี
- สามารถและเข้าใจการอ้างใช้ข้อมูลหรือตัวแปรที่ประกาศไว้
- สามารถและเข้าใจฟังก์ชันหมายเลข 09h และ 0Ah ของ DOS ว่าเป็นฟังก์ชันที่ทำงานอย่างไร

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 9 การประกาศข้อมูล (Data Declared)

ในบทที่แล้วได้ศึกษาเกี่ยวกับรูปแบบของการเขียนโปรแกรมภาษาแอสเซมบลีและการประกาศเซกเมนต์แล้ว ในบทนี้จะศึกษาเกี่ยวกับการประกาศข้อมูลภายในเซกเมนต์ข้อมูล และการใช้บริการหมายเลข 09h และ 0Ah ของระบบปฏิบัติการ DOS ในการแสดงผลข้อความและการอ่านข้อความจากผู้ใช้

9.1 การประกาศข้อมูล

การประกาศข้อมูลหรือตัวแปรในโปรแกรมภาษาแอสเซมบลีนั้น ทำได้โดยประกาศจองเนื้อที่ในหน่วยความจำในเซกเมนต์ข้อมูล แล้วตั้งเลเบลของข้อมูลนั้นไว้ ในการอ้างถึงข้อมูลในหน่วยความจำตำแหน่งนั้นสามารถอ้างโดยใช้เลเบลที่ประกาศไว้ได้ ดังนั้นการประกาศตัวแปรหรือข้อมูลนั้นจะมีลักษณะเช่นเดียวกับการประกาศเลเบลนั่นเอง

คำสั่งเทียมในการประกาศข้อมูล

คำสั่งเทียมที่ใช้ในการประกาศข้อมูลมีหลายคำสั่ง ดังตารางที่ 9.1 แสดงคำสั่งเทียมเหล่านี้ใช้ในระบุนขนาดในการจองหน่วยความจำ

ตารางที่ 9.1 แสดงคำสั่งเทียมสำหรับการระบุขนาดข้อมูลในการจองหน่วยความจำ

คำสั่งเทียม	ความหมาย
DB	Define Byte : ประกาศจองข้อมูลให้มีขนาดหน่วยละ 1 ไบต์
DW	Define Word : ประกาศจองข้อมูลให้มีขนาดหน่วยละ 1 เวิร์ด (2 ไบต์)
DD	Define Doubleword : ประกาศจองข้อมูลให้มีขนาดหน่วยละ 2 เวิร์ด
DQ	Define Quadword : ประกาศจองข้อมูลให้มีขนาดหน่วยละ 4 เวิร์ด
DT	Define Ten Bytes : ประกาศจองข้อมูลให้มีขนาดหน่วยละ 10 ไบต์

ในการประกาศจองข้อมูลนี้จะทำให้ assembler กันเนื้อที่ในเซกเมนต์นั้นตามข้อมูลที่ระบุตามหลังคำสั่งเทียมเหล่านี้ โดยจะกันหน่วยความจำที่มีขนาดของแต่ละหน่วยตามที่ระบุในคำสั่ง

รูปแบบของการประกาศข้อมูล

ในการประกาศข้อมูล (ตัวแปร) มักประกาศในเซกเมนต์ข้อมูล โดยจะระบุชื่อของตัวแปรนั้น พร้อมทั้งคำสั่งเทียมที่ใช้ระบุขนาดของข้อมูล จากนั้นจะระบุข้อมูลต่าง ๆ ที่จะใช้ตำแหน่งที่จะจองนั้น รูปแบบในการระบุเป็นดังนี้

```
variable_name Dx    data
```

ส่วนของโปรแกรมที่ 9.1 การประกาศข้อมูล

ตัวอย่างการประกาศข้อมูล

จากการประกาศข้อมูลในส่วนของโปรแกรมที่ 8.2 จะมีการจัดสรรเนื้อที่ในหน่วยความจำดังรูปที่ 9.1 สังเกตว่าในการประกาศ data1 กับ data2 นั้นการระบุข้อมูลเหมือนกันแต่ขนาดของข้อมูลต่างกัน ทำให้การจองเนื้อที่ในหน่วยความจำแตกต่างกันด้วย

```
dseg segment
data1 db 1,2
data2 dw 1,2
data3 db 'Hi',10,13
data4 dd 1234h
dseg ends
```

ส่วนของโปรแกรมที่ 9.2 ตัวอย่างการประกาศข้อมูล

DS:00h	01h	data1
DS:01h	02h	
DS:02h	01h	data2
DS:03h	00h	
DS:04h	02h	
DS:05h	00h	data3
DS:06h	48h	
DS:07h	69h	
DS:08h	0Ah	data4
DS:09h	0Ch	
DS:0Ah	34h	
DS:0Bh	12h	
DS:0Ch	00h	
DS:0Eh	00h	

รูปที่ 9.1 แสดงการจัดเรียงข้อมูลในหน่วยความจำจากการประกาศในส่วนของโปรแกรมที่ 9.2

การระบุไม่ระบุค่าของข้อมูลที่จองเนื้อที่

สามารถประกาศจองหน่วยความจำโดยไม่ระบุค่าเริ่มต้นได้โดยการระบุค่าเป็น '?' ดังเช่นในส่วนของโปรแกรมที่ 9.3 จะมีการจองเนื้อที่ไว้แต่ไม่มีการกำหนดค่าเริ่มต้น

```
data5 db ?
data6 dw ?
```

ส่วนของโปรแกรมที่ 9.3 การใช้จงหน่วยความจำโดยไม่ระบุค่าเริ่มต้น

การประกาศข้อมูลที่ซ้ำกัน

สามารถใช้คำสั่งเทียม `dup` เพื่อบอกการซ้ำกันของข้อมูลได้ รูปแบบของคำสั่งเทียม `dup` มีดังนี้

```
count dup (value)
```

ตัวอย่างของการประกาศที่ใช้คำสั่งเทียม `dup` ดังเช่นในส่วนของโปรแกรมที่ 9.4

<code>data7</code>	<code>db</code>	<code>10 dup (0)</code>
<code>data8</code>	<code>db</code>	<code>5 dup (4 dup (0))</code>
<code>data9</code>	<code>dw</code>	<code>5 dup (1, 2, 3 dup (4))</code>
<code>data10</code>	<code>db</code>	<code>20 dup (?)</code>

ส่วนของโปรแกรมที่ 9.4 การใช้คำสั่งเทียม `dup`

Assembler จะจองหน่วยความจำขนาด 10 ไบต์ ที่มีค่าเป็น 0 และจะให้เลเบล `data7` ชี้ไปที่ตำแหน่งเริ่มต้นของข้อมูลนี้ ในส่วนของ `data8` จะเป็นข้อมูลแบบไบต์จำนวน `4x5` ไบต์ ที่มีค่าเท่ากับ 0 เช่นเดียวกัน สังเกตว่าภายในเครื่องหมายวงเล็บของคำสั่งเทียม `dup` สามารถใส่ข้อมูลได้หลายค่า รวมทั้งกำหนดค่าแบบซ้ำกัน โดยใช้คำสั่ง `dup` อีกได้ ดังเช่นตัวแปร `data9` ในตัวแปร `data10` เป็นการประกาศจองหน่วยความจำไว้โดยไม่ระบุค่าเริ่มต้น

9.2 การอ้างใช้ข้อมูลที่ประกาศไว้

ในการอ้างใช้ข้อมูลหรือตัวแปรที่ประกาศไว้ สามารถอ้างโดยใช้ชื่อของเลเบลที่ประกาศไว้ได้ Assembler จะจัดการนำตำแหน่งของข้อมูลนั้นมาแทนค่าให้โดยอัตโนมัติ ยังสามารถอ้างค่าในหน่วยความจำโดยอ้างสัมพันธ์กับเลเบลที่กำหนดขึ้นได้ ส่วนของโปรแกรมที่ 9.5 เป็นโปรแกรมที่อ้างใช้ค่าของตัวแปรที่กำหนดในส่วนของโปรแกรมที่ 9.2 โดยหลังจากการทำงานของโปรแกรมค่าในหน่วยความจำจะเปลี่ยนไปตามรูปที่ 9.2

DS:00h	00h	data1
DS:01h	22h	
DS:02h	01h	data2
DS:03h	00h	
DS:04h	23h	
DS:05h	11h	
DS:06h	48h	data3
DS:07h	69h	
DS:08h	0Ah	
DS:09h	0Ch	
DS:0Ah	34h	data4
DS:0Bh	12h	
DS:0Ch	00h	
DS:0Eh	00h	

รูปที่ 9.2 แสดงการเปลี่ยนแปลงค่าหลังการทำงานของโปรแกรมที่ 9.5

```

mov    al,data1
mov    bx,data2
mov    data1,0
mov    [data2+2],1123h
mov    data1[1],22h
mov    cl,byte ptr data4[2]

```

ส่วนของโปรแกรมที่ 9.5 ตัวอย่างการเรียกใช้ตัวแปร

ค่าในรีจิสเตอร์ AL BX และ CL มีค่าเป็น 01h 01h และ 00h ตามลำดับ สังเกตว่าในการกำหนดค่าคงที่ให้กับตัวแปรในหน่วยความจำกระทำได้ทันทีโดยไม่ต้องระบุขนาด เนื่องจากในการประกาศตัวแปรได้ระบุกับ assembler แล้วว่าจะเป็นตัวแปรขนาดเท่าใด แต่ในกรณีที่ต้องการจะอ้างแตกต่างจากที่ระบุก็สามารถทำได้โดยต้องระบุขนาดของข้อมูลกำกับด้วย เช่นในคำสั่ง `mov cl, byte ptr data4[2]` เป็นการอ้างข้อมูลแบบ 8 บิต เพราะ CL เป็นรีจิสเตอร์ขนาด 8 บิต

9.3 การอ้างตำแหน่งของข้อมูล

สามารถอ้างถึงออฟเซตของข้อมูลที่ประกาศไว้ได้โดยใช้คำสั่งเทียม OFFSET ดังส่วนของโปรแกรมที่ 9.6

```

mov    bx,offset data1           ;bx = offset
mov    byte ptr [bx],10h
mov    bx,data2                 ;bx = value at data2

```

ส่วนของโปรแกรมที่ 9.6 การอ้างตำแหน่งของข้อมูล

การอ้างตำแหน่งข้อมูลโดยคิสัมพันธ์กับรีจิสเตอร์ BX

นอกจากการระบุตำแหน่งสัมพันธ์กับเลเบลโดยใช้ค่าคงที่แล้ว สามารถระบุตำแหน่งของข้อมูลสัมพันธ์กับเลเบลโดยใช้ค่าจากรีจิสเตอร์ BX ได้ ตัวอย่างเช่นส่วนของโปรแกรมที่ 9.7

```
mov bx,0
mov ah,data3[bx]      ;ah=data3[0]
inc bx
mov cl,data3[bx]      ;cl=data3[1]
mov bx,offset data3
mov dl,[bx+2]         ;dl=data3[2]
```

ส่วนของโปรแกรมที่ 9.7 การอ้างตำแหน่งของข้อมูลสัมพันธ์กับเลเบลโดยใช้ค่าจากรีจิสเตอร์ BX

ในคำสั่ง mov ก่อนบรรทัดที่ 5 อ้างหน่วยความจำโดยสัมพันธ์กับ data3 และค่าใน BX แต่ในคำสั่ง mov บรรทัดสุดท้ายของส่วนของโปรแกรมที่ 9.7 อ้างหน่วยความจำสัมพันธ์กับ BX ซึ่งเก็บออฟเซตของ data3

9.4 การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah

ฟังก์ชันหมายเลข 09h และ 0Ah ของ DOS เป็นฟังก์ชันที่ต้องมีการส่งแอดเดรสของข้อมูลในหน่วยความจำ การประกาศข้อมูลสำหรับฟังก์ชันหมายเลข 09h จะไม่มีความซับซ้อนมากนัก แต่สำหรับฟังก์ชันหมายเลข 0Ah การประกาศข้อมูลที่เหมาะสมจะทำให้เขียนโปรแกรมได้ง่ายมากขึ้น

9.5 การใช้บริการของ DOS หมายเลข 09h : การพิมพ์ข้อความ

ฟังก์ชันหมายเลข 09h นี้รับข้อมูลป้อนเข้าคือ

AH = 09h

DS : DX = ตำแหน่งของหน่วยความจำของข้อมูลที่จะแสดง โดยข้อมูลนี้จะต้องจบด้วยอักขระ '\$'

ถ้าต้องการพิมพ์ข้อความ "Hello world" สามารถประกาศข้อมูลในหน่วยความจำได้ดังนี้

```
dseg segment
mesg db 'Hello world',10,13,'$'
dseg ends
```

ส่วนของโปรแกรมที่ 9.8 ตัวอย่างการประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h

สามารถสั่งแสดงข้อมูลดังกล่าวได้โดย

```
mov ah,09h
mov dx,offset mesg
int 21h
```

ส่วนของโปรแกรมที่ 9.9 ตัวอย่างการการใช้บริการของ DOS หมายเลข 09h

อักขระหมายเลข 10 (Line feed) และ 13 (Carriage Return) คือรหัสควบคุมใช้ในการสั่งให้ขึ้นบรรทัดใหม่

9.6 การใช้บริการของ DOS หมายเลข 0Ah : การอ่านข้อความ

ฟังก์ชันนี้จะอ่านข้อความจากผู้ใช้จนกระทั่งผู้ใช้กดปุ่ม Enter โดยข้อมูลป้อนเข้าจะต้องระบุตำแหน่งของหน่วยความจำที่ใช้เก็บข้อมูล (บัฟเฟอร์) ของข้อความ ฟังก์ชันหมายเลข 0Ah นี้รับข้อมูลป้อนเข้าคือ

AH = 0Ah

DS : DX = ตำแหน่งของหน่วยความจำที่จะใช้เก็บข้อความ (บัฟเฟอร์)

บัฟเฟอร์จะต้องมีรูปแบบดังนี้

1. ไบต์แรกของหน่วยความจำเก็บค่าความยาวสูงสุดของข้อความที่อ่านได้ ความยาวนี้จะรวมรหัสขึ้นบรรทัดใหม่ด้วย

2. DOS จะเขียนความยาวจริงของข้อความที่อ่านเข้ามาได้ในไบต์ที่สอง

3. สำหรับไบต์ถัด ๆ ไปจะเป็นรหัสแอสกีของข้อความที่อ่านเข้ามา

การประกาศข้อมูลสำหรับการเรียกใช้ฟังก์ชันนี้จะสามารถประกาศได้ดังส่วนของโปรแกรมที่ 9.10

```
dseg segment
maxlen db 30 ;Maximum of 30 chars
msglen db ? ;2nd byte contains the real length
msg db 30 dup (?) ;Message recieved
dseg ends
```

ส่วนของโปรแกรมที่ 9.10 ตัวอย่างการประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 0Ah

เมื่อเรียกใช้บริการหมายเลข 0Ah จะส่งตำแหน่งของ maxlen ซึ่งเป็นตำแหน่งเริ่มต้นของบัฟเฟอร์ที่ประกาศไปให้กับ DOS จากนั้นสามารถอ่านความจริงของข้อความที่อ่านมาได้ทางตัวแปร msglen ตัวอย่างโปรแกรมที่ 9.11 แสดงการใช้งานบริการหมายเลข 0Ah ในการอ่านข้อความและแสดงข้อความนั้นออกมาโดยใช้บริการหมายเลข 09h

ในการรับข้อความนั้นบริการหมายเลข 0Ah จะเก็บอักขระขึ้นบรรทัดใหม่ให้ด้วย ดังนั้นจะต้องเพื่อขนาดบัฟเฟอร์ที่จะให้เก็บข้อความไว้ 1 ไบต์ด้วย แต่ในการคืนค่าความยาวของข้อความมาให้ บริการหมายเลข 0Ah นี้จะใส่ความยาวที่ไม่รวมอักขระขึ้นบรรทัดใหม่นี้ เมื่อรับข้อความเสร็จแล้ว เคอร์เซอร์จะอยู่ที่ต้นบรรทัดที่ป้อนข้อความนั้น ดังนั้นถ้าพิมพ์ข้อความเดิมซ้ำไปอีกครั้งจะทำให้ข้อความทับกันและไม่ทราบว่ามีกรพิมพ์ข้อความออกมาอย่างถูกต้องหรือไม่ ดังนั้นจึงใช้ฟังก์ชันหมายเลข 09h ส่งพิมพ์ชุดอักขระสำหรับการขึ้นบรรทัดใหม่ก่อนที่จะส่งพิมพ์ข้อความที่รับมา

ข้อความที่ส่งพิมพ์ด้วยบริการหมายเลข 09h จะต้องจบด้วยอักขระ '\$' ดังนั้นจึงต้องกำหนดค่าในไบต์สุดท้ายของข้อความที่รับมาด้วยอักขระ '\$' โปรแกรมนี้จะทำงานผิดพลาดถ้าภายในข้อความมีเครื่องหมาย '\$' อยู่ด้วย

```

;
; display string
;
dseg segment
;string buffer
maxlendb 30 ;29 chars + 1 return
msglendb ?
msg db 30 dup (?) ;29 chars + 1 return
;newline string
newline db 10,13,'$'
dseg ends
sseg segment stack
db 100h dup (?)
sseg ends
cseg segment
assume cs:cseg,ds:dseg,ss:sseg
start:
mov ax,dseg ;set DS
mov ds,ax
mov ah,0Ah ;read string
mov dx,offset maxlen
int 21h
mov ah,09h ;newline
mov dx,offset newline
int 21h
mov bl,msglen ;get string length
mov bh,0
mov msg[bx],'$' ;terminate string
mov ah,09h ;display it
mov dx,offset msg
int 21h
mov ax,4C00h ;bye-bye
int 21h
cseg ends
end start

```

โปรแกรมที่ 9.11 แสดงการรับข้อความและแสดงข้อความนั้นกลับมา

สรุป

การประกาศข้อมูลหรือตัวแปรในโปรแกรมภาษาแอสเซมบลีนั้น ทำได้โดยประกาศจองเนื้อที่ในหน่วยความจำในเซกเมนต์ข้อมูล แล้วตั้งเลเบลของข้อมูลนั้นไว้ ในการอ้างถึงข้อมูลในหน่วยความจำตำแหน่งนั้นสามารถอ้างโดยใช้เลเบลที่ประกาศไว้ได้ ดังนั้นการประกาศตัวแปรหรือข้อมูลนั้นจะมีลักษณะเช่นเดียวกับการประกาศเลเบลนั่นเอง ส่วนในการอ้างใช้ข้อมูลหรือตัวแปรที่ประกาศไว้ สามารถอ้างโดยใช้ชื่อของเลเบลที่ประกาศไว้ได้ Assembler จะจัดการนำตำแหน่งของข้อมูลนั้นมาแทนค่าให้โดยอัตโนมัติ ยังสามารถอ้างค่าในหน่วยความจำโดยอ้างสัมพันธ์กับเลเบลที่กำหนดขึ้นได้

การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah ฟังก์ชันหมายเลข 09h และ 0Ah ของ DOS เป็นฟังก์ชันที่ต้องมีการส่งแอดเดรสของข้อมูลในหน่วยความจำ การประกาศข้อมูลสำหรับฟังก์ชันหมายเลข 09h จะไม่มีความซับซ้อนมากนัก แต่สำหรับฟังก์ชันหมายเลข 0Ah การประกาศข้อมูลที่เหมาะสมจะทำให้เขียนโปรแกรมได้ง่ายมากขึ้น การใช้บริการของ DOS ฟังก์ชันหมายเลข 0Ah จะเกี่ยวข้องกับการอ่านข้อความส่วนฟังก์ชันหมายเลข 09h จะเกี่ยวข้องกับการพิมพ์ข้อความ

คำถามทบทวน

- จงอธิบายความหมายของคำสั่งเทียมสำหรับการระบุขนาดข้อมูลในการจองหน่วยความจำต่อไปนี้
DB, DW, DD, DQ, และ DT
- จงแสดงวิธีการจัดเรียงข้อมูลในหน่วยความจำจากการประกาศในส่วนของโปรแกรมที่แสดงอยู่ข้างล่างนี้


```
dseg      segment
data1     dw   1,2
data2     dw   1,2
data3     db   'Cs',14,15
data4     dd   3456h
dseg      ends
```
- คำสั่งเทียม dup มีไว้เพื่ออะไร
- อักขระหมายเลข 10 (Line feed) และ 13 (Carriage Return) หมายถึงอะไร
- จงอธิบายการประกาศข้อมูลแต่ละบรรทัดสำหรับการเรียกใช้ฟังก์ชันจากส่วนของโปรแกรมที่แสดงอยู่ข้างล่างนี้


```
dseg      segment
maxlen    db   60
msglen    db   ?
msg       db   40 dup (?)
dseg      ends
```
- ถ้าต้องการพิมพ์ข้อความ "Computer Science SDU" สามารถประกาศข้อมูลในหน่วยความจำได้อย่างไร
- ข้อความที่สั่งพิมพ์ด้วยบริการหมายเลข 09h จะต้องจบด้วยอักขระใดเสมอ

แผนบริหารการสอนประจำบทที่ 10

หัวข้อเนื้อหา

- คำสั่งกระโดด
- คำสั่งวนรอบ

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับคำสั่งกระโดดแบบไม่มีเงื่อนไข คำสั่งกระโดดที่พิจารณาค่าจากแฟล็ก และคำสั่งกระโดดที่พิจารณาค่าจากรีจิสเตอร์
- มีความรู้และความเข้าใจเกี่ยวกับกลุ่มคำสั่งวนรอบ เช่น LOOP LOOPZ และ LOOPNZ เป็นต้น

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 10 คำสั่งกระโดดและการกระทำซ้ำ (Jump and Iteration Loop)

ในบทนี้จะศึกษาเกี่ยวกับเรื่องคำสั่งกระโดด และคำสั่งเกี่ยวกับการทำซ้ำ เป้าหมายของบทนี้คือการนำคำสั่งเหล่านี้ไปใช้ในการสร้างโครงสร้างควบคุม (Control Structures) แบบต่าง ๆ ซึ่งจะศึกษาต่อไปในบทที่ 11

10.1 คำสั่งกระโดด

คำสั่งกระโดดเป็นคำสั่งที่สั่งให้หน่วยประมวลผลกระโดดไปทำงานที่ตำแหน่งอื่น รูปแบบทั่วไปของคำสั่งกระโดดคือ

Jxx label

คำสั่งกระโดดแบ่งได้เป็น 2 กลุ่ม คือ คำสั่งกระโดดแบบไม่มีเงื่อนไข และคำสั่งกระโดดแบบมีเงื่อนไข คำสั่งกระโดดแบบไม่มีเงื่อนไขคือคำสั่ง JMP ส่วนในกลุ่มของคำสั่งกระโดดแบบมีเงื่อนไขแบบคร่าว ๆ ออกเป็นสองกลุ่มคือกลุ่มซึ่งพิจารณาการกระโดดจากค่าในแฟล็ก และกลุ่มที่พิจารณาการกระโดดจากค่าในรีจิสเตอร์ ส่วนใหญ่คำสั่งกระโดดที่พิจารณาค่าในแฟล็กจะใช้ผลลัพธ์ที่ได้จากคำสั่ง CMP คำสั่งกระโดดต่าง ๆ สรุปได้ดังตารางที่ 10.1 แสดงคำสั่งกระโดดต่างๆ

คำสั่งกระโดด	ความหมาย	เงื่อนไข
คำสั่งกระโดดแบบไม่มีเงื่อนไข		
JMP	Jump	Always
คำสั่งกระโดดที่พิจารณาค่าจากแฟล็ก		
<i>Zero flag</i>		
JZ (JE)	Jump if Zero	ZF = 1
JNZ (JNE)	(Jump if Equal)	ZF = 0
	Jump if Not Zero	
	(Jump if Not Equal)	
<i>Overflow flag</i>		
JO	Jump if Overflow	OF = 1
JNO	Jump if Not Overflow	OF = 0
<i>Parity flag</i>		
JPO	Jump if Parity Odd	PF = 0
JPE	Jump if Parity Even	PF = 1

คำสั่งกระโดด	ความหมาย	เงื่อนไข
<i>Signs flag</i>		
JS	Jump if Sign	SF = 1
JNS	Jump if No Sign	SF = 0
<i>Carry flag</i>		
JC	Jump if Carry	CF = 1
JNC	Jump if No Carry	CF = 0
<i>Comparing unsigned numbers</i>		
JA (JNBE)	Jump if Above	(CF and ZF) = 0
JB (JNAE)	(Not Below or Equal)	CF = 1
JAE (JNB)	Jump if Below	CF = 0
JBE (JNA)	(Not Above or Equal)	(CF or ZF) = 1
	Jump if Above or Equal	
	(Not Below)	
	Jump if Below or Equal	
	(Not Above)	
<i>Comparing signed numbers</i>		
JG (JNLE)	Jump if Greater	ZF = 0 and SF = OF
JL (JNGE)	(Not Less or Equal)	SF <> OF
JGE (JNL)	Jump if Less	SF = OF
JLE (JNG)	(Not Greater or Equal)	(ZF = 1) of (SF <> OF)
	Jump if Greater or Equal	
	(Not Less)	
	Jump if Less or Equal	
	(Not Greater)	
<u>คำสั่งกระโดดที่พิจารณาค่าจาก</u>		
<u>รีจิสเตอร์</u>		
<i>Testing for CX</i>		
JCXZ	Jump if CX is equal to zero	CX = 0

คำสั่งต่าง ๆ เหล่านี้จะใช้ในการสร้างโครงสร้างควบคุมการทำงานของโปรแกรม โดยจะใช้ประกอบกับคำสั่ง CMP ดังที่ได้กล่าวมาแล้ว ยกเว้นคำสั่ง JCXZ จะนิยมใช้ประกอบกับกลุ่มคำสั่งประเภทการทำซ้ำ (LOOP) คำสั่งที่ใช้ควบคุมการกระโดดแบบมีเงื่อนไขที่ใช้การเปรียบเทียบระหว่างโอเพอร์แรนด์สองตัวของคำสั่ง

CMP มีสองกลุ่มคือ กลุ่มที่คิดการเปรียบเทียบเป็นการเปรียบเทียบของเลขไม่คิดเครื่องหมาย (JA JB JAE JBE) และกลุ่มที่คิดเป็นเลขคิดเครื่องหมาย (JG JL JGE JLE) ในการใช้คำสั่งกระโดดทั้งสองกลุ่มนี้จะต้องพิจารณาข้อมูลที่เปรียบเทียบกันด้วย

ตัวอย่าง

- (1)
- ```

cmp ah,10 ; เปรียบเทียบ ah กับ 10
jz lab1 ; ถ้าเท่ากันให้กระโดดไปที่ lab1
mov bx,2
lab1: add cx,10

```
- (2)
- ```

cmp    ah,10      ; เปรียบเทียบ ah กับ 10
jge    tenup     ; ถ้ามากกว่าหรือเท่ากับให้กระโดดไปที่ tenup
add    dl,'0'
jmp    endif     ; กระโดดไปที่ endif
tenup: add    dl,'A'
endif:

```
- (3)
- ```

getonechar:
mov ah,1 ; ใช้บริการหมายเลข 1 : อ่านอักขระ
int 21h
cmp al,'Q' ; เปรียบเทียบ al กับ 'Q'
jne getonechar ; ถ้าไม่เท่ากันให้กระโดดไปที่ getonechar (กลับไปรับตัวอักษรใหม่)

```
- (4)
- ```

mov    ah,02     ; บริการหมายเลข 2 : พิมพ์อักขระ
mov    dl,32     ; เริ่มที่ ช่องว่าง ASCII = 32 ( ' ')
printloop:
cmp    dl,128    ; เปรียบเทียบ dl กับ 128 (ASCII สุดท้าย)
ja     finish    ; ถ้ามากกว่ากระโดดไปที่ finish
int    21h      ; พิมพ์อักขระที่มี ASCII = dl
inc    dl        ; เพิ่ม dl
jmp    printloop
finish:

```

10.2 คำสั่งวนรอบ

คำสั่งที่ใช้กระทำซ้ำใน 8086 ยังมีอีกกลุ่มหนึ่งที่ใช้ค่าในรีจิสเตอร์ CX (Counter Register) ในการนับจำนวนครั้งของการทำงาน คำสั่งกลุ่มนี้คือ LOOP LOOPZ และ LOOPNZ

คำสั่ง LOOP

คำสั่ง LOOP จะลดค่าของรีจิสเตอร์ CX ลงหนึ่ง ถ้า CX มีค่าไม่เท่ากับศูนย์คำสั่ง LOOP จะกระโดดไปทำงานที่เลขที่ระบุ รูปแบบของคำสั่ง LOOP เป็นดังนี้

```
LOOP label
```

คำสั่ง LOOP จะลดค่าของรีจิสเตอร์ CX โดยไม่กระทบกับแฟล็ก นั่นคือคำสั่ง LOOP มีผลเหมือนคำสั่ง

```
DEC CX
```

```
JNZ label
```

แต่จะไม่มีผลกระทบต่อแฟล็ก

ตัวอย่างการใช้คำสั่ง LOOP

โปรแกรมตัวอย่างนี้คำนวณผลรวมของเลขตั้งแต่ 1 ถึง 20

```

mov    cx,20           ; ทำซ้ำ 20 ครั้ง
mov    bl,1           ; เริ่มจาก 1
mov    dx,0           ; กำหนดค่าเริ่มต้นให้กับผลรวม
addonumber:
add    dl,bl          ; บวก 8 บิตล่าง
adc    dh,0           ; รวมตัวทศ
inc    bl             ; ตัวถัดไป
loop   addonumber    ; ถ้า CX ลดลงแล้วไม่เท่ากับ 0 ทำซ้ำต่อไป

```

คำสั่ง JCXZ

ในกรณีที่ CX มีค่าเท่ากับศูนย์ก่อนการกระทำซ้ำโดยใช้คำสั่ง LOOP ผลลัพธ์จากการลดค่าจะมีค่าเท่ากับ 0FFFFh ทำให้จำนวนรอบของการทำงานไม่ถูกต้อง นิยมใช้คำสั่ง JCXZ ในการป้องกันความผิดพลาดในกรณีที่ค่าของรีจิสเตอร์ CX มีค่าเท่ากับ 0 โดยปกติถ้า CX มีค่าเท่ากับศูนย์ จะสั่งให้โปรแกรมกระโดดไปที่จุดสิ้นสุดการกระทำซ้ำ ดังตัวอย่าง

```

      initialization
      jcxz  endloop          ; CX =0 ?
label1:
      actions
      loop label1           ; loop
endloop:

```

คำสั่ง LOOPZ และ LOOPNZ

คำสั่ง LOOPZ และ LOOPNZ มีลักษณะทำงานเหมือนคำสั่ง LOOP แต่จะนำค่าของแฟล็กมาใช้ในการพิจารณาการกระโดดด้วย คำสั่ง LOOPZ จะลดค่าของรีจิสเตอร์ CX โดยไม่กระทบแฟล็กและจะกระโดดไปที่เลเบลที่ระบุเมื่อ CX มีค่าไม่เท่ากับศูนย์ และ แฟล็กศูนย์มีค่าเป็น 1 (ผลลัพธ์ของคำสั่งก่อนหน้านี้มีค่าเท่ากับศูนย์) สังเกตว่า คำสั่ง LOOPZ จะทำงานเหมือนคำสั่ง LOOP แต่แฟล็กศูนย์จะต้องมีค่าเป็นหนึ่งด้วย คำสั่งนี้ถึงจะกระโดดไปที่เลเบลที่กำหนด ในทำนองกลับกันคำสั่ง LOOPNZ จะกระโดดไปทำงานเมื่อ CX มีค่าไม่เท่ากับศูนย์ และแฟล็กศูนย์มีค่าเป็น 0 คำสั่งทั้งสองนิยมใช้ในการทำซ้ำที่ทราบจำนวนครั้งแต่มีเงื่อนไขในการทำซ้ำ

ตัวอย่างคำสั่ง LOOPZ และ LOOPNZ

ตัวอย่างนี้แสดงการใช้คำสั่ง LOOPNZ ในการค้นหาข้อมูลค่าหนึ่งจากชุดของข้อมูล ในตัวอย่างนี้จำนวนข้อมูลคือ 100 ค่า และข้อมูลที่ต้องการค้นหาเก็บที่รีจิสเตอร์ DX

```

.data
datalist      dw  100 dup (?)

.code
...
;

```

```

mov     bx,offset datalist      ; ให้ BX เก็บค่าตำแหน่งของ datalist
mov     cx,100                  ; ทำซ้ำ 100 ครั้ง
dec     bx,2                    ; ลดค่าของ BX เพราะใน loop มีการเพิ่มค่า

checkdata:
inc     bx,2                    ; BX ซ้ำไปยังข้อมูลตัวถัดไป
cmp     dx,[bx]                ; เปรียบเทียบ
loopnz  checkdata              ; ทำซ้ำถ้ายังไม่พบข้อมูลและยังไม่ครบข้อมูล
jz      found                  ; ค้นเจอข้อมูลกระโดดไปที่ found

; not found                    ; ไม่พบข้อมูล

...

found:
; found                        ; พบข้อมูล

...

```

ในตัวอย่างแรกนี้คำสั่ง DEC BX,2 ในโปรแกรมก่อนที่จะถึงส่วนที่ทำงานซ้ำเป็นการปรับค่าของ BX ให้สอดคล้องกับการปรับค่าในส่วนที่ทำงานซ้ำ การลดค่าของ BX ในกรณีนี้ทำให้การเปรียบเทียบครอบคลุมถึงค่าแรกของข้อมูลด้วย การใช้เทคนิคเช่นนี้ในโปรแกรมอาจทำให้โปรแกรมทำงานได้เร็วขึ้น แต่อาจทำให้ผู้อื่นที่มาอ่านโปรแกรมของเข้าใจผิดได้ ดังนั้นถ้าใช้เทคนิคต่าง ๆ ในโปรแกรม ควรใส่หมายเหตุให้ชัดเจน และโดยปกติยังสามารถใช้วิธีอื่นในการจัดการกับข้อมูลตัวแรกได้ ดังตัวอย่างถัดไป

ตัวอย่างนี้เป็นการค้นหาอักษรตัวแรกของข้อความที่ไม่ใช่ช่องว่าง โดยข้อความนี้มีความยาว 100 ตัวอักษร ในตัวอย่าง ตำแหน่งเริ่มต้นของข้อความเก็บอยู่ที่ BX

```

cmp     byte ptr [bx],' '      ; พิจารณาอักษรตัวแรก
jnz     found                  ; อักษรตัวแรกไม่ใช่ช่องว่าง
mov     cx,100                 ; ทำซ้ำ 100 ครั้ง

findnotspace:
inc     bx                    ; BX ซ้ำไปยังอักษรตัวถัดไป
cmp     byte ptr [bx],' '      ; เปรียบเทียบ
loopz   findnotspace          ; ทำซ้ำถ้ายังพบช่องว่างและยังไม่ครบข้อมูล
jnz     found                  ; ค้นเจออักษรตัวแรก

; not found                    ; ข้อความมีแต่ช่องว่าง

...

found:
; found                        ; พบอักษรตัวแรก

...

