เอกสารประกอบการสอน

รายวิชา ภาษาแอสแซมบลีและไมโครคอมพิวเตอร์

(Microprocessor and Assembly Language)

จุฑาวุฒิ จันทรมาลี

วท.บ., วท.ม.(วิทยาการคอมพิวเตอร์)

คณะวิทยาศาสตร์และเทคโนโลยี

มหาวิทยาลัยราชภัฏสวนดุสิต

2557

เอกสารประกอบการสอนวิชา : **ภาษาแอสเซมบลีและไมโครคอมพิวเตอร์**

เรียบเรียงโดย : จุฑาวุฒิ จันทรมาลี

ปีที่พิมพ์ : 2557

จำนวนหน้า : 278 หน้า

พิมพ์จำนวน : 450 เล่ม

ราคา : 300 บาท

พิมพ์ที่ : โครงการสวนดุสิตกราฟฟิคไซท์
มหาวิทยาลัยราชภัฏสวนดุสิต
ถนนนครราชสีมา เขตดุสิต กรุงเทพฯ 10300
โทร. 0-2244-5081, โทรสาร 0-2243-9113

คำนำ

เอกสารประกอบการสอน รายวิชา ภาษาแอสแซมบลีและไมโครคอมพิวเตอร์ รหัสวิชา 4123314 นี้ ได้เรียบเรียงขึ้นอย่างเป็นระบบ ครอบคลุมเนื้อหาสาระรายวิชา ในหมวดวิชาเฉพาะของมหาวิทยาลัย เพื่อใช้เป็นเครื่องมือสำคัญของผู้สอนในการใช้ประกอบการสอนของอาจารย์ ที่มุ่งเน้นให้ผู้เรียนมีความรู้ความเข้าใจในเนื้อหา

เอกสารเล่มนี้ ได้แบ่งเนื้อหาในการเรียนการสอนไว้ 15 สัปดาห์ ได้แก่ ระบบคอมพิวเตอร์ ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ ระบบจำนวนและภาษาสำหรับเขียนโปรแกรมคอมพิวเตอร์ สถาปัตยกรรมคอมพิวเตอร์เบื้องต้น คำสั่งโอนย้ายข้อมูล แฟล็กและคำสั่งคณิตศาสตร์ โปรแกรมภาษาแอสแซมบลี้เบื้องต้น การประกาศข้อมูล คำสั่งกระโดดและคำสั่งการกระทำซ้ำ โครงสร้างควบคุม โปรแกรมย่อยเบื้องต้น การกระทำระดับบิต การอ้างแอดเดรส การขัดจังหวะและคำสั่งจัดการกับสายข้อมูล คำสั่งตารางและการสร้างแมคโคร การฝึกทักษะการเขียนโปรแกรมภาษาแอสแซมบลีเบื้องต้น ผู้สอนควรได้ศึกษารายละเอียดแต่ละหัวข้อเรื่องที่สอนจากเอกสาร หนังสือ ตำรา หรือสื่ออื่นๆ เพิ่มเติมอีก หวังว่าเอกสารประกอบการสอนนี้คงอำนวยประโยชน์ต่อการเรียนการสอนตามสมควร หากท่านนำไปใช้มีข้อเสนอแนะ ผู้เขียนยินดีรับฝังข้อคิดเห็นต่างๆ และขอขอบคุณมา ณ โอกาศนี้

จุฑาวุฒิ จันทรมาลี

4 ตุลาคม 2557

สารบัญ

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

สารบัญภาพ

ภาพที่ **หน้า**

ภาพที่ **หน้า**

ภาพที่ **หน้า**

สารบัญตาราง

ตารางที่ **หน้า**

(20)

แผนบริหารการสอนประจำรายวิชา

รายวิชา ภาษาแอสแซมบลีและไมโครคอมพิวเตอร์ **รหัสวิชา** 4123314
(Microprocessor and Assembly Language)

จำนวนหน่วยกิต 3 (2-2-5) **ชั่วโมง**

เวลาเรียน คิด 15 สัปดาห์ (3 x 15) = 45 ชั่วโมง/ภาคเรียน 45 ชั่วโมง/ภาคเรียน
ศึกษาค้าคว้าด้วยตัวเอง = 5 x 15 = 75 ชม./ภาคเรียน

คำอธิบายรายวิชา

ศึกษาเกี่ยวกับความรู้ทั่วไปของไมโครโพรเซสเซอร์หรือไมโครคอนโทรลเลอร์ โครงสร้างของไมโครโพรเซสเซอร์ การเกิดอินเทอร์รัพต์ และการติดต่อสื่อสารกับหน่วยความจา การติดต่อรับส่งข้อมูลกับอุปกรณ์ภายนอก และการเขียนภาษาแอสแซมบลีสำหรับไมโครโพรเซสเซอร์ หรือไมโครคอนโทรลเลอร์

จุดมุ่งหมายรายวิชา

1. เพื่อให้ผู้เรียนมีความรู้ความเข้าใจ เกี่ยวกับไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ โครงสร้างของไมโครโพรเซสเซอร์ การเกิดอินเทอร์รัพต์ การเกิดอินเตอร์รัพต์ รีจีสเตอร์ของซีพียู 8086 การระบุตำแหน่งในรีจีสเตอร์ การเข้าถึงข้อมูลในหน่วยความจำ

2. เพื่อให้ผู้เรียนมีความรู้ความเข้าใจ สามารถที่จะใช้โปรแกรมดีบัก (Debug) ในการตรวจสอบการทำงานในระดับบิตคำสั่งและการจัดการกับบิตข้อมูลได้

3. เพื่อให้ผู้เรียนมีความรู้ความเข้าใจสามารถเขียนภาษาโปรแกรมแอสแซมบลีเบื้องต้นได้

เนื้อหา

แผนการสอนประจำสัปดาห์ที่ 1 3 ชั่วโมง/สัปดาห์

อธิบายถึงองค์ประกอบของระบบคอมพิวเตอร์ ชนิดของคอมพิวเตอร์ตลอดจนโครงสร้างของฮาร์ดแวร์และซอฟต์แวร์ที่นำมาใช้ร่วมกับระบบคอมพิวเตอร์

แผนการสอนประจำสัปดาห์ที่ 2 3 ชั่วโมง/สัปดาห์

อธิบายถึงของแตกต่างระหว่างไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ ชนิดของไมโครโพรเซสเซอร์ โครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์ การเกิดอินเตอร์รัพต์ การระบุตำแหน่งในรีจีสเตอร์และการเข้าถึงข้อมูลในหน่วยความจำ

แผนการสอนประจำสัปดาห์ที่ 3 3 ชั่วโมง/สัปดาห์

อธิบายความหมายของตัวเลขในหลักต่าง ๆ การแปลงค่าจากเลขฐานสิบเป็นเลขฐานสองการบวกและการลบเลขฐานสอง การคูณและการหารเลขฐานสอง เลขฐานแปดและเลขฐานสิบหก บิต, ไบต์, เวิร์ด และดับเบิ้ลเวิร์ด ระบบเลข 2' Complement ระบบเล BCD (Binary Code Decimal) และมาตรฐาน IEEE 754

แผนการสอนประจำสัปดาห์ที่ 4 3 ชั่วโมง/สัปดาห์

อธิบายภาษาต่าง ๆ สำหรับคอมพิวเตอร์ ระดับของภาษาสำหรับเขียนโปรแกรมการแปลภาษาสำหรับคอมพิวเตอร์และภาษาแอสแซมบลี

แผนการสอนประจำสัปดาห์ที่ 5 3 ชั่วโมง/สัปดาห์

อธิบายหน่วยประมวลผลกลาง หน่วยความจำ การเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ และของระบบไมโครโปรเซสเซอร์ตระกูล 80x86

แผนการสอนประจำสัปดาห์ที่ 6 3 ชั่วโมง/สัปดาห์

อธิบายคำสั่งในการโยนย้ายข้อมูล การใช้คำสั่ง MOV เครื่องมือในการทดลองการโปรแกรมภาษาแอสแซมบลี คำสั่งในการโยนย้ายข้อมูลและคำสั่งทั่วไป การทำงานของแฟล็ก (Flags) คำสั่งทางคณิตศาสตร์ กลุ่มคำสั่งบวกและลบ กลุ่มคำสั่งคูณและหาร กลุ่มคำสั่งแปลงขนาดตัวเลข

แผนการสอนประจำสัปดาห์ที่ 7 3 ชั่วโมง/สัปดาห์

อธิบายรูปแบบของโปรแกรมภาษาแอสแซมบลีแบบเก่า เปรียบเทียบกับรูปแบบของโปรแกรมภาษาแอสแซมบลีแบบใหม่ ขั้นตอนการแปลโปรแกรม การเรียกใช้บริการของ DOS ตารางรหัสแอสกี

แผนการสอนประจำสัปดาห์ที่ 8 3 ชั่วโมง/สัปดาห์

อธิบายรูปแบบและวิธีการประกาศข้อมูล การอ้างใช้ข้อมูลที่ประกาศไว้ การอ้างตำแหน่งของข้อมูล การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah การใช้บริการของ DOS หมายเลข 0Ah : การอ่านข้อความ และหมายเลข 09h : การอ่านพิมพ์ข้อความ คำสั่งกระโดดแบบไม่มีเงื่อนไข คำสั่งกระโดดที่พิจารณาค่าจากแฟล็ก คำสั่งกระโดดที่พิจารณาค่าจากรีจิสเตอร์ มีความรู้และความเข้าใจเกี่ยวกับกลุ่มคำสั่งวนรอบ เช่น LOOP LOOPZ และ LOOPNZ เป็นต้น

แผนการสอนประจำสัปดาห์ที่ 9 3 ชั่วโมง/สัปดาห์

อธิบายการสร้างโครงสร้างการตัดสินใจแบบ if–then–else การสร้าง repeat until loop การสร้าง while loop และการสร้าง for loop

แผนการสอนประจำสัปดาห์ที่ 10 3 ชั่วโมง/สัปดาห์

อธิบายคำสั่งที่รองรับการเรียกโปรแกรมย่อย การประกาศโปรแกรมย่อย คำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแสต็กและตัวอย่างการใช้งานโปรแกรมย่อย

แผนการสอนประจำสัปดาห์ที่ 11 3 ชั่วโมง/สัปดาห์

อธิบายคำสั่งทางตรรกศาสตร์ การประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์ การใช้คำสั่งเลื่อนบิต ตัวอย่างการใช้งานคำสั่งเลื่อนบิต การประยุกต์ใช้งานคำสั่งเลื่อนบิต คำสั่งหมุนบิต คำสั่งหมุนบิตที่ผ่านแฟล็กทด ตัวอย่างการใช้งานคำสั่งเกี่ยวกับการประมวลผลระดับบิต

แผนการสอนประจำสัปดาห์ที่ 12 3 ชั่วโมง/สัปดาห์

อธิบายเกี่ยวกับการอ้างแอดเดรสในเซกเมนต์ทั้งสี่ คือ CS, DS, SS และ ES การประยุกต์ใช้งานคำสั่งในการอ้างแอดเดรสแบบต่าง ๆ

แผนการสอนประจำสัปดาห์ที่ 13 3 ชั่วโมง/สัปดาห์

อธิบายกระบวนการขัดจังหวะและการใช้คำสั่งติดต่อกับอุปกรณ์ใน 8086 คำสั่งจัดการสายข้อมูลต่างๆ เช่น คำสั่ง MOVS, คำสั่ง REP, คำสั่ง STOS, คำสั่ง LODS, คำสั่ง REPZ, คำสั่ง CMPS, คำสั่ง REPNZ และคำสั่ง SCAS เป็นต้น

แผนการสอนประจำสัปดาห์ที่ 14 3 ชั่วโมง/สัปดาห์

อธิบายคำสั่ง XLAT ความหมายข้อดีและข้อเสียของการใช้แมคโครพารามิเตอร์ของแมคโครการใช้ LABEL ในแมคโครและแมคโครกับโปรแกรมย่อย

แผนการสอนประจำสัปดาห์ที่ 15 3 ชั่วโมง/สัปดาห์

อธิบายมีวิธีการดาวน์โหลดและติดตั้งโปรแกรม MASM32 รวมถึงวิธีการดาวน์โหลดและติดตั้งโปรแกรม EditPlus การประยุกต์ใช้งานคำสั่งและฝึกทักษะการเขียนโปรแกรมภาษาแอสแซมบลีและตรวจสอบความถูกต้องของโปรแกรมที่เขียนขึ้นมาเบื้องต้นได้

วิธีสอนและกิจกรรม

1. บรรยาย โดยการเสนอข้อมูลและปัญหา การตั้งคำถามรหว่างการเรียนการสอน
2. สืบเสาะหาความรู้ แบ่งผู้เรียนออกเป็นกลุ่ม แบ่งหัวข้อให้แต่ละกลุ่มไปศึกษา
3. ค้นคว้าเพิ่มเติม ทำแบบฝึกหัดและกิจกรรมทั้งในและนอกชั้นเรียน
4. วิธีการสาธิต การใช้เขียนโปรแกรมภาษาแอสแซมบลีเบื้องตั้น

สื่อการเรียนการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. โปรแกรมภาษาแอสแซมบลี MASM32

การวัดผลและประเมินผล

1. การวัดผล

1.1 คะแนนระหว่างภาครวม	ร้อยละ 60
1.1.1 คะแนนเข้าชั้นเรียน	ร้อยละ 10
1.1.2 คะแนนจัดทำรายงายกลุ่ม	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

(26)

แผนการสอนประจำสัปดาห์ที่ 1

หัวข้อเรื่อง ระบบคอมพิวเตอร์ (Computer System)

รายละเอียด

คอมพิวเตอร์ คือ อุปกรณ์ที่มนุษย์สร้างขึ้นเพื่ออำนวยความสะดวกและช่วยในการทำงานของมนุษย์ โดยมีการใช้งานที่แตกต่างกันออกไป พัฒนาการของคอมพิวเตอร์มีมาอย่างต่อเนื่องจนในปัจจุบันคอมพิวเตอร์เป็นที่นิยม และราคาถูกลงมากเมื่อเทียบกับสมัยก่อน อีกทั้งความสามารถและประสิทธิภาพของคอมพิวเตอร์ก็เพิ่มขึ้นทั้งในด้านความเร็วในการประมวลผลข้อมูล ความสามารถในการเก็บข้อมูลและปลอดภัยเพิ่มมากขึ้นด้วย

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 1 การจัดการเรียนการสอน จะเกี่ยวข้องกับองค์ประกอบของระบบคอมพิวเตอร์ ชนิดของคอมพิวเตอร์ ตลอดจนโครงสร้างของฮาร์ดแวร์และซอฟต์แวร์ที่นำมาใช้ร่วมกับระบบคอมพิวเตอร์ ซึ่งเป็นอุปกรณ์ที่มนุษย์สร้างขึ้นเพื่ออำนวยความสะดวกและช่วยในการทำงานของมนุษย์ โดยมีการใช้งานที่แตกต่างกันออกไป พัฒนาการของคอมพิวเตอร์มีมาอย่างต่อเนื่องจนในปัจจุบันคอมพิวเตอร์เป็นที่นิยม และราคาถูกลงมากเมื่อเทียบกับสมัยก่อน อีกทั้งความสามารถและประสิทธิภาพของคอมพิวเตอร์ก็เพิ่มขึ้นทั้งในด้านความเร็วและในการประมวลผลข้อมูล มีความสามารถในการเก็บข้อมูลจำนวนมากได้อย่างมีประสิทธิภาพและมีความเร็วเพิ่มมากขึ้นด้วย

1.1 องค์ประกอบของระบบคอมพิวเตอร์

1.1.1 ฮาร์ดแวร์ (hardware) เป็นอุปกรณ์ส่วนประกอบของเครื่องคอมพิวเตอร์ โดยแบ่งเป็น 5 ประเภท เช่น อุปกรณ์รับข้อมูล (input), อุปกรณ์นำส่งข้อมูล (output), อุปกรณ์ประมวลผลข้อมูล (system unit), อุปกรณ์จัดเก็บข้อมูล (storage device), และอุปกรณ์ที่ใช้ในการติดต่อสื่อสารและนำส่งข้อมูล (communication device) เป็นต้น

1.1.2 ซอฟต์แวร์ (software) คือ ส่วนของโปรแกรมภาษาชุดคำสั่งที่เขียนขึ้นมาเพื่อกำหนดขั้นตอนการทำงานต่างๆ ของระบบคอมพิวเตอร์ ตามหน้าที่หรือจุดประสงค์ที่ผู้เขียนโปรแกรมต้องการ เช่น Window, Winamp, Winzip, Word-Processor, Power-DVD เป็นต้น (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

1.1.3 ส่วนบุคคล (peopleware) คือ บุคลากรที่ส่วนเกี่ยวข้องกับคอมพิวเตอร์ เช่น ผู้ใช้งานทั่วไป, นักเขียนโปรแกรมภาษาชุดคำสั่ง, นักวิเคราะห์และออกระบบ เป็นต้น

1.1.4 ข้อมูล (data) คือ ข้อมูลที่เก็บอยู่ในระบบคอมพิวเตอร์เพื่อนำไปใช้งานให้ตรงตามวัตถุประสงค์ของผู้ใช้งาน อาจเป็นได้หลายรูปแบบ เช่น รหัสของข้อมูลในรูปแบบต่างๆ ตัวอักขระ ตัวเลข รูปภาพ เสียง และ วิดีโอ เป็นต้น

1.2 ชนิดของคอมพิวเตอร์ สามารถแบ่งคอมพิวเตอร์ตามรูปแบบการใช้งานได้เป็น 5 ชนิด ได้แก่

1.2.1 คอมพิวเตอร์ส่วนบุคคล (personal computer) เป็นคอมพิวเตอร์ตั้งโต๊ะทั่วไป โดยมีคุณสมบัติหรือขีดความสามารถ เช่น รับข้อมูลเข้า ประมวลผล ส่งข้อมูลออก และเก็บจัดเก็บข้อมูล เป็นต้น

1.2.2 คอมพิวเตอร์พกพา (mobile computer) และอุปกรณ์พกพา (mobile device) เป็นคอมพิวเตอร์ส่วนบุคคลที่ผู้ใช้สามารถพกพานำไปใช้ในที่ต่างๆ ส่วนอุปกรณ์พกพา หมายถึง อุปกรณ์คอมพิวเตอร์ที่มีขนาดเล็ก สามารถพกพาได้สะดวก แต่ยังมีขีดความสามารถจำกัดเมื่อเทียบกับคอมพิวเตอร์ส่วนบุคคล

1.2.3 เครื่องให้บริการขนาดกลาง (midrange servers) เป็นคอมพิวเตอร์ที่มีขนาดใหญ่ ราคาไม่แพงนักและมีประสิทธิภาพในการประมวลผลอย่างรวดเร็ว ทำงานบนระบบเครือข่าย (network) ที่มีการเชื่อมต่อและทำงานร่วมกันในระดับ 100 – 1000 เครื่อง

1.2.4 เมนเฟรม (mainframe) เป็นคอมพิวเตอร์ที่มีขนาดใหญ่ ราคาแพงและมีประสิทธิภาพความเร็วในการประมวลผลสูง รองรับการทำงานร่วมกันของเครื่องคอมพิวเตอร์บนระบบเครือข่ายมากกว่า 1000 เครื่อง สามารถเก็บข้อมูลและภาษาชุดคำสั่งจำนวนมากได้

1.2.5 ซุปเปอร์คอมพิวเตอร์ (supercomputer) เป็นคอมพิวเตอร์ที่มีการทำงานประสิทธิภาพและความเร็วในการประมวลผลเร็วที่สุดในชนิดของคอมพิวเตอร์ที่กล่าวมา อีกทั้งยังมีราคาแพงที่สุด เหมาะกับงานที่มีการคำนวณที่ซับซ้อนมากและความแม่นยำสูงสุด

ชนิดของคอมพิวเตอร์ สามารถแบ่งได้ตามขนาดและการใช้งานของคอมพิวเตอร์ได้ดังนี้

1. ซุปเปอร์คอมพิวเตอร์ (super computer) เป็นคอมพิวเตอร์ที่มีขนาดใหญ่ มีสมรรถนะสูง สามารถประมวลได้เร็ว และมีความสามารถในการเก็บข้อมูลขนาดใหญ่ เช่น สถิติประชากร การขุดเจาะน้ำมัน คอมพิวเตอร์ชนิดนี้มีราคาแพงที่สุด ส่วนใหญ่จะใช้งานในหน่วยงานหรือองค์กรที่มีการทำงานที่ต้องการความเร็วสูง เช่น งานวิเคราะห์ภาพถ่ายจากดาวเทียมของอุตุนิยมวิทยา หรือดาวเทียมที่ใช้สำรวจทรัพยากรใต้พื้นโลก งานวิเคราะห์และพยากรณ์ภูมิอากาศ งานสร้างแบบจำลองโมเลกุลของสารเคมีชนอดต่างๆ งานวิเคราะห์โครงสร้างภายในตัวอาคารที่มีความซับซ้อน เป็นต้น ซึ่งในประเทศไทยมีเครื่องซูเปอร์คอมพิวเตอร์รุ่น Cray YMP ใช้ในงานค้นคว้าวิจัย อยู่ที่ ณ ห้องปฏิบัติการคอมพิวเตอร์สมรรถภาพสูง (HPCC) ศูนย์เทคโนโลยีอิเล็กทรอนิกส์ และสำนักงานคอมพิวเตอร์แห่งชาติ ผู้ใช้งานส่วนมากมักเป็นนักวิจัยทางด้านวิศวกรรมและนักวิทยาศาสตร์ทั่วประเทศ โดยบริษัทผู้ผลิตที่ได้รับความหน้าเชื่อถือ ได้แก่ บริษัทเครย์รีเสิร์ซ (Cray Research), บริษัท เอ็นอีซี (NEC) เป็นต้น แสดงได้ดังภาพที่ 1.1 – 1.4

ภาพที่ 1.1 –1.4 แสดงภาพซุปเปอร์คอมพิวเตอร์ (super computer)
ที่มา: Lady Spor.2010

2. เมนเฟรม (mainframe) เป็นคอมพิวเตอร์ขนาดใหญ่ แต่มีขาดเล็กกว่าและมีสมรรถนะต่ำกว่าซุปเปอร์คอมพิวเตอร์ มีราคาแพง นิยมใช้งานกับภาคธุรกิจขนาดใหญ่ เช่น ธนาคาร โรงแรม หรือ ใช้เป็นเซิร์ฟเวอร์ขององค์การขนาดใหญ่ เป็นต้น การใช้ชื่อว่าเมนเฟรมคอมพิวเตอร์ ก็เพราะครั้งแรกที่สร้างคอมพิวเตอร์ลักษณะนี้ได้สร้างไว้บนฐานรองรับที่เรียกว่า คัสซี่ (Chassis) โดยต่อมาภายหลังได้ชื่อเรียกฐานรองรับนี้ว่า เมนเฟรม ซึ่งคอมพิวเตอร์เมนเฟรมที่มีชื่อเสียงมาก คือ เครื่องของบริษัท IBM (The International Business Machines Corporation) แสดงได้ดังภาพที่ 1.5 – 1.8

ภาพที่ 1.5 – 1.7 แสดงภาพเมนเฟรม (mainframe)
ที่มา: wordpress.2014

3. มินิคอมพิวเตอร์ (minicomputer) เป็นคอมพิวเตอร์ที่มีสมรรถนะรองลงมาจากเครื่องคอมพิวเตอร์เมนเฟรม ประมวลผลได้ช้าและราคาเครื่องก็ยังถูกกว่าเครื่องคอมพิวเตอร์เมนเฟรม เหมาะกับธุรกิจขนาดกลางและขนาดย่อมที่ต้องการความสามารถในการประมวลผลด้วยความเร็วสูง ราคาไม่แพงจนเกินไป เช่น หน่วยงานหรือองค์กร สถานบันการศึกษาในระดับอุดมศึกษาต่างๆ เป็นต้น แสดงได้ดังภาพที่ 1.9