```

ตัวอย่างโปรแกรม

ตัวอย่างที่ 1

โปรแกรมตัวอย่างต่อไปนี้เป็นโปรแกรมที่พิมพ์ค่าของรหัสแอสกีที่รับมาเป็นเลขฐานสิบหก. การแสดงตัวเลขเป็นเลขฐานสิบหกนั้นมีความยุ่งยากเพราะอักษร '0' ถึง 'F' ที่จะใช้ในการแสดงค่านั้นมีรหัสแอสกีที่แยกออกเป็นสองช่วง. ช่วงแรกเป็นช่วงของตัวเลขเริ่มที่รหัส 48 ของเลขศูนย์ อีกกลุ่มหนึ่งคือช่วงของตัวอักษรเริ่มที่รหัส 65 ของตัว 'A'. ดังนั้นในการแสดงผลจะต้องตรวจสอบตัวเลขในแต่ละหลักว่าอยู่ในช่วงใดโดยใช้การเปรียบเทียบ

```

;
; display ASCII in HEX
;
.model small
.dosseg

.data
newline      db      10,13,'$'

.stack 100h

.code
start:
    mov     ax,@data           ;set DS
    mov     ds,ax
    mov     ah,01h           ;Function 1 : Read char
    int     21h              ;AL=ASCII
    mov     ah,0
    mov     bl,16             ;div AL by 16
    div     bl
    mov     cl,al             ;first digit
    mov     bl,ah             ;second digit
    mov     dx,offset newline ;display newline
    mov     ah,09h
    int     21h

```

```
        mov    dl,cl                ;print first digit
        cmp    cl,9
        ja     overnine1           ;above 9
        add    dl,'0'
        jmp    print1
overnine1:
        add    dl,'A'-10
print1:
        mov    ah,2
        int    21h
        mov    dl,bl                ;print second digit
        cmp    bl,9
        ja     overnine2
        add    dl,'0'
        jmp    print2
overnine2:
        add    dl,'A'-10
print2:
        mov    ah,2
        int    21h
        mov    ax,4C00h
        int    21h
end     start
```


ตัวอย่างที่ 2

ตัวอย่างนี้เป็นโปรแกรมที่รับข้อความจากผู้ใช้ และรับตัวอักษรที่ผู้ใช้ต้องการตรวจสอบว่ามีในข้อความหรือไม่ โปรแกรมจะแสดงคำตอบว่า YES หรือ NO โปรแกรมนี้ค้นหาตัวอักษรโดยใช้คำสั่ง LOOPNZ

```

; Find a character in a string
.model small
.dosseg

.data
maxlen db 30
strlen db ?
str db 30 dup (?)

msg1 db 'Enter string :$'
msg2 db 10,13,'Enter character to find :$'
yesmsg db 10,13,'YES',10,13,$'
nomsg db 10,13,'NO',10,13,$'

.stack 100h

.code
start:

    mov ax,@data
    mov ds,ax

    mov dx,offset msg1 ;disp msg1
    mov ah,09h ;func 9
    int 21h

    mov dx,offset maxlen
    mov ah,0Ah ;read string
    int 21h

    mov dx,offset msg2 ;disp msg2
    mov ah,09h
    int 21h

```

```

        mov     ah,01h           ;read char
        int     21h
        mov     dl,al           ;dl=al=char

        mov     bx,offset str
        mov     cl,strlen
        mov     ch,0
        jcxz    notfound
        cmp     [bx],al         ;first char
        jz      found
        dec     cx               ;first char was compared
        jcxz    notfound       ;so we dec cx
compareloop:
        inc     bx               ;next char
        cmp     [bx],al
        loopnz  compareloop    ;loop
        jz      found           ;found?
notfound:
        mov     dx,offset nomsg ;set dx
        jmp     print
found:
        mov     dx,offset yesmsg ;set dx
print:
        mov     ah,09h          ;display msg
        int     21h             ;using func 09h
        mov     ax,4C00h
        int     21h
end     start

```

สังเกตว่าโปรแกรมนี้ใช้คำสั่ง JCXZ ในการป้องกันกรณีที่ใช้ป้อนอักขรเพียงตัวเดียวหรือไม่ป้อนเลย. เลือกที่จะทดสอบตัวอักษรตัวแรก แทนที่จะใช้วิธีลดค่าของ BX ในการจัดการเกี่ยวกับข้อมูลตัวแรก

สรุป

คำสั่งกระโดดแบ่งได้เป็น 2 กลุ่ม คือ คำสั่งกระโดดแบบไม่มีเงื่อนไข และคำสั่งกระโดดแบบมีเงื่อนไข คำสั่งกระโดดแบบไม่มีเงื่อนไขคือคำสั่ง JMP ส่วนในกลุ่มของคำสั่งกระโดดแบบมีเงื่อนไขแบบคร่าว ๆ ออกเป็นสองกลุ่มคือกลุ่มซึ่งพิจารณาการกระโดดจากค่าในแฟล็ก และกลุ่มที่พิจารณาการกระโดดจากค่าในรีจิสเตอร์ ส่วนใหญ่คำสั่งกระโดดที่พิจารณาค่าในแฟล็กจะใช้ผลลัพธ์ที่ได้จากคำสั่ง CMP

คำถามทบทวน

1. จงอธิบายพร้อมยกตัวอย่างคำสั่งกระโดดแบบไม่มีเงื่อนไขและคำสั่งกระโดดแบบมีเงื่อนไขว่ามีอะไรบ้าง
2. จงอธิบายพร้อมยกตัวอย่างคำสั่ง CMP กลุ่มที่คิดการเปรียบเทียบเป็นการเปรียบเทียบของเลขแบบไม่คิดเครื่องหมายว่ามีอะไรบ้าง
3. จงอธิบายพร้อมยกตัวอย่างคำสั่ง CMP กลุ่มที่กลุ่มที่คิดเป็นเลขคิดเครื่องหมายว่ามีอะไรบ้าง
4. จงอธิบายการโปรแกรมตัวอย่างนี้คำนวณผลรวมของเลขตั้งแต่ 25 ถึง 100 แต่ละบรรทัดสำหรับการเรียกใช้ฟังก์ชันจากส่วนของโปรแกรมที่แสดงอยู่ข้างล่างนี้

```

mov    cx,100
mov    bl,25
mov    dx,0
addonenumber:
add    dl,bl
adc    dh,0
inc    bl
loop  addonenumber

```

แผนบริหารการสอนประจำบทที่ 11

หัวข้อเนื้อหา

- การสร้างโครงสร้างการตัดสินใจแบบ if-then-else
- การสร้าง repeat until loop
- การสร้าง while loop
- การสร้าง for loop

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับรูปแบบของโปรแกรมภาษาแอสเซมบลีที่มีโครงสร้างแบบต่าง ๆ เช่น if-then-else, repeat until loop, while loop และ for loop เป็นต้น

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 11 โครงสร้างควบคุม (Control Structure)

เราได้ศึกษาคำสั่งกระโดดในบทที่ 10 โดยคำสั่งกลุ่มนี้ทำให้เราสามารถเขียนโปรแกรมให้มีการทำงานที่ซับซ้อนขึ้นได้ แต่การใช้คำสั่งเหล่านี้อย่างไม่เป็นระบบทำให้โปรแกรมที่เขียนขึ้นนั้นทำความเข้าใจได้ยาก และมีลักษณะเหมือนเส้นสปาเก็ตตี้ได้ ในบทนี้เราจะพิจารณาการใช้คำสั่งกระโดดมาสร้างเป็นโครงสร้างควบคุมรูปแบบต่าง ๆ การใช้คำสั่งกระโดดในลักษณะนี้จะทำให้โปรแกรมของเรามีความเป็นโครงสร้างมากขึ้น

11.1 การสร้างโครงสร้างการตัดสินใจแบบ if-then-else

รูปแบบของโครงสร้างที่ง่ายที่สุดคือโครงสร้างแบบ if-then-else รูปแบบของโปรแกรมภาษาแอสเซมบลีให้มีโครงสร้างแบบ if-then-else มีลักษณะดังนี้

```

    if condition is false then jump to elselabel
        then_actions
        jump to endif_label
elselabel:
    else_actions
endif_label:

```

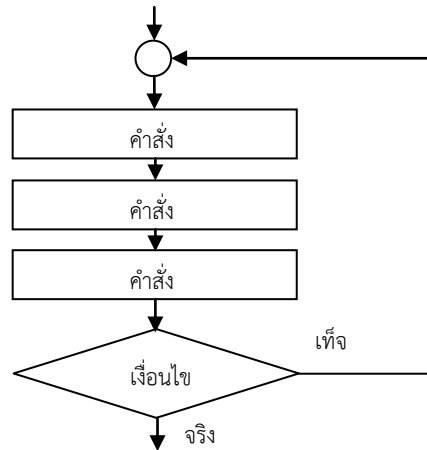
ตัวอย่างของโปรแกรมมีลักษณะโปรแกรมที่ 11.1

(a)	(b)
<pre> if AL<10 then DL:=AL+'0' else DL:=AL+'A'-10; </pre>	<pre> cmp al,10 jae abovenine mov dl,al add dl,'0' jmp endif abovenine: mov dl,al add al,'A'-10 endif: </pre>

โปรแกรมที่ 11.1 โปรแกรม (a) แสดงตัวอย่างของโปรแกรมที่เขียนด้วยโครงสร้างของภาษา pascal
โปรแกรม (b) แสดงโปรแกรมภาษาแอสเซมบลีที่เทียบเท่ากัน

11.2 การสร้าง repeat until loop

โครงสร้างของ repeat until loop ในภาษาระดับสูงทั่วไปมีลักษณะดังรูปที่ 11.2



รูปที่ 11.2 แสดงโครงสร้างควบคุมแบบ repeat until

เราสามารถเขียนโปรแกรมโดยใช้ภาษาแอสเซมบลีโดยมีโครงสร้างแบบ repeat until ได้ ดังตัวอย่าง

(a)

```

BL:=1;
CX:=0;
DX:=0;
Repeat
DX:=DX+BL*BL;

BL:=BL+1;
CX:=CX+1;
until (DX>100);
  
```

(b)

```

mov bl,1
mov cx,0
mov dx,0
startloop:
mov al,bl
mul bl ;result in ax
add dx,ax
inc bx
inc cx
cmp dx,100
jbe startloop
  
```

โปรแกรมที่ 11.2 โปรแกรม (a) แสดงตัวอย่างของโปรแกรมภาษา Pascal ที่ใช้โครงสร้างแบบ repeat until โปรแกรม (b) แสดงโปรแกรมภาษาแอสเซมบลีที่เทียบเท่ากัน

รูปแบบของโปรแกรมภาษาแอสเซมบลีที่เทียบเท่ากับ repeat until loop มีลักษณะเป็นดังนี้

startlabel:

action;

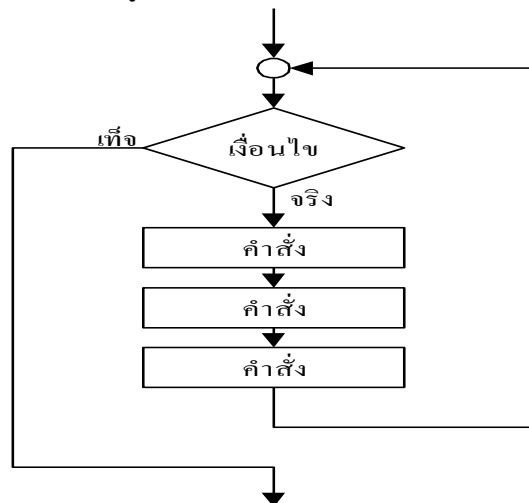
...

action;

if condition is false then jump to startlabel

11.3 การสร้าง while loop

โครงสร้างของ while loop ในภาษาระดับสูงทั่วไปมีลักษณะเป็นดังนี้



รูปที่ 11.3 โครงสร้างควบคุมแบบ while

เราสามารถเขียนโปรแกรมโดยใช้ภาษาแอสเซมบลีโดยมีโครงสร้างแบบ while loop ได้ ดังตัวอย่าง

(a)

```
while (DL<>13) and
      (CX<20) do
```

```
begin
```

```
  AX:=AX+DL;
```

```
  BX:=BX+1;
```

```
  DL:=DATA[BX];
```

```
  CX:=CX+1;
```

```
end;
```

(b)

```
startloop:
```

```
  cmp dl,13
```

```
  jz  endloop
```

```
  cmp cx,20
```

```
  jae endloop
```

```
  add al,dl
```

```
  adc ah,0
```

```
  inc bx
```

```
  mov dl,data[bx]
```

```
  inc cx
```

```
  jmp startloop
```

```
endloop:
```

โปรแกรมที่ 10.3

โปรแกรม (a) แสดง
ตัวอย่างของโปรแกรม
ภาษา pascal ที่ใช้
โครงสร้างแบบ while
loop

โปรแกรม (b) แสดง
โปรแกรม
ภาษาแอสเซมบลีที่
เทียบเท่ากัน

รูปแบบของโปรแกรม
ภาษาแอสเซมบลีที่
เทียบเท่ากับ while
loop มีลักษณะเป็นดังนี้

startlabel:

if condition is false then
jump to endlabel

action

...

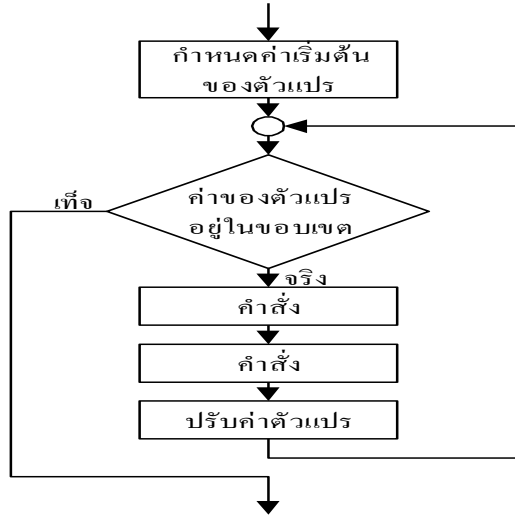
action

jump to startlabel

endlabel:

11.4 การสร้าง for loop

เราสามารถใช้คำสั่งกระโดดในการสร้างโครงสร้างแบบ for loop ได้ นอกจากนั้นเรายังสามารถใช้คำสั่งกลุ่ม LOOP ในการสร้างโครงสร้างแบบ for loop ได้เช่นเดียวกัน โครงสร้างของ for loop มีลักษณะดังรูปที่ 11.3



รูปที่ 11.3 โครงสร้างของ for loop

รูปแบบของโปรแกรมภาษาแอสเซมบลีที่มีโครงสร้างเป็นแบบ for loop มีลักษณะดังนี้

initialize index variables

startloop:

if index value is not in the range then jump to endloop

action

...

action

update index variable

jump to startloop

endloop:

ตัวอย่างของโปรแกรมที่เขียนโดยใช้รูปแบบลักษณะนี้เป็นดังโปรแกรมที่ 11.4

(a)

CX:=0;

for DL:=1 to 100 do

(b)

mov cx,0

mov dl,1

startloop:

cmp dl,100


```

ja    endloop
begin
if DL mod 7=0 then
mov  al,dl
mov  ah,0
mov  bl,7
div  bl

cmp  ah,0
jne  endif

CX:=CX+1;      inc  cx
endif:
End;           inc  dl
              jmp  startloop
endloop:

```

โปรแกรมที่ 11.4 โปรแกรม (a) แสดงตัวอย่างของโปรแกรมภาษา pascal ที่ใช้โครงสร้างแบบ for loop
 โปรแกรม (b) แสดงโปรแกรมภาษาแอสเซมบลีที่เทียบเท่ากัน

เรายังสามารถใช้คำสั่ง LOOP ในการสร้างโครงสร้างแบบ for loop ได้ เช่นเดียวกัน ดังต่อไปนี้

```

      set the value of CX
startloop:
      actions
      LOOP startloop

```

แต่การใช้คำสั่ง LOOP ในการสร้างโครงสร้างแบบ for loop ไม่สามารถสร้างโครงสร้างการกระทำซ้ำที่มีความซับซ้อนมาก ๆ ได้ เช่นการกระทำซ้ำที่มีวงรอบของการกระทำซ้ำซ้อนกันหลาย ๆ วง

สำหรับคำสั่ง LOOPZ และ LOOPNZ นั้น เราสามารถนำมาใช้ในการสร้างโครงสร้างควบคุมที่มีความซับซ้อนขึ้นได้ โดยโครงสร้างดังกล่าวจะมีลักษณะปนกันระหว่าง for loop และ while loop หรือ repeat until นั่นคือเงื่อนไขควบคุมการกระทำซ้ำจะขึ้นกับทั้งค่าของรีจิสเตอร์ (มีลักษณะคล้ายโครงสร้างแบบ for loop) และเป็นเงื่อนไขจริง ๆ (คล้าย repeat until loop และ while loop) ดังในตัวอย่างโปรแกรมต่อไปนี้

(a) AX:=0; CX:=100; repeat AX:=AX+data[BX]; BX:=BX+2; CX:=CX-1; until (data[BX]=0) or (CX=0);	(b) mov ax,0 mov cx,100 startloop: add ax,data[BX] add bx,2 cmp data[BX],0 loopnz startloop
----------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------

โปรแกรมที่ 11.5 โปรแกรม (a) โปรแกรมที่เขียนด้วยภาษา pascal
 โปรแกรม (b) โปรแกรมภาษาแอสเซมบลีที่เทียบเท่ากัน

สรุป

การใช้คำสั่งกระโดดมาสร้างเป็นโครงสร้างควบคุมรูปแบบต่าง ๆ การใช้คำสั่งกระโดดในลักษณะนี้จะทำให้โปรแกรมของเรามีความเป็นโครงสร้างมากขึ้น เช่น การสร้างโครงสร้างการตัดสินใจแบบ if-then-else, repeat until loop, while loop และ for loop เป็นต้น ดังนั้นผู้เขียนโปรแกรมภาษาแอสเซมบลีจำเป็นต้องทำความเข้าใจวิธีการและรูปแบบการเลือกการสร้างโครงสร้างควบคุมใช้งานให้เหมาะสมมากที่สุด

คำถามทบทวน

1. อธิบายพร้อมยกตัวอย่างการทำงานของโครงสร้างควบคุมรูปแบบ if-then-else
2. อธิบายพร้อมยกตัวอย่างการทำงานของโครงสร้างควบคุมรูปแบบ repeat until loop
3. อธิบายพร้อมยกตัวอย่างการทำงานของโครงสร้างควบคุมรูปแบบ while loop
4. อธิบายพร้อมยกตัวอย่างการทำงานของโครงสร้างควบคุมรูปแบบ for loop
5. จงเปรียบเทียบข้อดีและข้อเสียของรูปแบบโครงสร้างควบคุมแบบ repeat until และ while loop
6. จงเปลี่ยนโปรแกรมที่เขียนด้วยภาษาโปรแกรม pascal ที่ให้มาให้อยู่ในรูปของโปรแกรมภาษาแอสเซมบลีที่เทียบเท่ากัน

```

AX:=20;
CX:=100;
repeat
  AX:=AX+data[BX];
  BX:=BX+8;
  CX:=CX-2;
until (data[BX]=0) or
  (CX=0);

```

แผนบริหารการสอนประจำบทที่ 12

หัวข้อเนื้อหา

- คำสั่งที่รองรับการเรียกโปรแกรมย่อย
- การประกาศโปรแกรมย่อย
- คำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแอสตัก
- ตัวอย่างการใช้งานโปรแกรมย่อย

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับคำสั่งที่รองรับการเรียกโปรแกรมย่อยและการประกาศโปรแกรมย่อย
- มีความรู้และความเข้าใจเกี่ยวกับกลุ่มคำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแอสตัก

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 12 โปรแกรมย่อยขั้นต้น (The initial Subprogram)

รูปแบบของโปรแกรมภาษาแอสเซมบลีที่เขียนในบทก่อน ๆ จะมีส่วนของโปรแกรมหลายส่วนที่ซ้ำซ้อนกัน สามารถที่จะแยกส่วนย่อยเหล่านั้นเป็นโปรแกรมย่อยที่มีความอิสระจากโปรแกรมหลักได้ การแยกโปรแกรมเป็นโปรแกรมย่อยนี้ ทำให้สามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวก และการตรวจสอบและแก้ไขโปรแกรมยังสามารถกระทำได้ง่ายขึ้นด้วย

12.1 คำสั่งที่รองรับการเรียกโปรแกรมย่อย

การเรียกโปรแกรมย่อยมีความแตกต่างกับการกระโดดทั่วไป เนื่องจากภายหลังที่โปรแกรมย่อยทำงานเสร็จ หน่วยประมวลผลจะต้องสามารถกระโดดกลับมาทำงานในโปรแกรมหลักต่อไปได้ ดังนั้นการเรียกใช้โปรแกรมย่อยนั้น จะต้องมีการเก็บตำแหน่งของคำสั่งที่ทำงานอยู่เดิมด้วย และเมื่อจบโปรแกรมย่อยโปรแกรมจะต้องกระโดดกลับมาทำงานที่เดิม โดยใช้ข้อมูลที่เก็บไว้

คำสั่งของ 8086 ที่รองรับการใช้งานโปรแกรมย่อยคือคำสั่ง **CALL** และ คำสั่ง **RET** เมื่อผู้ใช้เรียกคำสั่ง CALL พร้อมทั้งระบุตำแหน่งของโปรแกรมย่อย หน่วยประมวลผลจะเก็บตำแหน่งของคำสั่งถัดไปที่จะกลับมาทำงานลงในแอสติก และจะกระโดดไปทำงานที่โปรแกรมย่อย เมื่อโปรแกรมย่อยทำงานเสร็จ โปรแกรมย่อยจะเรียกใช้คำสั่ง RET เพื่อกระโดดกลับมาทำงานในโปรแกรมหลักต่อไป เมื่อหน่วยประมวลผลประมวลผลคำสั่ง RET หน่วยประมวลผลจะดึงค่าตำแหน่งที่โปรแกรมจะกระโดดกลับไปทำงานจากแอสติก และกระโดดกลับไปทำงานต่อยังโปรแกรมหลัก

12.2 การประกาศโปรแกรมย่อย

สามารถสร้างโปรแกรมย่อยโดยการประกาศเลเบลที่จุดเริ่มต้นของโปรแกรมย่อยเท่านั้นก็ได้ แต่โดยทั่วไปแล้วจะประกาศโปรแกรมย่อยโดยใช้คู่ของคำสั่งเทียม PROC และ ENDP ดังตัวอย่างโปรแกรมที่ 12.1 โปรแกรมตัวอย่างนี้เป็นโปรแกรมที่สร้างโปรแกรมย่อยสำหรับพิมพ์เลขฐานสิบหกหนึ่งหลักชื่อ printhexdigit และใช้โปรแกรมย่อยนี้ในการเขียนโปรแกรมพิมพ์ค่ารหัสแอสกีของปุ่มที่รับค่าจากผู้ใช้ (ในตัวอย่างได้ละโปรแกรมบางส่วนไว้)

```
.model small
.dosseg

.stack 100h

.code

;procedure PrintHexDigit
```

```

;input  al : digit
;affect ah,dl
;
printhexdigit    proc    near
    mov    ah,2
    mov    dl,al
    add    dl,'0'
    cmp    al,10
    jb    printit
    add    dl,'A'-'0'-10
printit:
    int    21h
    ret
printhexdigit    endp
start:

; ...
; ละส่วนอ่านการกดปุ่มและส่วนหาค่าของตัวเลขของแต่ละหลักไว้
; ให้ตัวเลขหลักหน้าเก็บใน bh และหลักหลังเก็บใน bl
; ...

    mov    al,bh
    call    printhexdigit

    mov    al,bl
    call    printhexdigit

    mov    ax,4c00h
    int    21h
end    start

```

โปรแกรมที่ 12.1 ส่วนของโปรแกรมพิมพ์ค่ารหัสแอสกีเป็นเลขฐานสิบหก

รูปแบบของการประกาศโปรแกรมย่อยมีลักษณะดังนี้

```

proc_name  PROC          NEAR
           actions
proc_name  ENDP

```

คำสั่งเทียม NEAR ที่ต่อจากคำสั่งเทียม PROC เป็นการระบุว่าโปรแกรมย่อยนี้เป็นโปรแกรมย่อยที่อยู่ในเซกเมนต์เดียวกันกับโปรแกรมหลัก และมีการเรียกใช้แบบใกล้ ซึ่งจะมีผลในขั้นตอนการเก็บตำแหน่งที่จะกระโดดกลับมาทำงานหลังการทำงานของโปรแกรมย่อย

การทำงานของโปรแกรมย่อยจากตัวอย่างมีการเปลี่ยนแปลงค่าของรีจิสเตอร์ AH และรีจิสเตอร์ DL การเปลี่ยนค่าของรีจิสเตอร์ทั้งสองอาจทำให้เกิดผลข้างเคียงกับโปรแกรมหลักได้ถ้าโปรแกรมหลักมีการใช้งานรีจิสเตอร์ทั้งสองด้วยเช่นเดียวกัน ควรจะลดการเกิดผลข้างเคียงนี้โดยให้โปรแกรมย่อยเก็บค่าของรีจิสเตอร์ต่าง ๆ ที่โปรแกรมย่อยใช้และคืนค่าเดิมให้กับรีจิสเตอร์เหล่านั้นหลังการทำงาน นิยมใช้แอสตึกในการเก็บค่าของรีจิสเตอร์เป็นการชั่วคราวเนื่องจากสามารถเก็บข้อมูลลงในแอสตึกได้โดยไม่ต้องจองเนื้อที่ล่วงหน้า และการเก็บข้อมูลในลักษณะของแอสตึก มีความสอดคล้องกับการทำงานแบบโปรแกรมย่อย

12.3 คำสั่งเก็บข้อมูลและดึงข้อมูลจากแอสตึก : คำสั่ง PUSH และคำสั่ง POP

สามารถใช้คำสั่ง PUSH ในการเก็บข้อมูลลงในแอสตึก และคำสั่ง POP สำหรับการเรียกข้อมูลออกจากแอสตึก คำสั่งทั้งสองมีรูปแบบดังนี้

```
push regs16      push mem
pop  regs16      pop  mem
```

ข้อมูลที่จะเก็บลงในแอสตึกจะต้องมีขนาด 16 บิตเท่านั้น โดยการเก็บข้อมูลในแอสตึกจะมีลักษณะเป็นแบบใส่ที่หลังดึงออกก่อน การเก็บข้อมูลลงในแอสตึกจึงต้องระวังลำดับของการเก็บและการดึงข้อมูลออกไปด้วย

การทำงานของแอสตึกจะมีรีจิสเตอร์ SS และ SP เป็นรีจิสเตอร์ที่ใช้เก็บตำแหน่งของข้อมูลที่เก็บลงไปอันล่าสุด โดย SS จะเก็บเซกเมนต์ของแอสตึก และ SP จะเก็บออฟเซตของข้อมูลล่าสุด เมื่อมีการเก็บข้อมูลเพิ่มลงในแอสตึกหรือมีการดึงข้อมูลออกไปก็จะมีการปรับค่าของรีจิสเตอร์ทั้งสองนี้ การทำงานของแอสตึกมีลักษณะดังรูปที่ 12.1



รูปที่ 12.1 แสดงการทำงานของแอสตึก

โปรแกรมย่อยที่มีการเก็บค่าของรีจิสเตอร์ต่าง ๆ

สามารถแก้ไขโปรแกรมย่อยในตัวอย่างให้มีการรักษาค่าในรีจิสเตอร์ต่าง ๆ ได้ดังโปรแกรมที่ 12.2

```
;procedure PrintHexDigit
;input al : digit
;affect ah,dl
;
printhexdigit      proc      near
```

```

push    ax
push    dx
mov     ah,2
mov     dl,al
add     dl,'0'
cmp     al,10
jb     printit
add     dl,'A'-'0'-10

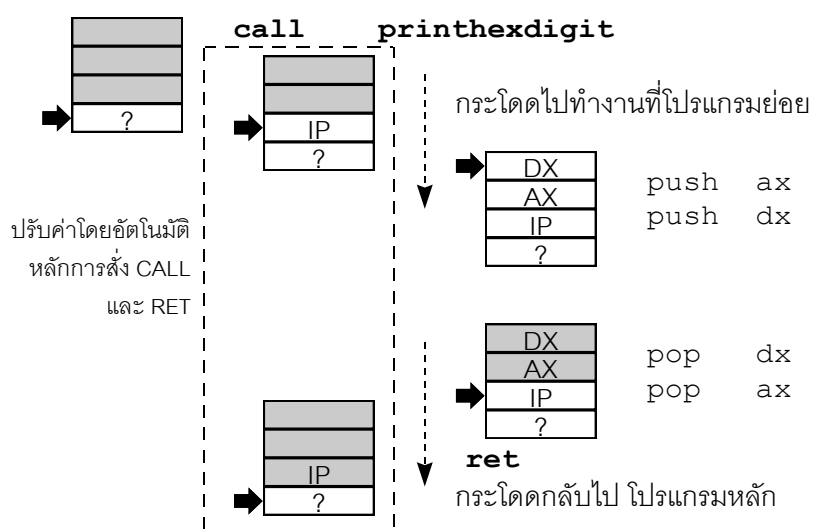
printit:
int     21h
pop     dx
pop     ax
ret

printhexdigit    endp

```

โปรแกรมที่ 12.2 ตัวอย่างโปรแกรมย่อยที่รักษาค่าของรีจิสเตอร์

สังเกตลำดับในการ push และ pop ของโปรแกรมย่อย ลำดับในการ push จะตรงกันข้ามกับลำดับในการ pop เนื่องจากการทำงานของแอสตักเป็นแบบข้อมูลที่ใส่ทีหลังจะถูกดึงออกก่อน การเปลี่ยนแปลงของแอสตักในการทำงานดังกล่าวแสดงได้โดยรูปที่ 12.2



รูปที่ 12.2 แสดงการเปลี่ยนแปลงของแอสตักในการเรียกใช้โปรแกรมย่อย

เมื่อมีการเรียกใช้โปรแกรมย่อยโดยคำสั่ง CALL ค่าของรีจิสเตอร์ IP ซึ่งเก็บตำแหน่งของคำสั่งถัดไปจะถูก PUSH ลงในแอสตักโดยอัตโนมัติ จากนั้นโปรแกรมจะกระโดดไปทำงานที่โปรแกรมย่อย ในโปรแกรมย่อยมีการเก็บ

ค่าของรีจิสเตอร์ AX และรีจิสเตอร์ DX ลงในแอสตัก หลังการทำงานโปรแกรมย่อยได้คืนค่าของรีจิสเตอร์ทั้งสอง โดยการดึงค่าจากแอสตัก และเรียกใช้คำสั่ง RET เพื่อกระโดดกลับไปทำงานยังโปรแกรมหลักต่อ สังเกตว่าถ้าเก็บและดึงค่าออกจากแอสตักได้ไม่ถูกต้องการกระโดดกลับไปทำงานยังโปรแกรมหลักอาจมีการผิดพลาดได้

12.4 ตัวอย่างการใช้งานโปรแกรมย่อย

โปรแกรมตัวอย่างต่อไปนี้สร้างโปรแกรมย่อยขึ้นมาสองโปรแกรมย่อย โปรแกรมย่อยแรกเป็นโปรแกรมย่อยที่พิมพ์เลขฐานสิบหกหนึ่งหลัก ส่วนโปรแกรมย่อยที่สองเป็นโปรแกรมย่อยที่พิมพ์เลขฐานสิบหกขนาด 1 ไบต์ โดยเรียกใช้งานโปรแกรมย่อยโปรแกรมแรก

```
.model small
.dosseg

.stack 100h

.code
;procedure PrintHexDigit
;display one hex digit
;input : al = the digit
printhexdigit proc near
    push ax
    push dx
    mov ah,2
    mov dl,al
    add dl,'0' ;make ascii from the number
    cmp al,10
    jb printit ;if above 9 adjust the ascii value
    add dl,'A'-'0'-10
printit:
    int 21h ;print the number
    pop dx
    pop ax
    ret
printhexdigit endp

;procedure PrintHexNumber
```

```
;display one hex number (1 byte)
;input : al = the number
printhexnumber    proc
    push    ax
    push    bx
    mov     bl,16          ;div by 16 to get 2 digits
    mov     ah,0
    div     bl             ;al=high digit , ah=low digit
    call    printhexdigit ;print the high digit
    mov     al,ah
    call    printhexdigit ;print the next digit
    pop     bx
    pop     ax
    ret
printhexnumber    endp
start:
    mov     ah,1
    int     21h
    call    printhexnumber
    mov     ax,4c00h
    int     21h
end    start
```

สรุป

การแยกโปรแกรมเป็นโปรแกรมย่อยนี้ ทำให้สามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวกและการตรวจสอบและแก้ไขโปรแกรมยังสามารถกระทำได้ง่ายขึ้นด้วย ส่วนการเรียกโปรแกรมย่อยมีความแตกต่างกับการกระโดดทั่วไป เนื่องจากภายหลังจากที่โปรแกรมย่อยทำงานเสร็จ หน่วยประมวลผลจะต้องสามารถกระโดดกลับมาทำงานในโปรแกรมหลักต่อไปได้ ดังนั้นการเรียกใช้โปรแกรมย่อยนั้นจะต้องมีการเก็บตำแหน่งของคำสั่งที่ทำงานอยู่เดิมด้วย และเมื่อจบโปรแกรมย่อยโปรแกรมจะต้องกระโดดกลับมาทำงานที่เดิม โดยใช้ข้อมูลที่เก็บไว้