ภาพที่ 1.9 แสดงภาพมินิคอมพิวเตอร์ (minicomputer)
ที่มา: suwanpaiboon.2014

4. ไมโครคอมพิวเตอร์ (microcomputer/personal computer) เป็นคอมพิวเตอร์ส่วนบุคคล หรือคอมพิวเตอร์แบบตั้งโต๊ะ(Desktop computer) มีขนาดเล็กกว่ามินิคอมพิวเตอร์ เหมาะกับการนำมาใช้งานที่ไม่จำเป็นต้องใช้ความเร็วสูงมาก แต่ในปัจจุบันความสามารถของคอมพิวเตอร์ส่วนบุคคลได้พัฒนาประสิทธิภาพได้สูงขึ้นมากและราคาไม่แพง เป็นที่นิยมนำมาใช้งานในครัวเรือน แสดงได้ดังภาพที่ 1.10 และภาพที่ 1.11 อีกทั้งยังได้รับการพัฒนาในรูปแบบที่สามารถพกพาสะดวกขึ้น ได้แก่

ภาพที่ 1.10 – 1.11 แสดงภาพไมโครคอมพิวเตอร์ (microcomputer/personal computer)
ที่มา: signalbattalion 3rd.2002

4.1 โน๊ตบุค (Notebook Computer/ Laptop) เป็นคอมพิวเตอร์ส่วนบุคคลขนาดเล็กประมาณสมุดโน๊ต ประหยัดไฟและมีราคาใกล้เคียงกับคอมพิวเตอร์ตั้งโต๊ะ ใช้แบตเตอร์รี่ในการสำรองไฟ พกพาได้สะดวก ปัจจุบันได้พัฒนาให้มีขนาดเล็กบางและน้ำหนักเบา อีกทั้งยังมีความสามารถเทียบเท่ากับคอมพิวเตอร์ตั้งโต๊ะที่ใช้งานในครัวเรือน แสดงได้ดังภาพที่ 1.12 – 1.14

ภาพที่ 1.12 – 1.14 แสดงภาพโน๊ตบุค (Notebook Computer/ Laptop)
ที่มา: rakuten.2013

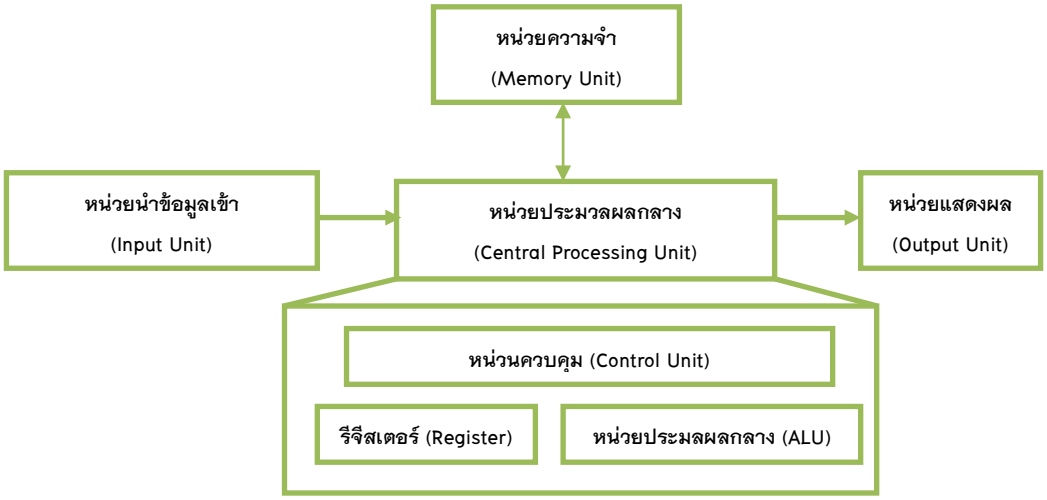
4.2 เทบเล็ต (tablet PC) มีลักษะคล้ายกระดานชนวน ที่ใช้เขียนตัวหนังสือในสมัยก่อน ลักษณะคล้ายสมุดจดบันทึกหรือสมุดโน๊ต ผู้ใช้งานสามารถที่จะขีดเขียนตัวหนังสือหรือวาดภาพลงไปบนหน้าจอได้โดยไม่จำเป็นต้องใช้คีย์บอร์ดเป็นอุปกรณ์นำเข้าข้อมูล เหมาะสำหรับผู้ใช้ที่ชอบงานเขียนมากกว่างานพิมพ์ ซึ่งในปัจจุบันเทบเล็ตสามารถบันทึกข้อมูลในรูปแบบเสียงได้อีกด้วย แสดงได้ดังภาพที่ 1.15 – 1.16

ภาพที่ 1.15 – 1.16 แสดงภาพเทบเล็ต (tablet PC)
ที่มา: rakuten.2013

4.3 พีดีเอ (PDA: Personal Digital Assistant) เป็นคอมพิวเตอร์ขนาดเล็กเท่าฝ่ามือ มีความเร็วในการประมวลช้ากว่าคอมพิวเตอร์ส่วนบุคคลท แต่ยังสามารถดูหนัง ฟังเพลง เล่นอินเตอร์เน็ต ดูปฏิทิน และยังสามารถเป็นสมุดนัดหมายได้ ซึ่งในบางรุ่นสามารถเป็นโทรศัพท์เคลื่อนที่ มีอุปกรณ์นำข้อมูลที่เรียกว่า สไตล์ลัส (stylus) ซึ่งมีลักษณะคล้ายปากกาใช้แรงกดลงไปบนหน้าจอพีดีเอ บางรุ่นสามารถบันทึกข้อมูลในรูปแบบเสียงได้อีกด้วย แสดงได้ดังภาพที่ 1.17 – 1.19

ภาพที่ 1.17 – 1.19 แสดงภาพพีดีเอ (PDA: Personal Digital Assistant)
ที่มา: Interactive Graphical SCADA System.2010

1.3 โครงสร้างของฮาร์ดแวร์ของระบบคอมพิวเตอร์ โดยทั่วไปแล้วคอมพิวเตอร์แต่ละรุ่น มีสถาปัตยกรรมของส่วนประกอบต่าง ๆ ในระบบฮาร์ดแวร์ที่ไม่เหมือนกัน แต่ถ้ามองโครงสร้างของคอมพิวเตอร์ในรูปแบบส่วนประกอบย่อย (Module) ต่างๆ แล้วจะมีส่วนประกอบต่าง ๆ ที่มีรูปแบบเหมือนกัน แสดงได้ดังภาพที่ 1.20



ภาพที่ 1.20 แสดงโครงสร้างของฮาร์ดแวร์ของระบบคอมพิวเตอร์

1.3.1 หน่วยประมวลผลกลาง (Central Processing Unit : CPU) หน่วยประมวลผลกลางจัดได้ว่าเป็นส่วนที่สำคัญที่สุดเปรียบเสมือนเป็นสมองของเครื่องคอมพิวเตอร์ โดยทำหน้าที่ในการคำนวณค่าต่าง ๆ ตามชุดคำสั่งที่ได้รับ และควบคุมการทำงานส่วนประกอบอื่น ๆ ทั้งหมด ในเครื่องไมโครคอมพิวเตอร์ หน่วยประมวลผลกลางจะถูกสร้างให้อยู่ในรูปของวงจรรวม (Integrated Circuit: IC) เพียงตัวเดียว ทำให้ง่ายต่อการนำไปใช้งาน ภายในหน่วยประมวลผลกลางจะมีส่วนประกอบย่อย ๆ ที่สำคัญ 3 ส่วนคือ

1. หน่วยคำนวณทางคณิตศาสตร์และตรรกะ (Arithmetic and Logic Unit : ALU) เป็นหน่วยที่ทำหน้าที่ประมวลผลโดยโดยวิธีทางคณิตศาสตร์ เช่น บวก ลบ คูณ หาร หรือ ทำหน้าที่ประมวลผลทางตรรกะ (logic) เช่น AND, OR, NOT, COMPLEMENT เป็นต้น อีกทั้งยังทำหน้าที่ในการเปรียบเทียบค่าต่าง ๆ อีกด้วย

2. หน่วยเก็บข้อมูลชั่วคราว (Register) เป็นหน่วยความจำขนาดเล็ก ทำหน้าที่เป็นที่พักข้อมูลชั่วคราวก่อนที่จะถูกนำไปประมวลผล โดยปกติแล้วในหน่วยประมวลผลกลางจะมีหน่วยเก็บข้อมูลชั่วคราว (Register) สำหรับเก็บข้อมูลได้ไม่เกิน 64 ตัว การอ้างอิงข้อมูลของหน่วยเก็บข้อมูลชั่วคราว (Register) จะมีความเร็วเท่ากับความเร็วของหน่วยประมวลผลกลาง เพราะเป็นหน่วยความจำส่วนที่อยู่ภายในตัวหน่วยประมวลผลกลาง จึงไม่จำเป็นต้องไปอ้างอิงถึงตำแหน่งภายนอกของหน่วยประมวลผล (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

3. หน่วยควบคุม (Control Unit) เป็นเสมือนหน่วยบัญชาการ ทำหน้าที่กำหนดจังหวะการทำงานต่าง ๆ ของระบบคอมพิวเตอร์ โดยไม่เว้นแม้แต่ส่วนประกอบอื่น ๆ ของหน่วยประมวลผลกลาง (CPU) นอกจากนี้ยังทำหน้าที่ควบคุมการส่งข้อมูลระหว่างหน่วยงานต่าง ๆ ในระบบคอมพิวเตอร์อีกด้วย

1.3.2 หน่วยความจำ (Memory Unit) เป็นหน่วยความจำที่ทำหน้าที่เก็บข้อมูลต่างๆ ในระบบคอมพิวเตอร์ ไม่ว่าจะเป็น ข้อมูลตัวเลขหรือข้อความ หรือชุดคำสั่งต่าง ๆ ที่ใช้ในการสั่งการทำงานระบบคอมพิวเตอร์ โดยทั่วไปแล้วหน่วยความจำจะถูกสร้างมาบนวงจรรวม (Integrated Circuit: IC) เพื่อให้มีความจุสูงแต่มีขนาดเล็ก ดังนั้นข้อมูลที่เก็บในหน่วยความจำจะมีสถานะเพียงแค่เปิดวงจร (0) หรือปิดวงจร (1) เท่านั้น หน่วยความจำสามารถแบ่งออกได้เป็นสองประเภทใหญ่ ๆ คือ

1. ROM (Read Only Memory) เป็นหน่วยความจำส่วนที่ CPU สามารถอ่านข้อมูลออกมาใช้งานได้ตามวิธีการปกติ แต่เมื่อต้องการจะเขียนข้อมูลลงไปจะต้องใช้วิธีการพิเศษ ทำให้ต้องมีวงจรในการเขียนข้อมูลโดยเฉพาะ ข้อดีของหน่วยความจำชนิดนี้ก็คือ ข้อมูลที่

เขียนลงไปจะคงอยู่ไปตลอดแม้ว่าจะมีการปิดเครื่องคอมพิวเตอร์ไปแล้วก็ตาม ดังนั้น ROM จึงมักจะถูกใช้เก็บชุดคำสั่งที่ใช้เริ่มต้นการทำงานของเครื่องคอมพิวเตอร์

2. RAM (Random Access Memory) เป็นหน่วยความจำที่ CPU สามารถอ่านเขียนข้อมูลได้ตามวิธีการปกติสามารถเปลี่ยนแปลงข้อมูลภายในได้ตลอดเวลา บางครั้ง RAM จึงมักจะถูกนำไปใช้เก็บข้อมูลระหว่างการทำงานของระบบ แต่ RAM ก็มีข้อเสียคือข้อมูลที่เก็บไว้ทั้งหมดจะสูญหายไปทันที หากไม่มีไฟฟ้าในการทำงาน

1.3.3 หน่วยนำข้อมูลเข้าและหน่วยแสดงผลข้อมูลออก (I/O Unit) เป็นหน่วยที่เปรียบเสมือนระบบประสาทของเครื่องคอมพิวเตอร์ ทำหน้าที่รับการติดต่อประสานงานกับภายนอกระบบ และแสดงผลที่ได้ออกมา เช่น คีย์บอร์ด จอภาพ ลำโพง อุปกรณ์บันทึกข้อมูลชนิดต่างๆ เป็นต้น โดยปกติแล้วเรามักจะแบ่งความหรูหราของเครื่องคอมพิวเตอร์โดยดูจากหน่วยนำข้อมูลเข้าและหน่วยแสดงผลเป็นส่วนใหญ่ เช่น ขนาดของจอภาพ ความจุของอุปกรณ์บันทึกข้อมูลชนิดต่างๆ เป็นต้น

1.4 ระบบซอฟต์แวร์ของคอมพิวเตอร์

ชุดคำสั่ง (software) เป็นเหมือนหัวใจ ในการสั่งงานให้คอมพิวเตอร์ทำตามสิ่งที่เขียนลงในชุดคำสั่ง (software) นั้น ๆ ซึ่งโดยรวมแล้วราคาของระบบคอมพิวเตอร์ที่ใช้งานอยู่ในปัจจุบันจะเป็นราคาของชุดคำสั่ง (software) เป็นส่วนใหญ่ ซึ่งความหมายโดยทั่วไปแล้ว ชุดคำสั่ง (software) คือ กลุ่มของคำสั่งที่กำหนดการให้คอมพิวเตอร์ทำงานอย่างใดอย่างหนึ่งเพื่อให้บรรลุเป้าหมายตามวัตถุประสงค์ที่ต้องการ" การพัฒนาระบบชุดคำสั่ง (software) ให้กับคอมพิวเตอร์จะมีการกำหนดประเภทของชุดคำสั่ง (software) ออกเป็น 3 ประเภท ดังนี้

1.4.1 ซอฟต์แวร์ระบบ (System Software) เป็นชุดคำสั่ง (software) ส่วนที่ทำหน้าที่ควบคุมการทำงานของฮาร์ดแวร์ (hardware) โดยตรงเพื่อทำให้ระบบคอมพิวเตอร์ทำงานได้เกิดประสิทธิภาพสูงที่สุด ทำหน้าที่ให้บริการการสั่งงานฮาร์ดแวร์ ประสานการทำงานระหว่างโปรแกรมชุดคำสั่ง (software) ประเภทอื่น ๆ อำนวยความสะดวกในเรื่องของการรวมรายละเอียดการสั่งงานที่ซับซ้อนให้เหลือเพียงการสั่งงานง่าย ๆ เช่น การอ่านไฟล์จากจานบันทึกข้อมูล (disk drive) การจัดการไฟล์ (file management) การจัดการผู้ใช้ (user account) การป้องกันและรักษาความปลอดภัย (protection and security) เป็นต้น

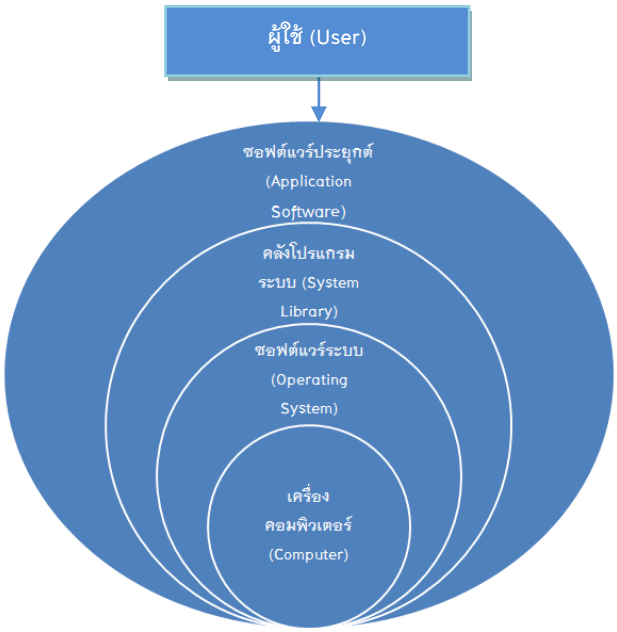
1.4.2 คลังโปรแกรมระบบ (System Library) เป็นส่วนที่รวบรวมการชุดคำสั่งการทำงานที่มีความซับซ้อน เพิ่มขึ้นมาจากชุดคำสั่งระบบ (System Software) ทำหน้าที่ให้บริการการทำงานกับชุดคำสั่งประยุกต์ (application software) เช่น ในซอฟต์แวร์ระบบ

อาจจะมีคำสั่งแค่อ่านเขียนไฟล์ แต่ในคลังโปรแกรมระบบอาจจะมีคำสั่งที่จัดการกับไฟล์หลาย ๆ ไฟล์ในรูปแบบของฐานข้อมูล เป็นต้น (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

1.4.3 ซอฟต์แวร์ประยุกต์ (Application Software) เป็นชุดคำสั่ง (software) ที่พัฒนาขึ้นมาเพื่อใช้กับงานกับงานด้านใดด้านหนึ่งโดยเฉพาะ ชุดคำสั่งประยุกต์ (application software) อาจจะเป็นโปรแกรมชุดคำสั่ง (software) ที่พัฒนาขึ้นมาเองหรือซื้อมาในรูปแบบโปรแกรมสำเร็จรูป (application program) ก็ได้ ซึ่งในปัจจุบันมีโปรแกรมสำเร็จรูป (application program) แบ่งตามกลุ่มตามลักษณะการที่ใช้งาน เช่น ด้านการประมวลผลคำ ด้านการวิเคราะห์ข้อมูล หรือตารางทำงาน ด้านการเก็บและเลือกค้นข้อมูลเป็นระบบฐานข้อมูล ด้านกราฟิก และนำเสนอข้อมูลด้าน การติดต่อสื่อสารทางไกล ด้านการพิมพ์ตั้งโต๊ะ ด้านการลงทุนและจัดการการเงิน ด้านวิทยาศาสตร์และวิศวกรรม ด้านการจำลอง เกมและการตัดสินใจ เป็นต้น

ความสัมพันธ์กันของส่วนประกอบต่าง ๆ

ความสัมพันธ์กันระหว่างฮาร์ดแวร์และซอฟต์แวร์ประเภทต่าง ๆ สามารถที่จะเขียนให้อยู่ในรูปของแผนภูมิหัวหอม แสดงได้ดังภาพที่ 1.2 โดยส่วนของฮาร์ดแวร์จะอยู่ชั้นในสุด ถัดมาจะเป็นส่วนของคลังโปรแกรมระบบ (System Library) และซอฟต์แวร์ประยุกต์ตามลำดับ โดยระบบในชั้นนอกจะเรียกใช้การบริการของระบบในชั้นในกว่าลงไปและคอยให้บริการแก่ระบบในชั้นที่อยู่ถัดออกมาจากตัวเอง (ธีรวัฒน์ ประกอบผล, 2537)



ภาพที่ 1.21 แสดงแผนภูมิหัวหอม

สรุป

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

คำถามทบทวน

1. จงอธิบายส่วนประกอบของระบบคอมพิวเตอร์ว่าควรมีส่วนประกอบอย่างน้อยที่สุดอะไรบ้าง
2. ชนิดของคอมพิวเตอร์ สามารถแบ่งได้ตามขนาดและการใช้งานของคอมพิวเตอร์ได้เป็นอย่างไร
3. จงอธิบายความแตกต่างระหว่างซุปเปอร์คอมพิวเตอร์และเมนเฟรมคอมพิวเตอร์
4. จงอธิบายความแตกต่างระหว่าง มินิคอมพิวเตอร์ (Minicomputer) คอมพิวเตอร์ส่วนบุคคล (Personal Computer) และโน๊ตบุค (Notebook) ว่ามีความเหมือนหรือแตกต่างกันอย่างไร
5. จงอธิบายความแตกต่างระหว่างเทปเล็ต (Tablet PC) และพีดีเอ (PDA)
6. จงอธิบายความแตกต่างระหว่าง ROM (Read Only Memory) และ ROM (Read Access Memory)
7. ชนิดของคอมพิวเตอร์ สามารถแบ่งได้ตามขนาดและการใช้งานของคอมพิวเตอร์ได้เป็นอย่างไร
8. ถ้าเรามองโครงสร้างของคอมพิวเตอร์ในรูปแบบ Module แล้วจะเห็นส่วนประกอบต่าง ๆ ที่มีรูปแบบเหมือนกันอย่างไร จงอธิบายพร้อมยกตัวอย่างประกอบ
9. การกำหนดประเภทของซอฟต์แวร์ แบ่งเป็นกี่ประเภท อะไรบ้าง
10. แผนภูมิหัวหอม คืออะไร ประกอบด้วยสิ่งใดบ้าง

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 10 ธันวาคม 2556,จาก: http://rirs3.royin.go.th/coinages/

ซุปเปอร์คอมพิวเตอร์. (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 10 ธันวาคม 2556, จาก: http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 2

หัวข้อเรื่อง ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์
(Microprocessor and Microcontroller)

รายละเอียด

คอมพิวเตอร์มีการพัฒนาอย่างต่อเนื่องเริ่มตั้งแต่ยุคแรกที่มีการใช้หลอดสุญญากาศแทนวงจรในการคำนวณต่อมาในปี ค.ศ. 1954 ได้มีการคิดค้นการใช้ทรานซิสเตอร์แทนหลอดสุญญากาศ โดยบริษัท Taxas Instrument และในช่วงทศวรรษที่ 1950 นี้เองก็ได้มีการคิดค้นวงจรรวม (Integrated Circuit) ขึ้นมาใช้ในกับเครื่องคอมพิวเตอร์ จึงทำให้เครื่องคอมพิวเตอร์มีขนาดเล็กลงและใช้พลังงานไฟฟ้าน้อยลง มีความเร็วเพิ่มมากขึ้น และมีการพัฒนาวงจรรวมขนาดใหญ่ที่เรียกว่า Very Large Scale Integrated Circuit ทำให้สามารถรวมการประมวลผลทั้งหมดไว้ในชิพตัวเดียว

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

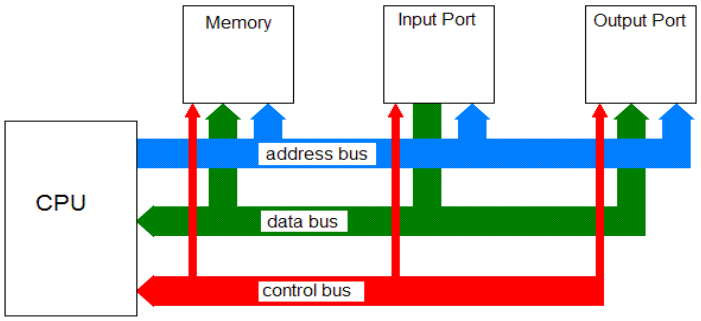
3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 2 การจัดการเรียนการสอน จะเกี่ยวข้องกับความแตกต่างระหว่างไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์ ชนิดของไมโครโพรเซสเซอร์ โครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์ การเกิดขัดจังหวะ การระบุตำแหน่งในรีจีสเตอร์และการเข้าถึงข้อมูลในหน่วยความจำ ในขณะที่ คอมพิวเตอร์มีการพัฒนาอย่างต่อเนื่องเริ่มตั้งแต่ยุคแรกที่มีการใช้หลอดสุญญากาศแทนวงจรในการคำนวณ ซึ่งในเวลาต่อมาในปี ค.ศ. 1954 ได้มีการคิดค้นการใช้ทรานซิสเตอร์แทนหลอดสุญญากาศ โดยบริษัท Taxas Instrument และในช่วงทศวรรษที่ 1950 นี้เองก็ได้มีการคิดค้นวงจรรวม (Integrated Circuit) ขึ้นมาใช้ในกับเครื่องคอมพิวเตอร์ จึงทำให้เครื่องคอมพิวเตอร์มีขนาดเล็กลงและใช้พลังงานไฟฟ้าน้อยลง มีความเร็วเพิ่มมากขึ้น และมีการพัฒนาวงจรรวมขนาดใหญ่ที่เรียกว่า Very Large Scale Integrated Circuit ทำให้สามารถรวมการประมวลผลทั้งหมดไว้ในชิพตัวเดียว เช่น ไมโครโพรเซสเซอร์ตระกูลเพนเทียม (Pentium) ของบริษัทอินเทล (Intel Corporation) และสามารถรวมไมโครโพรเซสเซอร์หลายๆ ตัวให้สามารถทำงานรวมกันได้ นอกจากนั้นจำนวนข้อมูลที่ใช้ในการประมวลผลมีการพัฒนาขึ้นอย่างต่อเนื่องจาก 8 บิต มาเป็น 32 บิต 64 บิตและ 128 บิตตามลำดับ