คำสั่งของ 8086 ที่รองรับการใช้งานโปรแกรมย่อยคือคำสั่ง CALL และคำสั่ง RET เมื่อผู้ใช้เรียกคำสั่ง CALL พร้อมทั้งระบุตำแหน่งของโปรแกรมย่อย หน่วยประมวลผลจะเก็บตำแหน่งของคำสั่งถัดไปที่จะกลับมาทำงานลงในแอสตีก และจะกระโดดไปทำงานที่โปรแกรมย่อย เมื่อโปรแกรมย่อยทำงานเสร็จ โปรแกรมย่อยจะเรียกใช้คำสั่ง RET เพื่อกระโดดกลับมาทำงานในโปรแกรมหลักต่อไป เมื่อหน่วยประมวลผลประมวลผลคำสั่ง RET หน่วยประมวลผลจะดึงค่าตำแหน่งที่โปรแกรมจะกระโดดกลับไปทำงานจากแอสตีก และกระโดดกลับไปทำงานต่อยังโปรแกรมหลักต่อไป

คำถามทบทวน

1. จงอธิบายการทำงานของคำสั่ง CALL และคำสั่ง RET
2. การประกาศโปรแกรมย่อยในภาษาแอสแซมบลีโดยทั่วไปมักจะใช้คู่ของคำสั่งเทียมอะไร
3. คำสั่งเก็บข้อมูลและดึงข้อมูลจากแอสตีกคำสั่ง PUSH และคำสั่ง POP มีขั้นตอนการทำงานอย่างไร
4. การทำงานของแอสตีกจะมีรีจิสเตอร์ที่เกี่ยวข้องอะไรบ้างและแต่ละรีจิสเตอร์มีหน้าที่และทำงานอย่างไร
5. จงอธิบายเหตุการณ์ที่เกิดขึ้นหลังจากมีการเรียกใช้โปรแกรมย่อยโดยคำสั่ง CALL

แผนบริหารการสอนประจำบทที่ 13

หัวข้อเนื้อหา

- คำสั่งทางตรรกศาสตร์
- การประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์
- คำสั่งเลื่อนบิต
- ตัวอย่างการใช้งานคำสั่งเลื่อนบิต
- การประยุกต์ใช้งานคำสั่งเลื่อนบิต
- คำสั่งหมุนบิต
- คำสั่งหมุนบิตที่ผ่านแฟล็กทด
- ตัวอย่างการใช้งานคำสั่งเกี่ยวกับการประมวลผลระดับบิต

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับคำสั่งทางตรรกศาสตร์ เช่น AND, OR XOR TEST เป็นต้น
- สามารถประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์ในการเขียนโปรแกรมภาษาแอสเซมบลีได้
- มีความรู้และความเข้าใจเกี่ยวกับการใช้คำสั่ง SHL (Shift Left) และคำสั่ง SHR (Shift Right)
- มีความรู้และความเข้าใจเกี่ยวกับการใช้คำสั่งหมุนบิตแบบต่างๆ เช่น ROL (Rotation Left), คำสั่ง ROR (Rotation Right), RCL (Rotate Carry Left) และคำสั่ง RCR (Rotate Carry Right)

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน

บทที่ 13 การกระทำระดับบิต (The Bit Level)

การจัดการกับข้อมูลที่เราได้ศึกษาในบทก่อน ๆ นั้นมีหน่วยย่อยในการจัดการเป็นไบนารี โดยเราไม่สามารถจัดการข้อมูลที่มีขนาดย่อยกว่านั้นได้ แต่ในการทำงานจริงในบางครั้งรูปแบบของข้อมูลที่เราต้องจัดการจะอยู่ในรูปของบิต เราจึงใช้คำสั่งเกี่ยวกับการจัดการระดับบิตในการประมวลผลข้อมูลกลุ่มนี้ และในบางกรณีคำสั่งกระทำระดับบิตสามารถช่วยให้การคำนวณต่าง ๆ ทำได้ง่ายขึ้นด้วย

13.1 คำสั่งทางตรรกศาสตร์

คำสั่งในกลุ่มนี้เป็นคำสั่งประมวลผลข้อมูลระดับบิต โดยจะนำค่าในแต่ละบิตของข้อมูลมาประมวลผลทางตรรกศาสตร์. คำสั่งในกลุ่มนี้ได้แก่ คำสั่ง AND คำสั่ง OR คำสั่ง XOR และคำสั่ง NOT รูปแบบการใช้งานของคำสั่ง AND คำสั่ง OR และคำสั่ง XOR จะมีลักษณะเหมือนกัน คือจะรับโอเปอร์เรนด์สองตัว และจะนำข้อมูลในโอเปอร์เรนด์ตัวแรกมากระทำกับข้อมูลตัวที่สอง และจะเก็บผลลัพธ์ของการกระทำนั้นในโอเปอร์เรนด์ตัวแรก ส่วนในกรณีของคำสั่ง NOT จะรับโอเปอร์เรนด์ตัวเดียว และจะทำการกลับค่าในบิตแล้วเก็บผลลัพธ์ลงในโอเปอร์เรนด์ตัวนั้นเลย ตารางค่าความจริงของการกระทำทางตรรกศาสตร์เป็นดังตารางที่ 13.1

ตารางที่ 13.1 ค่าของการกระทำทางตรรกศาสตร์

A	B	A and B	A or B	A xor B	not B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	0	1

คำสั่ง AND

ผลลัพธ์ของคำสั่ง AND จะมีบิตที่เป็น 1 เมื่อบิตของข้อมูลตัวตั้งทั้งสองตัวมีค่าเป็น 1 (ตาราง 12.1)

ตัวอย่างคำสั่ง

```

mov ax,1234h
and ax,2345h      ; ax = (0001 0010 0011 0100) and (0010 0011 0100 0101)
                  ; ax = (0000 0010 0000 0100) = 0204h

mov al,25h
and al,0Fh        ; al = 05h

mov bl,0AAh
and bl,80h        ; al = 80h

```

คำสั่ง OR

ผลลัพธ์ของคำสั่ง OR จะมีบิตที่เป็น 1 เมื่อบิตของข้อมูลตัวตั้งตัวใดตัวหนึ่งหรือทั้งสองตัวมีค่าเป็น 1 (ตาราง 13.1)

ตัวอย่างคำสั่ง

```

mov ax,1234h
or ax,2345h ; ax = (0001 0010 0011 0100 and (0010 0011 0100 0101)
; ax = (0011 0011 0111 0101) = 3375h

mov al,25h
or al,0Fh ; al = 2Fh

mov bl,55h
or bl,80h ; bl = 0D5h

```

คำสั่ง XOR

การทำงานของคำสั่ง XOR จะคล้ายกับคำสั่ง OR แต่ในกรณีที่ข้อมูลมีบิตที่เป็นหนึ่งทั้งคู่ ผลลัพธ์ที่ได้จะมีค่าเป็นศูนย์ (ตาราง 13.1) ลักษณะของการ XOR จะคล้ายกับการพิจารณาเหตุการณ์ที่เป็นไปได้ทั้งสองเหตุการณ์ แต่ไม่สามารถเป็นจริงพร้อมกันได้

ตัวอย่างคำสั่ง

```

mov ax,1234h
xor ax,2345h ; ax = (0001 0010 0011 0100 and (0010 0011 0100 0101)
; ax = (0011 0001 0111 0001) = 3171h

mov al,25h
xor al,0Fh ; al = 2Ah

mov bl,15h
xor bl,35h ; bl = 20h

```

คำสั่ง NOT

คำสั่ง NOT จะสลับบิตของโอเปอร์แรนด์จากศูนย์เป็นหนึ่งและหนึ่งเป็นศูนย์ (ตาราง 13.1)

ตัวอย่างคำสั่ง

```

mov ax,1234h
not ax ; ax = not(0001 0010 0011 0100)
; ax = (1110 1101 1100 1011) = 0EDCBh

```

คำสั่ง TEST

คำสั่ง TEST จะทำงานเหมือนคำสั่ง AND ทุกประการ แต่ผลลัพธ์จากการ AND จะไม่เขียนค่าลงในโอเปอร์แรนด์ตัวแรก ผลจากการใช้คำสั่งนี้จะปรากฏในแฟล็ก เรานิยมใช้คำสั่งนี้ในการทดสอบว่าข้อมูลในบิตที่ต้องการมีค่าเป็นหนึ่งหรือไม่ โดยเราจะพิจารณาผลลัพธ์จากแฟล็กทด

ตัวอย่างคำสั่ง

```

mov    al,7Ah
test   al,80h      ; test for the 7th bit
jz     biton       ; jump if 7th bit of al = 1
mov    bl,12h
test   bl,1        ; test for the 0th bit

```

การประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์

เราสามารถนำคำสั่งทางตรรกศาสตร์มาใช้ในการประมวลผลข้อมูลระดับบิตได้ จากตารางที่ 13.1 เราสามารถสร้างตารางที่ 13.2 ซึ่งแสดงผลของการใช้คำสั่งทางตรรกศาสตร์กับข้อมูลได้

ตารางที่ 13.2 ผลของการใช้คำสั่งทางตรรกศาสตร์กับข้อมูล

B	A and B	A or B	A xor B
0	0	A	A
1	A	1	~A

จากตารางเราจะพบว่าถ้าเราต้องการให้บิตใดของข้อมูลมีค่าเป็นหนึ่งโดยที่บิตอื่นมีค่าคงเดิม เราสามารถใช้คำสั่ง AND ได้ และถ้าเราต้องการจะทำให้บิตใดของข้อมูลมีค่าเป็นศูนย์โดยไม่มีผลกระทบต่อบิตอื่น ๆ เราสามารถใช้คำสั่ง OR สำหรับคำสั่ง XOR เราจะใช้ในกรณีที่ต้องการกลับบิตของข้อมูลจากศูนย์เป็นหนึ่ง

ตัวอย่างการประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์

โปรแกรมตัวอย่างต่อไปนี้จะเปลี่ยนบิตที่ 1 และ 2 ของ AL ให้มีค่าเป็นศูนย์ (การนับบิตจะนับบิตที่มีนัยสำคัญต่ำสุดเป็นบิตที่ 0) และเปลี่ยนบิตที่ 4 และบิตที่ 6 ให้มีค่าเท่ากับ 1 พร้อมทั้งกลับบิตที่ 3 ให้มีค่าตรงกันข้าม การทำงานควรมีลักษณะดังรูปที่ 13.1

```

AL      XXXX XXXX
and     1111 1001
or      0101 0000
xor     0000 1000
result  X1X1 X00X

```

รูปที่ 13.1 ขั้นตอนการแปลงค่าของ AL

โปรแกรมจะมีลักษณะดังนี้

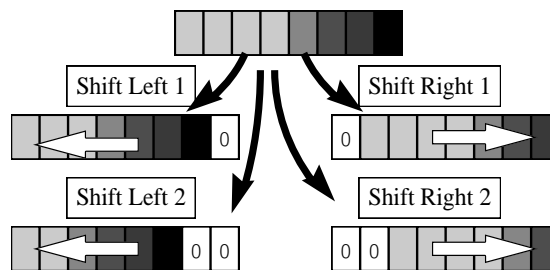
```

and    al,0F9h;clear bit 1&2
or     al,50h ;set bit 4&6
xor    al,08h ;switch bit 3

```

13.2 คำสั่งเลื่อนบิต

การประมวลผลอีกรูปแบบที่เราสามารถกระทำกับข้อมูลในระดับชั้นของบิตได้แก่การเลื่อนบิต ลักษณะการเลื่อนบิตเป็นดังรูปที่ 12.2 ในการเลื่อนบิตเราสามารถเลื่อนได้ทั้งทางซ้ายและทางขวา โดยคำสั่งสำหรับการเลื่อนบิตไปทางซ้ายได้แก่ คำสั่ง SHL (Shift Left) คำสั่งสำหรับการเลื่อนบิตไปทางขวาได้แก่ คำสั่ง SHR (Shift Right) เรานิยมใช้การเลื่อนบิตในการประมวลผลที่ต้องการประมวลผลข้อมูลทีละบิต และมีการประมวลผลเป็นแบบวงรอบ



รูปที่ 13.2 แสดงลักษณะของการเลื่อนบิต

รูปแบบของคำสั่งเลื่อนบิตมีลักษณะดังนี้

SHR	<i>regs,1</i>	SHR	<i>mem,1</i>
SHR	<i>regs,CL</i>	SHR	<i>mem,CL</i>
SHR	<i>regs,number</i>	SHR	<i>mem,number</i>

โดยรูปแบบของคำสั่ง SHL จะมีลักษณะเหมือนคำสั่ง SHR รูปแบบที่สามจะใช้ได้กับหน่วยประมวลผล 80286 ขึ้นไปเท่านั้นโดยในการที่เราจะใช้รูปแบบของคำสั่งของ 80286 ในโปรแกรมเราจะต้องระบุ คำสั่งเทียม 286 ลงในโปรแกรมด้วย โดยใส่คำสั่งนี้ก่อนหน้าการใช้งานคำสั่งครั้งแรก

ตัวอย่างการใช้งานคำสั่งเลื่อนบิต

โปรแกรมตัวอย่างต่อไปนี้เป็นโปรแกรมนับจำนวนบิตที่มีค่าเป็นหนึ่งใน AX โดยจะให้ผลลัพธ์ใน BL

```
mov    bl,0
mov    cx,16        ; 16 bits
```

procbits:

```
test   ax,1        ; test for last bit
jz     doprocbit   ; if last bit=0 jump
inc    bl
```

doprocbit:

```
shr    ax,1        ; next bit
loop  procbits
```


ความหมายทางคณิตศาสตร์ของการเลื่อนบิต

ตารางที่ 13.3 แสดงผลลัพธ์ของการเลื่อนบิตของข้อมูลต่าง ๆ จากตารางจะสังเกตเห็นได้ว่านอกจากการเลื่อนบิตจะมีความหมายโดยตรงคือการเลื่อนบิตไปทางซ้ายหรือทางขวาแล้ว การเลื่อนบิตยังมีความหมายทางคณิตศาสตร์อีกด้วย

ตารางที่ 13.3 ตัวอย่างผลลัพธ์ของการเลื่อนบิตของข้อมูลต่าง ๆ

ข้อมูล (ค่า)	เลื่อนบิตไป ทาง	จำนวน (บิต)	ผลลัพธ์ (ค่า)
0010 1110 (46)	ซ้าย	1	0101 1100 (92)
0010 1110 (46)	ซ้าย	2	1011 1000 (184)
0010 1110 (46)	ซ้าย	3	0111 0000 (112)
0110 0100 (100)	ขวา	1	0011 0010 (50)
0110 0100 (100)	ขวา	2	0001 1001 (25)
0110 0100 (100)	ขวา	3	0000 1100 (12)

สังเกตว่าการเลื่อนบิตไปทางซ้ายจะมีผลลัพธ์เหมือนกับการคูณด้วยกำลังของสอง ยกตัวอย่างเช่น การเลื่อนบิตไปทางซ้าย 1 บิตจะเหมือนกับการคูณด้วยสอง. และการ แต่เราจะต้องพิจารณากรณีที่ข้อมูลอยู่ในขอบเขตด้วย เช่นกรณีของการเลื่อน 0010 1110 ไปทางซ้าย 3 บิต (คูณด้วย 8) ผลลัพธ์ที่ได้จะมีความผิดพลาด การเลื่อนบิตไปทางขวาจะให้ผลลัพธ์ตรงกันข้ามกับการเลื่อนบิตไปทางขวา นั่นคือจะเสมือนการหารด้วยกำลังสอง (สังเกตว่าผลลัพธ์ที่ได้จะมีการปิดเศษเนื่องจากบิตที่เลื่อนจะหายไป เช่นในตัวอย่างที่เลื่อนบิตทางขวา 3 บิต)

คำสั่งเลื่อนบิตแบบคิดเครื่องหมาย : คำสั่ง SAL และคำสั่ง SAR

ถ้าเราใช้การเลื่อนบิตแทนการคูณหรือหารด้วยกำลังของสองกับตัวเลขแบบคิดเครื่องหมาย เราจะพบว่า การเลื่อนบิตไปทางซ้ายที่แสดงถึงการคูณนั้นยังสามารถใช้กับตัวเลขแบบคิดเครื่องหมายได้ เนื่องจากหลักที่เลื่อนเข้ามาแทนนั้นยังคงเป็นเลขศูนย์เหมือนในกรณีของเลขไม่คิดเครื่องหมาย แต่ในกรณีของการเลื่อนบิตไปทางขวาที่ใช้สำหรับการหารด้วยกำลังของสองนั้น บิตที่เลื่อนเข้ามาแทนอาจมีค่าเป็น 0 หรือ 1 ก็ได้ขึ้นกับเครื่องหมายของตัวเลขนั้น เราจึงมีคำสั่งเลื่อนบิตที่ใช้สำหรับเลขที่มองเป็นเลขคิดเครื่องหมาย คือคำสั่ง SAL (Shift Arithmetic Left) และ คำสั่ง SAR (Shift Arithmetic Right) คำสั่ง SAL จะทำงานเหมือนคำสั่ง SHL ทุกประการ ตัวอย่างการใช้งานคำสั่งเป็นดังตารางที่ 13.4

ตารางที่ 13.4 ตัวอย่างผลลัพธ์ของการเลื่อนบิตแบบคิดเครื่องหมาย

ข้อมูล (ค่า)	เลื่อนบิตไป ทาง	จำนวน (บิต)	ผลลัพธ์ (ค่า)
0010 1110 (46)	ซ้าย	1	0101 1100 (92)
1110 1000 (-24)	ซ้าย	2	1101 0000 (-48)
0110 0100 (100)	ขวา	1	0011 0010 (50)
1010 1100 (-84)	ขวา	2	1110 1011 (-21)
1010 1100 (-84)	ขวา	3	1111 0101 (-11)

การประยุกต์ใช้งานคำสั่งเลื่อนบิต

เราสามารถนำคำสั่งเลื่อนบิตไปใช้ในการคูณและหารข้อมูลได้ โดยการใช้คำสั่งเลื่อนบิตแทนการใช้คำสั่ง MUL ทำให้การคูณทำงานได้เร็วขึ้น และในบางกรณีเราจะเขียนโปรแกรมได้ง่ายขึ้นด้วย

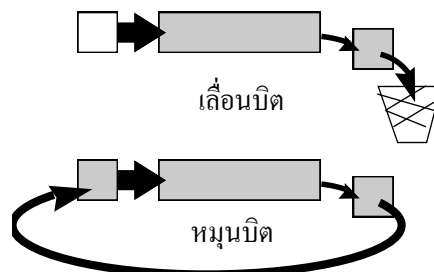
ตัวอย่างคำสั่ง

```
mov  bl,al
shl  al,1
add  bl,al      ;bl = al*3
```

```
mov  cl,2
shl  ax,cl
mov  bx,ax
shl  ax,1
add  bx,ax      ;bx=(ax*4)+(ax*8) = ax*12
```

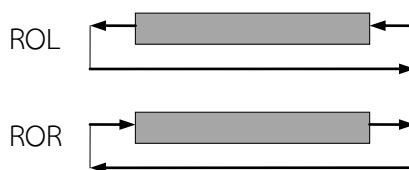
13.3 คำสั่งหมุนบิต

คำสั่งหมุนบิตมีความแตกต่างกับคำสั่งเลื่อนบิตในจุดที่ว่า บิตที่เลื่อนไปแล้วไม่ได้ถูกทิ้งหายไป แต่จะถูกนำมาใส่แทนบิตที่เลื่อนไป โดยลักษณะการทำงานคร่าว ๆ จะแสดงดังรูปที่ 13.3



รูปที่ 13.3 แสดงลักษณะของการเลื่อนบิต และ การหมุนบิต

เช่นเดียวกับคำสั่งเลื่อนบิต คำสั่งหมุนบิตมีลักษณะการหมุนสองแบบคือ หมุนไปทางซ้าย (คำสั่ง ROL : Rotate Left) และ หมุนไปทางขวา (คำสั่ง ROR : Rotate Right) รูปแบบของคำสั่งทั้งสองจะมีลักษณะเหมือนคำสั่งเลื่อนบิต การทำงานของคำสั่งทั้งสองแสดงได้ดังรูปที่ 13.4



รูปที่ 13.4 แสดงลักษณะการทำงานของคำสั่งหมุนบิต

เรานิยมใช้คำสั่งหมุนบิตแทนคำสั่งเลื่อนบิตในกรณีที่เราต้องการให้ค่าของข้อมูลกลับเหมือนเดิมหลังประมวลผลครบรอบ

ตัวอย่างคำสั่ง

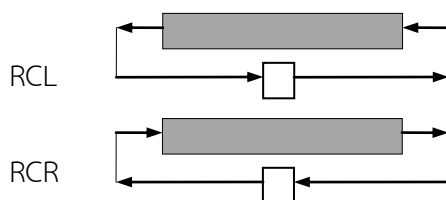
```
mov  al,4Ah
mov  cl,4
ror  al,cl      ;al = 0A4h
mov  bx,92EAh  ;bx=1001 0010 1110 1010
rol  bx,1      ;bx=0010 0101 1101 0101=35D5h
mov  cx,8
mov  dx,0
```

loophere:

```
xor  dx,ax
rol  ax,1
loop loophere
```

คำสั่งหมุนบิตที่ผ่านแฟล็กทาด

คำสั่งหมุนบิตอีกกลุ่มหนึ่งจะเป็นการหมุนโดยนำบิตไปผ่านแฟล็กทาด ลักษณะการทำงานจะเป็นดังรูปที่ 13.5 สังเกตว่าบิตที่เข้ามาแทนบิตที่หมุนไปจะนำมาจากแฟล็กทาด และบิตที่ถูกหมุนออกไปจะเข้าไปแทนค่าในแฟล็กทาด โดยคำสั่งหมุนบิตผ่านแฟล็กทาดคือคำสั่ง **RCL** (Rotate Carry Left) และคำสั่ง **RCR** (Rotate Carry Right)



รูปที่ 13.5 แสดงลักษณะการทำงานของคำสั่งหมุนบิตที่ผ่านแฟล็กทาด

สังเกตว่าบิตที่ล้นออกมาจะถูกนำไปพักที่แฟล็กทาด ก่อนที่จะนำมาแทนที่ในข้อมูล เรานิยมใช้คำสั่งหมุนบิตผ่านแฟล็กทาดในการเลื่อนบิตข้อมูลที่เก็บต่อเนื่องอยู่ในหลายรีจิสเตอร์ ในการใช้งานคำสั่งนี้เราจะต้องกำหนดค่าให้กับแฟล็กทาดเสียก่อน โดยใช้คำสั่ง **STC** และคำสั่ง **CLC**

ตัวอย่างการใช้งานคำสั่งหมุนบิตที่ผ่านแฟล็กทาด

ตัวอย่างต่อไปนี้เป็นการเล่นบิตของข้อมูลขนาด 32 บิตที่อยู่ในรีจิสเตอร์ DX,AX ไปทางซ้าย 1 บิต

```
clc
rcl  ax,1
rcl  dx,1
```

ตัวอย่างคำสั่ง การคูณข้อมูลขนาด 48 บิตที่เก็บในรีจิสเตอร์ BX,DX และ AX ต่อเนื่องกัน ด้วย 4

```
clc
rcl ax,1
rcl dx,1
rcl bx,1
clc
rcl ax,1
rcl dx,1
rcl bx,1
```

ตัวอย่างคำสั่ง การหารข้อมูลใน DX,AX ด้วย 2

```
clc
rcr dx,1
rcr ax,1
```

ตัวอย่างการใช้งานคำสั่งเกี่ยวกับการประมวลผลระดับบิต

โปรแกรมตัวอย่างต่อไปนี้เป็นโปรแกรมที่แสดงค่ารหัสแอสกีของปุ่มที่รับจากผู้ใช้เป็นเลขฐานสอง โปรแกรมย่อยที่แสดงข้อมูลเป็นรหัสเลขฐานสองใช้การเลื่อนบิตในการประมวลผล

```
.model small
.dosseg

.code

; Display Binary (input : al)
dispbinary proc near
    push ax
    push cx
    push dx
    mov cx,8
printloop:
    test al,80h ;test for 1
    jz printzero
    mov dl,'1'
    jmp printit
printzero:
    mov dl,'0'
```

```
printit: mov ah,2
         int 21h

         shl al,1
         loop printloop
         pop dx
         pop cx
         pop ax
         ret
dispbin endp

start:
        mov ah,1
        int 21h

        call dispbin

        mov ax,4C00h
        int 21h
end     start
```

สรุป

คำสั่งทางตรรกศาสตร์เป็นคำสั่งประมวลผลข้อมูลระดับบิต โดยจะนำค่าในแต่ละบิตของข้อมูลมาประมวลผลทางตรรกศาสตร์ คำสั่งในกลุ่มนี้ได้แก่ คำสั่ง AND คำสั่ง OR คำสั่ง XOR และคำสั่ง NOT รูปแบบการใช้งานของคำสั่ง AND คำสั่ง OR และคำสั่ง XOR จะมีลักษณะเหมือนกัน คือจะรับโอเปอร์แรนด์สองตัว และจะนำข้อมูลในโอเปอร์แรนด์ตัวแรกมากระทำกับข้อมูลตัวที่สอง และจะเก็บผลลัพธ์ของการกระทำนั้นในโอเปอร์แรนด์ตัวแรก ส่วนในกรณีของคำสั่ง NOT จะรับโอเปอร์แรนด์ตัวเดียว และจะทำการกลับค่าในบิตแล้วเก็บผลลัพธ์ลงในโอเปอร์แรนด์ตัวนั้นเลย ส่วนการประมวลผลอีกรูปแบบที่เราสามารถกระทำกับข้อมูลในระดับชั้นของบิตได้แก่ การเลื่อนบิต ในการเลื่อนบิตเราสามารถเลื่อนได้ทั้งทางซ้ายและทางขวา โดยคำสั่งสำหรับการเลื่อนบิตไปทางซ้ายได้แก่ คำสั่ง SHL (Shift Left) คำสั่งสำหรับการเลื่อนบิตไปทางขวาได้แก่ คำสั่ง SHR (Shift Right) เรานิยมใช้การเลื่อนบิตในการประมวลผลที่ต้องการประมวลผลข้อมูลที่ละบิต และมีการประมวลผลเป็นแบบวงรอบ

แผนบริหารการสอนประจำบทที่ 14

หัวข้อเนื้อหา

- การอ้างแอดเดรส
- แบบการอ้างแอดเดรส

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับการอ้างแอดเดรสใน เซกเมนต์ทั้งสี่ได้แก่ CS, DS, SS และ ES
- สามารถประยุกต์ใช้งานคำสั่งในการอ้างแอดเดรสแบบต่าง ๆ ได้

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 14 การอ้างแอดเดรส (Addressing Mode)

14.1 การอ้างแอดเดรส

8086 จะมองลักษณะของหน่วยความจำ โดยแบ่งหน่วยความจำเป็นกลุ่มๆในรูปแบบของ เซกเมนต์ ในหนึ่งเซกเมนต์จะชี้ได้ถึง 64 กิโลไบต์ เซกเมนต์ทั้งสี่ได้แก่ CS, DS, SS และ ES จะแสดงแอดเดรสของหน่วยความจำที่จะติดต่อกับ CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรม DS จะเก็บค่าตาตาเซกเมนต์ขณะนั้น SS ก็เก็บค่าสแต็กเซกเมนต์ขณะนั้น และ ES จะกำหนดเซกเมนต์ของข้อมูลรวมที่เรียกว่า global data segment

จุดสำคัญอีกจุดหนึ่งก็คือ เซกเมนต์จะแสดงตำแหน่งเหมือนกับเป็นพารากราฟ (paragraph) โดยจะเลื่อนไปทางซ้าย 4 บิต เพื่อที่จะกำหนดหรืออ้างแอดเดรสให้ครบ 20 เส้น โดยจุดเริ่มต้นของพารากราฟจะต้องมี 4 บิตหลังสุดเป็น 0 เช่น เป็น 00000H, 00010H, 00020H เป็นต้น

เพื่อที่จะทำการติดต่อกับข้อมูลหนึ่งไบต์หรือหนึ่งเวิร์ดนั้น 8086 ได้เตรียมค่า ออฟเซต เพื่อใช้อ้างตำแหน่งตั้งแต่จุดเริ่มต้นของเซกเมนต์แอดเดรส ตำแหน่งใดๆจะได้มาจากการบวกค่าเซกเมนต์รีจิสเตอร์กับค่าของออฟเซต 16 บิต เช่นถ้าเซกเมนต์มีค่า E89F และให้ออฟเซตมีค่า 0003H จะทำให้การอ้างแอดเดรสไปที่ E89F3H

ลักษณะในการอ้างแอดเดรส

การอ้างแอดเดรสโดยระบุตำแหน่งด้วย เซกเมนต์และออฟเซต เช่น

```
mov ax, [es:100h]
```

```
mov bl, es:[bx]
```

การอ้างข้อมูลทั่วไปโดยไม่ระบุเซกเมนต์ ออฟเซตที่ระบุจะคิดเทียบกับ DS เช่น

```
mov ax, B800h
```

```
mov es, ax
```

14.2 แบบการอ้างแอดเดรส

อ้างแบบรีจิสเตอร์ (Register addressing) ข้อมูลอยู่ในรีจิสเตอร์ เช่น

```
mov ax, bx
```

อ้างแบบค่าคงที่ (Immediate addressing) ข้อมูลอยู่ในคำสั่ง เช่น

```
mov ax, 10
```

อ้างโดยตรง (Direct addressing) เป็นการอ้างแอดเดรสแบบที่ระบุค่าออฟเซตโดยตรง เช่น

```
mov ax, [100h]
```

```
mov [200h], cl
```

```
mov cl, total
```

```
mov sum, ax
```

อ้างทางอ้อมโดยใช้รีจิสเตอร์ (Register indirect addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลอยู่ในรีจิสเตอร์ เช่น

```

mov  bx, offset buffer
mov  al, [bx]
mov  di, offset total
add  [di],al

```

อ้างแบบดัชนีโดยตรง (Direct indexed addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลได้จากการนำค่ารีจิสเตอร์ดัชนี (SI หรือ DI) มาบวกกับเลขคี่เครื่องหมายขนาดแปดบิต หรือเลขไม่คี่เครื่องหมายขนาดสิบหกบิต

```

.data
balance          dw   10 dup(?)
credit           dw   10 dup(?)
debit            dw   10 dup(?)
...
mov  cx, 10
mov  si, 0
calloop:         mov  ax, balance[si]
sub  ax, credit[si]
add  ax, debit[si]
mov  balance[si], ax
inc  si
inc  si
loop calloop

```

อ้างแบบสัมพันธ์กับฐาน (Base relative addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลได้จากการนำค่าคงที่ไปบวกกับค่าในรีจิสเตอร์ BX หรือ BP เช่น

```
.data
rec    dw    10 dup(4 dup(?))
...
mov    cx, 10
mov    bx, offset rec
updateloop:
mov    ax, [bx]      ; x
add    ax, [bx+2]    ; +y
add    ax, [bx+4]    ; +z
mov    [bx+6], ax    ; c = x+y+z
add    bx, 8
loop  updateloop
```

ถ้าคิดสัมพันธ์กับ BP ออฟเซตที่ได้จะคิดเทียบกับรีจิสเตอร์ SS (Stack Segment)

อ้างแบบดัชนีกับฐาน (Base indexed addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลได้จากการนำค่าของรีจิสเตอร์ฐาน (BX หรือ BP) รวมกับค่าของรีจิสเตอร์ดัชนี (SI หรือ DI)

```
mov    ax, [bx+si+2]
mov    [bx+di], al
inc    byte ptr [bx+si]
mov    dx, [bp+si+2]
mov    [bp+di+2], dx
```

ตัวอย่างโปรแกรมแสดงตัวอักษรแบบใหญ่

การแสดงผลตัวอักษรใหญ่นั้น ต้องมีการเก็บตัวอักษรที่จะพิมพ์ไว้ในหน่วยความจำ โดยการควรเก็บเป็นตารางจะง่ายต่อความเข้าใจ เช่น ตารางขนาด 8*8

```
.data
fontbuf db 0, 0, 0, 1, 0, 0, 0, 0
         db 0, 0, 1, 0, 1, 0, 0, 0
         db 0, 1, 0, 0, 0, 1, 0, 0
         db 1, 0, 0, 0, 0, 0, 1, 0
         db 1, 1, 1, 1, 1, 1, 1, 0
         db 1, 0, 0, 0, 0, 0, 1, 0
         db 1, 0, 0, 0, 0, 0, 1, 0
         db 0, 0, 0, 0, 0, 0, 0, 0
โดยต้องทำแบบนี้ทุกตัวอักษร( A - Z )
```

โปรแกรมย่อยสำหรับแสดงตัวอักษร หาตำแหน่งเริ่มต้นของตัวอักษร

```
mov  bx, offset fontbuf
mov  dh, 0
sub  dx, 'A'
mov  cl, 6
shl  dx, cl           ; dx = dx*16
add  bx, dx           ; bx = buf addr
```

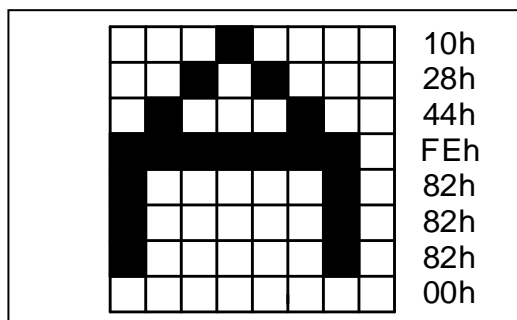
โดยตัวอักษรตัวแรกเริ่มที่ 'A' เริ่มที่ offset fontbuf แต่ละตัวตำแหน่งเพิ่มขึ้นทีละ 64 ไบต์ซึ่งมาจากการเก็บตัวอักษรในรูปของตาราง 8*8 ที่แสดงข้างต้น

การพิมพ์ตัวอักษรออกมา

```

        mov     si, 0
        mov     dh, 0
printline:
        mov     cx, 8
printonechar:
        mov     dl, [bx+si]
        call    printchar
        inc     si
        loop   printonechar
        call    printnewline
        inc     dh
        cmp     dh, 8
        jnz    printline

```



ซึ่งจากวิธีนี้ทำให้ใช้ 8 ไบต์เท่านั้นในการเก็บสำหรับหนึ่งตัวอักษร

```

.data
fontbuf db 10h, 28h, 44h, 0Feh
        db 82h, 82h, 82h, 00h
โดยที่ต้องสำหรับทุกตัวอักษร ( A - Z )

```

จากการที่ได้เปลี่ยนวิธีการเก็บตัวอักษรคือจากขนาด 64 ไบต์ เป็น 8 ไบต์ จึงต้องมีการแก้ไขส่วนของโปรแกรมแสดงผลใหม่ ดังนี้

```

        mov     si, 0
        mov     di, 0
printline:
        mov     dh, [bx+si]
        mov     cx, 8
printonechar:
        test    dh, 80h
        jz     printzero
        mov     dl, '#'
        jmp    printit
printzero:
        mov     dl, ' '
printit:
        call   printchar
        rol    dh, 1
        loop   printonechar
        call   printnewline
        inc    si
        inc    di
        cmp    di, 8
        jnz   printline

```

สรุป

8086 จะมองลักษณะของหน่วยความจำ โดยแบ่งหน่วยความจำเป็นกลุ่มๆในรูปแบบของ เซกเมนต์ ในหนึ่งเซกเมนต์จะชี้ได้ถึง 64 กิโลไบต์ เซกเมนต์ทั้งสิ้นได้แก่ CS, DS, SS และ ES จะแสดงแอดเดรสของหน่วยความจำที่จะติดต่อด้วย CS จะบรรจุค่าแอดเดรสเริ่มต้นของโปรแกรม DS จะเก็บค่าดาตาเซกเมนต์ขณะนั้น SS ก็เก็บค่าสแต็กเซกเมนต์ขณะนั้น และ ES จะกำหนดเซกเมนต์ของข้อมูลรวมที่เรียกว่า global data segment

คำถามทบทวน

1. ลักษณะการอ้างอิงแอดเดรสมีกี่แบบอะไรบ้าง
2. จงเขียนคำสั่งเพื่ออ้างอิงแอดเดรสต่อไปนี้
 - 2.1 อ้างแบบรีจิสเตอร์ (Register addressing)
 - 2.2 อ้างแบบค่าคงที่ (Immediate addressing)
 - 2.3 อ้างโดยตรง (Direct addressing)
 - 2.4 อ้างทางอ้อมโดยใช้รีจิสเตอร์ (Register indirect addressing)
 - 2.5 อ้างแบบดัชนีโดยตรง (Direct indexed addressing)
 - 2.6 อ้างแบบสัมพันธ์กับฐาน (Base relative addressing)
 - 2.7 อ้างแบบดัชนีกับฐาน (Base indexed addressing)
3. จงเขียนโปรแกรมแสดงตัวอักษร B และมีการเก็บตัวอักษรที่จะพิมพ์ไว้ในหน่วยความจำโดยเก็บเป็นตารางขนาด 8*8

แผนบริหารการสอนประจำบทที่ 15

หัวข้อเนื้อหา

- กระบวนการจัดจังหวะใน 8086
- คำสั่งติดต่อกับอุปกรณ์ใน 8086

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับกระบวนการจัดจังหวะและการใช้คำสั่งติดต่อกับอุปกรณ์ใน 8086

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 15 การขัดจังหวะ (Interrupted)

ปกติแล้วการทำงานของคอมพิวเตอร์จะประกอบด้วยการทำงานร่วมกันของหน่วยประมวลผลกลางกับอุปกรณ์รอบข้าง ซึ่งการทำงานร่วมกันนี้จำเป็นต้องมีการส่งผ่านข้อมูลระหว่างหน่วยประมวลผลกลางและอุปกรณ์อื่น ๆ อยู่เสมอ ดังนั้นรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับอุปกรณ์รอบข้างจึงต้องถูกออกแบบมาอย่างเหมาะสมกับการทำงานของอุปกรณ์ต่าง ๆ ซึ่งจะแบ่งรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับอุปกรณ์รอบข้างได้ดังนี้

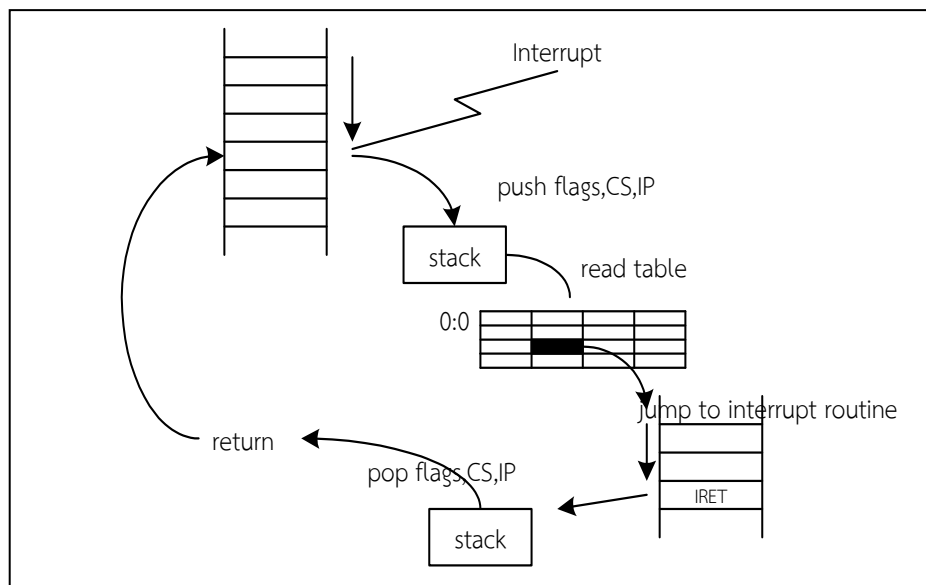
1. **Programmed I/O** หน่วยประมวลผลมีคำสั่งโดยเฉพาะสำหรับการส่งรับข้อมูลกับอุปกรณ์
2. **Memory-mapped I/O** ซึ่งการติดต่อแบบนี้นิยมใช้ในอุปกรณ์ที่ไม่มีคำสั่งพิเศษในการติดต่อกับอุปกรณ์ I/O การติดต่อจะเหมือนกับการเขียนและอ่านกับหน่วยความจำ
3. **Interrupt I/O** การที่ CPU ต้องตรวจสอบสถานะของ I/O ทำให้เสียเวลาจึงมีวิธีการติดต่อแบบนี้โดยให้ I/O รายงานแก่ CPU เองเมื่อมีการเปลี่ยนแปลงสถานะ เช่น มีข้อมูลเข้า ส่งข้อมูลเสร็จแล้ว มีลักษณะคือ I/O สร้างสัญญาณ Interrupt ให้กับ CPU เพื่อให้ CPU ประมวลผลเหตุการณ์ที่เกิดขึ้น
4. **Direct Memory Access I/O** โดยทั่วไป I/O ทำงานช้ามากเมื่อเทียบกับ CPU ดังนั้นการที่ CPU ต้องจัดการโอนย้ายข้อมูลเองจะเป็นการเสียเวลา จึงทำให้เกิดวิธีการติดต่อโดยตรงระหว่าง memory และ I/O ซึ่งในขณะนั้น CPU สามารถทำงานอื่นที่ไม่ต้องติดต่อกับ memory ได้ ขั้นตอนโดยทั่วไปมีขั้นตอนดังนี้
 - 4.1 หน่วยประมวลผลส่งงาน I/O และระบุตำแหน่งในหน่วยความจำที่จะให้ I/O เขียนผลลัพธ์หรืออ่านข้อมูล จากนั้น CPU จะไปทำงานอื่น
 - 4.2 I/O จะเขียนผลลัพธ์ที่ได้หรืออ่านข้อมูลจากหน่วยความจำโดยตรงโดยไม่ผ่าน CPU
 - 4.3 เมื่อ I/O ทำงานเสร็จจะแจ้ง CPU โดยการสร้างการขัดจังหวะ

15.1 กระบวนการขัดจังหวะใน 8086

กระบวนการขัดจังหวะหรือ interrupt เป็นกระบวนการหนึ่งที่อุปกรณ์รอบข้างจะเรียกร้องความสนใจจาก CPU เพื่อให้ CPU มาประมวลผลข้อมูลจากอุปกรณ์ที่ร้องขอนั้น ซึ่งกระบวนการนี้จะเริ่มจากอุปกรณ์รอบข้างส่งสัญญาณมาขอ interrupt จาก CPU เมื่อ CPU ได้รับสัญญาณ interrupt วงจรภายใน CPU จะพิจารณาว่าอุปกรณ์ใดที่ขอ interrupt โดยฮาร์ดแวร์ที่ควบคุมการ interrupt จะส่งค่าที่อยู่ระหว่าง 0 ถึง 255 ซึ่งเป็นหมายเลขที่ใช้แทนอุปกรณ์นั้นให้กับ CPU เมื่อ CPU ได้รับหมายเลขของอุปกรณ์ที่ส่งสัญญาณ Interrupt ก็ส่งผ่านการควบคุมไปยัง interrupt service routine โดย CPU ได้สำรองที่ 1024 ไบต์แรกในหน่วยความจำสำหรับ interrupt vector ข้อมูลที่อยู่ใน interrupt vector คือตัวชี้ที่บอกตำแหน่งที่อยู่ของ interrupt service routine ของอุปกรณ์แต่ละหมายเลขนั้น โดยที่ interrupt แต่ละหมายเลขจะเกี่ยวข้องกับ interrupt vector 4 ไบต์ เช่น interrupt vector เบอร์ 0 จะเกี่ยวข้องกับ vector ไบต์ 0 ถึง 3 โดยที่สองไบต์แรกจะชี้ไปยัง offset ของ interrupt service routine สองไบต์ต่อมาจะชี้ไปยัง segment ของ interrupt service routine

การ interrupt จะเก็บตำแหน่งที่อยู่ของคำสั่งถัดไป (return address) บน stack เนื่องจาก interrupt service routine อาจอยู่ในหน่วยความจำส่วนใดก็ได้ ดังนั้น CPU จึงต้องจัดการกับ interrupt คือจะต้องเก็บทั้ง offset และ segment ของโปรแกรมบน stack นอกจากนั้น ก็ยังเก็บค่าของ flag register ไว้บน stack ซึ่งใน flag register ก็มี interrupt enable flag (IF) บิตอยู่ด้วย การที่ CPU รับรู้ interrupt จากภายนอกได้หรือไม่ขึ้นอยู่กับค่านี้ ถ้า IF มีค่าเป็น 0 CPU จะไม่ยอมรับสัญญาณจาก maskable interrupt ขณะที่กำลัง execute interrupt service routine อยู่

เมื่อ CPU เก็บค่าของ flag register และ return address คือค่าใน register CS และ IP บน stack แล้ว ก็ใช้เบอร์ interrupt อ่านค่าตำแหน่งที่อยู่ของ interrupt service routine จาก interrupt vector แล้วย้ายการทำงานไปยัง interrupt service routine เมื่อทำมาถึงคำสั่ง IRET ซึ่งเป็นคำสั่ง return from interrupt ก็จะ pop ค่า 3 ค่าออกจาก stack และใส่ไว้ใน register IP, CS และ flag register ตามลำดับ แสดงได้ดังรูป 15.1



รูปที่ 15.1 แสดงการเก็บตำแหน่งที่อยู่ของคำสั่งถัดไป (return address) บน stack

ที่มาของการขัดจังหวะ

การขัดจังหวะของ CPU มีที่มาจาก 2 ลักษณะได้แก่

1. การขัดจังหวะที่มาจากระบบฮาร์ดแวร์ อุปกรณ์ต่างๆที่ต้องการสร้างการขัดจังหวะจะส่งสัญญาณมาที่ อุปกรณ์ควบคุมการขัดจังหวะ (Interrupt Controller) โดยสัญญาณที่เข้ามาที่อุปกรณ์ควบคุมนี้มีทั้งสิ้น 16 สัญญาณ (IRQ0 – IRQ16) จากนั้นอุปกรณ์ควบคุมการขัดจังหวะส่งสัญญาณมาที่ CPU พร้อมทั้งส่งหมายเลขของการขัดจังหวะมาพร้อมกันด้วย โดยถ้ามีการขัดจังหวะหลายอันที่ร้องขอพร้อมกันอุปกรณ์นี้จะจัดลำดับความสำคัญของอุปกรณ์และส่งสัญญาณที่มีความสำคัญสูงสุดก่อน อุปกรณ์ที่สร้างการขัดจังหวะกลุ่มนี้ได้แก่ นาฬิกาของระบบ แป้นพิมพ์ ฮาร์ดดิสก์ การ์ดเสียง ฯลฯ

2. การขัดจังหวะที่มาจากระบบซอฟต์แวร์ หรืออาจจะเรียกได้ว่าเป็นการขัดจังหวะที่เกิดขึ้นจากอุปกรณ์ภายในตัว CPU เองได้แก่ การเกิดความผิดพลาดในการทำงาน และการส่งคำสั่ง INT โดยคำสั่ง INT มีรูปแบบดังนี้

INT interrupt – type

โดยที่ interrupt – type คือเบอร์ของ interrupt ซึ่งมีค่าระหว่าง 0 ถึง 255

เมื่อคำสั่ง INT ถูก execute CPU จะดำเนินการดังต่อไปนี้คือ

1. Push ค่าของ flag register บน stack
2. Clear Trap Flag (TF) และ Interrupt Enable Flag (IF)
3. Push ค่าของ register CS บน stack
4. คำนวณตำแหน่งที่อยู่ใน interrupt vector ของ interrupt เบอร์นี้ โดยการคูณ interrupt-type ด้วย 4
5. Load ค่าใน word ที่สองใน interrupt vector สำหรับ interrupt เบอร์นี้ลงใน register CS ซึ่งก็คือค่าของ segment ของ interrupt service routine
6. Push ค่าใน register IP บน stack
7. Load ค่าใน word ที่หนึ่งใน interrupt vector เบอร์นี้ ใน register IP ซึ่งก็คือค่าของ offset ของ interrupt service routine
8. Execute interrupt service routine
 - ตัวอย่างการรับตัวอักษรจากแป้นพิมพ์และแสดงผล
 - interrupt 21H เป็นการเรียกใช้งานฟังก์ชัน โดยใช้ register AH บรรจุหมายเลขฟังก์ชัน

```

;DEMONSTRATE FUNCTION 01H
;READ KEYBOARD AND DISPLAY
CODE          SEGMENT      PAGE
              ASSUME      CS:CODE,DS:CODE
;*****
;* MACRO DEFINITION *
;*****
READ_KBD_ECHO  MACRO
    MOV  AH, 01H          ;รอรับการกดตัวอักษรหนึ่งตัวจากแป้นพิมพ์
    INT  21H             ;รหัสตัวอักษรที่กดถูกส่งไป register AL
    ENDM
DISP_CHAR     MACROCHARACTER
    MOV  DL, CHARACTER
    MOV  AH, 02H         ;แสดงผลตามค่าที่อยู่ใน register DL

```

```

        INT     21H
        ENDM

DISPLAY     MACROSTRING
        MOV    DX, OFFSET STRING
        MOV    AH, 09H                ;แสดงผลสายตัวอักษร (string) บนจอภาพ
                                           ;จนกว่าจะพบตัวอักษร

        INT     21H                ;'$', โดยregister DX จะบรรจุค่า offset
                                           ;(จาก segment DS)

        ENDM

TERMINATE   MACRO
        MOV    AH,4CH
        INT    21H
        ENDM

        ORG    0100H

START:     DISPLAY     SHOW                ;SHOW MESSAGE
        DISP_CHAR    10                ;SEND LINEFEED
FUNC_01H:  READ_KBD_ECHO ;THIS FUNCTION
        CMP         AL,0DH                ;IS IT A CR-RETURN
        JNE         FUNC_01H            ;NO,LOOP BACK
        DISP_CHAR    10                ;
        TERMINATE

SHOW       DB        'TYPE ANY KEY !',13,10
          DB        'PRESS RETURN TO TERMINATE',13,10,'$'

CODE       ENDS
          END     START

```

15.2 คำสั่งติดต่อกับอุปกรณ์ใน 8086

หน่วยประมวลผลของ 8086 สามารถติดต่อกับอุปกรณ์ได้ทั้งแบบ Programmed I/O, Interrupt I/O, และ Direct Memory Access โดยคำสั่งเขียนอ่านมีรูปแบบดังนี้

IN accumulator, DX
 IN accumulator, portnumber
 OUT DX, accumulator
 OUT portnumber, accumulator

โดยที่

accumulator คือ register AX หรือ AL ขึ้นกับว่าเป็นการติดต่อแบบ 8 บิต หรือ 16 บิต
 portnumber คือหมายเลขของอุปกรณ์ (หมายเลข I/O) เราสามารถระบุค่านี้โดยผ่านทาง register DX ได้ หมายเลขพอร์ตนี้มีค่าตั้งแต่ 0 ถึง 65535

ภายในเครื่องคอมพิวเตอร์จะมีลำโพงเล็กๆโดยที่โปรแกรมสามารถควบคุมเสียงที่ออกจากลำโพงนี้ โดยควบคุมผ่านพอร์ต 61H ใช้ในการนำข้อมูลออกไปให้โปรแกรมและค่านี้จะคงอยู่อย่างนั้นจนกระทั่งโปรแกรมเปลี่ยนค่า แต่ละบิตของพอร์ต 61H มีความหมายดังนี้

บิต 0 timer 2 gates (ควบคุมลำโพง)
 1 speaker direct control
 2 multiplex port 62H
 3 cassette motor control
 4 enable parity check on system board memory
 5 enable parity check on I/O board memory
 6 keyboard clock control
 7 keyboard clear/multiplex port 60H

การส่งงานลำโพงอย่างง่ายสามารถทำได้โดยส่งงานบิตที่ 1 โดยการกำหนดค่าลงในบิตนี้เปลี่ยนค่าไปมาระหว่าง 0 และ 1 ซึ่งจะทำให้ลำโพงเกิดการสั่นขึ้น การส่งงานนั้นต้องส่งงานโดยไม่ให้กระทบกับบิตอื่นๆ เช่น

in AL, 61H
 xor AL, 02H
 out 61H, AL

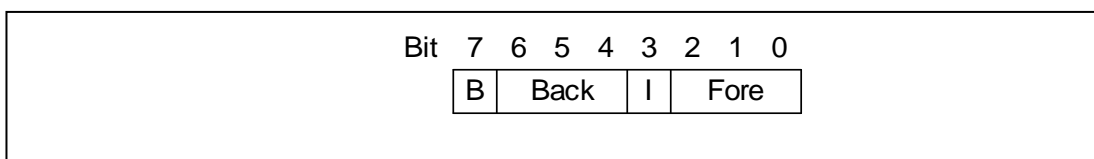
ในการสร้างเสียงต้องมีการหน่วงเวลาในการเปลี่ยนค่าบิตควบคุมลำโพง (บิตที่ 1) เพราะความถี่สูงเกินไป ลำโพงจะไม่สามารถเคลื่อนที่ทันและเพื่อให้ผู้ใช้สามารถได้ยินเสียงนั้นด้วย

ตัวอย่างโปรแกรมย่อยสำหรับสร้างเสียงอย่างง่าย

```
gensound    proc    near
; CX    delay [for freq]
; DX    duration
        push    ax
        push    dx
gensignal:
        in     al, 61h
        xor    al, 02h
        out    61h, al
        push    cx
delayloop:
        nop    ;no operation
        loop   delayloop
        pop    cx
        dec    dx
        or     dx, dx
        jnz    gensignal
        pop    dx
        pop    ax
        ret
gensound    endp
```

ตัวอย่างการติดต่อกับอุปกรณ์รอบข้าง : ระบบการแสดงผลแบบตัวอักษร

การแสดงผลแบบตัวอักษรใน IBM/PC นั้น ในแต่ละตัวอักษรที่แสดงจะมีค่าที่เกี่ยวข้องสองค่าคือ รหัสแอสกีของตัวอักษรนั้น และแสดงค่า display attribute ของตัวอักษร ซึ่งเป็นค่าที่บอกสีของตัวอักษรและลักษณะการแสดงผล โดยบิตที่ 0-3 แสดงสีของตัวอักษร และบิตที่ 4-7 แสดงสีของพื้นหลัง



ใช้บริการของ Video BIOS [Interrupt 10H]

ตัวอย่างแสดงรูปบนจอโดยใช้ BIOS interrupt เบอร์ 10H

```

; PROGRAM DRAW A DIAGONAL LINE OF SMILE FACES ON SCREEN
CODE_SEG      SEGMENT
               ASSUME     CS:CODE-SEG
DIAG  PROC  FAR
               PUSH  DS           ;SAVE RETURN ADDRESS
               MOV   AX, 0
               PUSH  AX
               PUSH  AX           ;SAVE REGISTERS AX
               PUSH  BX           ;SAVE REGISTERS BX
               PUSH  CX           ;SAVE REGISTERS CX
               PUSH  DX           ;SAVE REGISTERS DX
               STI                ;ENABLE INTERRUPT
               MOV   AH, 15       ;SET BH TO ACTIVE PAGE
               INT   10H
               MOV   CX, 1
               MOV   DX, 0       ;START AT ROW=0 COLUMN=0
CRSR:  MOV   AH, 2               ;MOVE CURSOR TO NEXT POSITION
               INT   10H
               MOV   AL, 2       ;CHARACTER DISPLAY
               MOV   AH, 10      ;WRITE CHARACTER TO SCREEN
               INT   10H
               INC   DH           ;POINT TO NEXT ROW
               INC   DL           ;POINT TO NEXT COLUMN
               CMP   DH, 25      ;BOTTOM OF SCREEN
               JNE   CRSR
               POP   DX
               POP   CX           ;RESTORE REGISTERS
               POP   BX
               POP   AX
               RET
DIAG  ENDP
CODE_SEG  ENDS
END

```

ในหน่วยแสดงผลแบบ VGA หน่วยความจำตั้งแต่ตำแหน่งที่ 0B800h: 0000h จะเป็นหน่วยความจำที่เก็บข้อมูลของหน้าจอ โดยการเก็บข้อมูลจะมีลักษณะเก็บทีละตัวอักษรเรียงกันไปตามลำดับ ซึ่งการเก็บข้อมูลจะเก็บตัวอักษรละ 2 ไบต์ ไบต์แรก (ไบต์ต่ำ) จะเก็บค่ารหัสแอสกี ไบต์ที่สองจะเก็บค่า display attribute การเก็บจะเก็บเรียงทีละบรรทัด บรรทัดละ 80 ตัวอักษร ตัวอย่างการคำนวณข้อมูลของตัวอักษรที่บรรทัดที่ 10 ตัวที่ 3 อยู่ที่ offset $10*(80*2)+3*2 = 1606$

ตัวอย่างโปรแกรมแสดงตัวอักษรทั้งหมดบนจอภาพ

CODE_SEG	SEGMENT		
START	PROC	FAR	
	ASSUME	CS:CODE_SEG	
	;SAVE RETURN ADDRESS		
	PUSH	DS	
	MOV	AX, 0	
	PUSH	AX	
	;CLEAR THE DISPLAY		
	MOV	AX, 0B0000H	
	MOV	ES, AX	;ES POINTS TO BASE OF ADAPTE
	MOV	DI, 0	
	MOV	AL, ' '	;BLANK CHARACTER
	MOV	AH, 07H	;NORMAL VIDEO
	MOV	CX, 2000	
	REP	STOSW	;BLANK OUT VIDEO DISPLAY
	;FILL THE DISPLAY WITH 256 CHARACTERS WITH A BLANK COLUMN		
	;AND LINE BETWEEN ALL CHARACTERS		
	MOV	AL, 0	;CHARACTER TO DISPLAY
	MOV	AH, 0	;COLUMN NUMBER
	MOV	DI, 160	;POINT TO FIRST COLUMN IN ROW 2
AGAIN:	MOV	ES:[DI], AL	;PUT CHARACTER IN MEMORY
	ADD	DI, 4	;POINTS TO NEXT POSITION
	ADD	AH, 2	
	CMP	AH, 80	;FINISHED THIS ROW ?
	JB	SAME_LINE	


```

;IF COLUMN NUMBER IS GREATER THAN 80 THEN SKIP A LINE
    ADD        DI, 160
    MOV        AH, 0
SAME_LINE:   CMP    AL, 255
    JE        FINISHED           ;256 CHARACTERS YET
    INC        AL                 ;UPDATE TO NEXT CHARACTER
    JMP        AGAIN

;AFTER DISPLAYING ALL THE CHARACTERS WAIT FOR A KEY TO BE
;PRESSED THEN BLANK THE DISPLAY AND RETURN TO DOS
    MOV        AH, 0             ;GET KEYBOARD INPUT
    INT        16H
    MOV        DI, 0             ;INITIAL OFFSET ADDRESS
    MOV        AL, ' '           ;BLANK CHARACTER
    MOV        AH, 07H           ;NORMAL DISPLAY ATTRIBUTE
    MOV        CX, 2000
    REP        STOSW             ;BLANK ADAPTER MEMORY
    RET

START        ENDP
CODE_SEG     ENDS

```

ตัวอย่างโปรแกรมย่อยเขียนตัวอักษรลงบนหน้าจอโดยตรง

```

WRITEONECHAR PROC     NEAR
;( DH, DL ) : ( ROW, COL )
;AL: CHAR      AH: ATTRIBUTE
    PUSH        DS
    PUSH        AX
    PUSH        BX
    PUSH        CX
    PUSH        DX
    MOV        AX, 0B800H
    MOV        DS, AX
    MOV        CL, DH
    PUSH        AX
    MOV        AL, 160
    MUL        CL           ; AX=ROW*160
    MOV        BX, AX

```

```
POP      AX
MOV      DH, 0
SHL      DX, 1
ADD      BX, DX
MOV      [BX], AX
POP      DX
POP      CX
POP      BX
POP      AX
POP      DS
RET
WRITEONECHAR      ENDP
```

สรุป

กระบวนการขัดจังหวะใน 8086 เป็นกระบวนการหนึ่งที่อุปกรณ์รอบข้างจะเรียกร้องความสนใจจาก CPU เพื่อให้ CPU มาประมวลผลข้อมูลจากอุปกรณ์ที่ร้องขอนั้น ซึ่งกระบวนการนี้จะเริ่มจากอุปกรณ์รอบข้างส่งสัญญาณมาขอ interrupt จาก CPU เมื่อ CPU ได้รับสัญญาณ interrupt วงจรภายใน CPU จะพิจารณาว่าอุปกรณ์ใดที่ขอ interrupt โดยฮาร์ดแวร์ที่ควบคุมการ interrupt จะส่งค่าที่อยู่ระหว่าง 0 ถึง 255 ซึ่งเป็นหมายเลขที่ใช้แทนอุปกรณ์นั้นให้กับ CPU เมื่อ CPU ได้รับหมายเลขของอุปกรณ์ที่ส่งสัญญาณ Interrupt ก็ส่งผ่านการควบคุมไปยัง interrupt service routine โดย CPU ได้สำรองที่ 1024 ไบต์แรกในหน่วยความจำสำหรับ interrupt vector ข้อมูลที่อยู่ใน interrupt vector คือตัวชี้ที่บอกตำแหน่งที่อยู่ของ interrupt service routine ของอุปกรณ์แต่ละหมายเลขนั้น โดยที่ interrupt แต่ละหมายเลขจะเกี่ยวข้องกับ interrupt vector 4 ไบต์ เช่น interrupt vector เบอร์ 0 จะเกี่ยวข้องกับ vector ไบต์ 0 ถึง 3 โดยที่สองไบต์แรกจะชี้ไปยัง offset ของ interrupt service routine สองไบต์ต่อมาจะชี้ไปยัง segment ของ interrupt service routine

คำถามทบทวน

1. จงอธิบายรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับอุปกรณ์รอบข้างต่อไปนี้
 - 1.1 Programmed I/O
 - 1.2 Memory-mapped I/O
 - 1.3 Interrupt I/O
 - 1.4 Direct Memory Access I/O
2. การขัดจังหวะของ CPU มีกี่ลักษณะอะไรบ้าง
3. เมื่อคำสั่ง INT ถูก execute CPU จะดำเนินการโดยมีขั้นตอนการทำงานอย่างไร
4. จงอธิบายหน่วยประมวลผลของ 8086 สามารถติดต่อกับอุปกรณ์ในแบบต่างๆ ต่อไปนี้ได้อย่างไร
 - 4.1 Programmed I/O
 - 4.2 Interrupt I/O
 - 4.3 Direct Memory Access
5. จงเขียนโปรแกรมแสดงตัวอักษรที่มีคำว่า "Computer Science Suan Dusit" บนจอภาพ

แผนบริหารการสอนประจำบทที่ 16

หัวข้อเนื้อหา

- คำสั่ง MOVS
- คำสั่ง REP
- คำสั่ง STO
- คำสั่ง LODS
- คำสั่ง REPZ
- คำสั่ง CMPS
- คำสั่ง REPNZ
- คำสั่ง SCAS

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับคำสั่งจัดการสายข้อมูลต่างๆ เช่น คำสั่ง MOVS, คำสั่ง REP, คำสั่ง STOS, คำสั่ง LODS, คำสั่ง REPZ, คำสั่ง CMPS, คำสั่ง REPNZ และคำสั่ง SCAS เป็นต้น

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

1. การวัดผลและประเมินผล

1.1 ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

2. การประเมินผล

2.1 วิธีตรวจผลงานต่างๆ ที่ให้ทำ

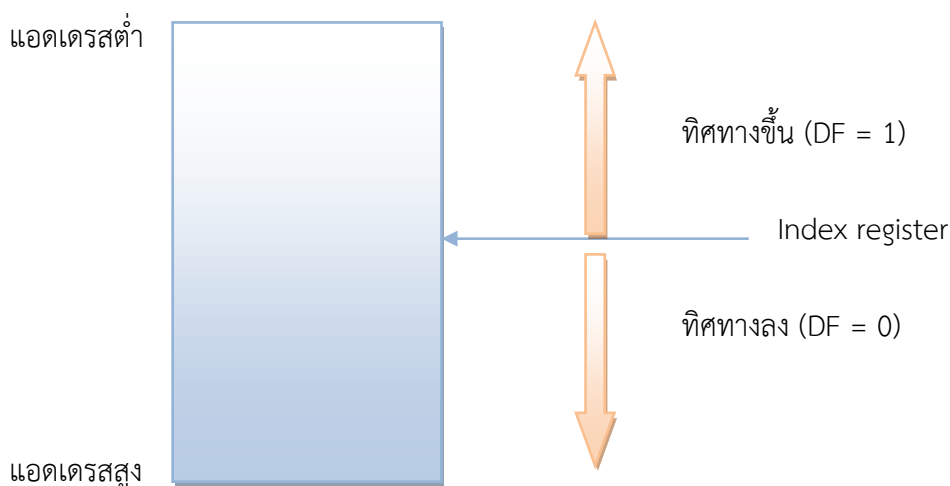
- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

2.2 ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 16 คำสั่งจัดการกับสายข้อมูล (Character Instruction)

คำสั่งจัดการกับสายข้อมูล เป็นคำสั่งที่ใช้จัดการกับข้อมูลเป็นที่เป็นชุดเรียงต่อกัน คำสั่งในกลุ่มนี้จะระบุตำแหน่งของข้อมูลด้วย Index register คือ SI (Source Index) และ DI (Destination Index) ประกอบด้วย DS และ ES โดย ข้อมูลต้นทางจะอยู่ที่ตำแหน่ง DS:SI และข้อมูลปลายทางจะอยู่ที่ ES:DI การทำงานคำสั่งในกลุ่มนี้ จะมีการปรับค่าของรีจิสเตอร์ที่เก็บตำแหน่งของข้อมูลให้โดยอัตโนมัติ เพื่อว่าเราจะสามารถใช้คำสั่งนั้นกระทำกับข้อมูลที่ตำแหน่งติดกันถัดไปได้โดยไม่ต้องปรับค่าของรีจิสเตอร์เอง ทิศทางในการปรับจะขึ้นกับค่าที่กำหนดในแฟล็กทิศทาง (DF : Direction flag)

ถ้า DF เป็น 0 แสดงว่ามีการเพิ่มตำแหน่งของการทำงาน โดยจะชี้ไปที่ข้อมูลตัวถัดไป คือ ค่าของรีจิสเตอร์ดัชนีจะเพิ่มขึ้น แต่ถ้า DF เป็น 1 แสดงว่าการทำงานจะเป็นการลดตำแหน่ง โดยจะชี้ไปที่ข้อมูลที่อยู่ก่อนหน้านี้ คือรีจิสเตอร์ดัชนีจะมีค่าลดลง



การกำหนดค่าของ DF สามารถทำได้ด้วยคำสั่ง STD (set direction flag) ซึ่งจะทำให้ ค่าใน DF เป็น 1 และ CLD (clear direction flag) ซึ่งจะทำให้ค่าใน DF เป็น 0 คำสั่งต่างๆ ในกลุ่มนี้

MOVSx	Move string
LODSx	Load string
STOSx	Store string
SCASx	Scan string
CMPSx	Compare string

x แทนได้ถึง B (byte) หรือ W (word) หรือ D (double) ซึ่งหมายถึงขนาดของข้อมูลที่ถูกกระทำด้วยคำสั่งนั้นๆ

16.1 คำสั่ง MOVS

คำสั่ง MOVS เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลจากแหล่งข้อมูลต้นทางไปยังแหล่งข้อมูลปลายทาง โดยมีรูปแบบดังนี้

```
MOVSB
MOVSW
MOVSD
```

คำสั่ง MOVS จะทำการคัดลอกข้อมูลจากตำแหน่ง DS:SI ไปยังตำแหน่ง ES:DI พร้อมทั้งปรับค่าในรีจิสเตอร์ DI และ SI คำสั่ง MOVSB ใช้ในการคัดลอกค่าขนาดไบต์ คำสั่ง MOVSW ใช้คัดลอกค่าขนาดเวิร์ด และคำสั่ง MOVSD ใช้คัดลอกค่าขนาดดับเบิลเวิร์ด

ตัวอย่าง โปรแกรมคัดลอกข้อมูลจำนวน 100 ตัวที่เริ่มต้นที่เลเบล d1 ไปยังเลเบล d2

```
.data
d1      dw    100 dup (N)
d2      dw    100 dup (?)

.code
mov     ax,@data
mov     ds,ax
mov     es,ax
mov     si,offset d1      ; source address
mov     di,offset d2      ; destination address
cld                                ; move in forward direction
mov     cs,100            ; 100 byte to be moved
copy:   movsw
        loop copy
...
```

16.2 คำสั่ง REP

คำสั่งการกระทำกับสายข้อมูลนั้นมักจะต้องทำงานซ้ำเป็นวงรอบเพราะในการเรียกคำสั่งแต่ละครั้งนั้นจะเป็นการกระทำกับข้อมูลทีละ 1 ตัวเท่านั้น ดังเช่นในตัวอย่างข้างต้นเป็นการใช้คำสั่ง MOVSW ร่วมกับคำสั่ง LOOP แต่เราก็อาจจะนำคำสั่ง REP มาใช้แทนได้

คำสั่ง REP ใช้หน้าหน้าคำสั่งอื่นๆ เพื่อให้ทำคำสั่งนั้นซ้ำ โดยในการกระทำคำสั่งแต่ละครั้งนั้นจะมีการลดค่าของรีจิสเตอร์ CX ลงครั้งละ 1 จนกว่าค่าของรีจิสเตอร์ CX จะเท่ากับ 0 ซึ่งก็คล้ายกับการทำงานของคำสั่ง LOOP นั่นเอง

ตัวอย่างคำสั่ง

```

mov cx,100
copy: movsw
loop copy

```

เราสามารถแทนบรรทัดข้างต้นทั้ง 3 ได้ด้วย

```

mov cx,500
rep movsw

```

16.3 คำสั่ง STOS

คำสั่ง STOS เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลขนาดไบต์จากรีจิสเตอร์ AL หรือ ขนาดเวิร์ดจากรีจิสเตอร์ AX หรือ ขนาดดับเบิลเวิร์ดจากรีจิสเตอร์ EAX ไปยังหน่วยความจำตำแหน่ง ES:DI พร้อมทั้งปรับค่ารีจิสเตอร์ DI โดยมีรูปแบบดังนี้

```

STOSB
STOSW
STOSD

```

ตัวอย่าง โปรแกรมกำหนดค่าเริ่มต้นเท่ากับ 0 ให้กับข้อมูล 100 ตัวที่เริ่มต้นที่หน่วยความจำตำแหน่ง d1

```

.data
d1 dw 100 dup (?)
.code
mov ax,@data
mov ds,ax
mov es,ax
mov ax,0
mov di,offset d1 ; destination address
cld ; fill them forwards
mov cx,100 ; number of locations to be filled
rep stosw
...