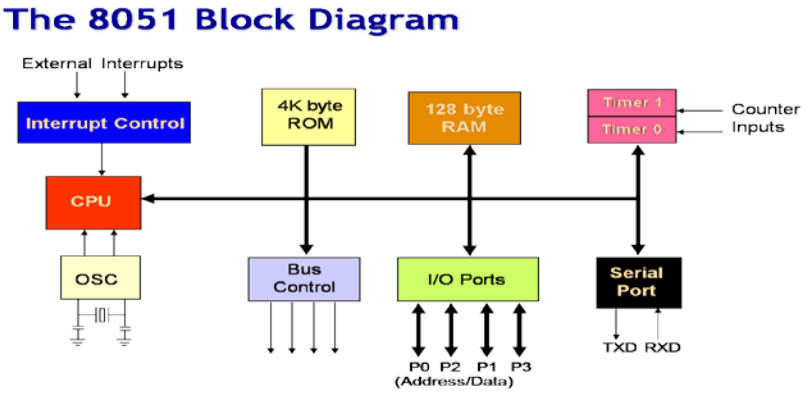
2.1 ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์

ไมโครโพรเซสเซอร์ (Microprocessor) เป็นอุปกรณ์อิเล็กทรอนิกส์ชนิดหนึ่ง ซึ่งมีลักษณะวงจรรวม (Integrated Circuit) หรือชิพ (Chip) โครงสร้างภายในจะมีวงจรรวมขนาดใหญ่ประกอบไปด้วย หน่วยคำนวณทางคณิตศาสตร์ (Arithmetic Unit) รีจิสเตอร์ (Register) บัสข้อมูล (Data Bus) บัสควบคุม (Control Bus) บัสที่อยู่ (Address Bus) รวมกันเป็นหน่วยประมวลผลกลาง (Central Processing Unit) เพื่อทำหน้าที่ในการประมวลผลตามโปรแกรมคำสั่งที่ป้อนเข้ามา แสดงได้ดังภาพที่ 2.1



ภาพที่ 2.1 แสดงบล็อกไดอะแกรมของไมโครโพรเซสเซอร์ (Block Diagram of Microprocessor)
ที่มา: (http://brainly.in/question/3843)

ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นชิพประมวลผลประเภทหนึ่งซึ่งทำหน้าตามโปรแกรมหรือชุดคำสั่งที่เขียนขึ้นมา 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 รวมอยู่ในชิพไมโครคอนโทรลเลอร์ แสดงได้ดังภาพที่ 2.2



ภาพที่ 2.2 แสดงบล็อกไดอะแกรมของโครคอนโทรลเลอร์ (Block Diagram of Microcontroller)
ที่มา: (https://aninditadhikary.wordpress.com/tag/microcontroller/)

2.2 ชนิดของไมโครโพรเซสเซอร์ จัดแบ่งตามลักษณะการใช้งานได้ 3 ประเภท

2.2.1 Dedicated or Embedded Controller เป็นไมโครโพรเซสเซอร์ที่ใช้ในอุปกรณ์อิเล็กทรอนิกส์โดยเฉพาะที่ เช่น ใช้ในการควบคุมการทำงานของเครื่องซักผ้า เตาไมโครเวฟ เครื่องคิดเลข และอื่นๆ โดยมากมักจะเรียกไมโครโพรเซสเซอร์ ประเภทนี้ว่า “ไมโครคอนโทรลเลอร์ (Micro Controller)” เช่น TMS-1000 ของบริษัท Texas Instrument สำหรับบริษัท Intel ได้เริ่มพัฒนาจาก ชิปขนาด 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 เป็นไมโครโพรเซสเซอร์ที่ได้รับการพัฒนให้สามารถนำฟังก์ชันบางอย่างที่ไม่มีในไมโครคอนโทรลเลอร์ เช่น ฟังก์ชันการทำงานแบบ Multiplexer และ Sequencer เข้ามาทำงานบนชิพประมวลผลได้ อีกทั้งยังสามารถควบคุมการทำงานด้วยโปรแกรมชุดคำสั่ง โดยเรียกโปรแกรมชุดคำสั่งสำหรับฟังก์ชันการทำงานนี้ว่า “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 ตำแหน่งที่แอดเดรสอยู่ต่อจากกัน ดังนั้น ถ้าไบต์แรกของ word ถูกจัดเก็บไว้ใน memory address เลขคู่ 8086 จะสามารถอ่านได้ภายในการปฏิบัติเพียงครั้งเดียว แต่ถ้าไบต์แรกของ 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

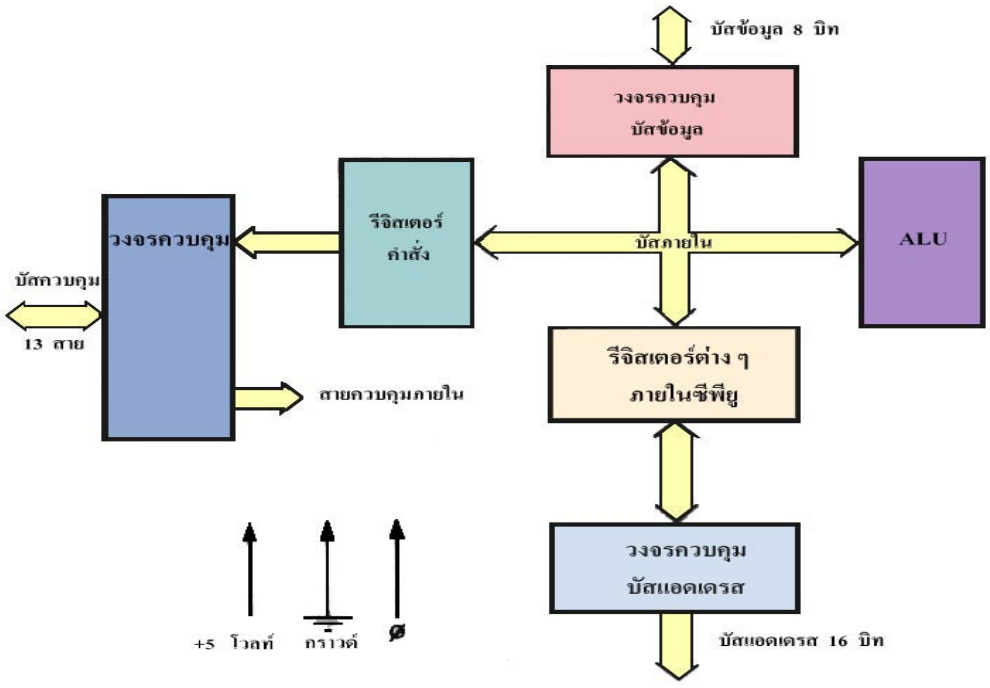
– 8088 มี address bus ขนาด 20 bits ดังนั้นจึงสามารถ address ตำแหน่งในหน่วยความจำได้สูงสุด 220 หรือ 1,048,576 ตำแหน่ง

- **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) ที่ได้รับการ ออกแบบให้ใช้เป็น 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 สำหรับการเริ่มต้นศึกษาโครงสร้างสถาปัตยกรรมของไมโครโพรเซสเซอร์จาก 8086 เพื่อให้เข้าใจได้โดยง่าย และไม่ซับซ้อนมากนักและก่อนที่จะเข้าไปถึงการเขียนโปรแกรมให้สามารถทำงานบนวิพตระกูล 8086 จะต้องทราบโครงสร้างการทำงานภายใน 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 สามารถเรียกข้อมูลได้ทีละ 1 เวิร์ด หรือ 2 ไบต์ และยังเตรียมพร้อมสำหรับการทำงานแบบหลาย CPU รวมทั้งการทำงานร่วมกับอุปกรณ์อื่นได้ด้วย แสดงได้ดังภาพที่ 2.3



ภาพที่ 2.3 แสดง Block Diagram ของ ซีพียู (CPU 8086)

ที่มา: (https://sites.google.com/site/kwantharathepsiri67/3)

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

1. BIU จะทำหน้าที่ส่งค่าแอดเดรสเพื่อไป fetch คำสั่งจากหน่วยความจำ, อ่านข้อมูลจากหน่วยความจำ และพอร์ต, และเขียนข้อมูลไปยังหน่วยความจำ และพอร์ต

2. EU ของ 8086 จะบอก location ที่จะไป fetch instruction, decode instruction และ execute instruction

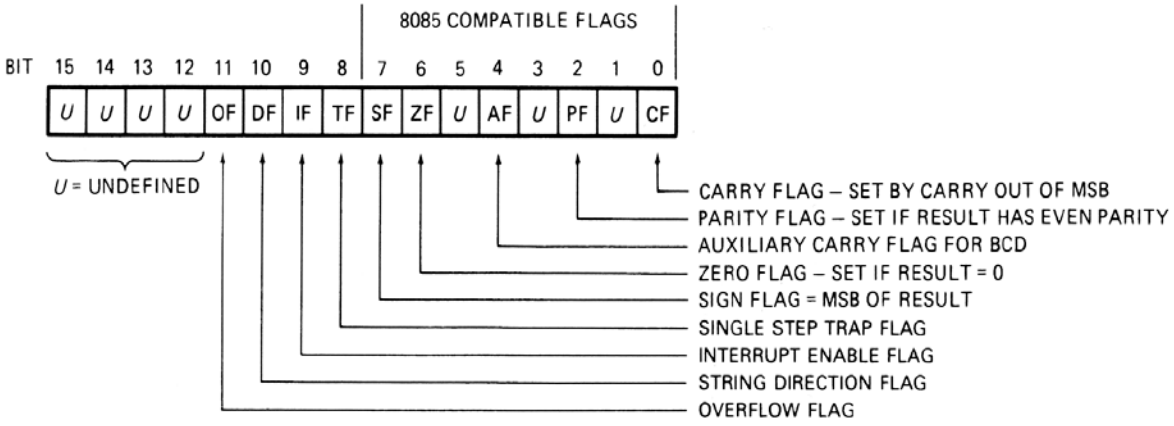
2.3.1 EU (Execution Unit) ประกอบด้วย

1. Control Circuitry ทำหน้าที่ควบคุม Operation ภายในทั้งหมด

2. Instruction Decoder ทำหน้าที่แปลความหมายของชุดคำสั่งที่นำมาจาก Memory ให้เป็นลำดับของภาษาเครื่องที่ EU สามารถปฏิบัติได้

3. ALU มีขนาด 16 bits ซึ่งมีฟังก์ชันในการ Add (บวก), Subtraction (ลบ), AND, OR, XOR, increment, decrement, complement และ shift

4. Flag Register เป็น flip–flop ที่แสดงสถานะของการ execute instruction หรือการควบคุมการทำงานของชุดคำสั่ง Flag Register ขนาด 16 บิต โดยที่ใน EU จะบรรจุ active flag 9 ตัว โดยเป็น flag แสดงสถานะ (Condition flags) จำนวน 6 บิต และ flag ควบคุม (Control flags) จำนวน 3 บิต สำหรับบิตส่วนที่เหลือก็ไม่ได้ใช้ แสดงได้ดังภาพที่ 2.4



ภาพที่ 2.4 แสดงสถานะแฟล็ก (Flag Status) ที่ใช้ในการตรวจสอบการทำงานของซีพียู (CPU 8086)
ที่มา: (http://www.cpe.ku.ac.th/~yuen/204323/gen_arch/86register.html)

4.1 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 มีตัวทดหลักเพิ่มเติมจากบิตที่มีความสำคัญสูงสุด หรือ MSB (Most Significant Bit)

1. CF (Carry Flag) จะถูก SET เมื่อมี CARRY OUT ของ MSB
2. PF (Parity Flag) จะถูก SET เมื่อผลลัพธ์มี parity เป็น EVEN
3. AF (Auxiliary Carry Flag) สำหรับ BCD
4. ZF (Zero Flag) จะถูก SET เมื่อผลลัพธ์ = 0
5. SF (Sign Flag) เป็นค่า MSB ของผลลัพธ์ (0 เป็นค่า +, 1 เป็นค่าลบ)
6. OF (Overflow Flag) จะถูก SET เมื่อผลลัพธ์เกิดการ Overflow

4.2 แฟล็กควบคุม (Control Flag) มีทั้งหมด 3 ตัว จะถูกใช้ในการควบคุมการปฏิบัติที่แน่นอนของโพรเซสเซอร์ ซึ่ง
แฟล็กเหล่านี้จะถูก SET หรือ RESET โดยเจตนาของชุดคำสั่งที่ใส่ไว้ในโปรแกรม

1. TF (Trap Flag) ถูกนำมาใช้สำหรับชุดคำสั่งทำงานในลักษณะ Single Step
2. IF (Interrupt Flag) ถูกนำมาใช้กำหนดการขัดจังหวะในชุดคำสั่งของโปรแกรม
3. DF (Direction Flag) ถูกนำมาใช้กับชุดคำสั่งที่เป็นสายอักขระ (String)

รูปแบบของแฟล็ก (flag) ได้ถูกออกแบบมาเพื่อแสดงผลของการกระทำทางคณิตศาสตร์และตรรกะ สถานะที่เป็นไปได้ของ flag จะมีได้แค่สองสถานะ คือ ถูกเซตทำให้มีค่าเป็น 1 หรือถูก รีเซตให้มีค่าเป็น 0 แฟล็กตัวทด (carry flag) จะสดงลักษณะบอกการทดหรือ การขอยืมของบิตสูงสุดของผลลัพธ์ , แฟล็กพาริตี้ (parity flag) จะใช้แสดงจำนวนคู่หรือคี่ของตัวเลขที่เป็น 1 ในรีจิสเตอร์ ถ้าเป็นคู่ flag นี้จะถูกเซตเป็น 1, แฟล็กตัวทดช่วย (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) จะกำหนดทิศทางของกลุ่มคำสั่งเกี่ยวกับสตริง ว่าจะมีทิศทางการทำงานจากแอดเดรสมากไปน้อยหรือจากแอดเดรสน้อยไปมาก

แฟล็กขัดจังหวะ (IE) ถ้าถูกเซตเป็น 1 แสดงว่าจะยอมรับขัดจังหวะจากภายนอก โดยหยุดการทำงานตามปกติของโปรแกรมไว้ก่อน และ Tap flag (TF) จะทำให้ 8086 ทำงานแบบทีละคำสั่งเพื่อการแก้ไขโปรแกรม

2.3.2 BIU (Bus Interface Unit) ประกอบด้วย

1. Queue : BIU จะจัดเก็บชุดคำสั่งที่ fetch มาจากหน่วยความจำได้สูงสุด 6 คำสั่ง โดยจะใช้เทคโนโลยีที่เรียกว่า FIFO (First–In First–Out) เมื่อ EU พร้อมที่จะทำงานกับชุดคำสั่งต่อไปก็เพียงแต่นำข้อมูลและชุดคำสั่งมาจาก Queue โดยไม่ต้องไปยุ่งเกี่ยวกับ Bus จึงสามารถทำงานได้เร็วขึ้นอีกหลายเท่า แทนที่จะต้องรอการอ่านข้อมูลและชุดคำสั่งมาจากหน่วยความจำภายนอก เว้นแต่ว่าเมื่อเมื่อพบคำสั่ง JMP (Jump) หรือ CALL ที่ต้องเรียกข้อมูลมาจากแอดเดรสที่กำหนดไว้ในโปรแกรม คุณสมบัติในการ Fetch คำสั่งต่อไปมาเก็บไว้ใน Queue ในขณะที่ EU กำลังexecute คำสั่งอื่นอยู่ เรียกว่า Pipelining

2. Segment Register : BIU จะส่งค่า address ขนาด 20 bits ออกไป ดังนั้นจึงสามารถที่จะระบุตำแหน่งใน Memory ได้ 220 = 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

1. CS (Code Segment)
2. SS (Stack Segment)
3. ES (Extra Segment)
4. 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 โดยที่ Stack เป็นส่วนของหน่วยความจำที่แยกไว้ต่างหาก เพื่อจัดเก็บค่าแอดเดรสเริ่มต้นของโปรแกรมย่อยและเก็บข้อมูลที่ใช้ในโปรแกรมย่อย ในขณะที่ทำการ execute Subprogram หรือ Procedure

2.4 การเกิดการขัดจังหวะ (Interrupt Handle)

การขัดจังหวะของ 8086 อาจจะมาจากหลาย ๆ แหล่งได้ เช่น การขัดจังหวะที่มาจากอุปกรณ์ภายนอกการขัดจังหวะทาง software หรือการขัดจังหวะที่มาจากตัว CPU 8086 เองก็ได้ การขัดจังหวะจากอุปกรณ์ภายนอกจะเข้ามาทางขา 2 ขา คือ INTR (interrupt request) และ NMI(non-maskable interrupt) โดยที่ INTR จะถูกกำหนดให้ทำงานได้โดยคำสั่ง IE (interrupt enable flag) ที่เป็นส่วนหนึ่งของ 8086 flag ถ้า flag IE ถูกเซตจะ ทำให้ 8086 สามารถจะรับการขัดจังหวะจาก INTR ได้ แต่ถ้า IF ถูกเคลียร์ 8086 จะไม่รับการขัดจังหวะจาก INTR ส่วน NMI จะเป็นการขัดจังหวะที่ไม่สามารถจะหยุดได้ มักจะใช้ในกรณีที่เกิดเหตุการณ์สำคัญ เช่น เกิดความผิดพลาดของพาริตี้หน่วยความจำ (memory parity) หรือเกิดไฟตก เป็นต้น

INTO (interrupt overflow) คือ การขัดจังหวะของ 8086 ที่เกิดจากข้อผิดพลาดของการหารและการทำงานทีละขั้น ข้อผิดพลาดจากการหาร (divide error) เกิดขึ้นเนื่องจากผลของการหารอาจจะมีค่ามากกว่าเนื้อที่ซึ่งเตรียมไว้เป็นที่เก็บผลลัพธ์ หรืออาจจะเกิดจากการหารด้วยเลข 0 ส่วนลักษณะของการทำงานทีละขั้น (single step) จะเป็นการขัดจังหวะที่เกิดเนื่องจากการ execute คำสั่งทีละคำสั่ง เหตุการณ์นี้เกิดขึ้นเนื่องจาก TF (trap flag) ในรีจิสเตอร์แฟล็กถูกเซตให้เป็น 1

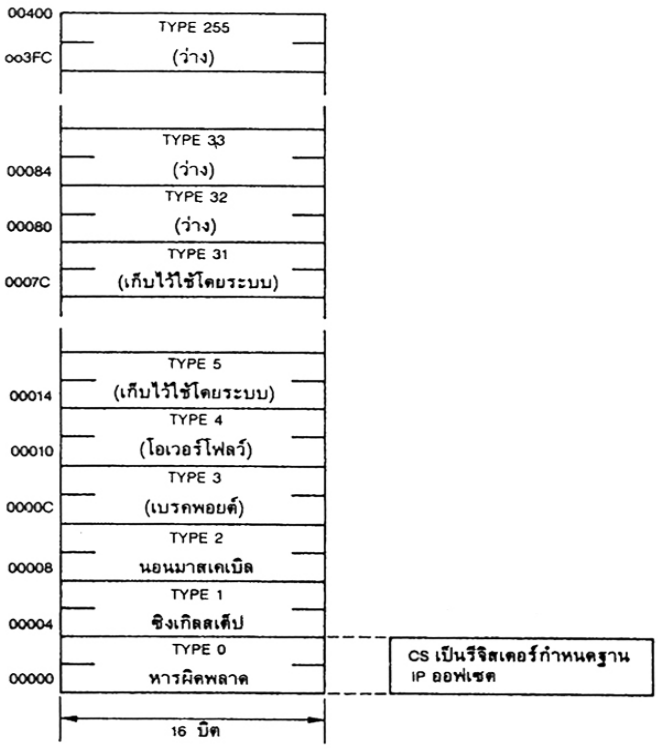
เมื่อเกิดการขัดจังหวะเหล่านี้ขึ้นมาแล้วก็จะเกิดเหตุการณ์เฉพาะอย่างขึ้นถ้ามีการขัดจังหวะจากภายนอก คำสั่งชุดสุดท้ายจะถูก execute จนเสร็จสิ้นก่อน (ยกเว้น ถ้าเป็นคำสั่ง MOV หรือ POP จะมีการ execute คำสั่งต่อไปก่อนแล้วจึงหยุด) ขัดจังหวะทุกตัวจะทำการเก็บค่าของแฟล็กรีจิสเตอร์ไว้ในสแตก เสร็จแล้ว 8086 จะทำการรีเซตแฟล็ก IF และ TF ให้เป็น 0 ทั้งนี้เพื่อป้องกัน INTR อินพุตที่เข้ามาใหม่ และป้องกันการเกิดการทำงานทีละขั้นในขณะที่อยู่ในโปรแกรมขัดจังหวะ เนื่องจากการที่แฟล็กถูกเก็บค่าไว้หลังจากโปรแกรมขัดจังหวะสิ้นสุดแล้วค่าต่าง ๆ ที่เก็บไว้จะถูกดึงกลับมาใช้งานได้ จากนั้น 8086 จะทำการเก็บค่าของ CS และ IP ลงบนสแตกเพื่อที่จะเก็บตำแหน่งของแอดเดรสส่งกลับ (return address) ในลักษณะเดียวกับคำสั่ง CALL และในขั้นตอนสุดท้าย CS และ IP จะถูกอ่านจากตาราง interrupt vector (ขึ้นกับหมายเลขการขัดจังหวะ) ตาราง interrupt vector ก็คือบริเวณของเนื้อที่ของหน่วยความจำเริ่มตั้งแต่แอดเดรส 0000 จำนวน หนึ่งกิโลไบต์ แสดงได้ดังภาพที่ 2.5

การขัดจังหวะแต่ละตัวจะชี้ไปยังตำแหน่งที่แน่นอนและสามารถที่จะกระโดดไปถึงที่กำหนดได้โดยอัตโนมัติ เมื่อเกิดการขัดจังหวะจากลักษณะของ divide error จะทำให้เกิดการขัดจังหวะ 0, single-step จะเป็นการขัดจังหวะ 1, การขัดจังหวะแบบ Non-maskable เป็นแบบการขัดจังหวะ 2 , bread-point จะเกิดการขัดจังหวะ 3 , และการเกิด over flow จะเป็นการขัดจังหวะ 4

ส่วน ขัดจังหวะ 5 ถึง 31 เตรียมไว้สำหรับ ผู้ใช้จะกำหนดเอาเองภายหลัง ส่วนการขัดจังหวะที่เหลือเราอาจจะควบคุมโดยใช้คำสั่ง INTR หรือ INT ได้

สำหรับ INTR จะตอบรับสัญญาณการขัดจังหวะจากภายนอก ส่วน INT จะรับได้โดยตัวคำสั่ง นั่นเอง CPU 8086 สามารถที่จะรับรู้ชนิดของการขัดจังหวะได้โดยการคูณค่าของคำสั่ง ด้วย 4 ผลลัพธ์ที่ได้จะบอกตำแหน่งของตารางขัดจังหวะ ซึ่งค่าของตารางจะถูกส่งค่าไปยัง CS และ IP ค่าต่าง ๆในตาราง เราจะต้องเป็นผู้กำหนดโดยกำหนดเป็นค่าของเซกเมนต์และค่าของออฟเซตที่แสดงตำแหน่งของการขัดจังหวะรูทีน เช่น การขัดจังหวะที่แอดเดรส 0–3 จะถูกต้องกำหนดเป็นค่าบอกแอดเดรสของ CS และ IP เพื่อจะชี้รูทีนของ divide error เป็นต้น

การขัดจังหวะรูทีน จะต้องเริ่มต้นด้วยการเก็บค่าของรีจิสเตอร์ต่าง ๆไว้ก่อน เพราะเราไม่ทราบว่าจะเกิดขัดจังหวะเมื่อถึงจุดใด และเราไม่ต้องการทำให้ค่ารีจิสเตอร์ต่าง ๆ เสียไปและเราต้อง ปิดท้ายรูทีนด้วยคำสั่งIRET คำสั่งนี้จะทำงานเหมือนคำสั่ง RET ธรรมดาแต่จะทำการอ่านค่าของแฟล็กออกมาจากสแตกด้วย (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)



ภาพที่ 2.5 แสดงการเกิดขัดจังหวะ ที่ตำแหน่ง 0000– 03FF ขนาด 1 กิโลไบต์ (1Kbytes)
ที่มา: (http://www.cpe.ku.ac.th/~yuen/204323/gen_arch/86interrupt.html)

2.5 รีจีสเตอร์ของซีพียู 8086 (CPU 8086 Register)

CPU 8086 ประกอบด้วยรีจิสเตอร์ที่เกี่ยวข้อง แบ่งออกเป็น 2 กลุ่ม (แสดงได้ดังภาพที่ 2.6) ดังนี้

1. กลุ่มข้อมูล (Data Group Register) แบ่งออกเป็น

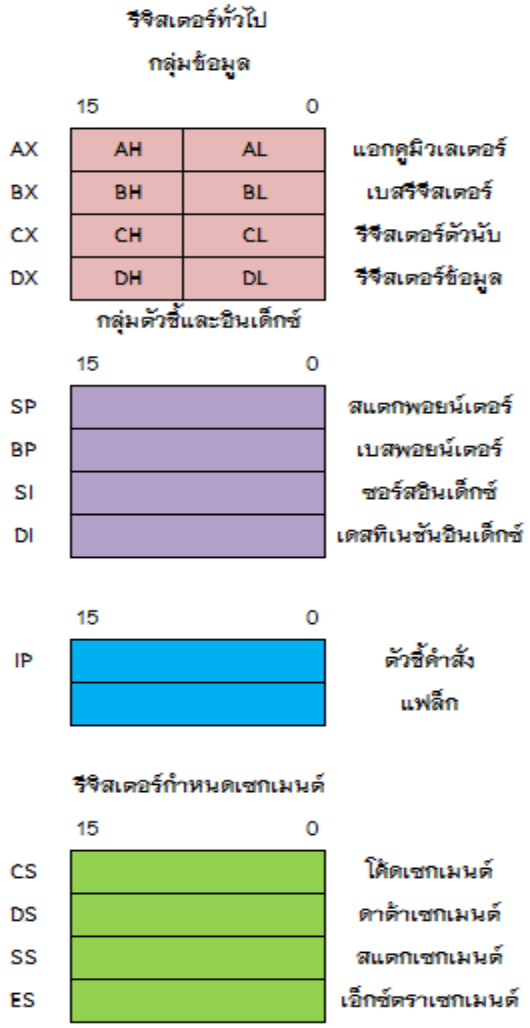
1.1 รีจีสเตอร์ทั่วไป (General–Purpose Register)

1.2 รีจิสเตอร์ตัวชี้ (Instruction Pointer Register)

1.3 แฟล็ก (flag)

2. กลุ่มกำหนดเซกเมนต์ (Segment Register) แบ่งออกเป็น

2.1 รีจิสเตอร์กำหนดเซกเมนต์ (Segment Register)



ภาพที่ 2.6 แสดงรีจีสเตอร์ที่ใช้ในไมโครโพรเซสเซอร์ 8086

ที่มา: (http://www.shsu.edu/~csc_tjm/spring2005/cs272/8086.html)

1.1 รีจีสเตอร์ทั่วไป (General–Purpose Register) รีจิสเตอร์ที่มีอยู่ทั้งหมดของ 8086 เป็นรีจิสเตอร์ขนาด 16 บิต ซึ่งกลุ่มข้อมูลจะแบ่งย่อยออกเป็น

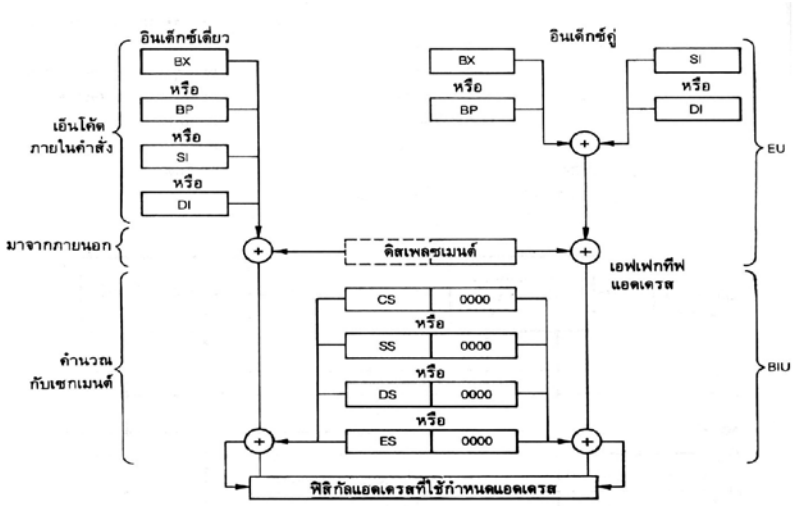
1.1.1 รีจิสเตอร์ AX (Accumulator) หรือแอกคูมิวเลเตอร์ ซึ่งจะประกอบด้วย AH และ AL โดย AX ทำหน้าที่ในการใช้คูณหารหรือเป็นรีจิสเตอร์เกี่ยวกับอินพุตเอาต์พุตที่เป็นเวิร์ดส่วน AL ทำหน้าที่ใช้ในการคำนวณ คูณหาร หรือทำงานเกี่ยวกับอินพุต (Input) เอาต์พุท (Output) แปลง ข้อมูลจัดการคำนวณแบบตัวเลขฐานสิบแบบ 8 บิต และ AH ใช้เป็นรีจิสเตอร์ในการคูณและหารได้

1.1.2 รีจิสเตอร์ BX (Base Register) หรือเบสรีจิสเตอร์ใช้ในการแปลงข้อมูล

1.1.3 รีจิสเตอร์ CX (Counter Register) หรือรีจิสเตอร์ตัวนับใช้ในคำสั่งจัดการเกี่ยวกับสตริง และการทำลูป CL ใช้เป็นตัวแปรสำหรับการเลื่อนบิตหรือหมุนบิต

1.1.4 รีจิสเตอร์ DX (Data Register) หรือรีจิสเตอร์ข้อมูลใช้ในคำสั่งคูณ หารเป็นเวิร์ด (Words) หรือใช้ในรูปแบบอ้างอินพุตเอาต์พุตแบบอ้อม แสดงได้ดังภาพที่ 2.6

1.2 รีจิสเตอร์ตัวชี้ (Instruction Pointer Register) รีจิสเตอร์ใช้งานทั่วไปในกลุ่มตัวชี้และอินเด็กซ์ประกอบด้วย SP หรือสแต็กพอยน์เตอร์ใช้กับคำสั่งสแตก สำหรับรีจิสเตอร์ต้นทางหรือ SI (Source Index) รีจิสเตอร์ ปลายทางหรือ DI (Destination Index) จะใช้กับคำสั่งที่เกี่ยวกับสตริง และเบสพอยน์เตอร์หรือ BP (base pointer) รีจิสเตอร์เหล่านี้สามารถที่จะนำมาใช้เป็นตัวอ้างถึง Physical address ได้ด้วยวิธีการต่างๆ ดังแสดงได้ดังภาพที่ 2.7

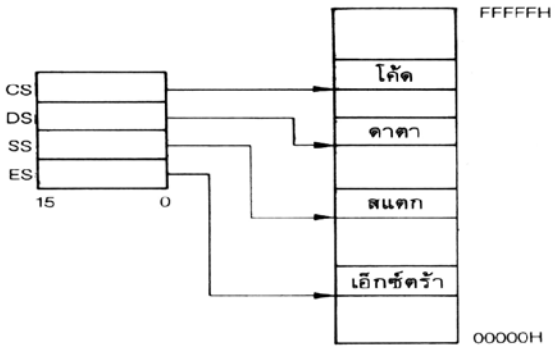


ภาพที่ 2.7 แสดงโครงสร้างรีจีสเตอร์ 8086 กับการคำนวณหาค่าตำแหน่งในหน่วยความจำ

ที่มา: (http://www.school.net.th/library/snet1/hardware/z8086/cpu8086.html)

อธิบายจากภาพที่ 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 ใช้ในการชี้ค่าตำแหน่งในสแตก (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 สตริงและการ ทำงานเป็น block หรือ การเชื่อมโยงข้อมูลในโครงสร้างแบบ array ในการ fetch คำสั่งของ 8086 จะใช้ IP (instruction pointer) ขนาด 16 บิต เป็นตัวชี้ตำแหน่ง address ของคำสั่งถัดไปที่จะถูกทำงาน ซึ่งความจริงแล้วค่าใน IP ไม่ได้กำหนดตำแหน่งโดยตรง แต่จะมีการคำนวณมาก่อน

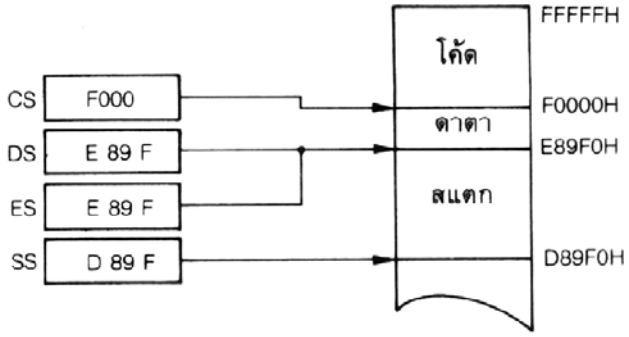
2. รีจิสเตอร์กำหนดเซกเมนต์ (Segment Register) ประกอบด้วยรีจีสเตอร์ย่อยมี 4 ตัว คือ CS (code segment), DS (data segment), SS (stack segment) และ ES (extra segment) ซึ่งจะนำมาใช้ในหลักการแบบบล็อกแอดเดรสของ 8086 หน่วยความจำ (Memory) จะได้รับการจัดสรรเป็นส่วนใหญ่ ๆ คือ code (หรือชุดคำสั่ง), ข้อมูล (ตัวเลข หรือตัวอักษร) และสแตกสำหรับการเก็บค่าแอดเดรสกลับคืนจากโปรแกรมย่อยเพื่อที่จะป้องกันการสับสน จึงกำหนดค่าให้แยกกันได้ซึ่ง 8086 จะมองลักษณะของหน่วยความจำ โดยแบ่งหน่วยความจำเป็นกลุ่ม ๆ ในรูปแบบของ เซกเมนต์ ในหนึ่งเซกเมนต์จะชี้ได้ถึง 64 กิโลไบต์ โดยเซกเมนต์รีจีสเตอร์ทั้งสี่ตัว จะแสดงแอดเดรสเริ่มต้นของหน่วยความจำที่จะติดต่อด้วย สังเกตได้จากภาพที่ 2.8 รีจีสเตอร์ CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรม, DS จะเก็บค่า Data segment ในขณะนั้น,SS จะเก็บค่า Stack segment ในขณะนั้น และ ES จะกำหนด Segment ของข้อมูลรวมที่เรียกว่า Global Data Segment แสดงได้ดังภาพที่ 2.8



ภาพที่ 2.8 แสดงการแบ่งกลุ่มของ Segment Register

ที่มา: (http://www.school.net.th/library/snet1/hardware/z8086/cpu8086.html)

เซกเมนต์จะแสดงตำแหน่งเหมือนกับ Paragraph โดยจะเลื่อนไปทางซ้าย 4 บิต เพื่อที่จะกำหนด หรืออ้างแอดเดรสให้ครบ 20 เส้น โดยจุดเริ่มต้นของ Paragraph จะต้องมี 4 บิต หลังสุดเป็น 0 เช่น เป็น 00000H, 00010H , 00020H เป็นต้น จากรูป ค่าของ SS จะชี้ตำแหน่งของเซกเมนต์ค่า D89F0 สังเกตเห็นว่าส่วนพื้นที่ ของหน่วยความจำอาจจะสลับหรือใช้บริเวณเดียวกันของหน่วยความจำได้ แสดงได้ดังภาพที่ 2.8



ภาพที่ 2.9 แสดงตำแหน่งเหมือนกับ Paragraph ใน Segment Register

ที่มา: (http://www.school.net.th/library/snet1/hardware/z8086/cpu8086.html)

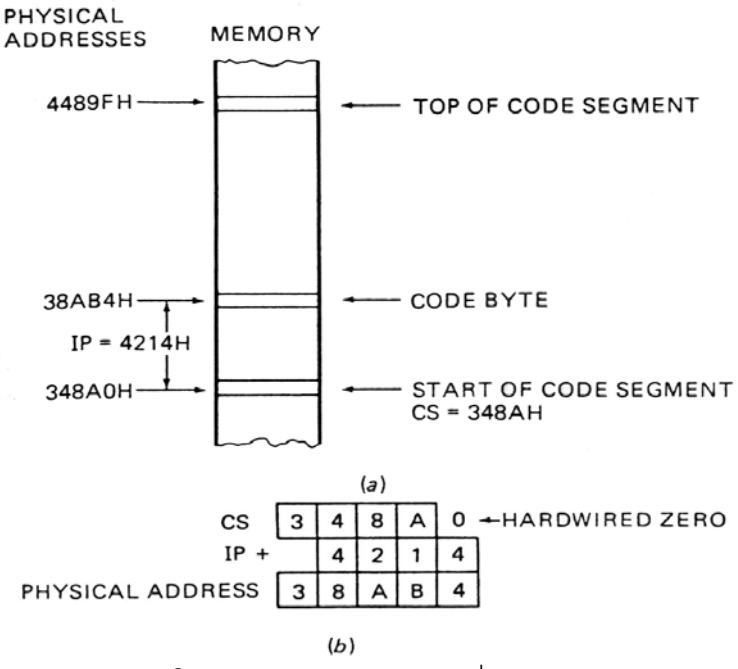
เพื่อที่จะทำการติดต่อกับข้อมูลหนึ่งไบต์หรือหนึ่งเวิร์ดนั้น 8086 ได้เตรียมค่าออฟเซต เพื่อใช้อ้างตำแหน่งตั้งแต่จุดเริ่มต้นของเซกเมนต์แอดเดรส ตำแหน่งใด ๆ จะได้มาจากการบวกค่าเซกเมนต์ รีจิสเตอร์กับค่าของออฟเซต 16 บิต เช่นถ้าเซกเมนต์มีค่า E89F จะให้ออฟเซตมีค่า 0003H จะทำให้การอ้างแอดเดรสไปที่ 89F3 การจะใช้เซกเมนตุรีจิสเตอร์และค่าออฟเซตตัวใดนั้นจะขึ้นอยู่กับชนิดของคำสั่งด้วย ดังแสดงในตารางที่ 2.1

ตารางที่ 2.1 แสดงชนิดของการอ้างอิงหน่วยความจำในเซกเมนต์รีจีสเตอร์

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

2.6 การระบุตำแหน่งในรีจีสเตอร์ (Addressing Data in Register)

การเข้าถึงข้อมูลในรีจิสเตอร์นับว่าเป็นหัวใจสำคัญในการเขียนโปรแกรมควบคุมการทำงานของไมโครโพรเซสเซอร์ 8086 สามารถ access เข้าไปในส่วนของโปรแกรม โดยใช้ค่าในรีจิสเตอร์ โดย **CS + IP –> physical address (20 bits)** แสดงได้ดังภาพที่ 2.10



ภาพที่ 2.10 แสดงการบวกค่าในรีจีสเตอร์ CS และ IP เพื่อคำนวณหา Physical Address
ที่มา: (http://www.school.net.th/library/snet1/hardware/z8086/cpu8086.html)

ในการ Access เข้าไปยังข้อมูล เพื่อนำมาใช้ในการทำงาน เรียกว่า “addressing mode” ซึ่งในภาษาแอสแซมบลี ใช้คำสั่ง MOV: format –> MOV destination, source เมื่อ

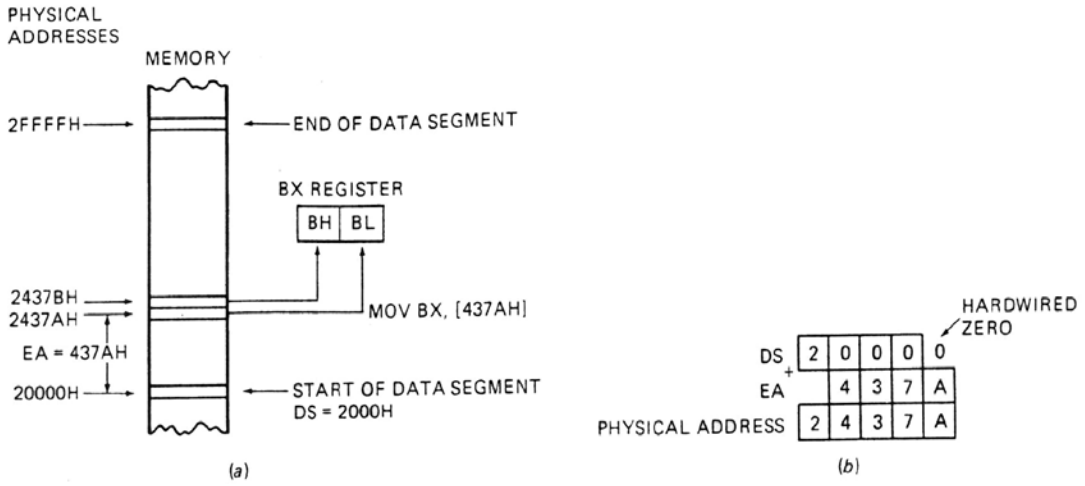
ชุดคำสั่งนี้ถูก execute 8086 จะทำการคัดลอก word หรือ byte จากตำแหน่งต้นทางไปยังจุดหมายปลายทาง Addressing Mode ใช้ในการระบุตำแหน่งของ operand ใน Memory ซึ่งสามารถแบ่งวิธีการเข้าถึงหน่วยความจำได้ดังนี้ คือ

2.6.1 Immediate Addressing Mode เป็นวิธีการอย่างง่ายในการกำหนดแอดเดรสในรีจิสเตอร์ กล่าวคือสามารถกำหนด operand ซึ่งเป็นค่าแอดเดรสให้กับชุดคำสั่งภาษาแอสแซมบลีได้โดยตรง ตัวอย่างเช่น ถ้าโปรแกรมต้องการที่จะใส่ค่า 437B (Hex) ลงใน CX Register ก็สามารถใช้คำสั่ง MOV CX, 437B (Hex) ได้โดยตรง

2.6.2 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 จะไม่ทราบว่าผู้เขียนโปรแกรมต้องการที่จะนำข้อมูลสำเนาไปเก็บไว้ในครึ่งใดของรีจิสเตอร์ CX ในกรณีเช่นนี้ โดยทั่วไป Assembler จะตรวจสอบพบ และจะบอกให้ทราบว่า Type error ดังนั้นการที่จะคัดลอกไบต์จาก AL ไปเก็บยังไบต์สูงของ CX จะใช้คำสั่ง MOV CH, AL และหากต้องการคัดลอกไบต์จาก AL ไปเก็บยังไบต์ต่ำของ CX จะใช้คำสั่ง MOV CL, AL

2.7 การเข้าถึงข้อมูลในหน่วยความจำ (Addressing Data in Memory)

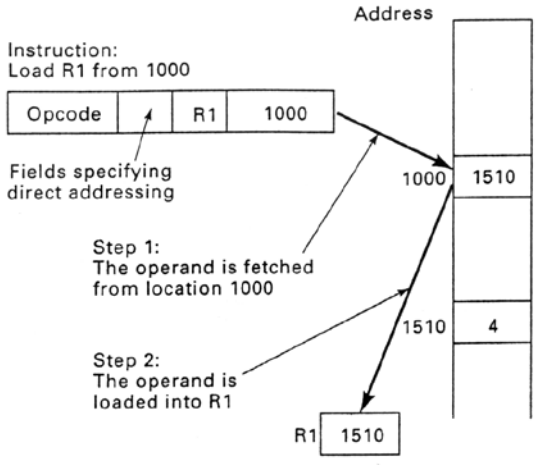
ที่ผ่านมาได้กล่าวถึง Addressing Mode ที่จะระบุตำแหน่งของ operand ในหน่วยความจำ ในกรณีที่ต้องการที่จะเข้าถึงข้อมูลในหน่วยความจำ 8086 จะต้องสร้าง physical address (20 bits) ซึ่งสามารถทำได้โดยการรวม Effective address (16 bits) เข้ากับ Segment Base Address (16 bits) แสดงได้ดังภาพที่ 2.11



ภาพที่ 2.11 แสดงการบวกค่าในรีจีสเตอร์ DS กับ EA เพื่อหา Physical Address
ที่มา: (http://www.school.net.th/library/snet1/hardware/z8086/cpu8086.html)

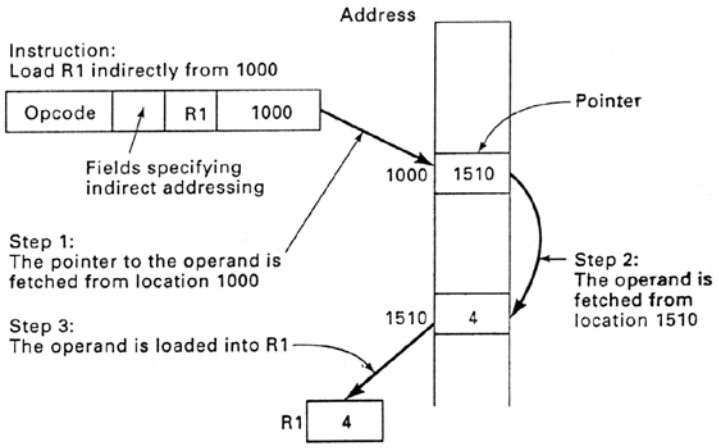
วิธีการเข้าถึงข้อมูลในหน่วยความจำในไมโครโพรเซสเซอร์ 8086 แบ่งได้เป็น 4 วิธี คือ

2.7.1 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 จะมีการทำงานแสดงได้ดังภาพที่ 2.12



ภาพที่ 2.12 แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยตรง (Direct Addressing Mode)
ที่มา: (http://www.school.net.th/library/snet1/hardware/z8086/cpu8086.html)

2.7.2 วิธีการเข้าถึงข้อมูลในหน่วยความจำโดยอ้อม (Indirect Addressing Mode) เป็นวิธีการเข้าถึงข้อมูลในหน่วยความจำที่แตกต่างจาก Direct Addressing ตรงที่แทนที่จะใส่ค่า effective address (16 bits) เข้าไปในชุดคำสั่งโดยตรง แต่จะใส่ค่า Address ของหน่วยความจำในตำแหน่งที่จัดเก็บ effective address เข้าไปในชุดคำสั่ง จะเห็นได้ว่าวิธีการนี้ค่าในแอดเดรสที่ระบุจะเป็นแอดเดรสที่บรรจุค่าที่ต้องการไว้ เราเรียกแอดเดรสภายในแอดเดรสที่ระบุว่า **“ตัวชี้ (Pointer)”** เช่นถ้าต้องการ Load รีจิสเตอร์ R1 ด้วยค่าในแอดเดรสที่แอดเดรส 1000 กำหนดไว้ จะมีการทำงาน แสดงได้ดังภาพที่ 2.13



ภาพที่ 2.13 แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยอ้อม (Indirect Addressing Mode)
ที่มา: (http://www.school.net.th/library/snet1/hardware/z8086/cpu8086.html)

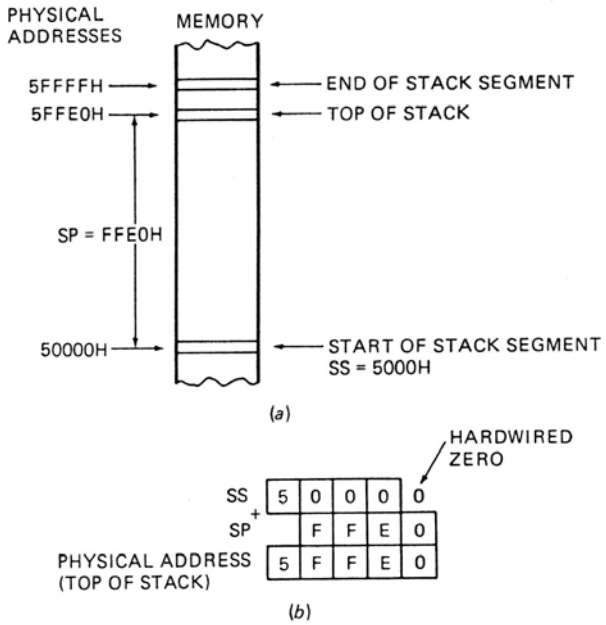
2.7.3 วิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้ตัวชี้ (Indexing Addressing Mode) บางครั้งในการทำงานซึ่งต้องการกระทำบางอย่างกับชุดข้อมูลเป็นจำนวนมากที่มีโครงสร้างการจัดเก็บไว้เป็นลำดับ อาจจะสามารถทำได้โดยใช้วิธี Indirect Addressing แต่อย่างไรก็ตามหากต้องการ Load ค่า 2 ค่าจาก 2 แอดเดรส เข้ามาในรีจิสเตอร์ตัวเดียวกัน 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 1เพื่อให้สามารถจัดการกับชุดข้อมูลที่มีโครงสร้างการจัดเก็บเรียงเป็นลำดับได้โดยอัตโนมัติ จึงทำให้มีความสามารถในการ Address โดยอัตโนมัติเรียกว่า **“autoindexing”**