```

16.4 คำสั่ง LODS

คำสั่ง LODS เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลขนาดไบต์จากหน่วยความจำตำแหน่ง DS:SI ไปยังรีจิสเตอร์ AL หรือ ขนาดเวิร์ดไปยังรีจิสเตอร์ AX หรือ ขนาดดับเบิลเวิร์ดไปยังรีจิสเตอร์ EAX ไปยัง พร้อมทั้งปรับค่ารีจิสเตอร์ SI โดยมีรูปแบบดังนี้

LODSB
LODSW
LODSD

ตัวอย่าง โปรแกรมหาค่าผลรวมของข้อมูลแบบไบต์ 100 ตัวที่เริ่มต้นที่ d1 และเก็บผลรวมที่ได้ไว้ใน DX

```
.data
d1      db    100 dup (?)

.code
mov     ax,@data
mov     ds,ax
...
mov     dx,0
mov     si,offset d1      ; destination address
cld                    ; forward direction
mov     cx,100           ; number of bytes to be moved
calsum: lodsb           ;move the next byte into AL
add     dl,al            ;add AL to DX
adc     dh,0
loop   calsum           ;if more to do, go back and repeat
...
```

โดยปกติเราจะไม่พบการใช้คำสั่ง REP ร่วมกับคำสั่ง LODS

ตัวอย่างคำสั่ง

```
.data
d1      dw    100 dup (?)

.code
mov     ax,@data
```



```

mov  ds,ax
...
mov  di,offset d1
cld
mov  cx,100
rep  lodsw
...

```

ข้างต้นเป็นตัวอย่างการใช้คำสั่ง LODS อ่านค่าจากหน่วยความจำตำแหน่ง d1 ทีละเวิร์ดไปเก็บยังรีจิสเตอร์ AX ซึ่งจะเห็นได้ว่าค่าในรีจิสเตอร์ AX จะถูกเขียนทับไปเรื่อยๆ โดยในที่สุดก็มีเพียงค่าสุดท้ายเท่านั้น ไม่มีประโยชน์อันใด

ตัวอย่าง โปรแกรมคำนวณค่าจำนวน 100 ค่าเพื่อใส่ลงในหน่วยความจำที่เริ่มต้นที่ตำแหน่ง d2 โดยมีค่าเท่ากับค่าในหน่วยความจำที่เริ่มต้นที่ตำแหน่ง d1 ที่มีลำดับเท่ากันคูณด้วย 2

```

.data
d1    db    100 dup (?)
d2    db    100 dup (?)

.code
mov   ax,@data
mov   ds,ax
...
mov   ax,ds
mov   es,ax
mov   si,offset d1    ;source address
mov   di,offset d2    ;destination address
cld                                     ;forward direction
mov   cx,100          ;number of bytes to be moved
calsum: lodsb         ;move next byte to AL
shl   al,1           ;multiply AL by 2
stosb                 ;store it in the new location
loop  calsum         ;if more to do ,go back and repeat
...

```

16.6 คำสั่ง REPZ

คำสั่ง REPZ มีการทำงานคล้ายกับคำสั่ง REP แต่คำสั่ง REPZ จะกระทำซ้ำเมื่อ zero flag มีค่าเป็น 1 และ CX มีค่าไม่เท่ากับ 0

16.7 คำสั่ง CMPS

คำสั่ง CMPS จะนำค่าในหน่วยความจำตำแหน่ง ES:DI มาลบออกจากค่าในหน่วยความจำตำแหน่ง DS:SI แล้วปรับค่าแฟล็กต่างๆ ที่เกี่ยวข้องโดยค่าของข้อมูลในหน่วยความจำยังคงเดิม และปรับค่าของรีจิสเตอร์ DI และ SI โดยอัตโนมัติ คำสั่ง CMPS มีรูปแบบดังนี้

CMPSB
CMPSW
CMPSD

คำสั่ง CMPS มักจะใช้ในการเปรียบเทียบ 2 สตริงว่าเหมือนกันหรือไม่ในการจะค้นหาว่าข้อมูลของสตริง 2 ตัวว่ามีค่าเท่ากันหรือไม่นั้น เราจะต้องทำการเปรียบเทียบข้อมูลไปที่ละตัวจนกว่าจะหมดข้อมูลหรือพบข้อมูลที่ไม่เท่ากันเป็นตัวแรก จากที่ผ่านมาเราใช้คำสั่ง REP ร่วมกับคำสั่งอื่นเพื่อให้คำสั่งนั้นซ้ำเท่าจำนวนที่เรากำหนด(ในรีจิสเตอร์ CX) แต่ถ้าเราต้องการให้หยุดกระทำคำสั่งเมื่อพบจุดที่แตกต่าง นั่นก็คือเราจะใช้คำสั่ง REPZ แทนคำสั่ง REP เพราะคำสั่ง REPZ จะหยุดกระทำเมื่อ Z-flag เป็น 0 นั่นคือเมื่อคำสั่ง CMPS ทำการเปรียบเทียบพบข้อมูลที่ต่างกันนั่นเอง

ตัวอย่าง โปรแกรมเปรียบเทียบข้อมูลในหน่วยความจำที่ตำแหน่งเริ่มต้นที่ d1 และในหน่วยความจำตำแหน่งเริ่มต้นที่ d2

.data		
d1	db	100 dup (?)
d2	db	100 dup (?)
.code		
mov	ax,@data	
mov	ds,ax	
...		
mov	bx,ds	
mov	es,bx	
mov	si, offset d1	;start address of first string
mov	di, offset d2	;start address of second string
cld		;forward direction

```

mov cx,100 ;the number of words to be compare
repz cmpsw
jnz differ
same: ...
...
jmp done
differ: ...
...
done: ...
...

```

16.8 คำสั่ง REPZ

คำสั่ง REPZ มีการทำงานคล้ายกับคำสั่ง REP แต่คำสั่ง REPZ จะกระทำซ้ำเมื่อ zero flag มีค่าเป็น 0 และ CX มีค่าไม่เท่ากับ 0

16.9 คำสั่ง SCAS

คำสั่ง SCAS จะค้นหาข้อมูลสำหรับค่าขนาดไบต์ที่กำหนดใน AL หรือขนาดเวิร์ดใน AX หรือขนาดดับเบิลเวิร์ดใน EAX โดยเริ่มต้นที่หน่วยความจำตำแหน่ง ES:DI จากนั้นจะมีการปรับค่าของรีจิสเตอร์ DI โดยอัตโนมัติ โดยมีรูปแบบดังนี้

```

SCASB
SCASW
SCASD

```

คำสั่ง SCAS มักจะใช้ในการค้นหาข้อมูลภายในสตริง ในการจะค้นหานั้น เราจะต้องทำการเปรียบเทียบข้อมูลไปทีละตัวจนกว่าจะหมดข้อมูลหรือพบข้อมูลที่ต้องการ ในทำนองเดียวกับคำสั่ง CMPS เราไม่สามารถใช้คำสั่ง REP ได้เพราะเราต้องการให้หยุดกระทำคำสั่งเมื่อพบข้อมูลที่ต้องการ เราจึงใช้คำสั่ง REPZ แทนคำสั่ง REP เพราะคำสั่ง REPZ จะหยุดกระทำเมื่อ Z-flag เป็น 1 นั่นคือเมื่อคำสั่ง SCAS ทำการเปรียบเทียบพบข้อมูลที่เหมือนกันนั่นเอง

ตัวอย่าง โปรแกรมค้นหาอักขระ 'X' จากข้อมูล 100 ตัว

```

.data
d1 db 100 dup (?)
.code
mov ax,@data
mov ds,ax

```

```

...
mov  bx,ds
mov  es,bx
mov  di,offset d1      ;first location to be searched
cld                          ;forward direction
mov  cx,100           ;the number of locations to be searched
      mov  al, 'X'
repnz scasb           ;do the search
jz   found
notfnd: ...
      jmp  done
found: ...
      ...
done: ...

```

ตัวอย่าง โปรแกรมสำหรับนำข้อมูลที่เริ่มต้นที่ d2 จำนวน 100 ตัวไปต่อท้ายข้อมูลที่เริ่มต้นที่ d1 โดยถือว่าข้อมูลที่เริ่มต้นที่ d1 ถูกปิดท้ายด้วย null string

```

...
mov  ax,@data
mov  ds,ax
mov  bx,ds
mov  es,bx
mov  al, 0
mov  di, offset d1      ;first location to be searched
cld                          ;forward direction
repnz scasb           ;do the search
      mov  si, offset d2      ;start address of second string
      mov  cx,100           ;number of bytes to be moved
rep  movsb           ;do the move
...

```

สรุป

คำสั่งจัดการกับสายข้อมูล เป็นคำสั่งที่ใช้จัดการกับข้อมูลเป็นที่เป็นชุดเรียงต่อกัน คำสั่งในกลุ่มนี้จะระบุตำแหน่งของข้อมูลด้วย Index register คือ SI (Source Index) และ DI (Destination Index) ประกอบด้วย DS และ ES โดย ข้อมูลต้นทางจะอยู่ที่ตำแหน่ง DS:SI และข้อมูลปลายทางจะอยู่ที่ ES:DI การทำงานคำสั่งในกลุ่มนี้ จะมีการปรับค่าของรีจิสเตอร์ที่เก็บตำแหน่งของข้อมูลให้โดยอัตโนมัติ เพื่อว่าเราจะสามารถใช้คำสั่งนั้นกระทำกับ ข้อมูลที่ตำแหน่งติดกันถัดไปได้โดยไม่ต้องปรับค่าของรีจิสเตอร์เอง ทิศทางในการปรับจะขึ้นกับค่าที่กำหนด ในแฟล็กทิศทาง (DF : Direction flag)

คำถามทบทวน

1. จงอธิบายการทำงานของรีจิสเตอร์ต่อไปนี้ SI DI DS ES และ DF
2. จงแสดงและจงอธิบายการทำงานของคำสั่งต่อไปนี้
 - 2.1 คำสั่ง MOVS
 - 2.2 คำสั่ง REP
 - 2.3 คำสั่ง STOS
 - 2.4 คำสั่ง LODS
 - 2.5 คำสั่ง REPZ
 - 2.6 คำสั่ง CMPS
 - 2.7 คำสั่ง REPNZ
 - 2.8 คำสั่ง SCAS
3. จงเขียนโปรแกรมสำหรับนำข้อมูลที่เริ่มต้นที่ data2 จำนวน 25 ตัวไปต่อท้ายข้อมูลที่เริ่มต้นที่ data1 โดยถือว่าข้อมูลที่เริ่มต้นที่ data1 ถูกปิดท้ายด้วย null string

แผนบริหารการสอนประจำบทที่ 17

หัวข้อเนื้อหา

- คำสั่ง XLAT
- แมคโคร
- พารามิเตอร์ของแมคโคร
- การใช้ LABEL ในแมคโคร
- แมคโครกับโปรแกรมย่อย

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับคำสั่ง XLAT
- มีความรู้และความเข้าใจเกี่ยวกับพารามิเตอร์ การใช้ LABEL ในแมคโคร
- สามารถประยุกต์ใช้งานคำสั่งแมคโคร (Macro) กับโปรแกรมย่อยได้

กิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 17 คำสั่งตารางและการสร้างแมโคร (Macro and Instruction Table)

17.1 คำสั่ง XLAT

คำสั่งด้านล่างนี้เป็นคำสั่งที่ผิดรูปแบบไม่สามารถใช้ได้

```
mov    al,[bx+al]
```

อาจจะมองได้ว่า ค่าใน AL เป็นหมายเลขช่องของตารางและค่าใน BX เป็นแอดเดรสของตาราง คำสั่งนี้ก็คือการนำเอาค่าในตารางที่ตำแหน่งด้วย BX ช่องที่ AL มาใส่ใน AL นั่นเอง ซึ่งในหลายๆครั้งมีความต้องการจะทำงานในลักษณะนี้ คำสั่งข้างต้นสามารถใช้งานได้ด้วยคำสั่ง XLAT แทน

ตัวอย่าง โปรแกรมแปลงค่าเลขฐานสิบหกหนึ่งหลักใน AL เป็นอักขระฐานสิบหก

```
.data
hextab db    '0123456789ABCDEF'
...

.code
...
mov    bx,offset hextab
xlat
...
```

17.2 แมโคร

แมโครคือส่วนของโปรแกรมที่ได้กำหนดชื่อไว้ เมื่อใดก็ตามที่มีการเรียกใช้ชื่อนั้นในโปรแกรม ส่วนของโปรแกรมที่มีชื่อดังกล่าวก็จะถูกนำไปแทนที่แมโครมีรูปแบบการประกาศดังนี้

```
name    MACRO parameters
...
ENDM
```

ตัวอย่างคำสั่ง

```

print    macro
    mov   bx,offset msg
    mov   ah,09h
    int   21h
    endm

.data
msg      db    'TEST$'

.code

...

print

...

```

เมื่อโปรแกรมข้างต้นเมื่อถูกขยายจะมีลักษณะดังนี้

```

.data
msg      db    'TEST$'

.code

...

;print

mov   bx,offset msg
mov   ah,09h
int   21h

...

```

17.3 พารามิเตอร์ของแมคโคร

พารามิเตอร์ของแมคโครนั้น จะถูกนำไปแทนที่ทุกจุดในตัวแมคโคร
ตัวอย่างคำสั่ง

```

print    macro prt
    mov   bx,offset prt
    mov   ah,09h
    int   21h
    endm

.data

```



```

msg1    db    'TEST$'
msg2    db    'TEST2$'
.code
...
print  msg1
print  msg2

```

เมื่อโปรแกรมข้างต้นเมื่อถูกขยายจะมีลักษณะดังนี้

```

.data
msg1    db    'TEST$'
msg2    db    'TEST2$'
.code
...
;print  msg1
mov     bx,offset msg1
mov     ah,09h
int     21h
endm
;print  msg2
mov     bx,offset msg2
mov     ah,09h
int     21h
endm
...

```

17.4 การใช้ LABEL ในแมคโคร

ตัวอย่างคำสั่ง

```

test1   macro
    mov  ...
label1:
...
endm
.code
...

```

```

test1
...
test1
...

```

เมื่อโปรแกรมข้างต้นเมื่อถูกขยายจะมีลักษณะดังนี้

```

.code
...
;test1
mov ...
label1:
...
;test1
mov ...
label1:
...

```

จะเห็นได้ว่าการซ้ำกันของ LABEL สามารถกำหนดให้ MASM สร้าง LABEL ที่มีชื่อไม่ซ้ำกันได้โดยจะต้องประกาศให้เป็น LABEL แบบ LOCAL

ตัวอย่างคำสั่ง

```

test1 macro
local label1
mov ...
label1:
...
endm

```

เมื่อโปรแกรมข้างต้นเมื่อถูกขยายจะมีลักษณะดังนี้

```

.code
...
;test1
mov ...

```

```
XXXX1:  
...  
;test1  
mov ...  
XXXX2:  
...
```

17.5 แมคโครกับโปรแกรมย่อย

คำสั่งในส่วนของแมคโครจะถูกนำไปแทนที่ในตำแหน่งที่มีการเรียกใช้ ในการทำงานจริงจะไม่มีกรกระโดดไปทำงาน แต่ถ้ามีการเรียกใช้แมคโครหลายครั้ง โปรแกรมส่วนนั้นก็จะถูกขยายออกมาหลายครั้ง ส่วนโปรแกรมย่อยจะเป็นส่วนของโปรแกรมที่มีเพียงชุดเดียว การเรียกใช้โปรแกรมย่อยจะเป็นการกระโดดไปทำงานในโปรแกรมย่อยนั้น

แมคโครมีข้อดี คือ ลดเวลาในการกระโดดไปทำงานสำหรับส่วนของคำสั่งที่มีขนาดเล็ก แต่แมคโครก็มีข้อเสีย คือ ถ้าส่วนของแมคโครมีขนาดใหญ่หรือต้องเรียกใช้หลายครั้ง จะทำให้โปรแกรมที่ได้จากการแปลมีขนาดใหญ่

สรุป

แมคโครคือส่วนของโปรแกรมที่ได้กำหนดชื่อไว้ เมื่อใดก็ตามที่มีการเรียกใช้ชื่อนั้นในโปรแกรม แมคโครมีข้อดี คือ ลดเวลาในการกระโดดไปทำงานสำหรับส่วนของคำสั่งที่มีขนาดเล็ก แต่แมคโครก็มีข้อเสียคือถ้าส่วนของแมคโครมีขนาดใหญ่หรือต้องเรียกใช้หลายครั้ง จะทำให้โปรแกรมที่ได้จากการแปลมีขนาดใหญ่ไปด้วย

คำถามทบทวน

1. จงอธิบายการทำงานของคำสั่ง XLAT
2. จงแสดงวิธีการใช้งานการใช้ LABEL ในแมคโคร
3. แมคโครกับโปรแกรมย่อยคืออะไร มีข้อดีและข้อเสียอย่างไร

แผนบริหารการสอนประจำบทที่ 18

หัวข้อเนื้อหา

- การดาวน์โหลดและติดตั้งโปรแกรม MASM32
- การดาวน์โหลดและติดตั้งโปรแกรม EditPlus

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจวิธีการดาวน์โหลดและติดตั้งโปรแกรม MASM32
- มีความรู้และความเข้าใจวิธีการดาวน์โหลดและติดตั้งโปรแกรม EditPlus
- สามารถประยุกต์ใช้งานคำสั่งและฝึกทักษะการเขียนโปรแกรมภาษาแอสเซมบลีและตรวจสอบความถูกต้องของโปรแกรมที่เขียนขึ้นมาเบื้องต้นได้

กิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 18 การฝึกทักษะการเขียนโปรแกรมภาษาแอสเซมบลีเบื้องต้น (Skill Training Programs in a Primary Assembly)

บทนี้จะเกี่ยวข้องกับการเรียนรู้และการฝึกทักษะให้กับนักพัฒนาโปรแกรม ที่ต้องการเขียนโปรแกรมภาษาระดับต่ำ (Low Level Language) หรือบางครั้งอาจเรียกว่าการเขียนโปรแกรมภาษาเครื่อง โดยเฉพาะการเขียนโปรแกรมภาษาแอสเซมบลีเบื้องต้นซึ่งเหมาะสำหรับนักศึกษาที่ยังไม่เคยเขียนโปรแกรมหรือเขียนโปรแกรมแล้วแต่ยังไม่ค่อยเข้าใจเท่าที่ควรซึ่งเขียนโปรแกรมภาษาแอสแซมบลีนี้จัดเป็นภาษาระดับต่ำ ถ้าเทียบกับภาษาระดับสูง เช่น C++, JAVA , VISUAL BASIC เป็นต้น แต่ภาษา assembly ก็จะมีคุณสมบัติที่ดีกว่าภาษาอื่นตรงที่จะใช้เวลาในการทำงานของเครื่องน้อยกว่า

Microsoft Macro Assembler (อักษรย่อ MASM) โปรแกรมสำหรับแปลภาษาแอสแซมบลี (assembler) ใช้กับเครื่องคอมพิวเตอร์ที่ใช้ซีพียูตระกูล x86 แต่เดิมผลิตโดยบริษัทไมโครซอฟต์สำหรับใช้กับระบบปฏิบัติการดอส (MS-DOS) และเป็นโปรแกรมสำหรับแปลภาษาแอสแซมบลีที่นิยมใช้มากที่สุด

ขั้นตอนเบื้องต้นในการฝึกทักษะและการเขียนโปรแกรมภาษาแอสเซมบลีเบื้องต้นประกอบด้วย

18.1 การดาวน์โหลดและติดตั้งโปรแกรม MASM32

วิธีการดาวน์โหลดและติดตั้งโปรแกรม MASM32

ขั้นตอนที่ 1. เข้าเว็บไซต์ไปที่ http://dusithost.dusit.ac.th/~juthawut_cha/home.htm เลือก **การฝึกทักษะการเขียนโปรแกรมเบื้องต้น (Programming Skills)**

1. ไปที่ ดาวน์โหลด โปรแกรมสำหรับฝึกทักษะการเรียนรู้การเขียนโปรแกรมภาษาแอสแซมบลี คลิก Download MASM32 Version 9, ดาวน์โหลดไฟล์ m32v9r.zip ขนาด 3.37 MB

ฝึกทักษะการเขียนโปรแกรม (Programming Skills)



ดาวน์โหลด 1.android-sdk-windows



ดาวน์โหลด 1.MASA32v9r.zip 2.EditPlus



Load 1.Universal-USB-Installer-1.9.0.2



ดาวน์โหลด1.Adobe Dreamweaver CS3



ดาวน์โหลด 1.PHP & MySQL



ดาวน์โหลด 1.jdk-7u7-windows-i586



ดาวน์โหลด 1.Microsoft Visual C++ 6.0



ดาวน์โหลด 1.Microsoft Visual Basic 6.0



ดาวน์โหลด 1. Python

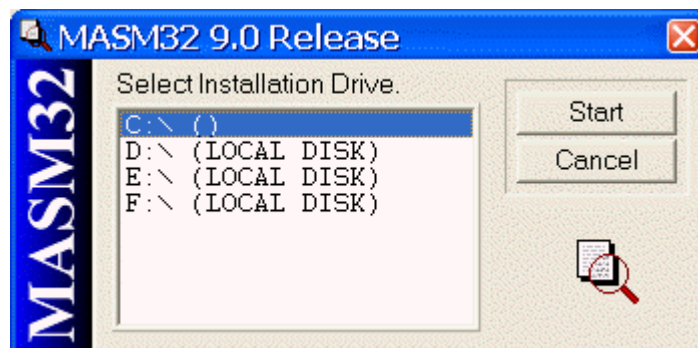
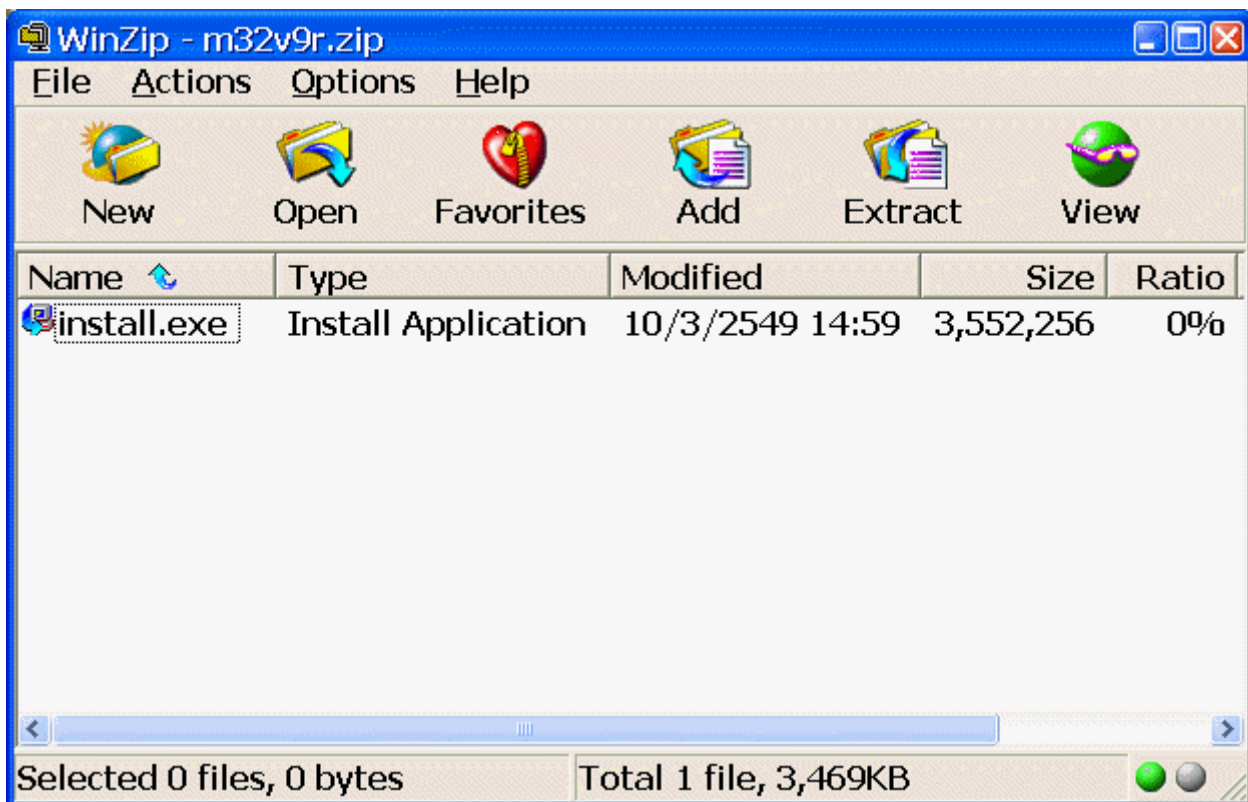


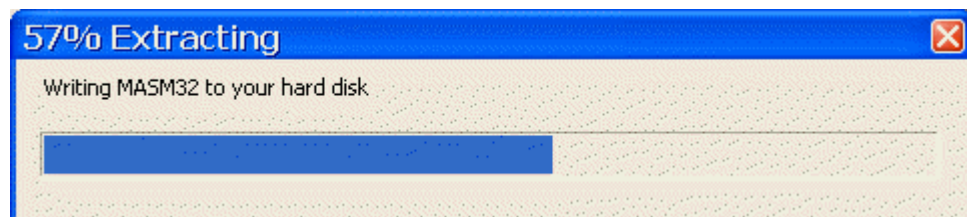
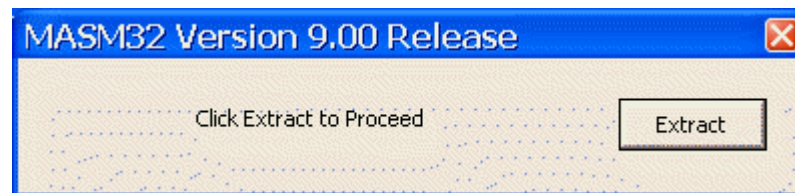
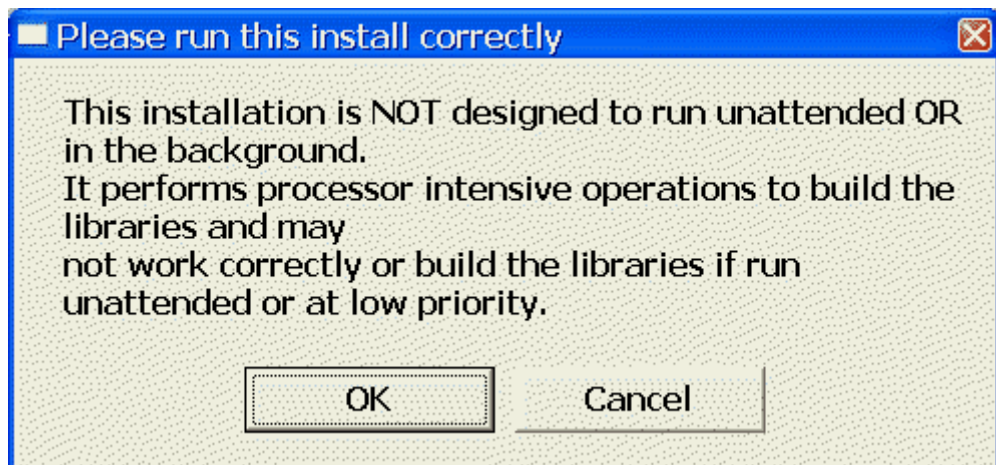
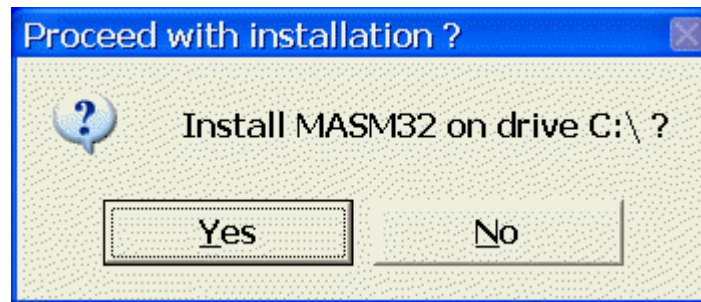
ดาวน์โหลด 1. Perl Programming

Last Update 17/10/2012 Contact Web Master: jchantharamalee@yahoo.com Tel 084-2055511, 089-4760432

2. Uncompress the file **m32v9r.zip** and run the installer file **install.exe** to start the installation process until finish.

2. ขยายไฟล์ **m32v9r.zip** และรันตัวติดตั้ง **install.exe** เพื่อเริ่มกระบวนการติดตั้งไปจนเสร็จ






```

E:\WINDOWS\system32\cmd.exe
-----
                BUILD THE LIBRARIES FOR MASM32

                IMPORTANT

If you have programs running that use a lot of processor resource
or in any other way interfere with processor intensive operations
like the following library building operations, close them down
now before you continue with this instalation as they may prevent
the library building operations from successfully completing.
-----

This operation performs the following actions,

    1. Builds the IMPORT Libraries for system API calls.
    2. Builds the MASM32 static library.
    3. Builds the FPULIB static library.
    4. Installs the VKdebug files into MASM32.
    5. Builds the MSVCRT library and include file

-----
Press any key to build the libraries
-----

```

```

E:\WINDOWS\system32\cmd.exe
Assembling: vtile.asm
Assembling: w2b_ex.asm
Assembling: wait_key.asm
Assembling: wordcnt.asm
Assembling: wordrep.asm
Assembling: writdisk.asm
Assembling: writline.asm
Assembling: wshell.asm
Assembling: xordata.asm
Microsoft (R) Library Manager Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

    1 file(s) copied.
    1 file(s) copied.
Volume in drive C has no label.
Volume Serial Number is 1212-15E0

Directory of C:\masm32\lib
05/17/2006  06:49 PM                132,498 masm32.lib
              1 File(s)                132,498 bytes
              0 Dir(s)                750,190,592 bytes free
Volume in drive C has no label.
Volume Serial Number is 1212-15E0

Directory of C:\masm32\include
12/05/2005  12:04 PM                14,358 masm32.inc
              1 File(s)                14,358 bytes
              0 Dir(s)                750,190,592 bytes free
Press any key to continue . . .

```

```

E:\WINDOWS\system32\cmd.exe
Assembling: FpuRound.asm
Assembling: FpuSin.asm
Assembling: FpuSinh.asm
Assembling: FpuSize.asm
Assembling: FpuSqrt.asm
Assembling: FpuState.asm
Assembling: FpuSub.asm
Assembling: FpuTan.asm
Assembling: FpuTanh.asm
Assembling: FpuTexpX.asm
Assembling: FpuTrunc.asm
Assembling: FpuXexpY.asm
Microsoft (R) Library Manager Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

    1 file(s) copied.
    1 file(s) copied.

FPU.LIB and FPU.INC have been copied
to the LIB and INCLUDE directories.

Volume in drive C has no label.
Volume Serial Number is 1212-15E0

Directory of C:\masm32\fpulib
05/17/2006  06:50 PM                19,982 fpu.lib
              1 File(s)                19,982 bytes
              0 Dir(s)                750,108,672 bytes free
Press any key to continue . . .