2.7.4 วิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้สแต็ก (Stack Addressing Mode) เป็นวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้สแต็ก (Stack Addressing) ซึ่งสแต็กประกอบด้วยข้อมูลที่ถูกจัดเก็บเรียงเป็นลำดับไว้ในหน่วยความจำ โดยข้อมูลแรกจะถูกใส่ (Push) เข้าไปในสแต็ก และจะลงไปอยู่ที่ส่วนล่างสุดของสแต็ก ข้อมูลต่อๆ ไปก็จะถูกนำไปจัดเก็บในแอดเดรสถัดมา การที่จะเข้าไปเรียกข้อมูลในสแต็กจึงจำเป็นต้องมีแอดเดรสที่ระบุว่าเป็นส่วนบนสุดของสแต็ก ซึ่งเรียกว่า **“Stack Pointer”** กระบวนการเข้าไปยังสแต็กสามารถทำได้โดยนำค่า

SS (Stack Segment) + SP (Stack Pointer) –> Physical address (Stack’s top address)

แสดงได้ดังภาพที่ 2.14



ภาพที่ 2.14 แสดงวิธีการเข้าถึงข้อมูลในหน่วยความจำโดยใช้สแต็ก (Stack Addressing Mode)
ที่มา: (https://www.google.co.th/search?q= StacktAddressingMode&source)

สรุป

ไมโครโพรเซสเซอร์ (microprocessor) เป็นอุปกรณ์อิเล็กทรอนิกส์ชนิดหนึ่งซึ่งมีลักษณะวงจรรวม(Integrated Circuit) หรือชิพ (Chip) โครงสร้างภายในจะมีวงจรรวมขนาดใหญ่ ส่วนไมโครคอนโทรลเลอร์ (microcontroller) เป็นชิพประมวลผลประเภทหนึ่งซึ่งทำหน้าตามโปรแกรมหรือขุดคำสั่งที่เขียนขึ้นมา ภายในประกอบด้วยวงจรรวมขนาดใหญ่ (Very Large Scale Integrated Circuit) การนำไมโครโพรเซสเซอร์ หลายๆ ตัวให้สามารถทำงานรวมกันได้ในเวลาเดียวกัน ทำให้ประสิทธิภาพการทำงานและความเร็วนั้นเพิ่มมากขึ้น

CPU 8086 ประกอบด้วยรีจิสเตอร์ที่เกี่ยวข้อง ออกเป็น 2 กลุ่ม คือ

1. กลุ่มข้อมูล (Data Group Register) แบ่งออกเป็น
   1.1 รีจีสเตอร์ทั่วไป (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 ประกอบด้วยอะไรบ้าง
4. จงอธิบายข้อแตกต่างที่สำคัญของไมโคโพรเซสเซอร์ 8086 และ 8088
5. ถ้า Code Segment (8086) เริ่มต้นที่ 90400H ค่าที่ถูกจัดเก็บในรีจีสเตอร์ CS คือค่าใด
6. ถ้า Code Segment (8086) เริ่มต้นที่ 90400H จงหา Physical Address ของหน่วยความจำที่จะถูก Fetch เข้ามาใช้งาน ถ้าในขณะนั้นรีจีสเตอร์ IP เก็บค่า 0345CH
7. ถ้า Stack Segment Register มีค่า 0500H และ Stack Pointer Register มีค่า 8225H จงหา Physical Address ส่วนบนสุดของสแต็ก (Top of Stack)
8. จงอธิบายข้อได้เปรียบของการใช้รีจีสเตอร์ในหน่วยประมวลผลกลาง (CPU) เป็นที่เก็บข้อมูลชั่วคราว แทนที่จะเก็บข้อมูลไว้ในหน่วยความจำ
9. จงอธิบายความแตกต่างระหว่างคำสั่ง MOV AX, 1234H กับ MOV AX, [1234H]

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 15 ธันวาคม 2556,จาก: http://rirs3.royin.go.th/coinages/

รีจีสเตอร์ (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 15 ธันวาคม 2556, จาก: http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 3

หัวข้อเรื่อง ระบบเลขจำนวน (Numeric System)

รายละเอียด

คอมพิวเตอร์เป็นเครื่องจักรที่กลไกการทำงานพื้นฐานเป็นสองสถานะ (Binary) คือ เปิดวงจรกับปิดวงจร ซึ่งสามารถแทนสถานะดังกล่าวได้ด้วยตัวเลขโดดสองตัวคือ 0 กับ 1 ข้อมูลแบบอื่นของคอมพิวเตอร์จะเกิดจากการประกอบรวมกันของเลข 0 กับ 1 เท่านั้น โดยเรียกระบบเลขจำนวนที่ประกอบด้วยตัวเลข 0 กับ 1 เท่านั้นว่า “เลขฐาน 2”

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 3 การจัดการเรียนการสอน จะเกี่ยวข้องกับความหมายของตัวเลขในหลักต่าง ๆ การแปลงค่าจากเลขฐานสิบเป็นเลขฐานสองการบวกและการลบเลขฐานสอง การคูณและการหารเลขฐานสอง เลขฐานแปดและเลขฐานสิบหก บิต, ไบต์, เวิร์ด และดับเบิ้ลเวิร์ด ระบบเลข 2’ Complement ระบบเล BCD (Binary Code Decimal) และมาตรฐาน IEEE 754 ซึ่งคอมพิวเตอร์เป็นเครื่องจักรที่มีกลไกการทำงานพื้นฐานเป็นสองสถานะ (Binary) คือเปิดวงจรกับปิดวงจร ซึ่งสามารถแทนสถานะดังกล่าวได้ด้วยตัวเลขโดดสองตัวคือ 0 กับ 1 ข้อมูลแบบอื่นของคอมพิวเตอร์จะเกิดจากการประกอบรวมกันของเลข 0 กับ 1 เท่านั้น โดยเรียกระบบเลขจำนวนที่ประกอบด้วยตัวเลข 0 กับ 1 เท่านั้นว่า “เลขฐาน 2” ส่วนการนับของมนุษย์โดยปกตินั้น 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 ก็คือค่าของเลขโดดนั้นคูณด้วยสิบยกกำลังของตำแหน่งนั้น เช่น 12345 หมายความว่า ค่า 5 อยู่ในตำแหน่งหลักหน่วยซึ่งค่าของสิบยกกำลังของหลักหน่วยคือ 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

$$12345 = (1 \times 10^4) + (2 \times 10^3) + (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0)$$
$$= 10000 + 2000 + 300 + 40 + 5$$

จะเห็นว่าเลขกำลังของสิบจะเริ่มต้นจากศูนย์ที่หลักหน่วย แล้วเพิ่มขึ้นหนึ่งทุกครั้งในหลักถัดมาทางด้านซ้ายมือ ในกรณีที่เลขเป็นจำนวนทศนิยม ให้เริ่มกำลังศูนย์ที่หลักหน่วย แล้วลดกำลังลงหนึ่งทุกครั้งในหลักถัดไปทางด้านขวามือ ส่วนทางด้านซ้ายมือก็จะเป็นไปในรูปแบบเดิม เช่น 12.34 จะสามารถเขียนได้เป็น

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

สามารถใช้หลักการเดียวกันนี้กับเลขฐานสองเพื่อหาค่าของจำนวนดังกล่าวในรูปของเลขฐานสิบ (ในความเป็นจริงแล้วสามารถที่จะนำไปใช้ได้กับเลขทุกฐาน) เช่น 1011.01_2 จะเขียนได้เป็น

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

3.2 การแปลงค่าจากเลขฐานสิบเป็นเลขฐานสอง

การแปลงเลขฐานสิบเป็นฐานสองจะมีขั้นตอนอยู่สองขั้นตอนคือ การแปลงเลขส่วนที่อยู่หน้าทศนิยมและการแปลงเลขส่วนที่อยู่หลังทศนิยม

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

ตัวอย่าง จงเปลี่ยนค่า 13_{10} ให้เป็นเลขฐานสอง

$2\overline{)13}$

6 เศษ 1

3 เศษ 0

1 เศษ 1

0 เศษ 1

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

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

ตัวอย่าง จงเปลี่ยนค่า 0.25_{10} ให้เป็นเลขฐานสอง

0.25

x 2

0.50

x 2

1.00

.01

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

ดังนั้น จากตัวอย่างข้างต้นสามารถสรุปได้ว่า $13.25_{10} = 1101.01_2$

3.3 การบวกและการลบเลขฐานสอง

การบวกเลขฐานสองมีหลักการเหมือนกับการบวกเลขฐานสิบ การบวกเลขในฐานสิบนั้นเมื่อผลบวกในหลักใดมีค่ามากกว่า 9 ก็จะต้องมีการทดเลข 1 ไปยังหลักถัดไป ซึ่งหลักเกณฑ์การทดเลขนี้ยังสามารถใช้ได้กับเลขฐานสอง เพียงแต่ว่าเลขโดดที่สูงที่สุดของ

เลขฐานสองคือ 1 ดังนั้นถ้าผลบวกมีค่าเกิน 1 ก็จะมีการทดไปยังหลักถัดไปทางซ้าย รูปแบบการบวกเป็นดังนี้

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 ทด 1

ตัวอย่าง จงบวกเลข 1011.101_2 กับ 110.011_2

1 0 1 1 . 1 0 1

\+ 1 1 0 . 0 1 1

1 0 0 1 0 . 0 0 0

การลบเลขเป็นการดำเนินการที่ผกผันกับการบวก ในการลบ ถ้ามีการลบเลขที่มากกว่าจากเลขที่น้อยกว่า ต้องมีการขอยืมจากเลขในหลักถัดไปทางซ้ายมา 1 รูปแบบการลบเป็นดังนี้

0 – 0 = 0

0 – 1 = 1 ขอยืม 1

1 – 0 = 1

1 – 1 = 0

ตัวอย่าง จงลบเลข 1001.11 กับ 101.1

1 0 0 1 . 1 1

– 1 0 1 . 1 0

1 0 0 . 0 1

3.4 การคูณและการหารเลขฐานสอง

การคูณและการหารของเลขฐานสอง ก็มีหลักการเช่นเดียวกับเลขฐานสิบ เพียงแต่มีสูตรคูณแค่แม่ 0 กับ 1 เท่านั้น อีกทั้งการหารด้วยศูนย์ก็ไม่มีความหมายเช่นเดียวกับการหารในระบบเลขฐานสิบ ตารางการคูณและการหารของระบบเลขฐานสองคือ

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1

0 ÷ 1 = 0

1 ÷ 1 = 1

ตัวอย่าง จงคูณเลขฐานสอง 1.01 x 10.1

$$
\begin{array}{r}
1\ .\ 0\ 1 \\
\times\quad 1\ 0\ .\ 1\ 0 \\
\hline
1\ 0\ 1 \\
0\ 0\ 0 \\
1\ 0\ 1 \\
\hline
1\ 1\ .\ 0\ 0\ 1
\end{array}
$$

ตัวอย่าง จงหารเลขฐานสอง 11001 ÷ 101

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

3.5 เลขฐานแปดและเลขฐานสิบหก

ถึงแม้ว่าระบบคอมพิวเตอร์จะเข้าใจแต่ระบบเลขฐานสองเพียงอย่างเดียว แต่ในทางปฏิบัติจะเกิดปัญหากับผู้ใช้งานคอมพิวเตอร์เป็นอย่างมาก เนื่องจากระบบเลขฐานสองมีจำนวนเลขโดดน้อยจึงต้องมีจำนวนหลักมากขึ้นเพื่อแทนค่าตัวเลขต่าง ๆ ทำให้จดจำได้ยาก จึงมีความพยายามในการรวมหลักของเลขฐานสองหลาย ๆ หลักเข้าด้วยกันเป็นเลขฐานที่ใหญ่ขึ้น เพื่อให้ง่ายต่อการจดจำ ซึ่งการแปลงเลขฐานที่ได้นี้กลับเป็นเลขฐานสองจะทำได้อย่างง่ายดาย เนื่องจากแต่ละหลักของเลขฐานดังกล่าวแทนเลขฐานสองที่มีจำนวนหลักแน่นอน

โดยปกติแล้วมักจะรวมเลขฐานสองจำนวนสามหรือสี่หลักเป็นเลขฐานใหม่ เมื่อเรารวมเลขฐานสอง 3 หลักจะได้เลขที่มี 8 รูปแบบแตกต่างกันคือ 000, 001, 010, 011, 100, 101, 110, และ 111 และใช้ตัวเลขโดดตั้งแต่เลข 0 ถึงเลข 7 แทนเลขฐานสองในแต่ละแบบได้ ซึ่งระบบเลขจำนวนที่มีตัวเลขโดด 8 ตัวก็คือเลขฐานแปดนั่นเอง โดยที่เลขโดดแต่ละเลขของระบบเลขฐานแปดจะแทนรูปแบบของเลขฐานสองจำนวนสามหลักที่มีค่าเท่ากัน

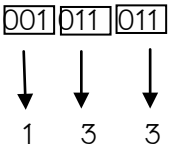
ในกรณีที่มีเลขฐานสองจำนวนสี่หลัก ก็จะได้เลขฐานสองที่มี 16 รูปแบบแตกต่างกัน ซึ่งสามารถใช้เลขฐานสิบหกแทนแต่ละรูปแบบของเลขฐานสองได้ แต่เนื่องจากเรามีเลขโดดใช้งานกันแค่สิบตัว ดังนั้นจึงมีการนำตัว A – F มาแทนค่าเลขโดดที่มีค่า 10 – 15 แทน ทำให้เลขฐานสิบหกมีเลขโดดคือ 0 – 9 และ A – F

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

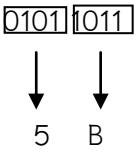
ในการแปลงเลขฐานสองให้เป็นเลขฐานแปดหรือฐานสิบหกนั้น เราจะต้องจับกลุ่มเลขฐานสองให้ได้จำนวนหลักตามที่เลขโดดของฐานที่เราจะแปลงไป เช่น จับกลุ่มสามหลักสำหรับการแปลงเป็นฐานแปด เป็นต้น การจับกลุ่มจะเริ่มจับจากหลักทางด้านขวามือสุดก่อน ในกรณีที่เหลือเลขไม่ครบจำนวนหลักที่ต้องการให้เติมศูนย์ไปทางด้านซ้ายมือเรื่อย ๆ จนกว่าจะได้จำนวนหลักที่ต้องใช้ จากนั้นจึงแปลงฐานเลขโดยแปลงทีละกลุ่มก็จะได้เลขฐานแปดหรือฐานสิบหกตามต้องการ

ตัวอย่าง จงแปลงเลขฐานสอง 1011011 ให้เป็นเลขฐานแปดและฐานสิบหก

แปลงเป็นเลขฐานแปด



แปลงเป็นเลขฐานสิบหก



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

3.6 บิต, ไบต์, เวิร์ด และดับเบิ้ลเวิร์ด

ข้อมูลต่าง ๆ ในระบบคอมพิวเตอร์มักจะไม่ได้มาจากเลขฐานสองเพียงแค่หลักเดียว เพื่อความสะดวกในการเรียกและความกะทัดรัดของจำนวนที่จะต้องใช้เรียก (คงไม่สะดวกนักถ้าต้องมีการเรียกกันว่า เลขฐานสอง 32 หลัก) ดังนั้นจึงได้มีการตั้งชื่อเฉพาะเรียกกลุ่มของเลขฐานสองที่มีจำนวนหลักตั้งแต่หนึ่งหลักขึ้น โดยชื่อต่าง ๆ มีดังนี้

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	“ดับเบิ้ลเวิร์ด” (double word)

3.7 ระบบเลข 2’ Complement

การคำนวณเลขฐานสองที่คุ้นเคยกัน มักจะเป็นเลขฐานสองที่มีค่าเป็นบวกอยู่เสมอ แต่ในในการใช้งานจริงจะเป็นทั้งจำนวนเป็นบวกและลบ ดังนั้นระบบคอมพิวเตอร์จึงต้องมีกลไกบางอย่างเพื่อระบุเครื่องหมายของตัวเลขต่าง ๆ ที่อยู่ในระบบ หนึ่งในวิธีการระบุเครื่องหมายของตัวเลขในระบบคอมพิวเตอร์ที่ได้รับความนิยมมากที่สุดก็คือการใช้วิธี 2’ Complement

วิธี 2’ Complement จะเริ่มจากการกำหนดจำนวนหลักสูงสุดของตัวเลข (จำนวนบิตสูงสุด) กระบวนการทางคณิตศาสตร์ใด ๆ ก็ตาม ถ้าทำให้เกิดการทดเลขเลยบิตซ้ายสุดที่กำหนดเลขทดดังกล่าวจะหายไป 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 คือบิตที่ถูกกำหนดให้เป็นบิตนัยสำคัญสูงที่สุดจะเป็นบิตที่บอกเครื่องหมายของตัวเลข เช่น กำหนดให้ใช้ตัวเลขขนาด 6 บิต ดังนั้นบิดที่ 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 ถ้า**เป็นการคูณกันระหว่างตัวเลขขนาด 1 ไบต์ด้วยกัน ผลลัพธ์ที่ได้จะต้องใช้ที่เก็บขนาด 2 ไบต์** (ดูตัวอย่างในเรื่องการคูณเลขฐานสอง) ซึ่งมากเกินกว่าที่จะเก็บได้ ดังนั้นถ้าต้องการเก็บผลลัพธ์ไว้เป็นขนาด 1 ไบต์ ก็จะต้องให้ตัวตั้งและตัวคูณมีขนาดไม่เกิน 1 นิบเบิล (4 บิต)

3.8 ระบบเลขบีซีดี (Binary Code Decimal)

ระบบเลขบีซีดี (BCD) มีชื่อเรียกอีกชื่อหนึ่งว่า Packed Decimal เป็นวิธีการหนึ่งที่จะทำให้การแปลงเลขระหว่างฐานสองกับฐานสิบง่ายขึ้นโดยใช้วิธีเดียวกับการสร้างเลขฐานสิบหก โดยจัดกลุ่มให้เลขฐานสอง 4 บิต เป็นเลขฐานสิบ 1 หลัก แต่จำนวนเลขโดดของเลขฐานสิบมีค่าน้อยกว่ารูปแบบของเลขฐานสองที่เป็นไปได้ใน 4 บิต ดังนั้นจึงมีเลขฐานสองบางรูปแบบที่ไม่ได้ถูกใช้งาน สาเหตุที่ต้องจัดกลุ่มเลขฐานสองให้เป็น 4 บิตแทนที่จะเป็นค่าอื่น ๆ เพราะจำนวนเลขโดดของเลขฐานสองมีค่าเท่ากับ 10 ซึ่งอยู่ระหว่าเลขฐานสอง 3 บิตกับ 4 บิต นั่นเอง

ปกติแล้วมักจะไม่นำเลขบีซีดี (BCD) ไปคำนวณแบบซับซ้อน มักจะนำไปบวกลบกันเท่านั้นซึ่งการบวกลบกันของเลขบีซีดี (BCD) จะต้องทำทีละ 1 นิบเบิล และต้องปรับค่าผลลัพธ์เพื่อเลี่ยงเลขฐานสองที่ไม่ได้ใช้งานด้วย (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

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 ค่าอินฟินิตี้ด้วย ซึ่งในการคำนวณเลขตามมาตรฐาน IEEE754 นั้นมีความซับซ้อนมาก ดังนั้นจึงไม่ขอกล่าวไว้ในเอกการประกอบการสอนเล่มนี้

สรุป

ระบบจำนวนเลขที่ใช้กับคอมพิวเตอร์ จะทำงานบนพื้นฐานของสองสถานะ (Binary) คือเปิดวงจรกับปิดวงจร ซึ่งสามารถแทนสถานะดังกล่าวได้ด้วยตัวเลขโดดสองตัวคือ 0 กับ 1 เราเรียกระบบเลขจำนวนที่ประกอบด้วยตัวเลข 0 กับ 1 นี้ว่า “เลขฐาน 2” ข้อมูลต่าง ๆ ในระบบคอมพิวเตอร์มักจะไม่ได้มาจากเลขฐานสองเพียงแค่หลักเดียว เพื่อความสะดวกในการเรียกและความกระทัดรัดของจำนวนที่จะต้องใช้เรียก (คงไม่สะดวกนักถ้าต้องมีการเรียกกันว่าเลขฐานสอง 32 หลัก) ดังนั้นจึงได้มีการตั้งชื่อเฉพาะเรียกกลุ่มของเลขฐานสองที่มีจำนวนหลักตั้งแต่หนึ่งหลักขึ้น โดยชื่อต่าง ๆ มีดังนี้คือ บิต, ไบต์, เวิร์ด และดับเบิ้ลเวิร์ด

คำถามทบทวน

1. จงเปลี่ยนค่า 123_{10} ให้เป็นเลขฐานสอง
2. จงเปลี่ยนค่า 0.125_{10} ให้เป็นเลขฐานสอง
3. จงคูณเลขฐานสอง 1.101 x 10.11
4. จงเปลี่ยนค่า CAD_{16} ให้เป็นเลขฐานสอง
5. จงเปลี่ยนค่า FAB_{16} ให้เป็นเลขฐานแปด
6. จงเปลี่ยนค่า $ABCDEF_{16}$ ให้เป็นเลขฐานสิบ
7. จงเปลี่ยนค่า 110111011_2 ให้เป็นเลขฐานแปดและฐานสิบหก

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 19 ธันวาคม 2556,จาก: http://rirs3.royin.go.th/coinages/

เลขฐาน (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 19 ธันวาคม 2556, จาก: http://th. wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 4

หัวข้อเรื่อง ภาษาสำหรับเขียนโปรแกรมคอมพิวเตอร์
(Language for Computer Programming)

รายละเอียด

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

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 4 การจัดการเรียนการสอน จะเกี่ยวข้องกับภาษาต่าง ๆ สำหรับคอมพิวเตอร์ ระดับของภาษาสำหรับเขียนโปรแกรมการแปลภาษาสำหรับคอมพิวเตอร์และภาษาแอสแซมบลี ซึ่งเป็นที่เข้าใจว่าเครื่องคอมพิวเตอร์สามารถทำงานได้กับระบบเลขฐานสองเพียงอย่างเดียว ซึ่งเลขฐานสองดังกล่าว ไม่ได้หมายถึงข้อมูลเท่านั้น แต่ยังหมายถึงคำสั่งต่าง ๆ ของเครื่องคอมพิวเตอร์ด้วย ดังนั้นการเขียนโปรแกรมขึ้นมาสักหนึ่งโปรแกรม ผู้เขียนจะต้องเรียบเรียงคำสั่งต่าง ๆ ให้อยู่ในรูปของลำดับของเลขฐานสองต่าง ๆ ตามแต่ผู้ที่ผลิตเครื่องคอมพิวเตอร์นั้นจะกำหนดไว้ จะเห็นว่าการเขียนโปรแกรมแบบนี้ไม่มีความสะดวกเลย จึงมีผู้คิดค้นภาษาสำหรับเขียนโปรแกรมขึ้นมาเพื่อให้ง่ายในการเขียนโปรแกรม (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

4.1 ภาษาต่าง ๆ สำหรับคอมพิวเตอร์

ในบรรดาภาษาสำหรับเขียนโปรแกรมนั้น ภาษาสำหรับการเขียนโปรแกรมด้วยเลขฐานสองหรือฐานสิบหก เป็นภาษาที่เก่าแก่ที่สุดซึ่งเครื่องคอมพิวเตอร์สามารถจะเข้าใจและปฏิบัติตามได้เลยทันที เราเรียกภาษาแบบนี้ว่า **“ภาษาเครื่อง (Machine Language)”** ซึ่งภาษา