```

```

E:\WINDOWS\system32\cmd.exe
Assembling: FpuTanh.asm
Assembling: FpuTexpX.asm
Assembling: FpuTrunc.asm
Assembling: FpuXexpY.asm
Microsoft (R) Library Manager Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

    1 file(s) copied.
    1 file(s) copied.

FPU.LIB and FPU.INC have been copied
to the LIB and INCLUDE directories.

Volume in drive C has no label.
Volume Serial Number is 1212-15E0

Directory of C:\masm32\fpulib
05/17/2006  06:50 PM                19,982 fpu.lib
              1 File(s)                19,982 bytes
              0 Dir(s)                750,108,672 bytes free
Press any key to continue . . .
    1 file(s) copied.
    1 file(s) copied.
    1 file(s) copied.
Microsoft (R) Library Manager Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

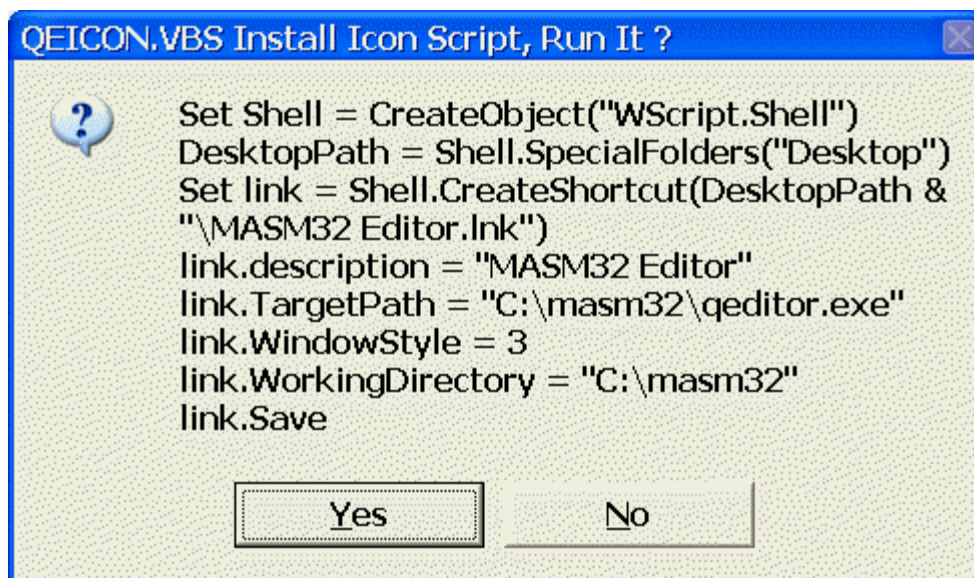
    Creating library msvcrt.lib and object msvcrt.exp
Press any key to continue . . .

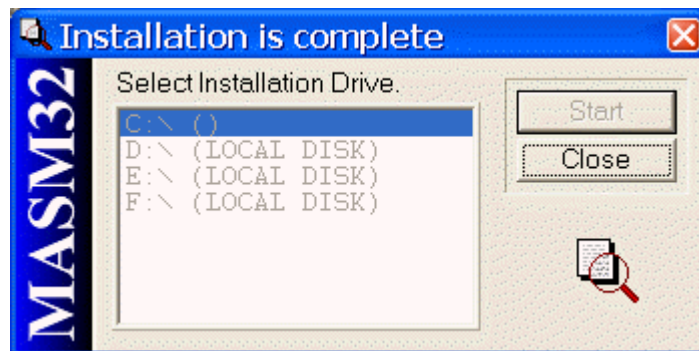
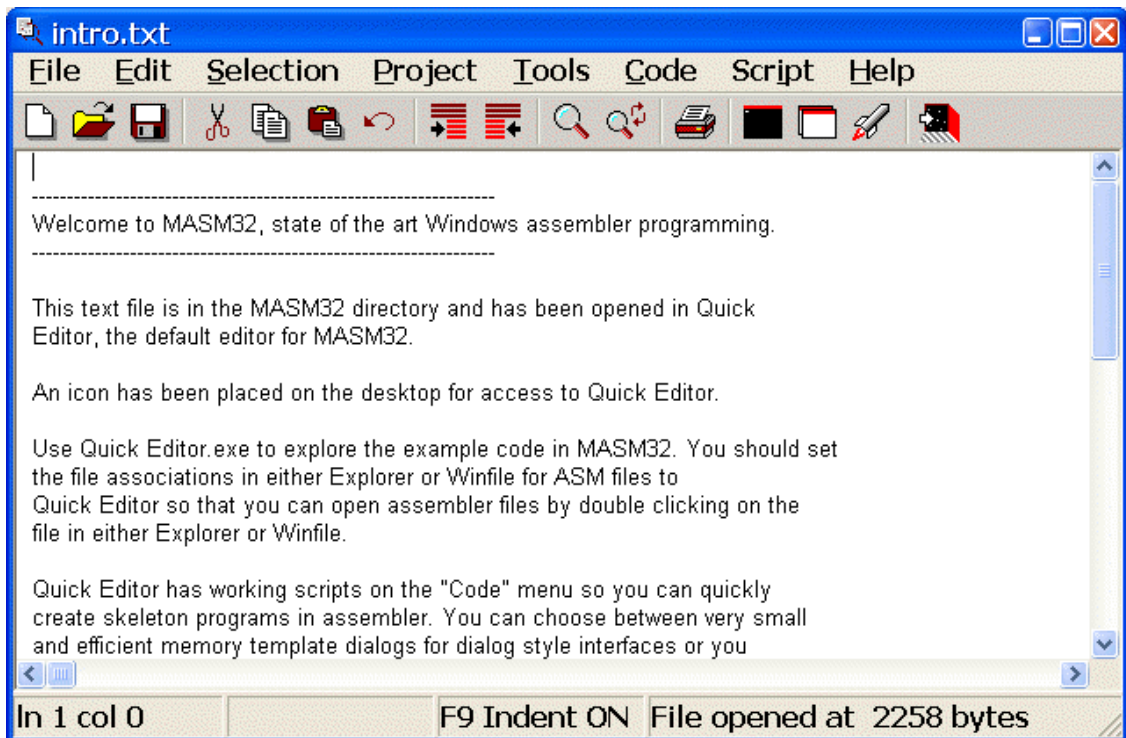
```

```

E:\WINDOWS\system32\cmd.exe
Directory of C:\masm32\fpulib
05/17/2006  08:50 PM                19,982 fpu.lib
                1 File(s)                19,982 bytes
                0 Dir(s)                742,129,664 bytes free
Press any key to continue . . .
1 file(s) copied.
1 file(s) copied.
1 file(s) copied.
Microsoft (R)
Copyright (C)
Creating 1
Press any key
Microsoft (R)
Copyright (C)
Assembling:
Volume in dr
Volume Serial
Directory of C:\masm32\testinst
03/10/2006  06:45 AM                4,707 testinst.asm
03/10/2006  06:48 AM                2,218 testinst.inc
05/17/2006  08:50 PM                2,975 testinst.obj
05/17/2006  08:50 PM                2,048 testinst.exe
                4 File(s)                11,948 bytes
                0 Dir(s)                741,621,760 bytes free

```





3. Do NOT use the linker link.exe (32 bit) in the masm32/bin directory. Use the linker version 5.60 to generate 16-bit DOS applications.

http://www.scs.carleton.ca/~sivarama/asm_book_web/free_MASM.html

3. อย่าใช้ตัวเชื่อม link.exe (32 bit) ที่อยู่ในไดเรกทอรี masm32/bin ให้ใช้ตัวเชื่อม link.exe รุ่น 5.60 เพื่อสร้างโปรแกรม 16-bit สำหรับระบบปฏิบัติการดอส (DOS)

http://www.scs.carleton.ca/~sivarama/asm_book_web/free_MASM.html

3.1. Download **lnk563.exe** size 274 KB from the following URL:

<http://download.microsoft.com/download/vc15/Update/1/WIN98/EN-US/Lnk563.exe>

http://www.scs.carleton.ca/~sivarama/asm_book/source/win_nasm_readme.pdf

3.1 ดาวนโหลดไฟล์ `lnk563.exe` ขนาด 274 KB จากเส้นทางดังนี้:

<http://download.microsoft.com/download/vc15/Update/1/WIN98/EN-US/Lnk563.exe>

http://www.scs.carleton.ca/~sivarama/asm_book/source/win_nasm_readme.pdf



4. Copy this file to `C:\MASM32\BIN` and run it. Answer Yes(y) when asked whether to overwrite existing files.

<http://www.intelligent-systems.info/classes/ee360/tutorial.htm>

4. คัดลอกไฟล์นี้ไปไว้ที่ `C:\MASM32\BIN` และรันไฟล์ ให้ตอบใช่ (y) เมื่อโปรแกรมถามว่าจะเขียนทับหรือไม่

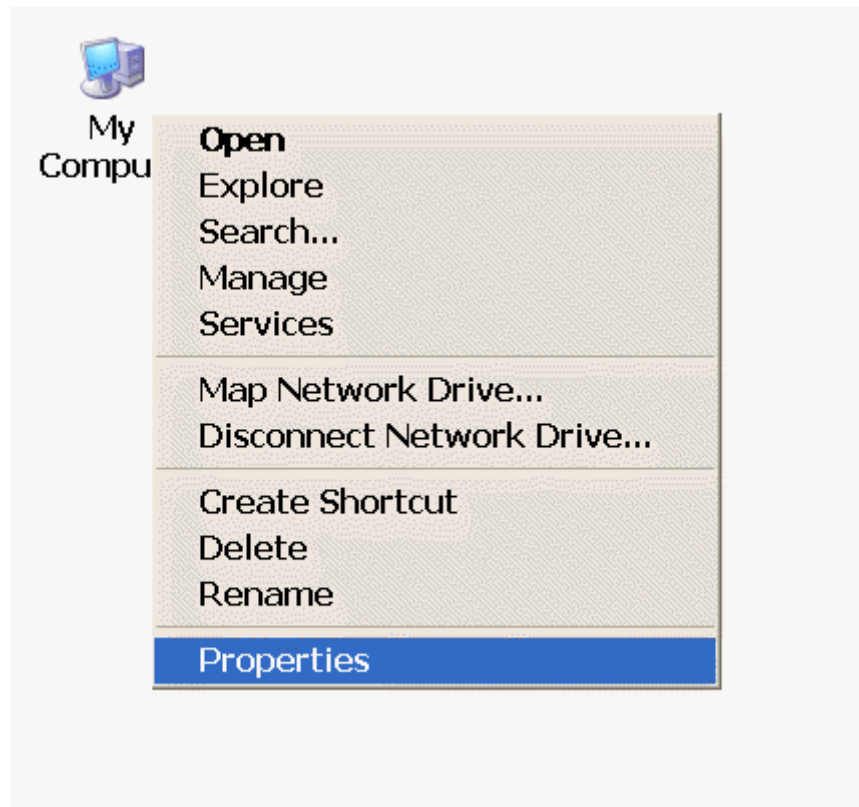
<http://www.intelligent-systems.info/classes/ee360/tutorial.htm>

A screenshot of a Windows command prompt window. The title bar reads "C:\masm32\bin\Lnk563.exe". The window content shows the following text:

```
PKSFX (R) FAST! Self Extract Utility Version 2.04g 02-01-93  
Copr. 1989-1993 PKWARE Inc. All Rights Reserved. Shareware version  
PKSFX Reg. U.S. Pat. and Tm. Off.  
  
Searching EXE: C:/MASM32/BIN/LNK563.EXE  
Inflating: CVPACK.EXE  
PKSFX: (W18) Warning! LINK.EXE already exists. Overwrite (y/n)?_
```

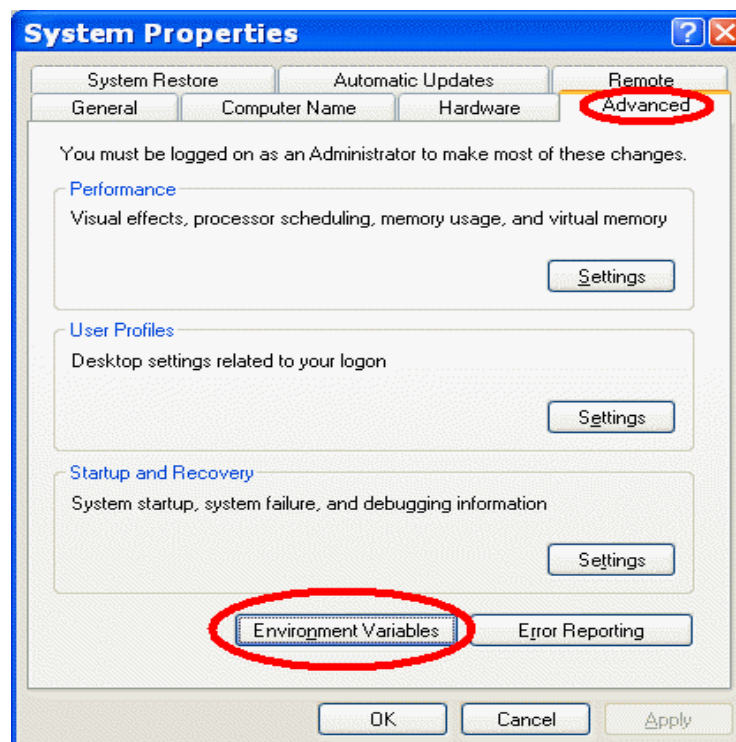
5. Right-click **my Computer** and click on **Poperties**.

5. คลิกขวา **my Computer** และคลิกที่ **Poperties**



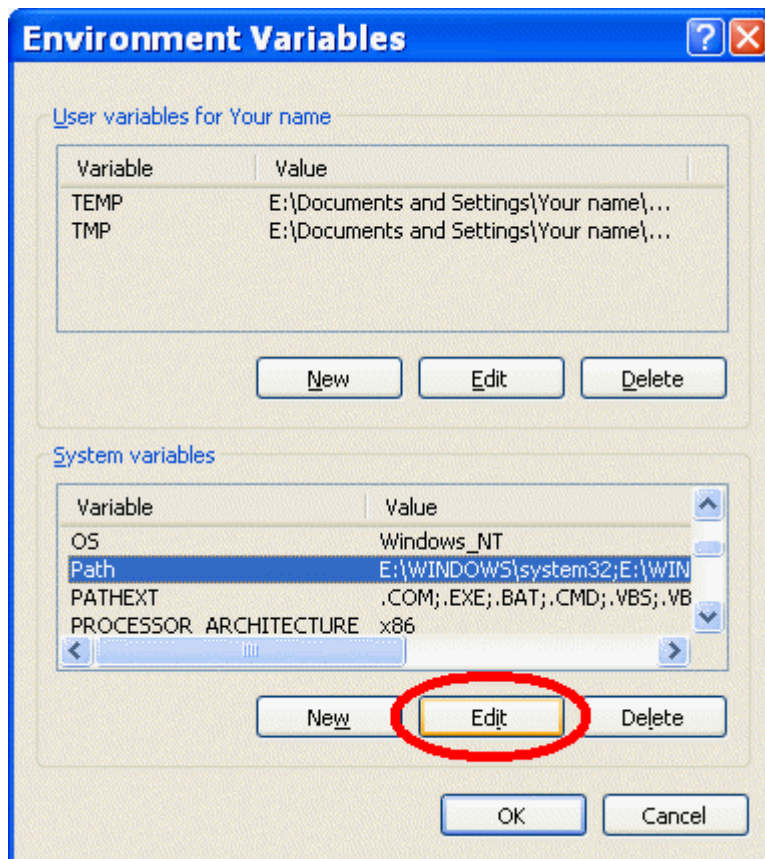
6. On the Advanced tab, click Environment Variables.

6. ที่แท็บ Advanced, คลิก Environment Variables



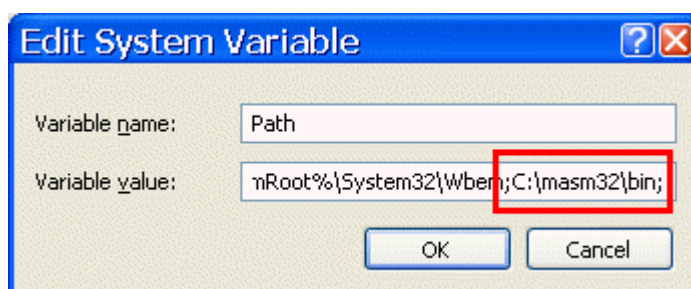
7. Under **System variables** box, select variable **Path** and click on **Edit**.

7. ในกรอบ **System variables**, ใน System variables เลือก **Path** คลิก **Edit**



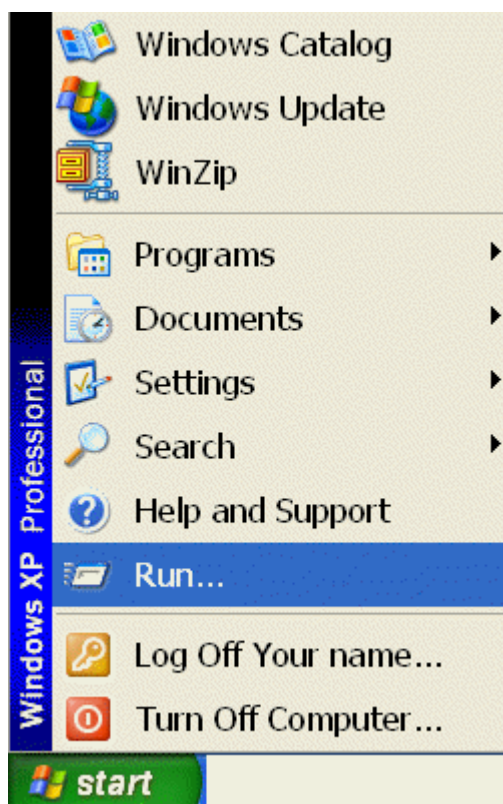
8. In **Variable value** edit box add the new paths separated by semicolons (;) **C:\masm32\bin**, click OK, click OK and click Ok again.

8. ในช่อง **Variable value** ใส่เส้นทางใหม่ขึ้นด้วยเครื่องหมายอัฒภาค (;) ดังนี้ **C:\masm32\bin**, คลิก OK, คลิก OK, และคลิก OK



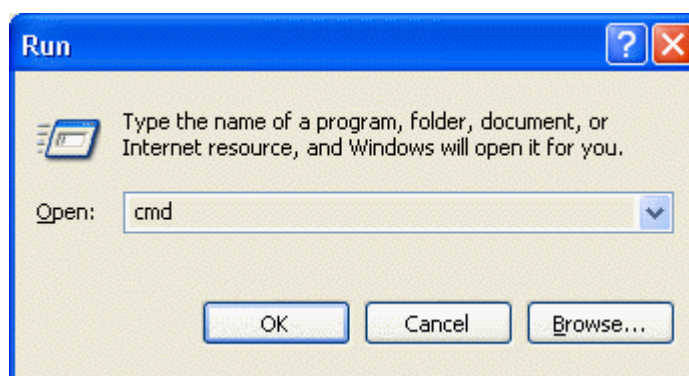
9. Click **Start**, and then click **Run**.

9. คลิก **Start**, และคลิก **Run**



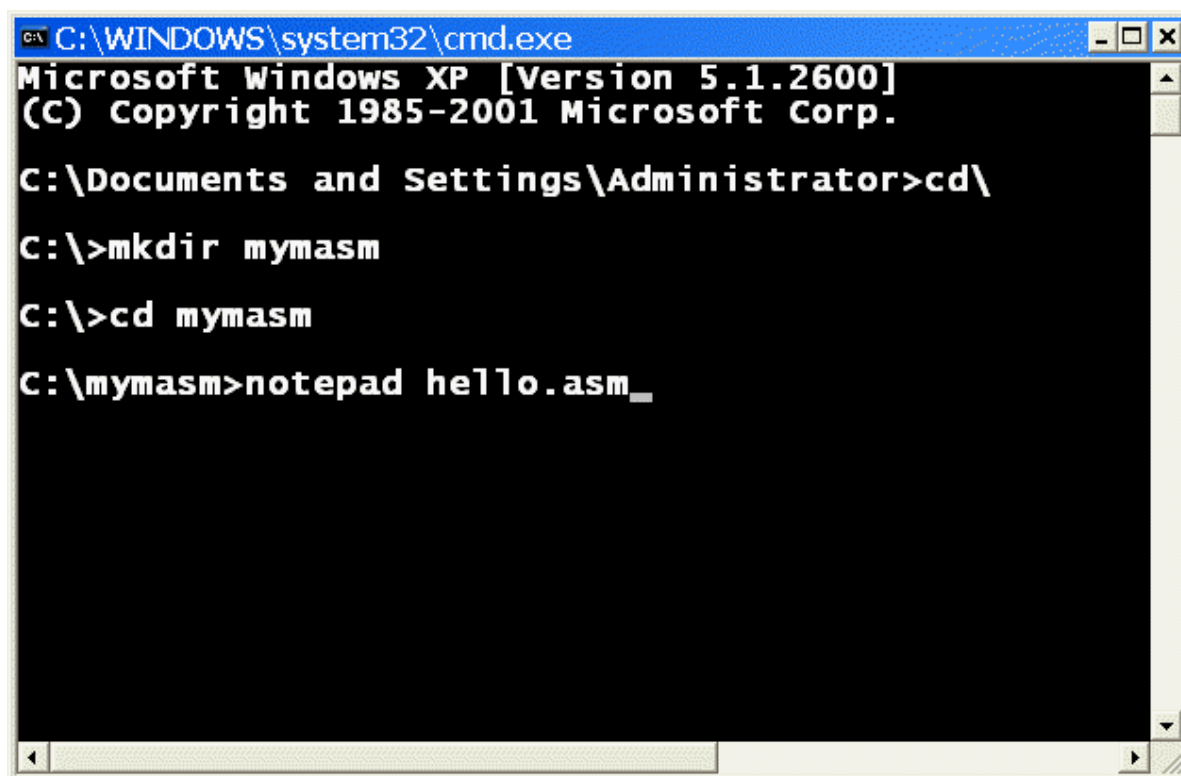
10. In the Open box, type **cmd**, and then click OK.

10. ในช่อง Open, พิมพ์ **cmd**, และคลิก OK



11. Create a folder to hold your Assembly **source code** file, using the **mkdir** command, and then change into the working directory, using the **cd** command. Finally, create your Assembly source code file by typing **notepad hello.asm**.

11. สร้างโฟลเดอร์เพื่อเก็บโค้ดโปรแกรมภาษาแอสเซมบลี, ใช้คำสั่ง **mkdir**, และใช้คำสั่ง **cd** เพื่อเข้าไปในโฟลเดอร์ที่สร้างไว้ ขั้นสุดท้ายสร้างโค้ดภาษาแอสเซมบลีโดยพิมพ์คำสั่งดังนี้ **notepad hello.asm**



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd\

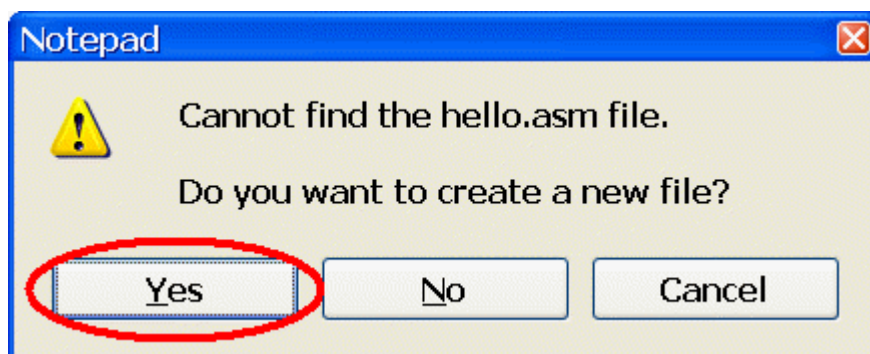
C:\>mkdir myasm

C:\>cd myasm

C:\myasm>notepad hello.asm_
```

12. Click **Yes** when Notepad asked if you want to create the new file because the file doesn't exist.

12. คลิก **Yes** ถ้าโปรแกรมถามว่าท่านต้องการที่จะสร้างไฟล์ใหม่หรือไม่เพราะไม่มีไฟล์อยู่



13. In `hello.asm` type the following Assembly source code, and then save it:

13. ในไฟล์ `hello.asm` พิมพ์โค้ดภาษาแอสเซมบลี, และบันทึกไฟล์

```
;hello.asm
```

```
;To assemble and link
```

```
;For TASM, type:
```

```
;tasm hello.asm
```

```
;tlink hello.obj
```

```
;For MASM, type:
```

```
;ml hello.asm
```

```
;This program prints the message "Hello World!" on the screen.
```

```
.model small
```

```
.stack
```

```
.data
```

```
Message db "Hello World!",10,13,"$"
```

```
.code
```

```
start:
```

```
;Get segment of Message.
```

```
mov ax,SEG Message
```

```
mov ds,ax
```

```
;Write 'Hello World!' to the screen.
```

```
mov ah,9
```

```
mov dx,OFFSET Message
```

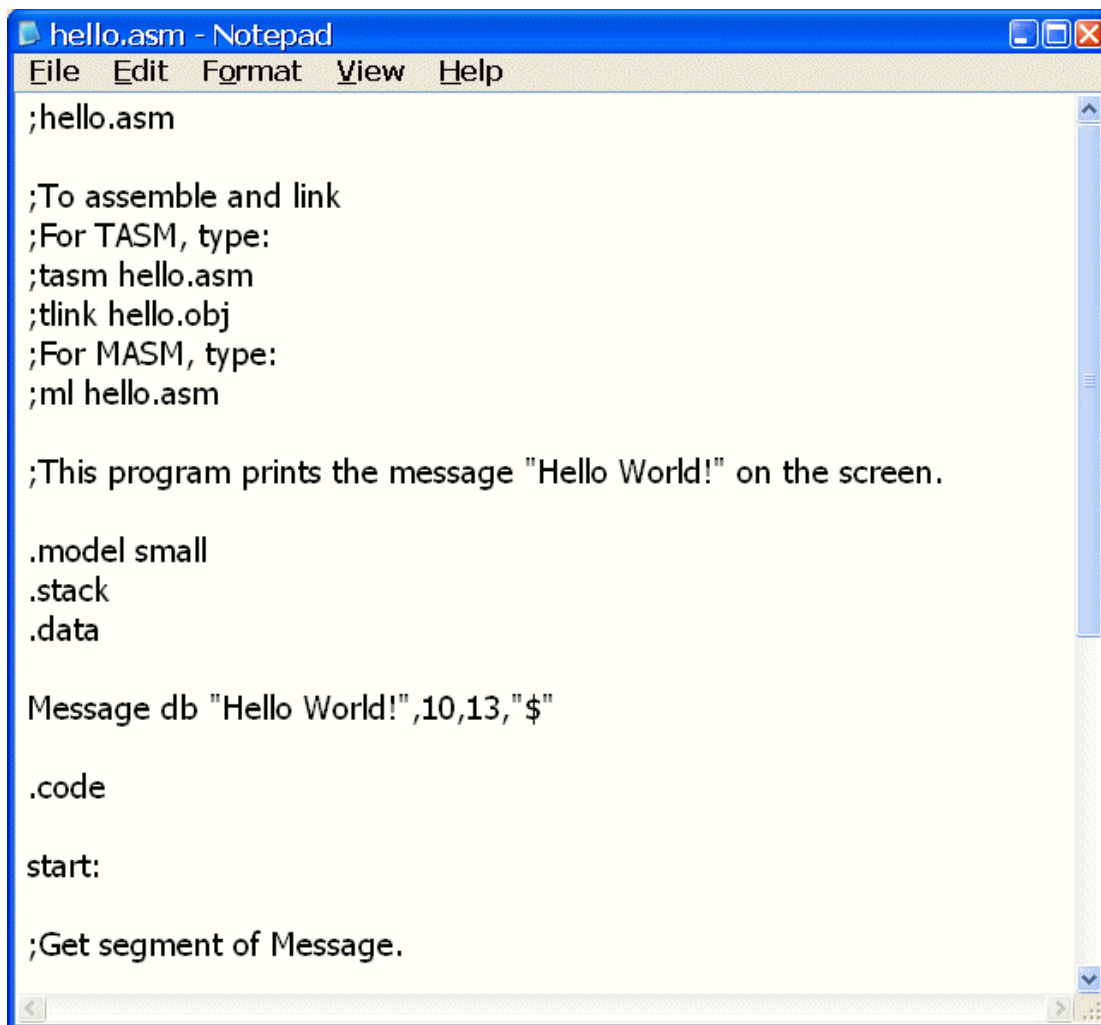
```
int 21h
```

```
;Terminate program.
```

```
mov ax,4c00h
```

```
int 21h
```

```
end start
```



```
hello.asm - Notepad
File Edit Format View Help
;hello.asm

;To assemble and link
;For TASM, type:
;tasm hello.asm
;tlink hello.obj
;For MASM, type:
;ml hello.asm

;This program prints the message "Hello World!" on the screen.

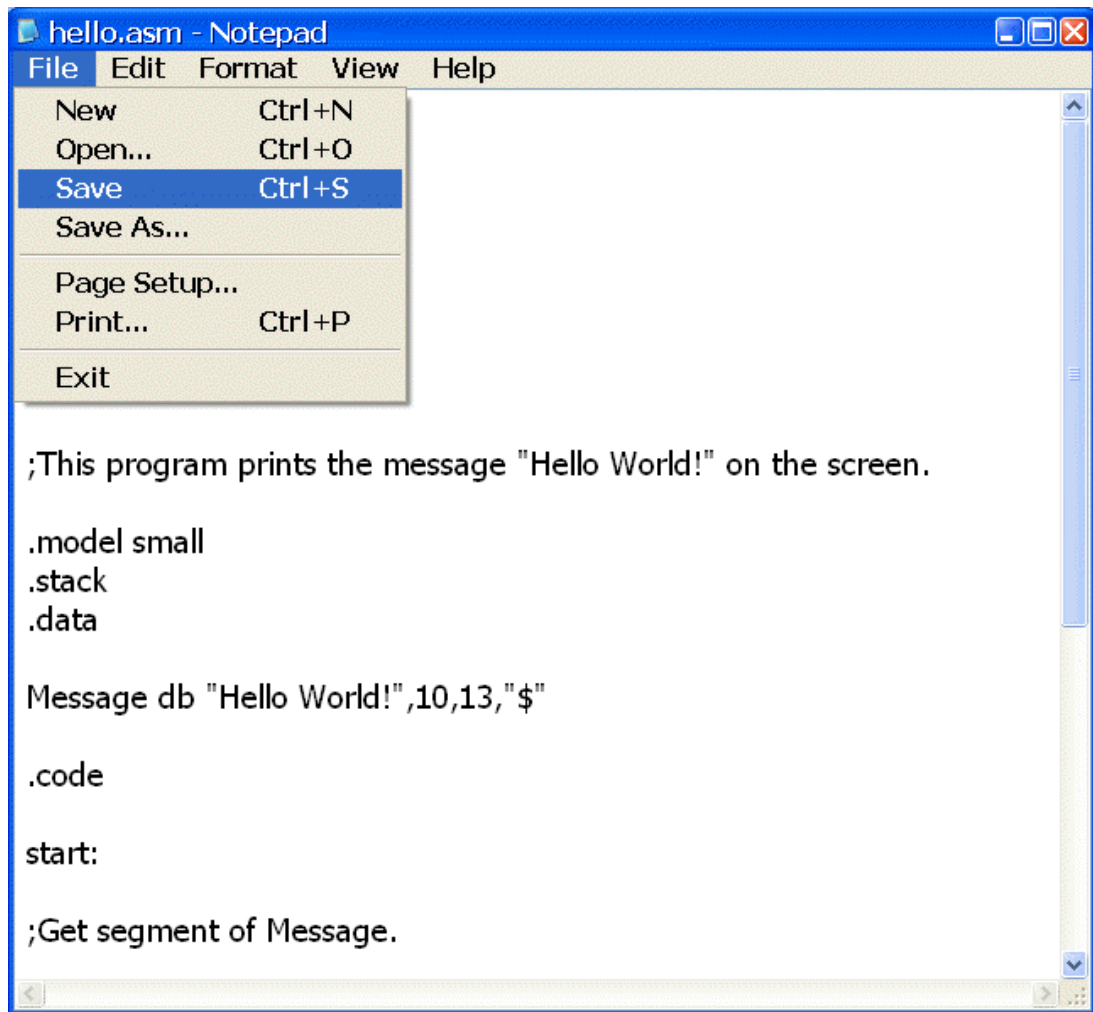
.model small
.stack
.data

Message db "Hello World!",10,13,"$"

.code

start:

;Get segment of Message.
```



```

;This program prints the message "Hello World!" on the screen.