ดังกล่าวไม่สื่อความหมายที่ชัดเจนกับมนุษย์ซึ่งเป็นผู้ที่เขียนโปรแกรม จึงมีการคิดค้นสัญลักษณ์ต่าง ๆ ขึ้นมาแทนที่ภาษาเครื่อง โดยสัญลักษณ์ที่คิดขึ้นในตอนแรกจะเป็นการแทนค่าภาษาเครื่องหนึ่งคำสั่งต่อสัญลักษณ์หนึ่งตัว เช่น 00111110 แทนว่า LD A เป็นต้น ซึ่งเรียกภาษาสัญลักษณ์แบบนี้ว่า **"ภาษานีโมนิค (Mnemonic)"** หรือ **"ภาษาแอสแซมบลีพื้นฐาน"** แต่ทว่าสัญลักษณ์ที่ได้นี้ ก็ยังยากที่จะใช้เขียนโปรแกรม จึงได้มีการสร้างสัญลักษณ์คำสั่งใหม่ให้มีความง่ายขึ้น โดยเทียบเคียงสัญลักษณ์กับภาษาอังกฤษเพื่อให้สื่อความหมายมากที่สุด จึงเกิดเป็นภาษาสำหรับเขียนโปรแกรมขึ้นมากมายหลายภาษาขึ้นอยู่กับสัญลักษณ์และรูปแบบที่ใช้ เช่น **Fortran, C, Pascal, และ Basic** เป็นต้น (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

4.2 ระดับของภาษาสำหรับเขียนโปรแกรม

เมื่อมีการสร้างสัญลักษณ์แทนภาษาเครื่อง ทำให้มีการแบ่งระดับของภาษาสำหรับเขียนโปรแกรมออกเป็นระดับต่าง ๆ สองระดับคือ ภาษาระดับสูง และภาษาระดับต่ำ

ภาษาระดับสูงเป็นภาษาที่เน้นการสื่อความหมายกับผู้เขียนโปรแกรมเป็นหลัก ทำให้ภาษาในระดับนี้มีความง่ายในการเขียนและทำความเข้าใจ ส่วนภาษาระดับต่ำเน้นที่ความง่ายที่เครื่องคอมพิวเตอร์จะนำคำสั่งต่าง ๆ ไปปฏิบัติซึ่งมีอยู่ด้วยกันแค่สองภาษาคือ ภาษาเครื่องกับภาษาแอสแซมบลี

4.3 การแปลภาษาสำหรับคอมพิวเตอร์

ภาษาสำหรับเขียนโปรแกรมที่อยู่ในรูปของสัญลักษณ์ต่าง ๆ นั้น ไม่สามารถนำไปให้เครื่องคอมพิวเตอร์ปฏิบัติได้ทันที เพราะเครื่องคอมพิวเตอร์เข้าใจแต่ภาษาเครื่องเท่านั้น ดังนั้นเราจะต้องมีขั้นตอนต่าง ๆ ในการแปลสัญลักษณ์ต่าง ๆ ให้กลายเป็นภาษาเครื่อง ซึ่งขั้นตอนการแปลจะมีสองแบบคือ

- **Interpret** เป็นการแปลแบบคำสั่งต่อคำสั่ง โดยดึงสัญลักษณ์ของภาษาระดับสูงมาทีละคำสั่งแล้วแปลคำสั่งนั้น จากนั้นจึงส่งคำสั่งที่แปลได้ไปให้เครื่องคอมพิวเตอร์ทำงานทันที เมื่อคอมพิวเตอร์ทำงานเสร็จแล้วจึง จะเริ่มแปลคำสั่งถัดไปเรื่อย ๆ โดยไม่มีการเก็บภาษาเครื่องของคำสั่งที่แปลไปแล้ว ดังนั้นถ้ามีการวนที่คำสั่งใด จะต้องแปลคำสั่งนั้นใหม่ทุกครั้งที่ทำงานผ่านคำสั่งนั้น ข้อดีของการแปลคือแต่ละคำสั่งจะผ่านขั้นตอนต่าง ๆ อย่างสมบูรณ์ในการแปลภาษา จึงสามารถพบข้อผิดพลาดได้ง่าย

- **Compile** เป็นกระบวนการแปลสัญลักษณ์ของภาษาระดับสูงเป็นภาษาเครื่องทีเดียวทั้งโปรแกรม แล้วจึงนำผลลัพธ์ที่ได้ทั้งหมดไปให้เครื่องคอมพิวเตอร์ปฏิบัติต่อเนื่องกันไปโดยไม่มีการแปลสัญลักษณ์ใด ๆ อีก ซึ่งภาษาเครื่องที่ได้อาจจะถูกเก็บไว้เพื่อนำกลับมาให้เครื่องคอมพิวเตอร์ทำงานได้อีกโดยที่ไม่ต้องผ่านกระบวนการแปลภาษาอีกครั้ง สัญลักษณ์ของภาษาระดับสูงแต่ละตัวจะถูกแปลเป็นภาษาเครื่องเพียงครั้งเดียว ดังนั้นเมื่อเครื่องคอมพิวเตอร์ทำงานตามโปรแกรมที่เขียนไว้ จึงทำงานได้เร็วกว่าการแปลแบบ Interpret แต่วิธีนี้ก็มีข้อเสียคือ ข้อผิดพลาดบางประการของโปรแกรมจะตรวจพบเมื่อเครื่องคอมพิวเตอร์ทำงานตามโปรแกรมแล้วเท่านั้น ซึ่งเมื่อมีการแก้ไขให้ถูกต้องก็จะต้องมีการแปลสัญลักษณ์ทั้งหมดใหม่อีกครั้ง
- **Assembling** จริง ๆ แล้ว Assembling เป็นการ Compile โปรแกรมแบบหนึ่ง แต่ถูกตั้งชื่อใหม่ตามชื่อของภาษาที่ถูกนำมาแปลซึ่งก็คือภาษาแอสแซมบลี การทำงานของ Assembling มักจะอยู่ในรูปการแทนค่าสัญลักษณ์ต่าง ๆ โดยเทียบกับตาราง เพราะภาษาแอสแซมบลีเป็นการแทนค่าภาษาเครื่องในแบบคำสั่งต่อคำสั่งนั่นเอง

4.4 ภาษาแอสแซมบลี

ภาษาแอสแซมบลีจริง ๆ แล้วก็คือภาษานีโมนิคที่มีการเพิ่มเติมคำสั่งพิเศษในการจัดการเรื่องการอ้างอิงหน่วยความจำนั่นเอง ภาษาแอสแซมบลีจัดเป็นภาษาระดับต่ำภาษาหนึ่ง ซึ่งสัญลักษณ์ที่ใช้มักจะถูกกำหนดโดยผู้ผลิต IC หน่วยประมวลผลกลาง ดังนั้นเมื่อมีการเปลี่ยนหน่วยประมวลผลกลาง ภาษาแอสแซมบลีที่ใช้ ก็จะต้องเปลี่ยนตามไปด้วย เมื่อพัฒนาโปรแกรมเป็นภาษาแอสแซมบลีแล้ว ก็ต้องนำไป Assembling ด้วยตัวแปลภาษาชื่อ Assembler ซึ่งจะแปลภาษาแอสแซมบลีให้เป็นภาษาเครื่องอีกทีหนึ่ง

ข้อดี ของการใช้ภาษาแอสแซมบลี	**ข้อเสีย** ของการใช้ภาษาแอสแซมบลีคือ
1. สั่งงานระบบคอมพิวเตอร์ได้โดยตรง	1. เป็นภาษาระดับต่ำ พัฒนาโปรแกรมได้ยาก
2. สามารถเขียนโปรแกรมที่มีขนาดเล็กได้	2. ใช้กับการพัฒนาโปรแกรมที่ไม่ใหญ่มาก
3. โปรแกรมภาษาทำงานได้รวดเร็ว	3. ไม่มีโครงสร้างข้อมูลในระดับสูง
	4. ไม่สามารถนำไปใช้กับ CPU แบบอื่นได้

สรุป

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

คำถามทบทวน

1. ภาษานีโมนิค (Mnemonic) เรียกอีกอย่างหนึ่งว่าภาษาอะไรและมีความสำคัญอย่างไรในการเขียนภาษาโปรแกรมในยุคแรกๆ
2. การสร้างสัญลักษณ์แทนภาษาเครื่อง ทำให้มีการแบ่งระดับของภาษาสำหรับเขียนโปรแกรมออกเป็นกี่ระดับอะไรบ้าง
3. Interpret และ Compile แตกต่างกันอย่างไรจงอธิบาย
4. Assembling คืออะไร จงอธิบายมาพอสังเขป
5. จงอธิบายข้อดีและข้อเสียข้อดีของการใช้ภาษาแอสแซมบลี

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 22 ธันวาคม 2556,จาก: http://rirs3.royin.go.th/coinages/

ภาษาแอสแซมบลี. (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 22 ธันวาคม 2556, จาก: http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 5

หัวข้อเรื่อง สถาปัตยกรรมคอมพิวเตอร์เบื้องต้น
(Introduction to computer architecture)

รายละเอียด

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

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

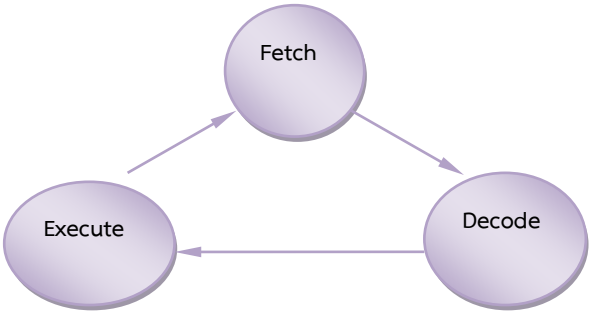
3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 5 การจัดการเรียนการสอน จะเกี่ยวข้องกับหน่วยประมวลผลกลาง หน่วยความจำ การเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ และของระบบไมโครโปรเซสเซอร์ตระกูล 80x86 ซึ่งระบบคอมพิวเตอร์ประกอบด้วยหน่วยประมวลผลกลาง (CPU) หน่วยความจำ (Memory) อุปกรณ์รับข้อมูล (Input Units) และอุปกรณ์แสดงผล (Output Unit) โดยเริ่มพิจารณาการเชื่อมโยงของอุปกรณ์ต่าง ๆ และโครงสร้างภายในขององค์ประกอบเหล่านั้น

5.1 หน่วยประมวลผลกลาง

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



ภาพที่ 5.1 แสดงขั้นตอนการทำงานของหน่วยประมวลผล

หน่วยประมวลผลกลางจะทำงานตามชุดคำสั่ง (instructions) ที่อ่านมาจากหน่วยความจำหลักเท่านั้น และเรียกสถาปัตยกรรมของระบบคอมพิวเตอร์ที่มีการเก็บโปรแกรมและข้อมูลไว้ในหน่วยความจำหลัก โดยที่หน่วยประมวลผลมีทำหน้าที่ที่เรียกว่า Stored Program Architecture หรือคอมพิวเตอร์แบบวอนนอยแมน (von Neumann Computer) โดยตั้งชื่อเพื่อเป็นเกียรติให้กับ John von Neumann [1]

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

5.2 หน่วยความจำ

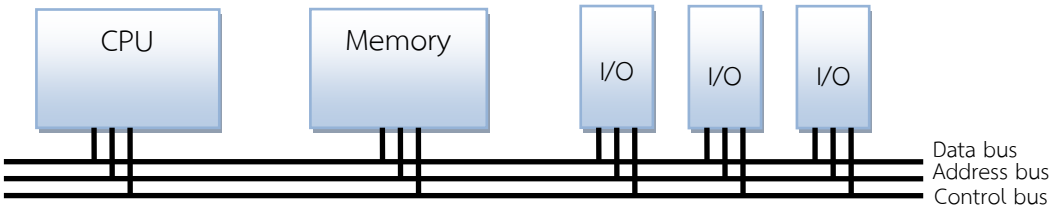
การเก็บข้อมูลในหน่วยความจำมีหน่วยที่เล็กที่สุดในการเก็บข้อมูลคือบิต แต่ในการเรียกข้อมูลจากหน่วยความจำนั้นจะกระทำในรูปของข้อมูลที่มีขนาดใหญ่กว่า คือจะมีขนาด 8 บิต หรือ 1 ไบท์ ภายในหน่วยความจำของระบบคอมพิวเตอร์หนึ่ง ๆ จะประกอบด้วยหน่วยย่อย ๆ ขนาด 8 บิตเหล่านี้อยู่มากมาย แต่ละหน่วยย่อย ๆ จะมีหมายเลขเฉพาะตัว เพื่อให้หน่วยประมวลผลสามารถใช้อ้างถึงเมื่อจะอ่านหรือเขียนข้อมูลลงไปในหน่วยย่อยหน่วยนั้นได้ จะต้องระบุหมายเลขซึ่งเรียกว่า **แอดเดรส (Address)** ของข้อมูลชิ้นนั้นให้ได้ด้วย

---

[1] ผู้บุกเบิกเทคโนโลยีคอมพิวเตอร์เชื่อว่าชื่อของคอมพิวเตอร์แบบวอนนอยแมน ให้เกียรติกับ von Neumann ผู้เขียนแนวคิดเกี่ยวกับคอมพิวเตอร์รูปแบบนี้มากไป โดยให้ความสำคัญกับ J.Presper Eckert และ John Mauchly วิศวกรผู้สร้างเครื่องคอมพิวเตอร์ลักษณะนี้ขึ้นมาน้อยเกินไป

5.3 การเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ

บัส (Bus) : ช่องทางสื่อสาร อุปกรณ์ต่าง ๆ จะเชื่อมต่อกับโดยผ่านทางกลุ่มของสายสัญญาณ ที่เราเรียกว่า “บัส” อุปกรณ์ต่าง ๆ จะส่งและรับสัญญาณผ่านทางกลุ่มสายสัญญาณชุดเดียวกัน แสดงดังภาพที่ 5.2



ภาพที่ 5.2 แสดงการเชื่อมต่อของอุปกรณ์ต่าง ๆ ผ่านระบบบัส

ระบบบัสสามารถแบ่งการทำงานออกได้ 3 กลุ่ม คือ

1. บัสข้อมูล (Data bus) ใช้สำหรับส่งรับข้อมูล
2. บัสตำแหน่งหรือแอดเดรสบัส (Address bus) ใช้สำหรับส่งรับตำแหน่งที่ใช้สำหรับอ้างอิงข้อมูลจากหน่วยความจำหรือจากอุปกรณ์ I/O
3. บัสควบคุม (Control bus) ใช้สำหรับส่งสัญญาณควบคุม

แอดเดรสบัส (Address bus) ในระบบคอมพิวเตอร์ที่หน่วยประมวลผลเชื่อมต่อกับอุปกรณ์อื่น ๆ ผ่านทางบัส ข้อมูลต่าง ๆ ที่ส่ง/รับกันระหว่างอุปกรณ์ต่าง ๆ นั้นจะส่งผ่านทางบัสข้อมูล ดังนั้นการที่หน่วยประมวลผลจะติดต่อกับหน่วยความจำหรืออุปกรณ์รับและแสดงผลข้อมูลที่ต้องการได้นั้น ทั้งหน่วยความจำ และ อุปกรณ์รับหรือแสดงผลข้อมูลทุกอุปกรณ์ จะต้องมีหมายเลขเฉพาะ หมายเลขนี้สำหรับหน่วยความจำก็คือแอดเดรส ส่วนอุปกรณ์อินพุตและอุปกรณ์เอาท์พุตก็มีหมายเลขเฉพาะสำหรับอุปกรณ์หนึ่ง ๆ เช่นเดียวกัน โดยเรียกว่า หมายเลข I/O (I/O address) เมื่อหน่วยประมวลผลต้องการติดต่อกับหน่วยความจำที่ตำแหน่งใด หรือติดต่อกับอุปกรณ์ใดก็จะส่งแอดเดรสของหน่วยความจำนั้น หรือของอุปกรณ์นั้นมา ในการเลือกว่าหมายเลขแอดเดรสที่ส่งมาเป็นของหน่วยความจำหรือของอุปกรณ์อินพุตเอาท์พุตโดยหน่วยประมวลผลจะส่งสัญญาณระบุมาทางสัญญาณในบัสควบคุม

5.4 สถาปัตยกรรมของระบบไมโครโปรเซสเซอร์ตระกูล 80x86

5.4.1 ความเป็นมา ไมโครโปรเซสเซอร์ตระกูล 80x86 เป็นไมโครโปรเซสเซอร์ที่พัฒนาขึ้นโดยบริษัท Intel โดยมีการพัฒนามาตั้งแต่รุ่น 4040 ซึ่งเป็นไมโครโปรเซสเซอร์ขนาด 4 บิต จนกระทั่งในปัจจุบันได้พัฒนาเป็นไมโครโปรเซสเซอร์รุ่น Pentium รายละเอียดคร่าว ๆ ของไมโครโปรเซสเซอร์รุ่นต่าง ๆ เป็นดังต่อไปนี้

1. 8086 เป็นไมโครโปรเซสเซอร์ขนาด 16 บิต รุ่นแรกที่ Intel ผลิตขึ้น สามารถอ้างหน่วยความจำได้ 1 MByte มีริจิสเตอร์ภายในขนาด 16 บิต

2. 8088 เนื่องจากในขณะนั้นอุปกรณ์ต่าง ๆ โดยมากเป็นอุปกรณ์ขนาด 8 บิต บริษัท Intel จึงได้ผลิตไมโครโปรเซสเซอร์ 8088 ซึ่งมีสถาปัตยกรรมภายในเหมือน 8086 แต่มีการติดต่อกับระบบภายนอกเป็นแบบ 8 บิต

3. 80186 เป็นหน่วยประมวลผลซึ่งเพิ่มการจัดการเกี่ยวกับอุปกรณ์รอบข้างเข้าไปเพื่อให้เป็นหน่วยประมวลผลสำหรับงานควบคุมต่าง ๆ

4. 80286 เป็นหน่วยประมวลผลขนาด 16 บิตที่มีความเร็วในการทำงานสูงขึ้น ขยายขอบเขตการอ้างหน่วยความจำเป็น 24 MByte เพิ่มความสามารถในการจัดการหน่วยความจำ

5. 80386 เป็นหน่วยประมวลผลขนาด 32 บิต อ้างหน่วยความจำได้ถึง 4 กิกะไบต์ มีความสามารถในการจัดการหน่วยความจำที่ซับซ้อน และสามารถใช้หน่วยความจำแบบเสมือนได้ มาตรฐานของชุดคำสั่งและกรรมวิธีในการจัดการหน่วยความจำของหน่วยประมวลผลรุ่นนี้ยังคงใช้เป็นมาตรฐานอยู่จนถึงปัจจุบันนี้ แม้แต่ระบบปฏิบัติการ Windows 95 ยังสามารถนำมาทำงานได้บนหน่วยประมวลผลรุ่นนี้ แต่จะทำงานได้ช้ามากเท่านั้นเอง

6. 80386SX เป็นหน่วยประมวลผลซึ่งมีสถาปัตยกรรมภายในเหมือน 80386 แต่มีระบบบัสภายนอกเป็น 16 บิต หน่วยประมวลผลรุ่นนี้ได้รับการออกแบบขึ้นคล้ายกับ 8088

7. 80486 ได้รับการพัฒนาต่อจากรุ่น 80386 โดยทางบริษัท Intel ได้มีการเพิ่มหน่วยประมวลผลเลขทศนิยมเข้าไป

8. 80486SX เป็นหน่วยประมวลผลรุ่น 80486 ซึ่งตัดความสามารถในส่วนของการประมวลผลเลขทศนิยมออก

9. Pentium, Pentium Pro, Pentium II เป็นหน่วยประมวลผลรุ่นล่าสุดของบริษัท Intel มีการเพิ่มความสามารถในการประมวลผลให้สูงขึ้น โดยใช้เทคโนโลยีต่าง ๆ เช่นการประมวลผลแบบไปป์ไลน์ การประมวลผลแบบซูเปอร์สเกลาร์ เป็นต้น นอกจากไมโครโปรเซสเซอร์ตระกูล 80x86 แล้ว ยังมีหน่วยประมวลผลกลางที่ผลิตและพัฒนาโดยบริษัทต่าง ๆ อีกหลายตระกูลเช่น

ตระกูล 68000 และ PowerPC ของบริษัทโมโตโรล่าตระกูล Alpha ของบริษัท DEC และตระกูล Sparc ของบริษัท Sun Microsystem เป็นต้น

5.4.2 ลักษณะทั่วไปของไมโครโปรเซสเซอร์ 8086

ระบบบัส ไมโครโปรเซสเซอร์ 8086 มีแอดเดรสบัสขนาด 20 บิต ทำให้สามารถอ้างแอรดเดรสได้ 1 เมกะไบต์และมีบัสข้อมูลขนาด 16 บิต ซึ่งทำให้การอ่านและเขียนข้อมูลครั้งละ 2 ไบต์ หน่วยประมวลผลทางคณิตศาสตร์และตรรกศาสตร์ภายใน 8086 สามารถประมวลผลได้กับข้อมูลขนาด 16 บิต รีจิสเตอร์ภายในไมโครโปรเซสเซอร์ 8086 มีขนาด 16 บิต

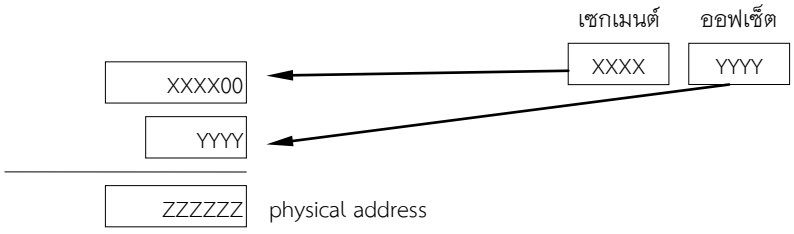
การจัดการหน่วยความจำ ภายในหน่วยประมวลผล 8086 มีรีจิสเตอร์ขนาด 16 บิต แต่มีแอดเดรสบัสขนาด 20 บิต ด้วยสาเหตุดังกล่าวหน่วยประมวลผลจะไม่สามารถเก็บตำแหน่งข้อมูลภายในหน่วยความจำได้ภายในรีจิสเตอร์เพียงตัวเดียว ดังนั้นการจัดเก็บตำแหน่งของข้อมูลภายในหน่วยความจำ 8086 จึงต้องเก็บด้วยรีจิสเตอร์ 2 ตัว ซึ่งมีวิธีการจัดเก็บ แบบ **เซกเมนต์:ออฟเซ็ต (segment:offset)** แอดเดรสที่แท้จริง (physical address) ขนาด 20 บิต จะถูกจัดเก็บด้วยรีจิสเตอร์ขนาด 16 บิต 2 ตัว ค่าที่เก็บในรีจิสเตอร์ตัวแรกเรียกว่าเซกเมนต์ (segment) ส่วนค่าที่เก็บในรีจิสเตอร์อีกตัวเรียกว่าออฟเซ็ต (offset)

การอ้างถึงตำแหน่งภายในหน่วยความจำแบบเซกเมนต์ : ออฟเซ็ตนั้น อาจเปรียบได้เสมือนกับการที่แบ่งหน่วยความจำเป็นส่วนย่อย ๆ โดยส่วนย่อยนี้เรียกว่า เซกเมนต์ การที่มีการอ้างถึงตำแหน่งใด ๆ มักจะอ้างถึงเซกเมนต์ที่ตำแหน่งข้อมูลนั้นอยู่ และระบุระยะห่างของหน่วยความจำที่คิดเทียบกับจุดเริ่มต้นของเซกเมนต์ที่ระบุไป

การแปลงแอดเดรสที่เก็บอยู่ในรูปของ เซกเมนต์ : ออฟเซ็ต เป็นแอดเดรสขนาด 20 บิต มีขั้นตอนดังนี้

1. เลื่อนบิตของค่าเซกเมนต์ไปทางซ้าย 4 บิต ดังนั้นจากข้อมูล 16 บิต เราจะได้ข้อมูล 20 บิต ที่มี 4 บิตทางขวาเป็น 0 ในทุกหลัก

2. นำค่าออฟเซ็ตมาบวกเข้ากับค่าเซกเมนต์ที่เลื่อนบิตแล้ว จะได้แอดเดรสขนาด 20 บิต ที่จะนำไปอ้างตำแหน่งที่แท้จริงของข้อมูล ขั้นตอนทั้งสอง แสดงได้ดังภาพที่ 5.3