.model small
.stack
.data

Message db "Hello World!",10,13,"$"

.code

start:

;Get segment of Message.

```

14. To assemble and link, use the command: `ml hello.asm`

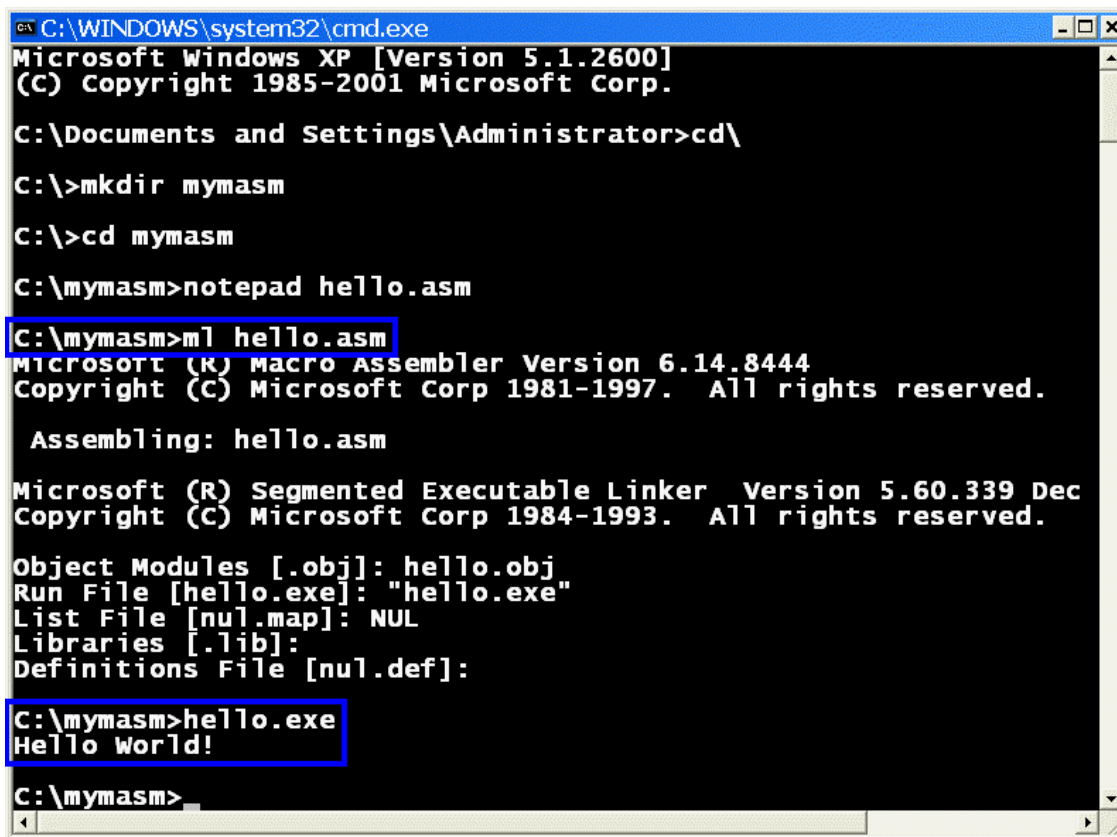
14. การแปลโค้ดและเชื่อม, ใช้คำสั่ง: `ml hello.asm`

15. To run the program, use the command: `hello`

15. การรันโปรแกรม, ใช้คำสั่ง: `hello`

16. The program prints the message **Hello World!** on the next line.

16. โปรแกรมจะแสดงผลข้อความ **Hello World!** ที่บรรทัดถัดไป



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd\

C:\>mkdir myasm

C:\>cd myasm

C:\myasm>notepad hello.asm

C:\myasm>ml hello.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: hello.asm

Microsoft (R) Segmented Executable Linker Version 5.60.339 Dec
Copyright (C) Microsoft Corp 1984-1993. All rights reserved.

Object Modules [.obj]: hello.obj
Run File [hello.exe]: "hello.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:

C:\myasm>hello.exe
Hello World!

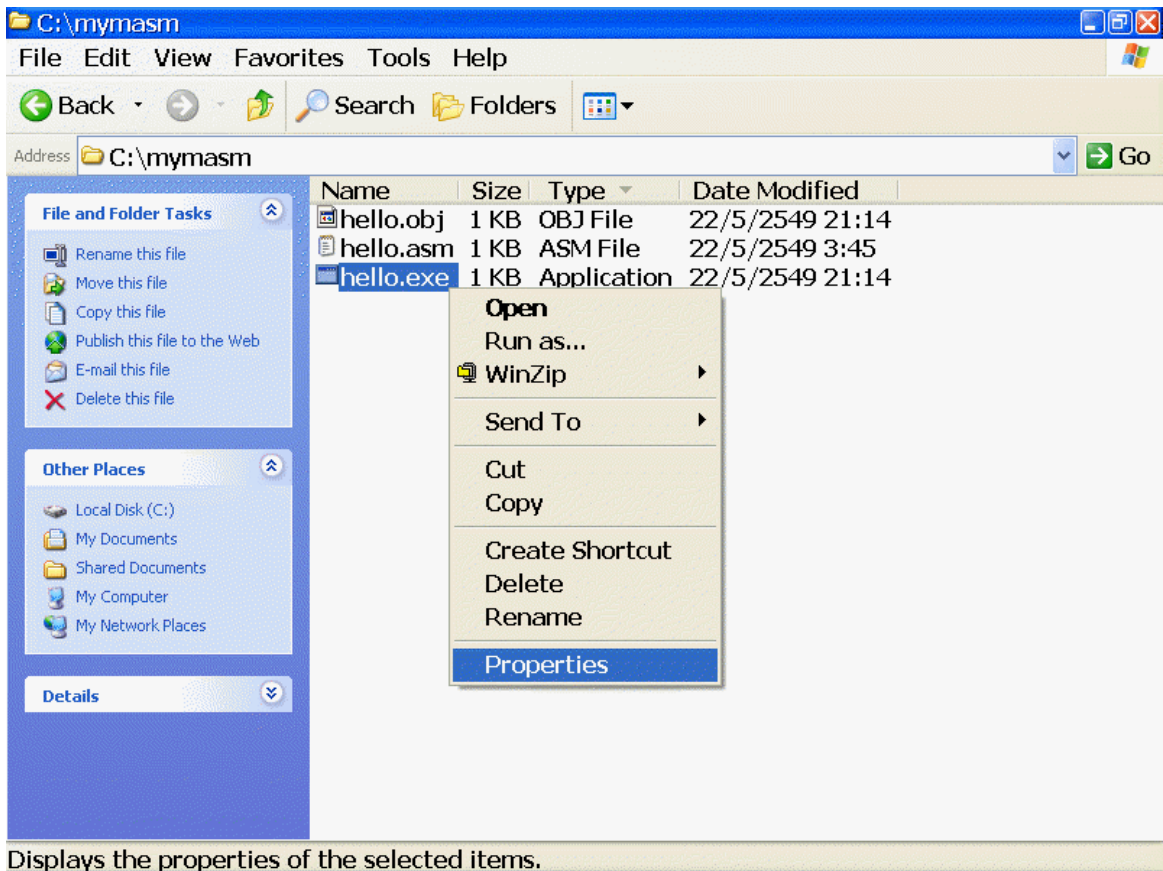
C:\myasm>
```

Running from icon

รันโปรแกรมโดยไอคอน

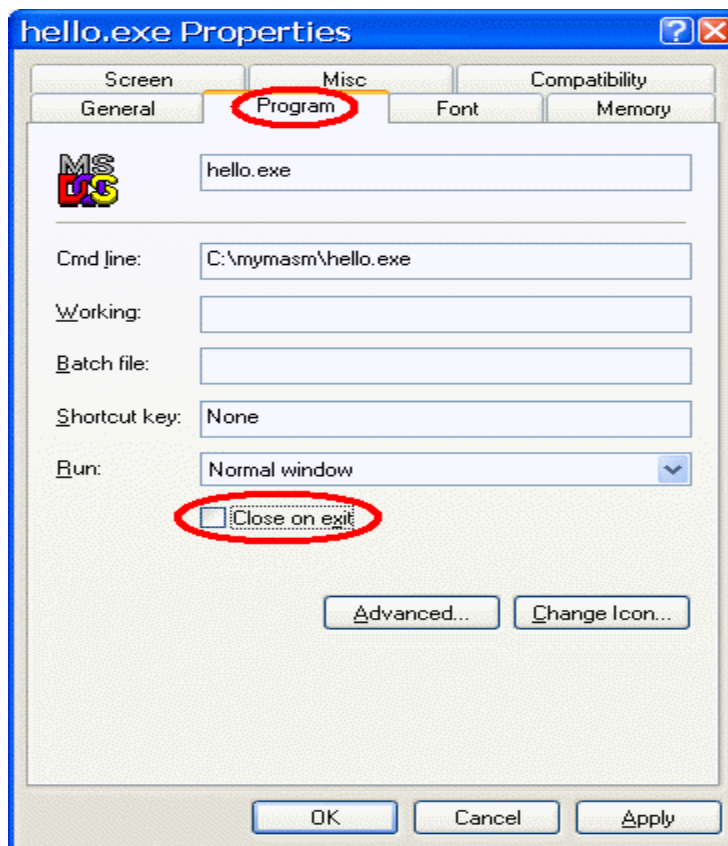
17. In Windows Explorer in the folder you saved the Pascal source code right click on the executable file `hello.exe`, and choose **Properties**.

17. ใน Windows Explorer ในโฟลเดอร์ที่ท่านได้เก็บโค้ดภาษา Pascal คลิกขวาที่ไอคอนไฟล์ `hello.exe`, และเลือก **Properties**



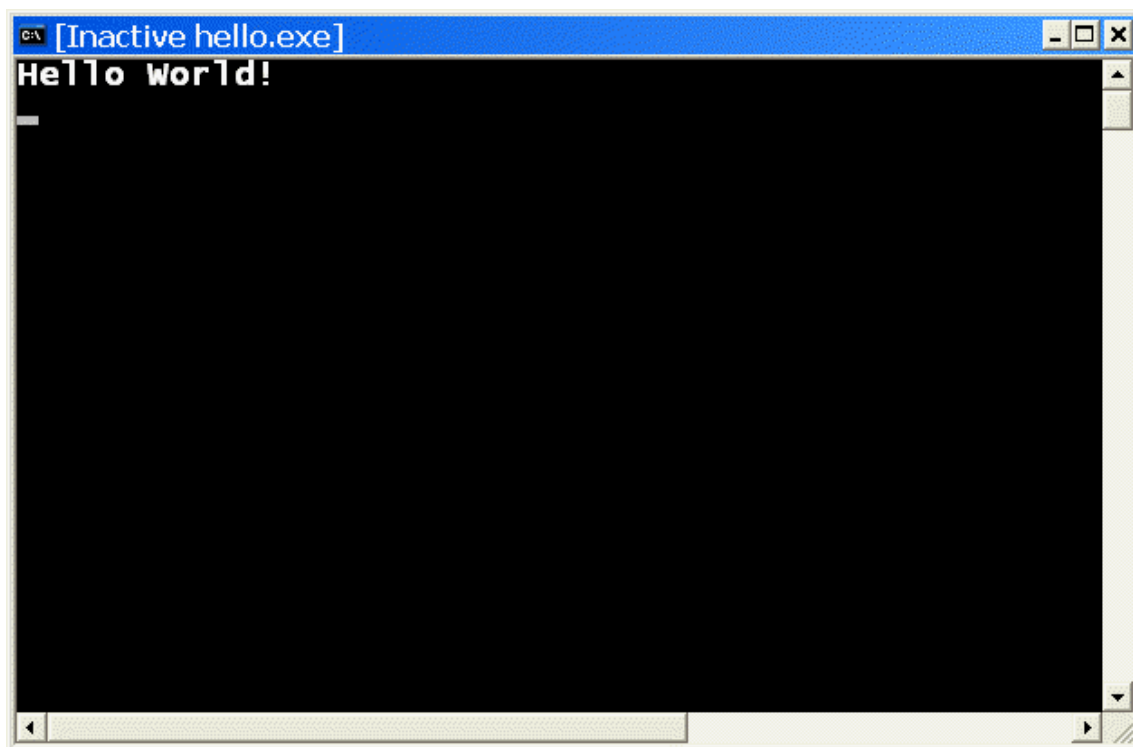
18. Click the **Program** tab, and uncheck the box marked **Close on exit** and then click "OK" to close the box.

18. คลิกแท็บ **Program**, คลิก **Close on exit** ให้เครื่องหมายถูกหายไปและคลิก **OK**



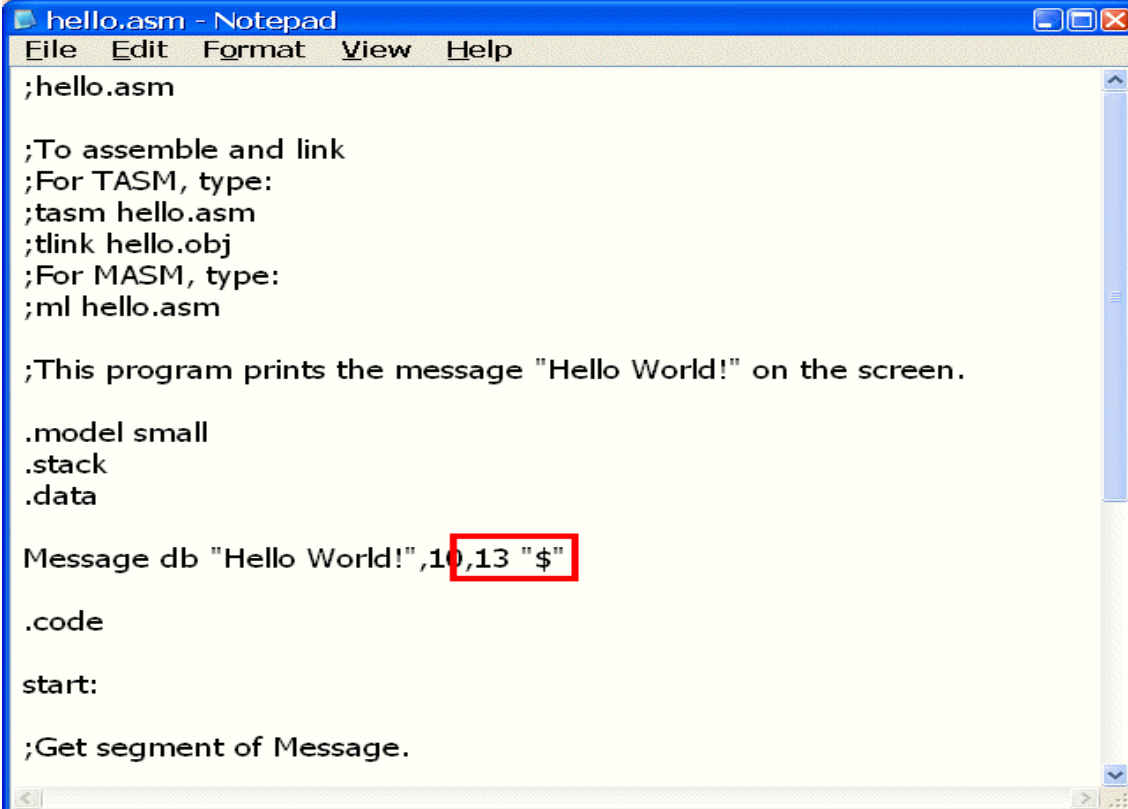
19. This will stop the DOS window from closing automatically when the program runs by DOS program icon.

19. โปรแกรมจะรันค้างไว้โดยไม่ปิดเองถ้ารันโดยไอคอนของดอส



20. If you forget the comma, the compiler will give you an error message when you attempt to compile the program.

20. ถ้าท่านลืมเครื่องหมายจุลภาค (,), ตัวแปลภาษาจะแสดงข้อความแจ้งความผิดพลาดเมื่อท่านแปลโค้ด



```
hello.asm - Notepad
File Edit Format View Help
;hello.asm

;To assemble and link
;For TASM, type:
;tasm hello.asm
;tlink hello.obj
;For MASM, type:
;ml hello.asm

;This program prints the message "Hello World!" on the screen.

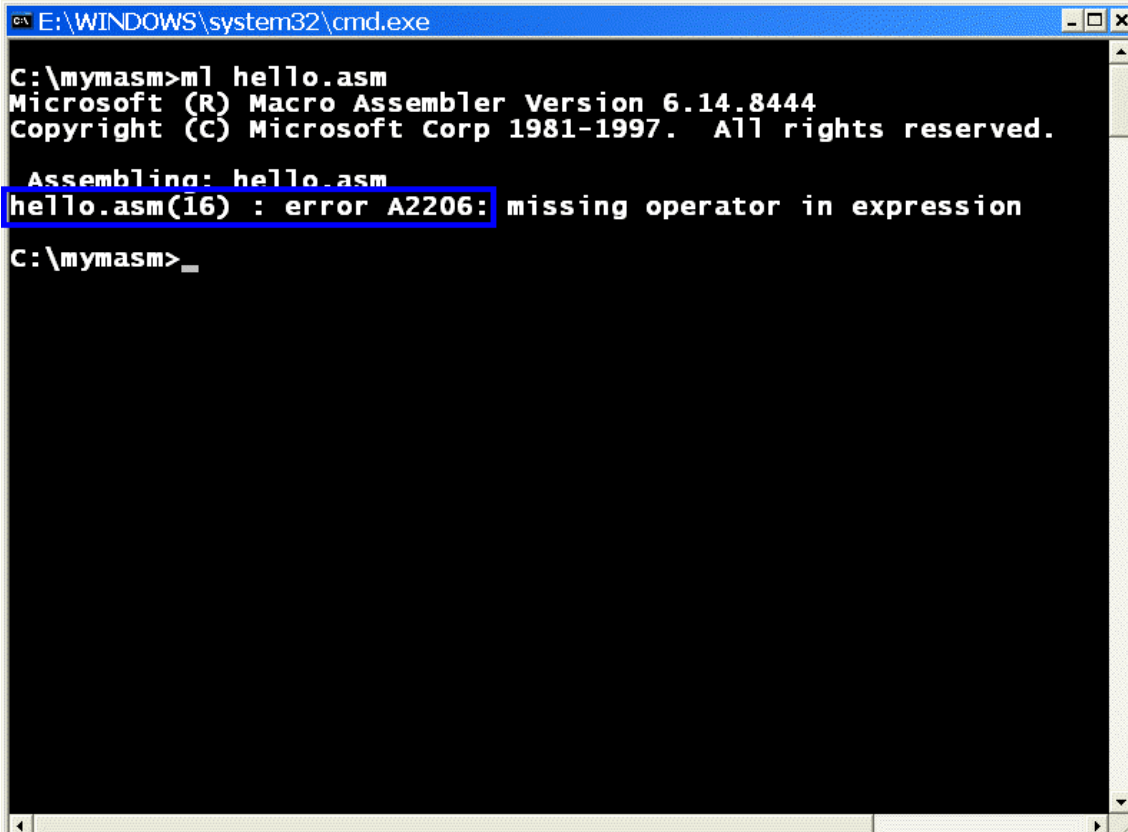
.model small
.stack
.data

Message db "Hello World!",10,13 "$"

.code

start:

;Get segment of Message.
```



```
E:\WINDOWS\system32\cmd.exe
C:\mymasm>ml hello.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: hello.asm
hello.asm(16) : error A2206: missing operator in expression
C:\mymasm>_
```

18.2 การดาวน์โหลดและติดตั้งโปรแกรม EditPlus

EditPlus คือโปรแกรมพิมพ์ข้อความรวมทั้งโค้ดภาษาโปรแกรมและโค้ดภาษาเว็บใช้ในวินโดวส์ มีคุณสมบัติมากมายเหมาะสำหรับเขียนเว็บภาษา HTML และเขียนโปรแกรมภาษาต่างๆ

วิธีการดาวน์โหลดและติดตั้ง โปรแกรม EditPlus

ขั้นตอนที่ 1. เข้าเว็บไซต์ไปที่ http://dusithost.dusit.ac.th/~juthawut_cha/home.htm เลือก **การฝึกทักษะการเขียนโปรแกรมเบื้องต้น (Programming Skills)**

1. ไปที่ ดาวน์โหลด โปรแกรมสำหรับฝึกทักษะการเรียนรู้การเขียนโปรแกรมภาษาแอสเซมบลี คลิก **Download** ดาวน์โหลดไฟล์ **epp220_en.exe** ขนาด 939 KB

ฝึกทักษะการเขียนโปรแกรม (Programming Skills)

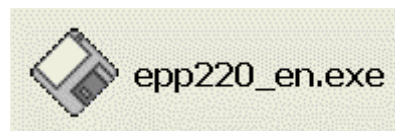
ดาวน์โหลด 1.android-sdk-windows ดาวน์โหลด 1.MASA32r.zip 2.EditPlus Load 1.Universal-USB-Installer-1.9.0.9 ดาวน์โหลด1.Adobe Dreamweaver CS3 ดาวน์โหลด 1.PHP & MySQL

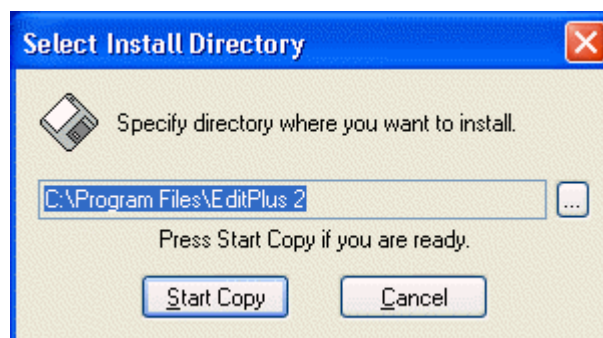
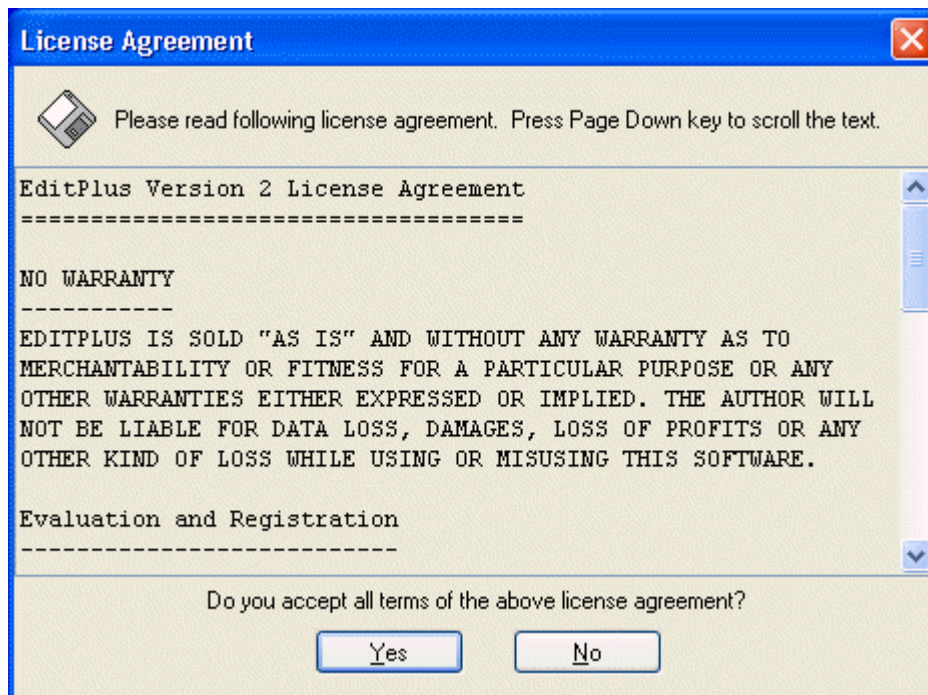
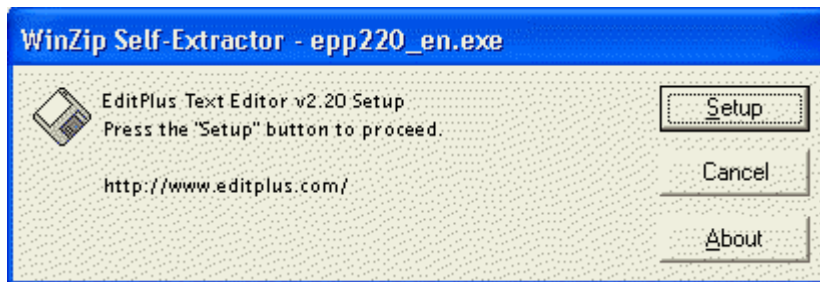
ดาวน์โหลด 1.jdk-7u7-windows-i586 ดาวน์โหลด 1.Microsoft Visual C++ 6.0 ดาวน์โหลด 1.Microsoft Visual Basic 6.0 ดาวน์โหลด 1.Python ดาวน์โหลด 1.Pert Programming

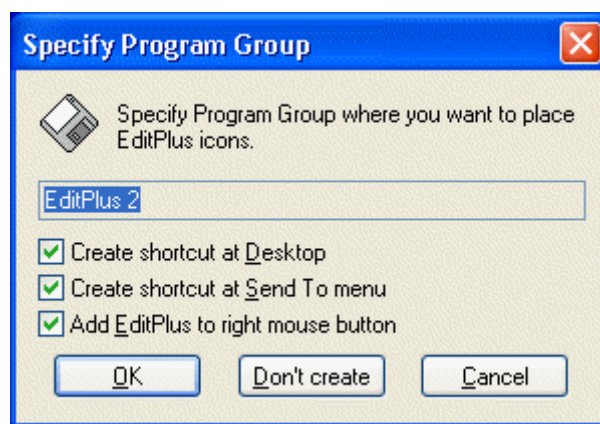
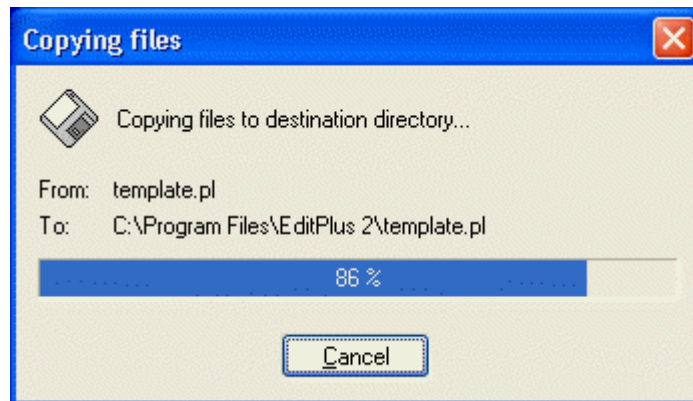
Last Update 17/10/2012 Contract Web Master: jchantharamalee@yahoo.com Tel 084-2055511, 089-4760432

2. Double-click on the saved file icon **epp220_en.exe** to start the installation process.

2. ดับเบิลคลิกไอคอนของไฟล์ **epp220_en.exe** เพื่อเริ่มกระบวนการติดตั้งโปรแกรม

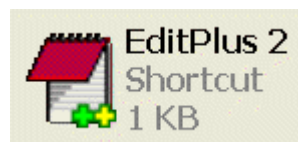


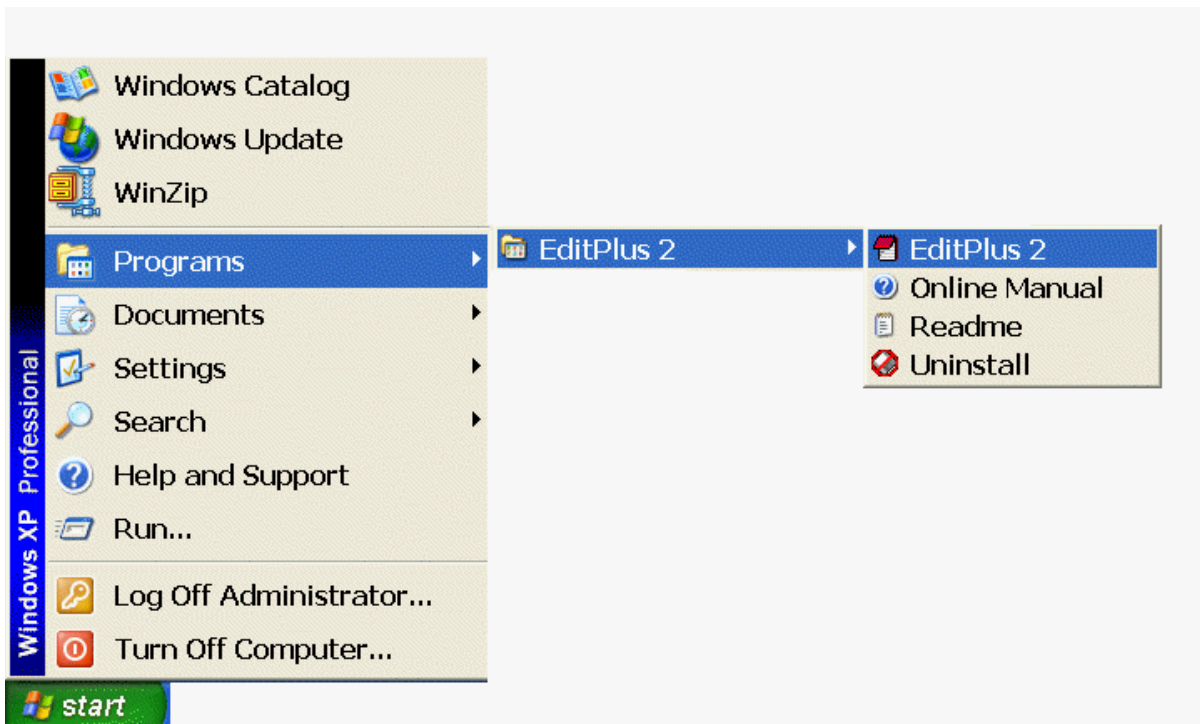




3. Click the program icon on your desktop or the shortcut on the **Start menu** to launch the program.

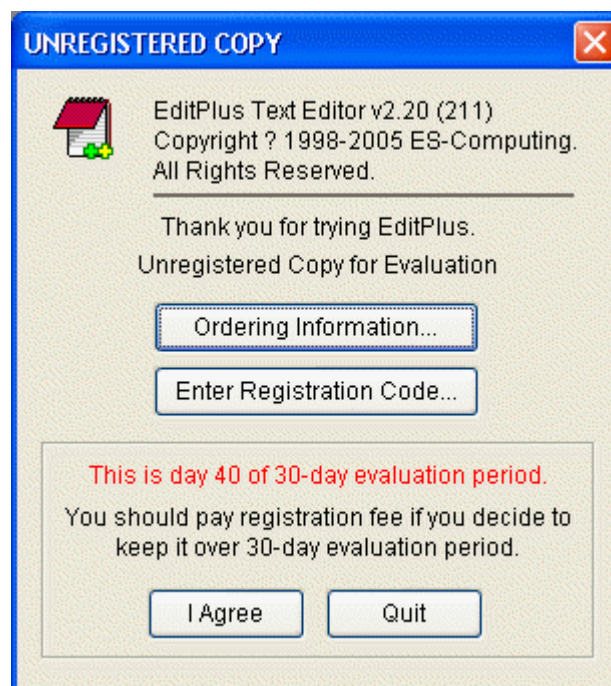
3. คลิกไอคอนโปรแกรมบนหน้าจอหรือคลิกชื่อโปรแกรมที่เมนู **Start** เพื่อรันโปรแกรม





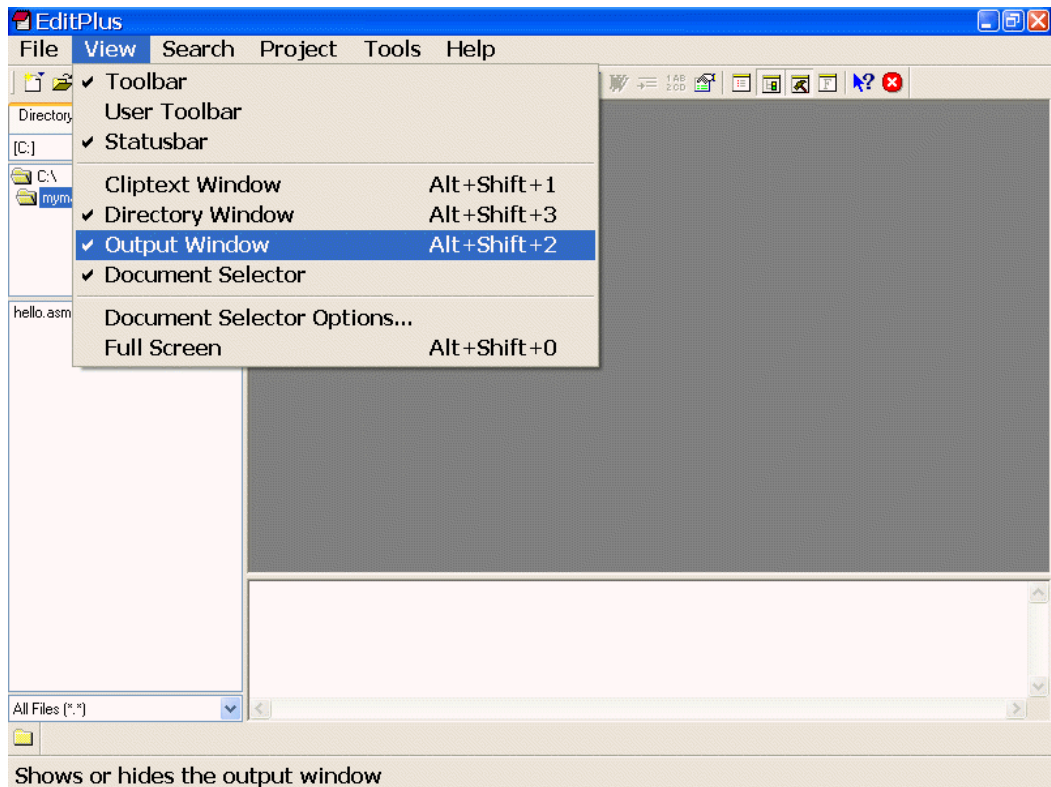
4. It will bring up a nag screen each time it is run until you register the program.

4. โปรแกรมจะแสดงข้อความเตือนว่ายังไม่ริจิสเตอร์ในกรอบสีเหลี่ยม



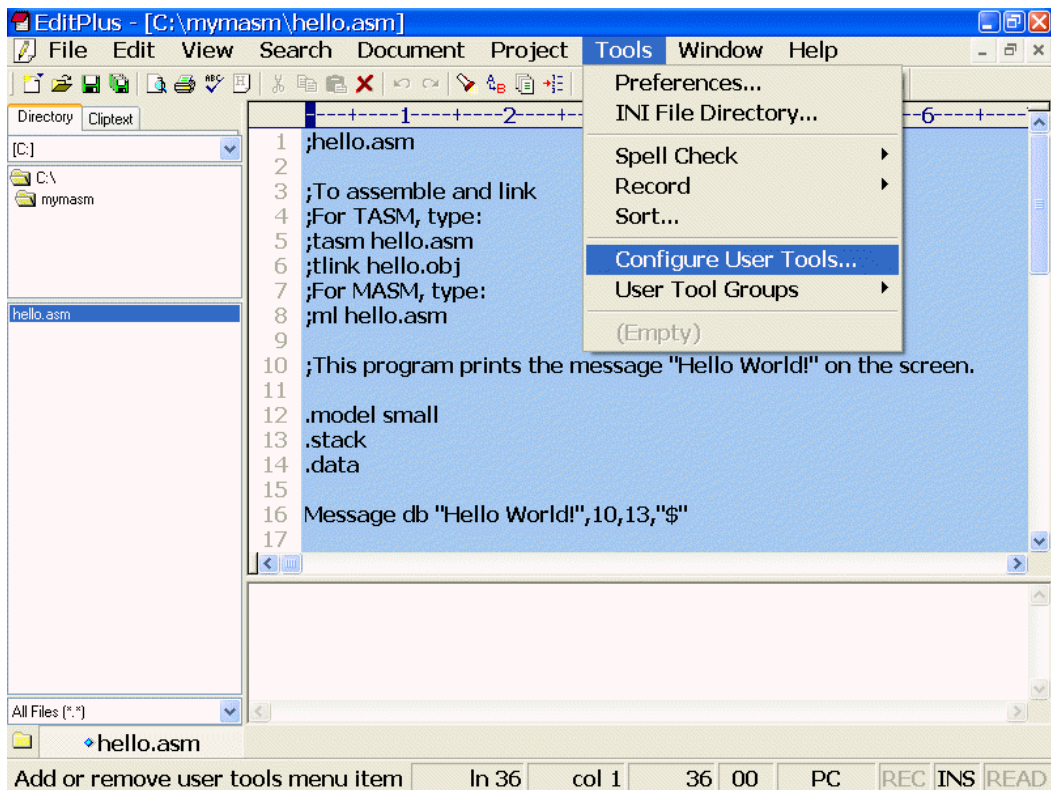
5. From the **View** menu, select **Output Window** to show it.

5. ที่เมนู **View**, เลือก **Output Window** เพื่อแสดงให้เห็น



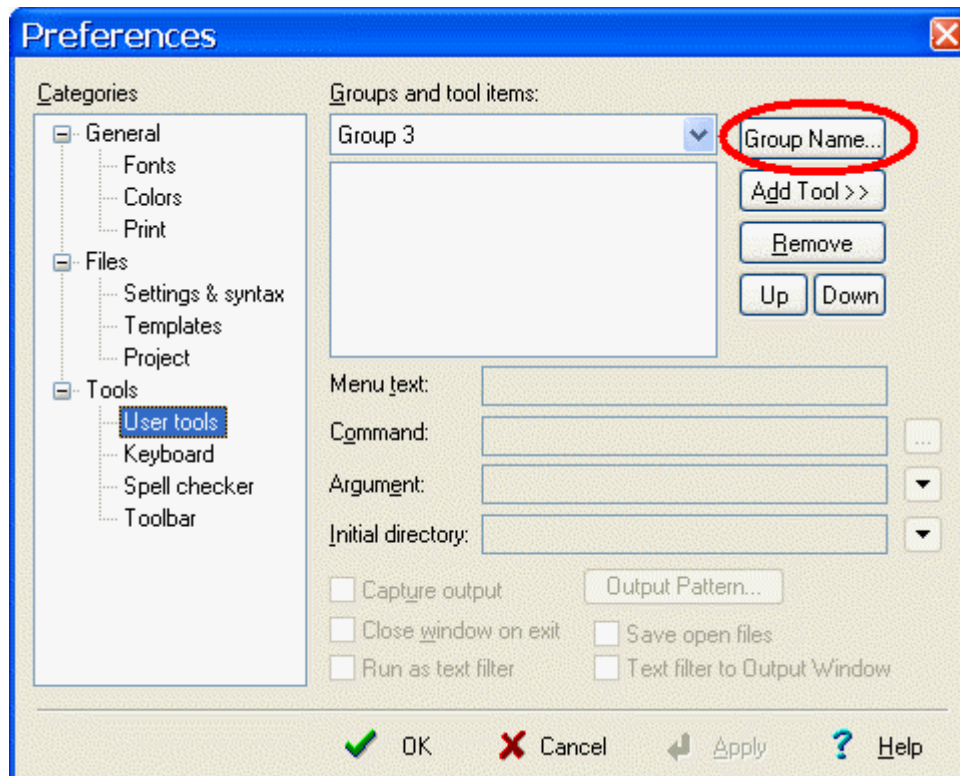
6. From the **Tools** menu, select **Configure User Tools**.

6. ที่เมนู **Tools**, เลือก **Configure User Tools**



7. In **Preferences** dialog box On **User tools** page click on the **Group Name** button to change the default group name Group 3.

7. ในกรอบ Preferences ที่หน้า User tools คลิกปุ่ม Group Name เพื่อเปลี่ยนชื่อ



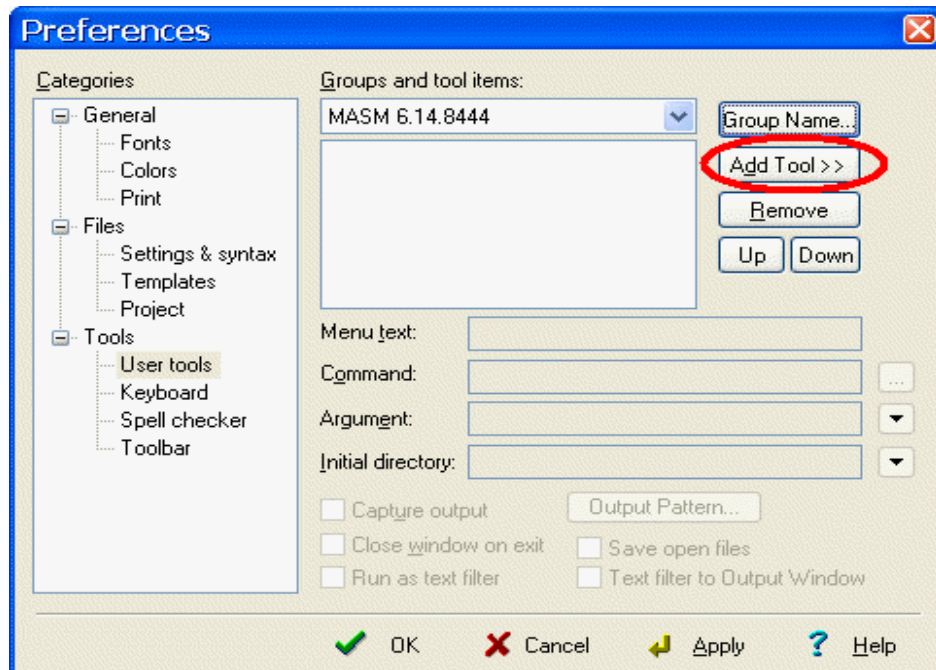
8. In the New Name text box, type **MASM 6.14.8444**

8. ในช่อง New Name, พิมพ์ **MASM 6.14.8444**



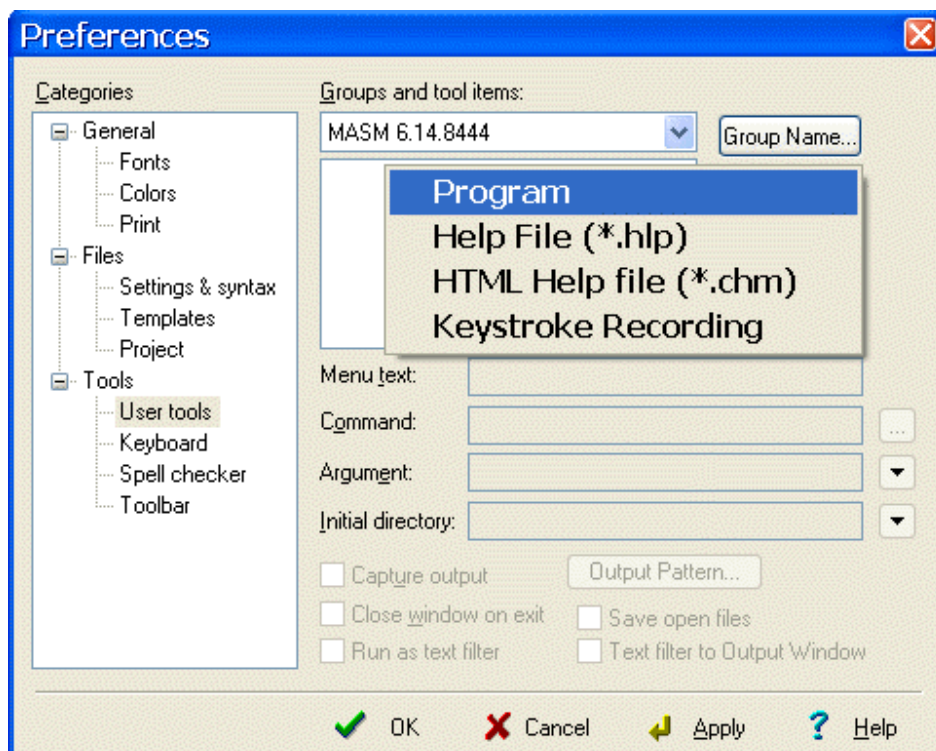
9. Select the **Add Tool** button.

9. คลิกปุ่ม **Add Tool**



10. Select **Program** from the popup menu.

10. เลือกเมนู **Program**



11. Set the options like this:

Menu text: MASM

Command: C:\masm32\bin\ml.exe

Argument: \$(FilePath)

Initial directory: \$(FileDir)

Capture output: ON

and then click Apply.

11. เซ็ตค่าต่างๆดังนี้

Menu text: MASM

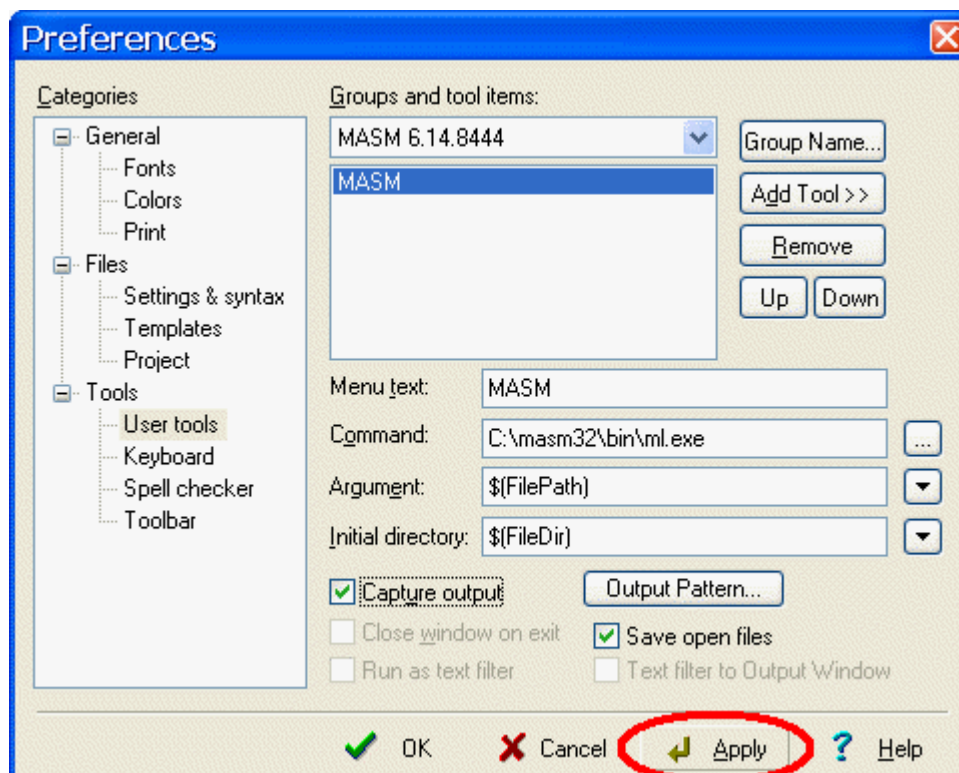
Command: C:\masm32\bin\ml.exe

Argument: \$(FilePath)

Initial directory: \$(FileDir)

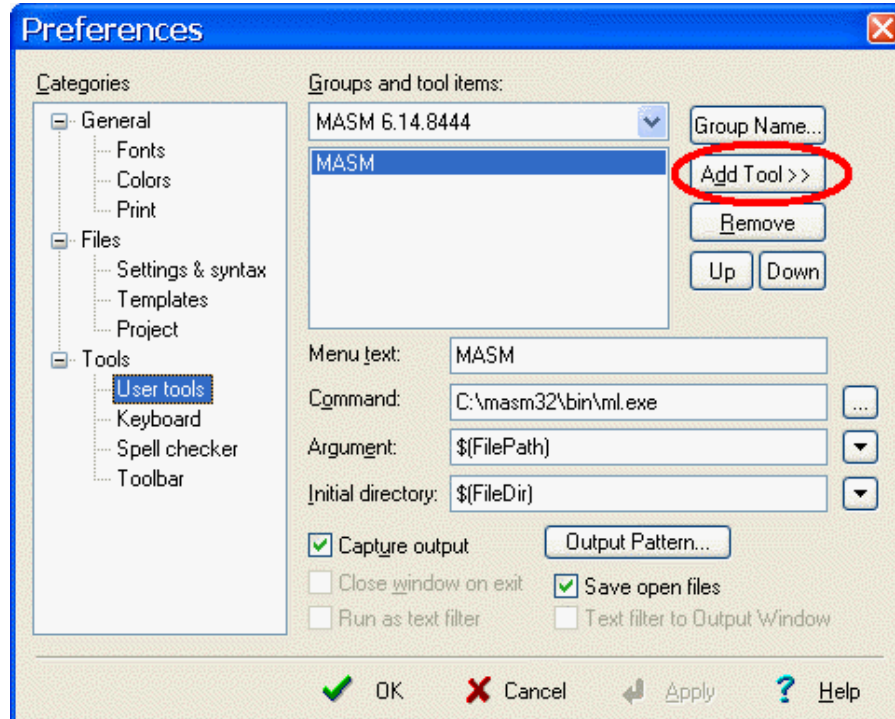
Capture output: ON

หลังจากนั้นคลิกปุ่ม Apply



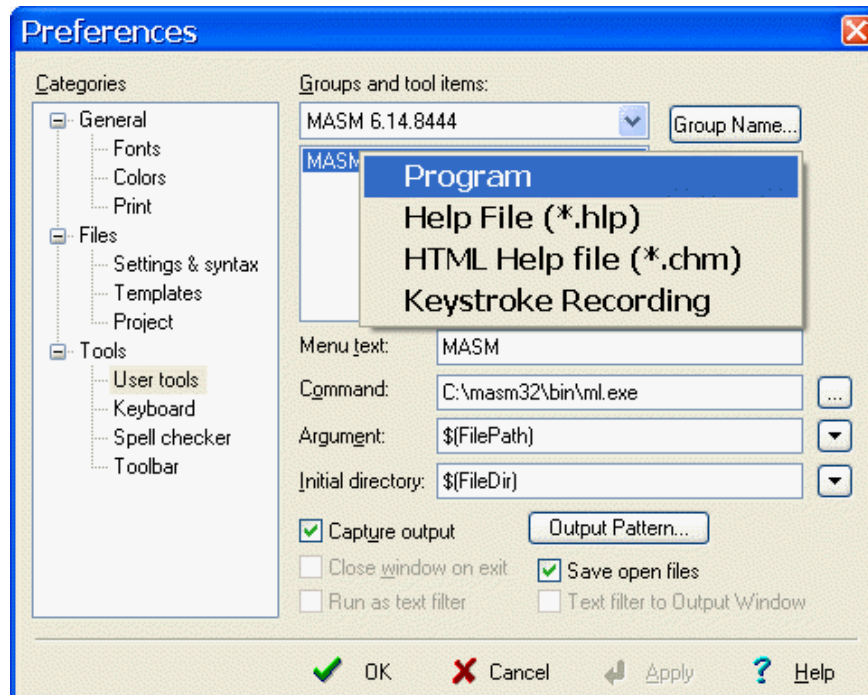
12. Select the Add Tool button.

12. คลิกปุ่ม Add Tool



13. Select Program from the popup menu.

13. เลือกเมนู Program



14. Set the options like this:

Menu text: **Run**

Command: **\$(FileNameNoExt)**

Argument:

Initial directory: **\$(FileDir)**

Capture output: **ON**

and then click OK.

14. เซ็ตค่าต่างๆดังนี้

Menu text: **Run**

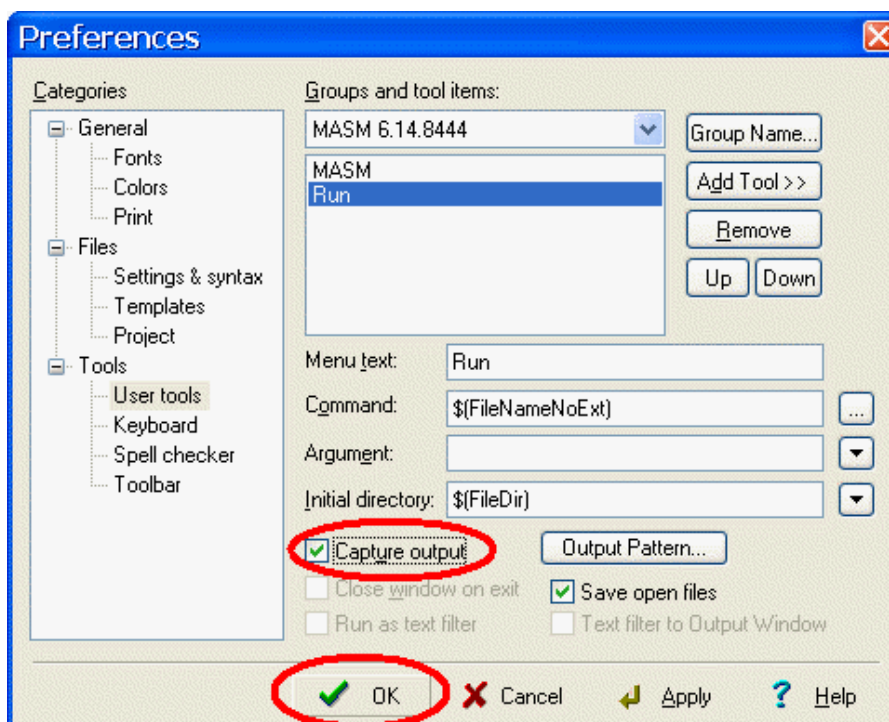
Command: **\$(FileNameNoExt)**

Argument:

Initial directory: **\$(FileDir)**

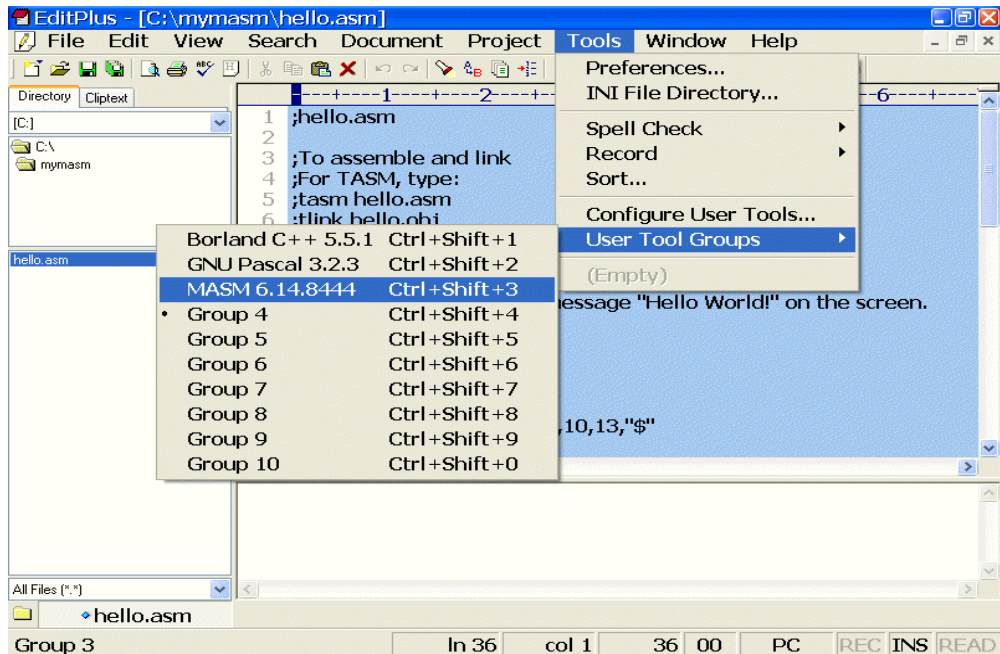
Capture output: **ON**

หลังจากนั้นคลิกปุ่ม OK



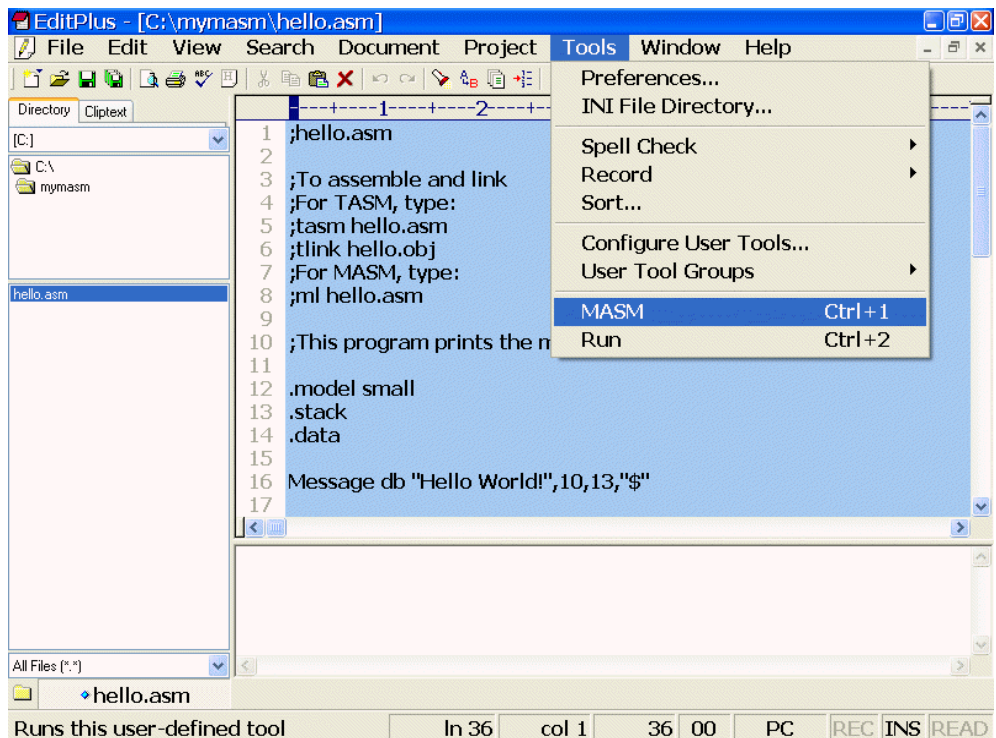
15. From the **Tools** menu, select **User Tool Groups**, and then **MASM 6.14.8444**

15. ที่เมนู **Tools**, เลือก **User Tool Groups**, เลือก **MASM 6.14.8444**



16. To compile, open the **source file**, select **MASM** from the **Tools** menu.

16. การแปลโค้ดให้เปิดไฟล์โค้ดภาษา **Assembly** และเลือก **MASM** จากเมนู **Tools**



17. The result will be shown in the Output Window at the bottom.

17. โปรแกรมจะแสดงผลเป็นข้อความที่หน้าต่าง Output ด้านล่าง

```

;hello.asm
;To assemble and link
;For TASM, type:
;tasm hello.asm
;tlink hello.obj
;For MASM, type:
;ml hello.asm

----- MASM -----
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: C:\mymasm\hello.asm

Microsoft (R) Segmented Executable Linker Version 5.60.339 Dec 5 1994
Copyright (C) Microsoft Corp 1984-1993. All rights reserved.

Object Modules [.obj]: hello.obj
Run File [hello.exe]: "hello.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:

Output completed (0 sec consumed) - Normal Termination

```

18. To run, select **Run** from the **Tools** menu.

18. การรันโปรแกรมให้เลือก **Run** ที่เมนู **Tools**

```

;hello.asm
;To assemble and link
;For TASM, type:
;tasm hello.asm
;tlink hello.obj
;For MASM, type:
;ml hello.asm

----- MASM -----
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: C:\mymasm\hello.asm

Microsoft (R) Segmented Executable Linker Version 5.60.339 Dec 5 1994
Copyright (C) Microsoft Corp 1984-1993. All rights reserved.

Object Modules [.obj]: hello.obj
Run File [hello.exe]: "hello.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:

Output completed (0 sec consumed) - Normal Termination

```

19. The result will be shown in the Output Window at the bottom.

19. โปรแกรมจะแสดงผลพีธีในหน้าต่าง Output ด้านล่าง

```

1 ;hello.asm
2
3 ;To assemble and link
4 ;For TASM, type:
5 ;tasm hello.asm
6 ;tlink hello.obj
7 ;For MASM, type:
8 ;ml hello.asm

```

----- Run -----
Hello World!
Output completed (0 sec consumed) - Normal Termination

20. If you forget the comma, the compiler will give you an error message when you attempt to compile the program.

20. ถ้าท่านลืมเครื่องหมายจุลภาค (,), ตัวแปลภาษาจะแสดงข้อความแจ้งความผิดพลาดเมื่อท่านแปลโค้ด

```

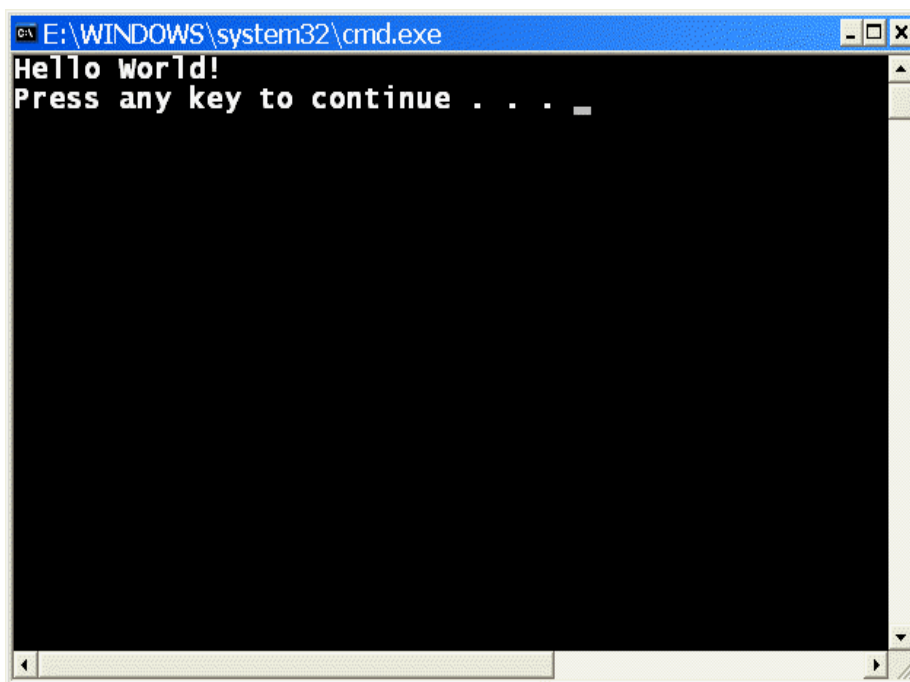
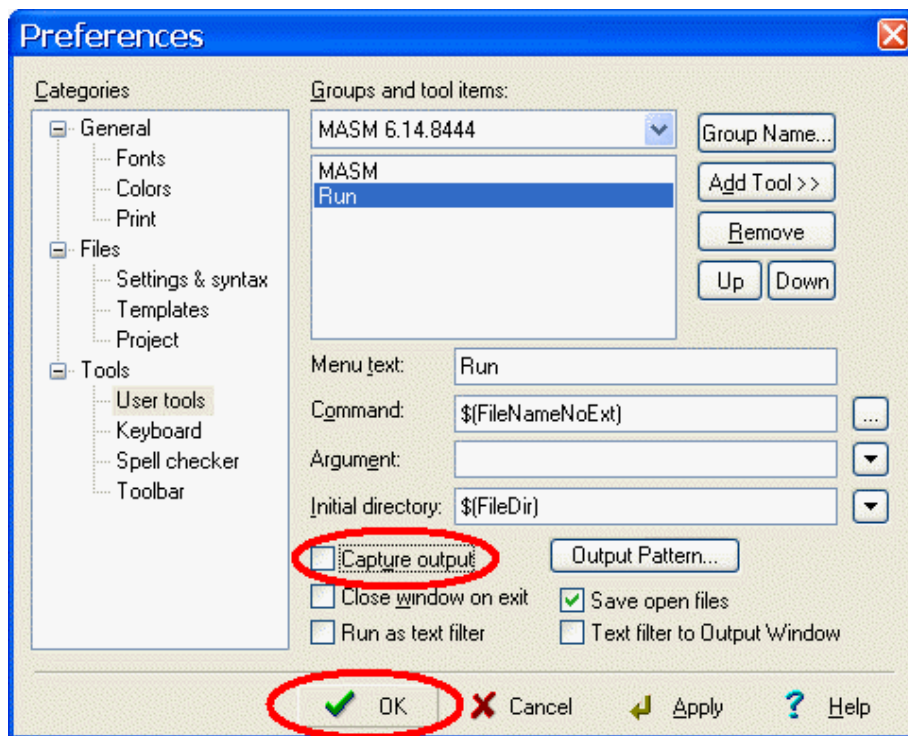
10 ;This program prints the message "Hello World!" on the screen.
11
12 .model small
13 .stack
14 .data
15
16 Message db "Hello World!",10,13 "$"
17
18 .code
19
20 start:
21
22 ;Get segment of Message.
23 mov ax,SEG Message
24 mov ds,ax

```

----- MASM -----
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.
Assembling: C:\mymasm\hello.asm
C:\mymasm\hello.asm(16) : error A2206: missing operator in expression
Output completed (0 sec consumed) - Normal Termination

21. Uncheck the **Capture output** box of the **Run** menu will show the output in a command-prompt window.

21. ถ้าไม่มีเครื่องหมายถูกข้างหน้า **Capture output** ในเมนู **Run** โปรแกรมจะแสดงผลพีธีในหน้าต่างแบบ command-prompt



4.int mouse ใช้ int 33h

สรุป

Microsoft Macro Assembler เป็นโปรแกรมสำหรับแปลภาษาแอสเซมบลี (assembler) ใช้กับเครื่องคอมพิวเตอร์ที่ใช้ซีพียูตระกูล x86 สำหรับใช้กับระบบปฏิบัติการดอส (MS-DOS) และเป็นโปรแกรมสำหรับแปลภาษาแอสเซมบลีที่นิยมใช้มากที่สุด ดังนั้นนักพัฒนาภาษาโปรแกรมภาษาแอสเซมบลีเบื้องต้นจำเป็นที่จะต้องเรียนรู้และฝึกทักษะการเขียนภาษาโปรแกรมและตรวจสอบความถูกต้องอย่างต่อเนื่อง เพื่อทำความเข้าใจและทดสอบโปรแกรมที่ต้นเขียนขึ้นมาซึ่งจะทำให้เข้าใจรูปแบบและวิธีการทำงานอย่างเป็นขั้นตอน รวมถึงช่วยให้สามารถพัฒนาโปรแกรมภาษาขั้นสูงที่เกี่ยวข้องต่อไป

คำถามทบทวน

1. จงยกตัวอย่างโปรแกรมที่ใช้ในการเขียนโปรแกรมภาษาแอสเซมบลีในปัจจุบันนอกเหนือจากโปรแกรม MASM32 หรือ EditPlus ว่ามีวิธีการดาวน์โหลดและติดตั้งเพื่อใช้งานได้อย่างไร
2. จงเขียนโปรแกรมภาษาแอสเซมบลีอ่านค่าจากคีย์บอร์ด (Keyboard) ตัวอักษรภาษาอังกฤษพิมพ์ใหญ่และให้แสดงผลเป็นจากแป้นพิมพ์ออกทางหน้าจอเป็นตัวอักษรภาษาอังกฤษพิมพ์เล็ก
3. จงเขียนโปรแกรมภาษาแอสเซมบลีที่พิมพ์ String โดยใช้ฟังก์ชันเบอร์ 9 ของอินเทอร์พอร์ตเบอร์ 21H
4. จงเขียนโปรแกรมภาษาแอสเซมบลีอ่านค่าจากคีย์บอร์ด (Keyboard) โดยใช้ฟังก์ชันเบอร์ 1 ของอินเทอร์พอร์ตเบอร์ 21H แล้วเปลี่ยนเป็นเลขฐาน 10 และพิมพ์ค่าออกทางจอภาพ (Monitor) โดยใช้ฟังก์ชันเบอร์ 2 ของอินเทอร์พอร์ตเบอร์ 21H
5. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 2 เป็นเลขฐาน 8 และแสดงผลออกทางจอภาพ (Monitor)
6. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 2 เป็นเลขฐาน 10 และแสดงผลออกทางจอภาพ (Monitor)
7. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 2 เป็นเลขฐาน 16 และแสดงผลออกทางจอภาพ (Monitor)
8. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 8 เป็นเลขฐาน 2 และแสดงผลออกทางจอภาพ (Monitor)
9. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 8 เป็นเลขฐาน 10 และแสดงผลออกทางจอภาพ (Monitor)
10. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 8 เป็นเลขฐาน 16 และแสดงผลออกทางจอภาพ (Monitor)
11. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 10 เป็นเลขฐาน 2 และแสดงผลออกทางจอภาพ (Monitor)
12. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 10 เป็นเลขฐาน 8 และแสดงผลออกทางจอภาพ (Monitor)

- 13 จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 10 เป็นเลขฐาน 16 และแสดงผลออกทางทางจอภาพ (Monitor)
14. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 16 เป็นเลขฐาน 2 และแสดงผลออกทางทางจอภาพ (Monitor)
15. จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 16 เป็นเลขฐาน 8 และแสดงผลออกทางทางจอภาพ (Monitor)
- 16 จงเขียนโปรแกรมภาษาแอสเซมบลีที่เปลี่ยนค่าจากเลขฐาน 16 เป็นเลขฐาน 10 และแสดงผลออกทางทางจอภาพ (Monitor)

บรรณานุกรม

- Agarwal R.K. 80x86 Architecture and Programming Vol 2 Architecture Reference. USA : Prentice Hall Inc, 1991.
- Bradley, David J. Assembly Language Programming for the IBM Personal Computer. USA : Prentice Hall Inc, 1984.
- Holzner, Steven. Creating Utilities with Assembly Language 10 Best for IBM PC & XT. USA : Brady Communication Company, 1986.
- Jermann, William H. The Structure and Programming of Microcomputer. USA : Alfred Publishing Co Inc, 1982.
- Thorne M. Computer Organization and Assembly Language Programming 2nd Edition. Benjamin Cumming Publishing Company : USA, 1991.
- Williams Dave. The Programming Technical Reference MS-DOS, IBM PC & Compatibles. Singapore : John Wiley & Sons (SEA), 1990.
- ฉวีวรรณ โสภจารย์. โครงสร้างคอมพิวเตอร์และการเขียนโปรแกรมภาษาแอสเซมบลี. กรุงเทพฯ : ภาควิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์ สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ, 2548.
- ชูชัย ธนสารตั้งเจริญ, กาธร พานิชปฐมพงษ์. ภาษาแอสเซมบลี 80286/80386 (PC). กรุงเทพฯ : สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.
- ธีรวัฒน์ ประกอบผล. ระบบคอมพิวเตอร์และภาษาแอสเซมบลี. กรุงเทพฯ : สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.
- ศิริวรรณ ฉันทาดิสัย. หลักการเขียนโปรแกรมภาษาแอสเซมบลี 8088. กรุงเทพฯ : สำนักพิมพ์ประการพริก, 2534.