ภาพที่ 5.3 แสดงการแปลงแอดเดรสจากแบบเซกเมนต์ : ออฟเซ็ต เป็น แอดเดรสขนาด 20 บิต

ตัวอย่าง เช่น 123Ah : 22B6h แปลงเป็นแอดเดรสขนาด 20 บิตได้ดังนี้

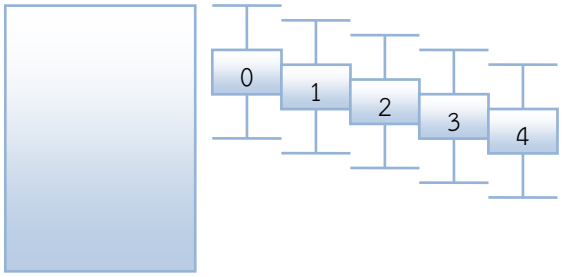
เลื่อนบิตของ 123Ah ไป 4 บิต ได้ 123A0h

นำ 22B6h มาบวก 22B6h

จะได้แอดเดรสขนาด 20 บิตคือ 14656h

ในทางกลับกันที่ตำแหน่ง แอดเดรส 14656h ก็จะสามารถอ้างแบบ เซกเมนต์ : ออฟเซ็ต ได้เป็น 123Ah : 22B6h เช่นเดียวกันแต่ตำแหน่งแอดเดรส 14656h สามารถอ้างแบบ เซกเมนต์ : ออฟเซ็ต ค่าอื่นก็ได้เช่น 1465h : 0006h หรือ 1460h : 0056h และยังอ้างโดยใช้คู่เซกเมนต์ : ออฟเซ็ต คู่อื่น ๆ ได้อีกหลายคู่

ผลจากการใช้การอ้างแอดเดรสแบบ เซกเมนต์ : ออฟเซ็ต ทำให้ลักษณะของหน่วยความจำที่ 8086 มองเห็นจะมีลักษณะเป็นส่วน ๆ ที่อ้างอิงตามค่าของเซกเมนต์ โดยแต่ละส่วนนี้จะมีขนาดส่วนละ 64 กิโลไบท์ ส่วนของหน่วยความจำขนาด 64 กิโลไบต์นี้เรียกว่า เซกเมนต์ การจัดเรียงตัวของเซกเมนต์ต่าง ๆ ในหน่วยความจำจะจัดเรียงเป็นส่วน ๆ ที่มีการเหลื่อมกัน แสดงได้ดังภาพที่ 5.4



ภาพที่ 5.4 แสดงลักษณะการเหลื่อมกันของเซกเมนต์

เนื่องจากการอ้างถึงข้อมูลใด ๆ ในหน่วยความจำของไมโครโปรเซสเซอร์ 8086 จะต้องอ้างตำแหน่งเป็นคู่เซกเมนต์ : ออฟเซต์ การอ้างถึงข้อมูลในตำแหน่งต่าง ๆ อาจทำได้ยุ่งยากเพราะต้องมีการระบุทั้งเซกเมนต์และออฟเซ็ต 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 ตัวนี้จะใช้ประกอบกับค่าออฟเซ็ตต่าง ๆ เพื่อระบุตำแหน่งของโปรแกรม (code) ข้อมูล (data) และแสต็ก (stack) ส่วนรีจิสเตอร์ ES มีหน้าที่เก็บเซกเมนต์ของข้อมูลที่ใช้ในการสั่งงานคำสั่งพิเศษบางประเภท เช่น คำสั่งเกี่ยวกับข้อความ ดังนั้นจะเห็นได้ว่า ถึงแม้ไมโครโปรเซสเซอร์ 8086 จะสามารถมองหน่วยความจำได้รวมถึง 1 เมกะไบต์ แต่โปรแกรมที่ทำงานอยู่จะมองเห็น

หน่วยความจำได้พร้อม ๆ กันแค่เพียง 4 เซกเมนต์เท่านั้น นั่นคือ เซกเมนต์ของโปรแกรม เซกเมนต์ของข้อมูล 2 เซกเมนต์ และ เซกเมนต์ของแสต็ก

แสต็ก (Stack) ภายในหน่วยความจำของระบบไมโครโปรเซสเซอร์ 8086 จะมีหน่วยความจำส่วนหนึ่งที่ถูกกันเนื้อที่ไว้สำหรับเป็น แสต็กโดยเซกเมนต์ของแสต็กจะถูกชี้โดยรีจิสเตอร์ SS ลักษณะเฉพาะของหน่วยความจำแบบแสต็กคือการที่ระบบจะเก็บข้อมูลและอ่านข้อมูลออกไปแบบ เข้าก่อน ออกทีหลัง (First In Last Out : FILO) โดยอาจมองลักษณะเหมือนการวางซ้อนข้อมูลเหมือนซ้อนจาน ข้อมูลที่ถูกนำมาเก็บก่อนจะอยู่ทางด้านล่าง ข้อมูลถัดไปจะวางซ้อนอยู่ด้านบน ข้อมูลที่อยู่ทางด้านล่างจะไม่สามารถอ่านออกไปได้ถ้ามีข้อมูลอื่นที่เก็บทีหลังและยังไม่ได้อ่านออกไป ระบบจะใช้แสต็กในการเรียกโปรแกรมย่อยเป็นส่วยใหญ่

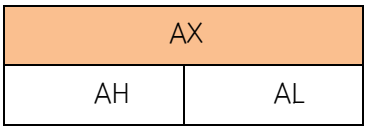
5.4.3 รายละเอียดของส่วนประกอบภายในไมโครโปรเซสเซอร์

หน่วยประมวลผลทางคณิตศาสตร์และตรรกศาสตร์ (ALU) ไมโครโปรเซสเซอร์ 8086 มี ALU ที่สามารถประมวลผลได้ครั้งละ 16 บิต ด้วยความสามารถในการประมวลผลที่ละ 16 บิตนี้ ทำให้เราเรียกไมโครโปรเซสเซอร์ 8086 ว่าเป็นไมโครโปรเซสเซอร์ขนาด 16 บิต

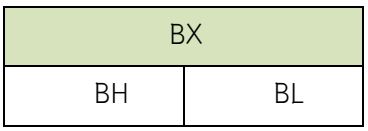
หน่วยความจำชั่วคราว (รีจิสเตอร์ – Register) รีจิสเตอร์ใน 8086 มีทั้งขนาด 16 บิต และ 8 บิต โดยจะแบ่งเป็นกลุ่ม ๆ ได้ดังนี้

1.รีจิสเตอร์สำหรับใช้งานทั่วไป (General–Purpose Registers) รีจิสเตอร์ในกลุ่มนี้ ผู้เขียนโปรแกรมสามารถนำไปใช้งานได้ตามความต้องการ โดยในกลุ่มนี้จะมีรีจิสเตอร์ที่ใช้ขนาด 16 บิต อยู่ 4 ตัว โดยรีจิสเตอร์ขนาด 16 ทั้ง 4 ตัวจะแบ่งได้เป็นรีจิสเตอร์ขนาด 8 บิต อีก 8 ตัว รีจิสเตอร์ขนาด 16 บิตและคู่รีจิสเตอร์ขนาด 8 บิต มีดังต่อไปนี้

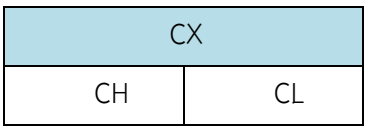
1.1. รีจิสเตอร์ AX (Accumulator Register)



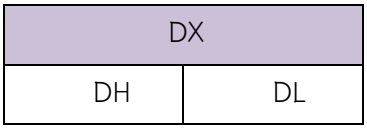
1.2. รีจิสเตอร์ BX (Base Register)



1.3. รีจิสเตอร์ CX (Counter Register)



1.4. รีจิสเตอร์ DX (Data Register)



สำหรับรีจิสเตอร์ทั้ง 4 ตัวนี้ นอกจากจะมีไว้สำหรับใช้งานทั่วไปได้แล้ว ยังมีหน้าที่เฉพาะอื่น ๆ อีก ซึ่งจะกล่าวถึงในการสอนประจำสัปดาห์ที่ 6 ต่อไป

2. รีจิสเตอร์สำหรับอ้างอิง (Index Register) ไมโครโปรเซสเซอร์ 8086 มีรีจิสเตอร์สำหรับอ้างอิง 2 ตัว คือ SI (Source Index) และ DI (Destination Index) รีจิสเตอร์ในกลุ่มนี้ใช้สำหรับการอ้างตำแหน่งแบบอ้างอิงและใช้ในคำสั่งที่เกี่ยวกับข้อความ แต่ผู้ใช้สามารถนำไปใช้งานทั่วไปได้ด้วยเช่นกัน

รีจิสเตอร์สำหรับการชี้ (Pointer Register) รีจิสเตอร์กลุ่มนี้คือ SP และ BP รีจิสเตอร์ SP ใช้ประกอบกับรีจิสเตอร์ SS มีหน้าที่ชี้ตำแน่งปัจจุบันของแสต็ก โดยรีจิสเตอร์ BP ส่วนใหญ่จะใช้เพื่อชี้ตำแหน่งของแสต็กส่วนของการส่งพารามิเตอร์นิยมใช้ในส่วนโปรแกรมย่อย

3. เซกเมนต์รีจิสเตอร์ (segment register) เซกเมนต์รีจิสเตอร์ทั้ง 4 ตัวคือ CS DS ES และ SS ใช้ประกอบกับค่าของออฟเซ็ตเพื่อชี้ตำแหน่งของโปรแกรมข้อมูลปรกติ ข้อมูลพิเศษและแสต็ก ตามลำดับ

4. แฟล็ก (flag) ไมโครโปรเซสเซอร์ 8086 จะเก็บลักษณะของผลลัพธ์ของการคำนวณทางคณิตศาสตร์ไว้ใน แฟล็ก

5. รีจิสเตอร์อื่น ๆ ของระบบ นอกจากรีจิสเตอร์ต่าง ๆ ที่ผู้ใช้สามารถกำหนดและใช้งานได้แล้ว ยังมีรีจิสเตอร์อีกกลุ่มหนึ่งซึ่งผู้เขียนโปรแกรมไม่สามารถเรียกใช้ได้ รีจิสเตอร์ในกลุ่มนี้ เช่น IP ซึ่งเป็นรีจิสเตอร์ที่ใช้ประกอบกับ CS เพื่อชี้ตำแหน่งของคำสั่งที่จะทำงานต่อไปหรือ IR ซึ่งรีจิสเตอร์ที่เก็บคำสั่งปัจจุบันที่ไมโครโปรเซสเซอร์อ่าน (fetch) ขึ้นมาจากหน่วยความจำ

5.4.4 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 เป็นต้น

5.4.5 การอินเตอร์รัพท์ (Interrupt) การอินเตอร์รัพท์หรือการขัดจังหวะ คือการสั่งให้หน่วยประมวลผลหยุดการทำงานชั่วคราว แล้วกระโดดไปทำงานบางอย่างเพื่อตอบสนองการ

ขัดจังหวะนั้น ตัวอย่างของการขัดจังหวะ เช่น อุปกรณ์บางชิ้นได้รับข้อมูล หรือ ข้อมูลได้รับเขียนเก็บลงในฮาร์ดดิสก์เรียบร้อยแล้ว เป็นต้น เมื่อหน่วยประมวลผลตอบสนองการขัดจะหวะเรียบร้อยแล้ว ก็จะคืนสู่สถานะเดิมและกลับไปประมวลผลงานเก่าที่ประมวลผลค้างไว้ เสมือนไม่มีอะไรเกิดขึ้น การขัดจังหวะนี้มี 2 ประเภทคือ ซอฟต์แวร์อินเตอร์รัพท์ และฮาร์ดแวร์อินเตอร์รัพท์ นิยมใช้ซอฟต์แวร์อินเตอร์รัพท์ในการเรียกใช้การบริการต่าง ๆ ของระบบ ส่วนฮาร์ดแวร์อินเตอร์รัพท์จะนิยมใช้ในการแจ้งการเปลี่ยนสถานะของอุปกรณ์อินพุตเอาท์พุตต่าง ๆ

5.4.6 สถาปัตยกรรมของระบบคอมพิวเตอร์สมัยใหม่ เทคโนโลยีของสถาปัตยกรรมคอมพิวเตอร์สมัยใหม่ได้พัฒนาไปอย่างรวดเร็วมาก เครื่องคอมพิวเตอร์ที่ใช้ในปัจจุบันมีประสิทธิภาพมากกว่าเครื่องคอมพิวเตอร์เมื่อ 2–3 ปีก่อนหลายเท่า ทั้งนี้เนื่องจากมีการวิจัยและสร้างหน่วยประมวลผลกลางและระบบคอมพิวเตอร์ที่มีประสิทธิภาพสูงขึ้นมาก

เทคโนโลยีของหน่วยประมวลผลกลาง

1. หน่วยประมวลผลแบบ RISC ชุดคำสั่งของหน่วยประมวลผลยุคเก่ามีลักษณะเป็นแบบ CISC : Complex Instruction Set Computer นั่นคือชุดคำสั่งจะหนึ่ง ๆ จะมีความซับซ้อนมาก การที่ชุดคำสั่งซับซ้อนทำให้การออกแบบส่วนควบคุมภายในหน่วยประมวลผลทำได้ยาก ในปัจจุบันหน่วยประมวลผลต่าง ๆ ได้เปลี่ยนแนวทางในการพัฒนาไปเป็นแบบ RISC : Reduced Instruction Set Computer โดยเน้นชุดคำสั่งที่มีความซับซ้อนน้อยลง แต่มีความเร็วในการทำงานสูงขึ้น การทำให้ชุดคำสั่งมีรูปแบบที่ง่ายขึ้นทำให้การออกแบบส่วนควบคุมทำได้ง่ายขึ้นและยังสามารถใช้วิธีการแบบไปป์ไลน์ (Pipeline) และซูปเปอร์สเกลาร์ (Superscalar) ในการเพิ่มประสิทธิภาพของหน่วยประมวลผลได้ง่ายขึ้นด้วย

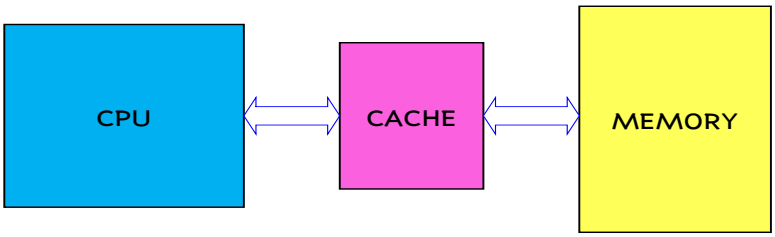
2. ไปป์ไลน์ (Pipeline) หน่วยประมวลผลรุ่นใหม่จะมีการประมวลผลแบบไปป์ไลน์ ซึ่งจะมีการ fetch decode และ execute คำสั่งที่เหลื่อมเวลากัน แสดงได้ดังภาพที่ 5.5 การประมวลผลเหลื่อมกันนี้ทำให้ประสิทธิภาพของการประมวลผลสูงขึ้นมาก

ภาพที่ 5.5 แสดงการทำงานแบบไปป์ไลน์เทียบกับการทำงานแบบปกติ

3. ซูเปอร์สเกลาร์ (Superscalar) ในหน่วยประมวลผลที่มีประสิทธิภาพสูงบางรุ่นจะประมวลผลชุดคำสั่งหลายชุดคำสั่งได้พร้อมกัน การที่หน่วยประมวลผลประมวลผลคำสั่งได้หลายชุดพร้อมกันนี้เรียกว่า ซูเปอร์สเกลาร์

ระบบบัสสมัยใหม่ ระบบคอมพิวเตอร์สมัยก่อน หน่วยประมวลผลมีความเร็วในการประมวลผลไม่มากนักทำให้การโอนย้ายข้อมูลระหว่างหน่วยประมวลผลกับหน่วยความจำกระทำได้โดยไม่ก่อให้เกิดการเสียเวลา แต่การพัฒนาของหน่วยประมวลผลเป็นไปอย่างรวดเร็วกว่าการพัฒนาของหน่วยความจำมากทำให้ปัจจุบันอัตราการประมวลผลของหน่วยประมวลผลสูงกว่าอัตราการโอนย้ายข้อมูลระหว่าหน่วยประมวลผลกับหน่วยความจำมาก ซึ่งสถานการณ์เช่นนี้ก่อให้เกิดปัญหาในรูปแบบคอขวด (Bottleneck problem) ขึ้นนั่นคือจุดเชื่อมต่อระหว่างหน่วยความจำกับหน่วยประมวลผลทำให้ประสิทธิภาพของระบบลดลง ซึ่งเรียกคอขวดระหว่างหน่วยประมวลผลกับหน่วยความจำว่า คอขวดของวอนนอยแมน (von Neumann Bottleneck)

วิธีการแก้ปัญหานี้คือ การใช้หน่วยความจำที่มีความเร็วสูงและขนาดเล็กมาเป็นบัฟเฟอร์ (ที่พักข้อมูลชั่วคราว) ระหว่างหน่วยความจำและหน่วยประมวลผล หน่วยความจำที่มีความเร็วสูงนี้เรียกว่า **หน่วยความจำแคช** (cache memory) ในหน่วยประมวลผลปัจจุบันหลายรุ่น ได้มีการบรรจุหน่วยความจำแคชลงไปภายในไมโครโปรเซสเซอร์ด้วย ลักษณะของบัสที่มีการใช้หน่วยความจำแคช แสดงได้ดังภาพที่ 5.6



ภาพที่ 5.6 แสดงลักษณะของบัสที่มีการใช้หน่วยความจำแคช

สรุป

สถาปัตยกรรมไมโคร (Micro Architecture) เป็นลักษณะของ เครื่องคอมพิวเตอร์ส่วนใหญ่ที่มีการใช้งานมาตั้งแต่ยุคเริ่มต้น จนถึงยุคปัจจุบัน ได้รับการออกแบบโครงสร้างและการทำงานโดยจอนวอนนิวแมน(John Von Neumann) ซึ่งเป็นผู้นำในการออกแบบเครื่องคอมพิวเตอร์ โดยเครื่องคอมพิวเตอร์ ที่เขาได้ออกแบบ มีส่วนประกอบที่สำคัญ 3 อย่างด้วยกัน คือ หน่วยประมวลผลกลาง (Central Processing Unit) หน่วยความจำ (Main Memory) และหน่วยเชื่อมต่ออุปกรณ์ภายนอก (Input/Output) สำหรับไมโครโปรเซสเซอร์ ในตระกูล x86 ออกแบบโดยใช้หลักการเดียวกันกับที่ วอนนิวแมนได้กำหนดไว้ กล่าวคือ การประมวลผลทั้งหมด ที่หน่วยประมวลผลกลาง สำหรับข้อมูล ไม่ว่าจะเป็นข้อมูลที่ใช้สำหรับประมวลผล หรือข้อมูลที่เป็นคำสั่งก็ตาม จะถูกเก็บไว้ในหน่วยความจำจนกว่าจะมีการเรียกใช้งานจากหน่วยประมวลผลกลาง สำหรับส่วนต่อเชื่อมกับอุปกรณ์ภายนอกนั้น การทำงานของหน่วยประมวลผลกลางก็จะคล้ายกับการทำงานกับหน่วยความจำ ทั้งนี้เพราะทั้งหน่วยความจำและอุปกรณ์ภายนอก เมื่อมีการทำงานกับหน่วยประมวลผลกลางก็มีลักษณะของการเรียกใช้ข้อมูลและนำ ข้อมูลไปเก็บ ต่างกันเพียงสถานที่ และ วิธีการเข้าถึงเท่านั้น บัสของระบบ หน่วยงานต่าง ๆ ภายในเครื่องคอมพิวเตอร์นั้น ต่อเชื่อเข้าด้วยกันด้วยระบบบัส สำหรับโปรเซสเซอร์ในตระกูล 80x86 มีด้วยกัน 3 กลุ่มอันได้แก่ ดาต้าบัส (Data Bus) แอดเดรสบัส (Address Bus) และคอมโทรลบัส (Control Bus) เป็นต้น

คำถามทบทวน

1. ขั้นตอนการทำงานของหน่วยประมวลผลมีทั้งหมดกี่ขั้นตอนอะไรบ้าง
2. เราสามารถแบ่งกลุ่มของบัสออกเป็นกี่กลุ่มอะไรบ้าง จงยกตัวอย่างพร้อมวาดรูปประกอบคำอธิบาย
3. จงแสดงวิธีการแปลงแอดเดรส 234Ah : 44B6h เป็นแอดเดรสขนาด 20 บิต
4. โปรเซสเซอร์ในตระกูล 80x86 ประกอบไปด้วยกลุ่มข้อมูลอะไรบ้าง จงยกตัวอย่างพร้อมวาดรูปประกอบคำอธิบาย
5. รีจิสเตอร์สำหรับใช้งานทั่วไป (General–Purpose Registers) มีอะไรบ้าง ผู้เขียนโปรแกรมสามารถนำรีจิสเตอร์ในกลุ่มไปใช้งานได้อย่างไร จงอธิบายพร้อมยกตัวอย่างประกอบ
6. การทำงานของ SI (Source Index) และ DI (Destination Index) มีหน้าที่ต่างกันอย่างไร
7. รีจิสเตอร์สำหรับการชี้ (Pointer Register) มีอะไรบ้าง พร้อมอธิบายการทำงาน
8. รีจิสเตอร์ใน 8086 มีทั้งขนาด 16 บิต และ 8 บิต แบ่งเป็นกลุ่ม ๆ ได้อย่างไรบ้าง
9. จงอธิบายความแตกต่างระหว่างหน่วยประมวลผลแบบ RISC กับแบบ CISC
10. วิธีการทำงานแบบไปป์ไลน์ (Pipeline) และซูปเปอร์สเกลาร์ (Superscalar) แตกต่างกันอย่างไร

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 28 ธันวาคม 2556,จาก: http://rirs3.royin.go.th/coinages/

ไมโครโปรเซสเซอร์. (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 28 ธันวาคม 2556, จาก: http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 6

หัวข้อเรื่อง คำสั่งโอนย้ายข้อมูล แฟล็กและคำสั่งคณิตศาสตร์
(Command Transfer, Flags and Mathematics Instruction)

รายละเอียด

ศึกษาเกี่ยวกับคำสั่งภาษาแอสแซมบลีเบื้องต้น โดยจะเป็นคำสั่งเกี่ยวกับการโอนย้ายข้อมูล ทั้งจากรีจิสเตอร์และหน่วยความจำ ในตอนท้ายของบทนี้จะเป็นการแนะนำโปรแกรม DEBUG ซึ่งจะเป็นเครื่องมือที่เราจะใช้ทดลองการโปรแกรมภาษาแอสแซมบลีเบื้องต้น การใช้คำสั่งคำนวณทางคณิตศาสตร์และการแสดงสถานะของผลลัพธ์ของการคำนวณนั้นในแฟล็ก สถานะที่เก็บในแฟล็กจะใช้สำหรับการจัดการกับผลการคำนวณนั้น ๆ รวมถึงใช้ในคำสั่งเกี่ยวกับการกระโดดแบบมีเงื่อนไขด้วย

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 6 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่งในการโยนย้ายข้อมูล การใช้คำสั่ง MOV เครื่องมือในการทดลองการโปรแกรมภาษาแอสแซมบลี คำสั่งในการโยนย้ายข้อมูล และคำสั่งทั่วไป การทำงานของแฟล็ก (Flags) คำสั่งทางคณิตศาสตร์ กลุ่มคำสั่งบวกและลบ กลุ่มคำสั่งคูณและหาร กลุ่มคำสั่งแปลงขนาดตัวเลข ซึ่งคำสั่งภาษาแอสแซมบลีเบื้องต้น จะเป็นคำสั่งเกี่ยวกับการโอนย้ายข้อมูล ทั้งจากรีจิสเตอร์และหน่วยความจำ ซึ่งในเนื้อหาจะมีการแนะนำการใช้โปรแกรม DEBUG ซึ่งจะเป็นเครื่องมือที่ใช้ทดลองในการเขียนโปรแกรมภาษาแอสแซมบลีเบื้องต้น

6.1 คำสั่งในการโอนย้ายข้อมูล

คำสั่ง MOV

คำสั่ง MOV เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลจากแหล่งข้อมูลต้นทางไปยังแหล่งข้อมูลปลายทาง โดยมีรูปแบบดังนี้

```
MOV     reg, reg
MOV     reg, mem
```

reg : register
mem : memory

MOV mem, reg imm : immediate (ค่าคงที่)

MOV reg, imm

MOV mem, imm

ในการคัดลอกข้อมูลจะกระทำจากตัวถูกดำเนินการ (operand) ตัวหลังไปยังตัวถูกดำเนินการ (operand) ตัวหน้า ซึ่งจะเห็นว่าไม่สามารถคัดลอกข้อมูลจากหน่วยความจำไปยังหน่วยความจำได้ (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

ข้อจำกัดของคำสั่ง MOV

* ตัวถูกดำเนินการ (Operand) ทั้งสองตัวต้องมีขนาดเท่ากัน
* ไม่สามารถคัดลอกค่าคงที่ (immediate) ไปยังเซกเมนต์รีจิสเตอร์ได้โดยตรง ต้องกระทำผ่านทางรีจิสเตอร์อื่น ๆ เช่น AX เป็นต้น
* การคัดลอกค่าคงที่ไปยังหน่วยความจำจะต้องระบุขนาดของหน่วยความจำด้วย

ตัวอย่าง

MOV	AX,100h	กำหนดค่ารีจิสเตอร์ AX ให้เป็น 100h จากนั้นคัดลอกค่าจาก AX ไปยัง BX และคัดลอกจาก BX ไปยัง DX ตามลำดับ
MOV	BX,AX	
MOV	DX,BX	
MOV	AX,1234h	กำหนดค่า 1234h ให้กับ AX และ 5678h ให้กับ DX จากนั้น คัดลอกค่าในรีจิสเตอร์ DL (มีค่าเท่ากับ 78h เพราะอะไร?) ไปให้กับรีจิสเตอร์ AL และคัดลอกค่าจาก DH (มีค่าเท่ากับ 56h เพราะอะไร?) ไปยังรีจิสเตอร์ DH
MOV	DX,5678h	
MOV	AL,DL	
MOV	BH,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

คู่รีจิสเตอร์ 16 บิต กับ 8 บิต ในการใช้งานรีจิสเตอร์ทั่วไปเราจะต้องคำนึงถึงเรื่องของคู่รีจิสเตอร์ 16 บิต กับ 8 บิตด้วย จากที่เราได้ทราบมาแล้วว่ารีจิสเตอร์ 8 บิตที่เราใช้ได้นั้น เป็นส่วนหนึ่งของรีจิสเตอร์ทั่วไปขนาด 16 บิต เช่น รีจิสเตอร์ AH เป็นไบต์สูงรีจิสเตอร์ AX (บิตที่ 8–15) เป็นต้น ดังนั้นถ้าเรากำหนดค่าให้กับรีจิสเตอร์ 8 บิตก็จะมีการเปลี่ยนแปลงกระทบถึงรีจิสเตอร์ 16 บิตที่รีจิสเตอร์นั้นประกอบอยู่ด้วย และในทางกลับกันการเปลี่ยนแปลงค่าในรีจิสเตอร์ 16 บิตก็มีผลกระทบมาถึงรีจิสเตอร์ 8 บิตด้วย

ตัวอย่าง

คำสั่ง		**ค่าในริจิสเตอร์หลังการทำงานของคำสั่ง**		
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

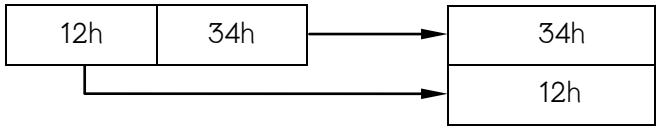
การโอนย้ายข้อมูลกับหน่วยความจำ โดยปกติการโอนย้ายข้อมูลกับหน่วยความจำนั้น โดยบุเฉพาะออฟเซ็ตของหน่วยความจำที่เราต้องการจะโอนย้ายข้อมูลด้วย ค่าของออฟเซ็ตนั้นจะถูกนำมาประกอบกับเซกเมนต์รีจิสเตอร์ DS เพื่อใช้เป็นตำแหน่งในหน่วยความจำที่แท้จริงที่จะอ่านเขียนข้อมูลด้วย

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

หลังการทำงาน รีจิสเตอร์ AX มีค่าเท่ากับ 6734h และ BX มีค่าเท่ากับ 3467h

ข้อสังเกตในการจัดเรียงไบต์เมื่อเก็บข้อมูลในหน่วยความจำ

สังเกตว่าในการเก็บค่าในหน่วยความจำเมื่อเราเก็บค่าเป็น 16 บิต การเรียงไบต์ในหน่วยความจำจะเก็บค่าในไบต์ที่มีนัยสำคัญสูงไว้ในไบต์ที่มีแอดเดรสสูงกว่าและไบต์ที่มีนัยสำคัญต่ำไว้ในแอดเดรสที่มีแอดเดรสต่ำกว่า แสดงได้ดังภาพที่ 6.1



ภาพที่ 6.1 แสดงการเรียงไบต์ข้อมูลในหน่วยความจำ

ลักษณะของการเก็บข้อมูลโดยเรียงไบต์ที่มีนัยสำคัญต่ำไว้ที่แอดเดรสต่ำและไบต์ที่มีนัยสำคัญสูงไว้ที่แอดเดรสสูงเรียกว่าเป็นการเรียงแบบ **C**

ในการกำหนดออฟเซ็ตของหน่วยความจำที่จะอ่านและเขียนข้อมูลนั้น จากตัวอย่างข้างต้นเป็นการระบุโดยใส่หมายเลขออฟเซ็ตโดยตรง และสามารถที่จะระบุออฟเซ็ตโดยผ่านทางค่าในรีจิสเตอร์ 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) สำหรับระบุว่าการอ้างถึงหน่วยความจำในตำแหน่งนั้นเป็นแบบไบต์ หรือ เวิร์ด

ตัวอย่าง

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

```
MOV    BX,104h
MOV    WORD PTR [BX],2345h
```

6.2 เครื่องมือในการทดลองการโปรแกรมภาษาแอสแซมบลี : โปรแกรม DEBUG

โปรแกรม DEBUG เป็นโปรแกรมสารพัดประโยชน์สำหรับการทดลองเกี่ยวกับการเขียนโปรแกรมภาษาแอสแซมบลี โปรแกรมนี้มีอยู่ทั้งในระบบปฏิบัติการ DOS และ Windows

เมื่อเรียกโปรแกรม DEBUG โดยพิมพ์ DEBUG ที่ DOS prompt จะเห็นเครื่องหมายเตรียมพร้อมของโปรแกรม DEBUG เป็นเครื่องหมายขีด

```
-
```

ผู้เขียนโปรแกรมสามารถจะพิมพ์คำสั่งต่าง ๆ ลงไปได้ที่ prompt นี้

6.3 คำสั่งในการโอนย้ายข้อมูล

คำสั่งทั่วไป

คำสั่งแสดงความช่วยเหลือ : ?

ผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรม DEBUG แสดงรายการคำสั่งต่าง ๆ ได้โดยใช้คำสั่ง ?

คำสั่งจัดการกับรีจิสเตอร์ : R (register)

คำสั่ง R คือคำสั่งให้โปรแกรม DEBUG แสดงค่าในรีจิสเตอร์รวมถึงกำหนดค่าให้กับรีจิสเตอร์ต่าง ๆ ด้วย ถ้าใช้คำสั่ง R โดยไม่ระบุชื่อรีจิสเตอร์ โปรแกรม DEBUG จะแสดงค่าในรีจิสเตอร์ออกมาให้เห็น

```
-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 สามารถระบุชื่อรีจิสเตอร์ต่อท้ายคำสั่ง 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)

ผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรม DEBUG แสดงค่าข้อมูลในหน่วยความจำได้ โดยใช้คำสั่ง D (dump) สามารถระบุตำแหน่งของหน่วยความจำที่เริ่มต้นแสดงข้อมูลได้ ถ้าไม่กำหนดโปรแกรม 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
-
```

การทดลองโปรแกรมภาษาแอสแซมบลีใน DEBUG

ผู้เขียนโปรแกรมสามารถป้อนโปรแกรมภาษาแอสแซมบลีเบื้องต้นได้ในโปรแกรม DEBUG และสามารถสั่งให้โปรแกรมนั้นทำงานเพื่อสังเกตผลการทำงานได้ อีกทั้งยังสามารถสั่งให้โปรแกรมทำงานทีละบรรทัดได้ด้วย

คำสั่งสำหรับแปลโปรแกรมภาษาแอสแซมบลี : A (assemble)

ผู้เขียนโปรแกรมสามารถใช้คำสั่ง A (assemble) เพื่อสั่งให้โปรแกรม DEBUG รับคำสั่งภาษาแอสแซมบลีแล้วแปลคำสั่งนั้นเป็นภาษาเครื่องเก็บไว้ในหน่วยความจำได้ ในการสั่งคำสั่ง assemble จะระบุตำแหน่งที่โปรแกรมภาษาเครื่องได้แปลแล้วจะถูกเขียนลงไปทันที แต่ถ้าไม่ระบุตำแหน่ง โปรแกรมที่เขียนจะถูกเก็บไว้ที่ตำแหน่งถัดจากการแปลครั้งสุดท้าย ในการป้อนโปรแกรมโดยป้อนโปรแกรมต่อเนื่องไปได้เรื่อย ๆ เมื่อป้อนเสร็จแล้วให้กดปุ่ม Enter ว่างไปหนึ่งบรรทัดโปรแกรม 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 ถ้ามีการป้อนโปรแกรมผิดโปรแกรม DEBUG จะรายงานว่าโปรแกรมผิดให้ผู้เขียนโปรแกรมทราบและสามารถป้อนโปรแกรมเข้าไปใหม่ได้

คำสั่งสำหรับแสดงโปรแกรมภาษาแอสแซมบลีจากภาษาเครื่อง : U (unassemble)

ผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรม DEBUG แสดงคำสั่งภาษาเครื่องที่อยู่ในหน่วยความจำเป็นรหัสคำสั่งภาษาแอสแซมบลีโดยใช้คำสั่ง unassemble ทำการระบุตำแหน่งที่จะเริ่มแสดงได้

```
-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 จะแสดงทั้งรหัสภาษาเครื่องและรหัสนีโมนิค (รหัสภาษา assembly) ด้วยการสั่งให้โปรแกรมทำงานผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรมทำงานได้ทั้งสิ้น 3 รูปแบบโดยโปรแกรมจะเริ่มทำงานที่ตำแหน่ง CS:IP เท่านั้น

คำสั่งเริ่มทำงาน : G (go)

เมื่อมีการสั่งให้โปรแกรมทำงานโดยใช้คำสั่ง g โปรแกรมจะเริ่มทำงานทันที และโปรแกรมจะทำงานจนกระทั่งจบ

```
-G
Program terminated normally
```

ข้อสังเกต โปรแกรมตัวอย่างจะมีคำสั่ง INT 20h ระบุอยู่ตอนท้าย คำสั่งนี้เป็นคำสั่งเรียกใช้บริการของระบบเพื่อที่จะจบโปรแกรม ถ้าไม่มีคำสั่งนี้ปิดท้าย โปรแกรมจะทำงานตามคำสั่งต่อไปเรื่อย ๆ จนกว่าจะพบคำสั่งที่สั่งให้โปรแกรมหยุดการทำงาน ดังนั้นเวลาทดลองเขียนโปรแกรมในโปรแกรม DEBUG ควรจะใส่คำสั่ง INT 20h ปิดท้ายไว้ทุกครั้ง

คำสั่งตามรอยการทำงาน (คำสั่งให้ทำงานทีละบรรทัดแบบที่หนึ่ง) : T (trace)

คำสั่งนี้โปรแกรมจะทำงานไปหนึ่งคำสั่งแล้วจะกลับมาที่โปรแกรม DEBUG และแสดงค่าของรีจิสเตอร์ต่าง ๆ รวมถึงผลจากคำสั่งนั้นทันที ทำให้สามารถตรวจสอบสถานะของโปรแกรมได้ตลอดขั้นตอนการทำงาน

```
-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 ภายหลังการทำงานของทุก ๆ คำสั่ง รวมถึงในกรณีที่มีการเรียกใช้โปรแกรมย่อยหรือเรียกใช้บริการของระบบด้วย การให้โปรแกรมทำงานไปหยุดที่บรรทัดถัดไปเลยโดยไม่ต้องคำนึงถึงการเรียกโปรแกรมย่อยต่าง ๆ ผู้เขียนโปรแกรมสามารถใช้คำสั่ง P (proceed) แทนคำสั่ง trace ได้

คำสั่งทำงานต่อจนถึงบรรทัดถัดไป (คำสั่งให้ทำงานทีละบรรทัดแบบที่สอง) : P (proceed)

ผลโดยทั่ว ๆ ไปของคำสั่งนี้เหมือนกับการใช้คำสั่ง T แต่การใช้คำสั่งนี้จะสะดวกกว่าเมื่อโปรแกรมที่ทำงานมีการเรียกโปรแกรมย่อยหรือมีการเรียกใช้บริการจากระบบ โดยโปรแกรมจะกลับมาที่ DEBUG เมื่อทำงานถึงบรรทัดถัดไป

การตั้งค่าให้กับรีจิสเตอร์ IP

เนื่องจากโปรแกรม DEBUG จะประมวลผลคำสั่งที่ตำแหน่ง CS:IP เสมอ แต่เมื่อโปรแกรมทำงานเสร็จ รีจิสเตอร์ IP จะชี้ที่ออฟเซ็ตของคำสั่งถัดไปซึ่งเป็นคำสั่งที่ไม่ใช่ส่วนของโปรแกรมของผู้เขียน ดังนั้นภายหลังการทำงานของโปรแกรม ผู้เขียนโปรแกรมควรตรวจสอบว่ารีจิสเตอร์ IP ชี้ไปยังจุดเริ่มต้นของโปรแกรมที่เขียนไว้หรือไม่ ซึ่งโดยปกติแล้วโปรแกรมจะเริ่มต้นที่ออฟเซ็ต 100h ถ้ารีจิสเตอร์ IP ชี้ไปที่อื่นและตั้งค่าให้รีจิสเตอร์ IP ได้โดยใช้คำสั่ง R

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

ตัวอย่างการทดลองโปรแกรมภาษาแอสแซมบลี

ยกตัวอย่างการทดลองโปรแกรมภาษาแอสแซมบลีต่อไปนี้ด้วยโปรแกรม 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

ผู้เขียนโปรแกรมสามารถที่จะป้อนโปรแกรมและสั่งให้ DEBUG แปลโปรแกรมลงในหน่วยความจำโดยใช้คำสั่ง A ไม่ระบุแอดเดรสเริ่มต้นในครั้งแรกโปรแกรมจะถูกแปลลงในตำแหน่ง CS:100h และถ้าสั่งครั้งถัดไป โปรแกรมจะแปลตามลำดับต่อเนื่องกับการสั่งครั้งก่อน

```
-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 จะแสดงข้อความขึ้นดังตัวอย่าง จากโปรแกรมดังกล่าวผู้เขียนโปรแกรมจะต้องระบุขนาดของการคัดลอกข้อมูลด้วย โดยสามารถใส่คำสั่งที่แก้ไขแล้วลงไปในบรรทัดใหม่ได้ทันที

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

เมื่อใส่โปรแกรมเสร็จจะต้องกดปุ่ม Enter ว่าง ๆ ไปเพื่อบอกโปรแกรม DEBUG ว่าโปรแกรมสิ้นสุดแล้วและสามารถเรียกดูโปรแกรมที่ใส่ลงไปได้ด้วยคำสั่ง 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
```

ผู้เขียนโปรแกรมจะเห็นทั้งโปรแกรมภาษาแอสแซมบลีและโปรแกรมภาษาเครื่องที่แปลแล้ว

ติดตามการทำงานของโปรแกรม

ผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรมทำงานโดยใช้คำสั่ง G (go) และตรวจสอบค่าในหน่วยความจำได้ แต่โปรแกรมที่จะทดลองนั้นต้องมีการสั่งให้โปรแกรมจบการทำงาน มิฉะนั้นโปรแกรมจะทำงานเลยไปถึงโปรแกรมอื่น ๆ ที่อยู่ในหน่วยความจำ ดังนั้นอาจจะใส่คำสั่ง INT 20h ปิดท้ายโปรแกรมไว้เพื่อสั่งให้โปรแกรมจบการทำงาน จากตัวอย่าง การแสดงแอดเดรสถัดไปของคำสั่งคือออฟเซ็ตที่ 0120h (สามารถหาได้โดยใช้คำสั่ง U แล้วสังเกตค่าออฟเซ็ตของคำสั่งอื่น ๆ ถัดจากโปรแกรมที่ป้อน) ดังนั้นผู้เขียนโปรแกรมจะต้องป้อนคำสั่ง 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
                ...
```

เมื่อใส่คำสั่งให้โปรแกรมจบการทำงานแล้ว ผู้เขียนโปรแกรมสามารถใช้คำสั่ง G (go) เพื่อสั่งให้โปรแกรมทำงานและใช้คำสั่ง D (dump) เพื่อสังเกตค่าในหน่วยความจำหรือ คำสั่ง R (register) เพื่อดูค่าในรีจิสเตอร์อีกทีได้

```
-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.SQRWV..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
```

จะสังเกตเห็นว่าเมื่อมีการลับมายังโปรแกรม DEBUG หลังจากโปรแกรมที่ผู้เขียนป้อนเข้าไปทำงานเรียบร้อยแล้ว รีจิสเตอร์ต่าง ๆ จะถูกกำหนดค่าเริ่มต้องใหม่รวมทั้งรีจิสเตอร์ IP ด้วย ดังนั้นถ้าสังเกตผลของการทำงานจะได้จากค่าจากหน่วยความจำเท่านั้น อีกทั้งยังสามารถสั่งให้โปรแกรมทำงานทีละบรรทัดโดยใช้คำสั่ง T หรือ P แต่จะต้องระวังในเรื่องของการตั้งค่าเริ่มต้นของรีจิสเตอร์ IP ให้มาชี้หรือแสดงที่ตำแหน่งเริ่มต้นของโปรแกรมด้วย

```
-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
-
```

สังเกตว่าในคำสั่งสุดท้ายผู้เขียนโปรแกรมจะใช้คำสั่ง P เพราะคำสั่ง INT 20h เป็นการเรียกใช้บริการของระบบ และโปรแกรมจะกระโดดไปทำงานที่โปรแกรมย่อยของระบบซึ่ง

อาจจะยาวมาก ถ้าใช้คำสั่ง T จะได้เห็นการกระโดดไปทำงาน ณ ตำแหน่งที่ต้องการและใช้คำสั่ง G เพื่อให้โปรแกรมทำงานจนจบต่อจากการทำงานนั้นก็ได้

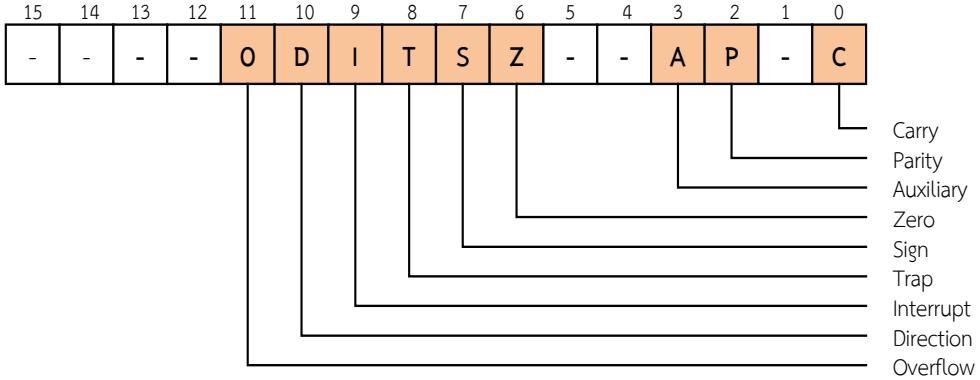
```
-T
AX=5678  BX=0204  CX=1234  DX=7856  SP=FFE8  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=0120  NV UP DI PL NZ NA
PO NC
14FF:0120 CD20        INT   20
-T
AX=5678  BX=0204  CX=1234  DX=7856  SP=FFE2  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=00C9  IP=0FA8  NV UP 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 ผู้เขียนโปรแกรมควรสังเกตค่าของรีจิสเตอร์ CS และ IP ก่อนที่จะเริ่มสั่งให้โปรแกรมทำงานใหม่เสมอ โดยปกติในโปรแกรม DEBUG ค่าของรีจิสเตอร์ CS จะมีค่าเท่ากันกับเซกเมนต์รีจิสเตอร์ตัวอื่น ๆ (DS ES และ SS) และมักนิยมป้อนหรือเขียนโปรแกรมที่จะทดสอบลงใน ณ ตำแหน่ง CS:100h เป็นตำแหน่งเริ่มต้นในการเขียนโปรแกรม

การใช้คำสั่งคำนวณทางคณิตศาสตร์และการแสดงสถานะของผลลัพธ์ของการคำนวณนั้นในแฟล็ก สถานะที่เก็บในแฟล็กจะใช้สำหรับการจัดการกับผลการคำนวณนั้น ๆ รวมถึงใช้ในคำสั่งเกี่ยวกับการกระโดดแบบมีเงื่อนไขด้วย

6.4 แฟล็ก (Flags)

แฟล็กเปรียบเสมือนรีจิสเตอร์ตัวหนึ่ง แต่แทนที่จะใช้เก็บค่าต่าง ๆ แฟล็กจะเก็บสถานะของการคำนวณทางคณิตศาสตร์ที่ผ่านมา ตัวอย่างของสถานะของการคำนวณ เช่น การมีบิตทดมีการเก็บค่าล้นหลักหรือผลลัพธ์มีค่าเป็นศูนย์เป็นต้น ใน 8086 แฟล็กจะมีขนาด 16 บิต โดยในแต่ละบิตจะเก็บค่าของสถานะการคำนวณแบบหนึ่ง ๆ แสดงได้ดังภาพที่ 6.4



ภาพที่ 6.2 แสดงแฟล็กต่าง ๆ

ค่าในบิตของแฟล็กนั้น ๆ จะมีค่าเป็น 1 เมื่อสถานะนั้นเป็นจริง จะเรียกสถานะที่แฟล็กเป็น 1 ว่า แฟล็กเซ็ต (flag set) และถ้าแฟล็กมีค่าเป็นศูนย์ (0) เรียกว่าแฟล็กเคลียร์ (flag cleared) โดยทั่วไปแล้วคำสั่งที่จะมีผลกับแฟล็กจะเป็นคำสั่งเกี่ยวกับการคำนวณทางคณิตศาสตร์ ส่วนคำสั่งในกลุ่มของการโอนย้ายข้อมูล เช่น คำสั่ง MOV จะไม่เปลี่ยนแปลงค่าในแฟล็ก

ความหมายของแฟล็ก แต่ละบิตเป็นดังต่อไปนี้

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. พาริตี้แฟล็ก (Parity flag)

พาริตี้แฟล็กจะมีค่าเป็นหนึ่งเมื่อในผลลัพธ์มีจำนวนบิตที่มีค่าเป็น 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

แฟล็กเครื่องหมายจะเซ็ตเมื่อผลลัพธ์มีค่าเป็นลบ (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)

แฟล็กทดจะเซ็ตเมื่อการคำนวณมีการทดหรือมีการยืม โดยพิจารณาค่าของข้อมูลแบบ*ไม่คิดเครื่องหมาย* แฟล็กทดนี้ยังใช้ในการเก็บบิตข้อมูลในคำสั่งเกี่ยวกับการเลื่อนบิตด้วย

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. โอเวอร์โฟล์แฟล็ก (Overflow flag)

ในการพิจารณาโอเวอร์โฟล์แฟล็กพิจารณาค่าของข้อมูลเป็นแบบคิดเครื่องหมาย โดยโอเวอร์โฟล์แฟล็กจะมีค่าเป็นหนึ่งเมื่อผลลัพธ์มีความผิดพลาด 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. อินเตอร์รัพท์แฟล็ก (Interrupt Flag)

แฟล็กนี้ใช้ระบุว่าหน่วยประมวลผลจะตอบสนองการขัดจังหวะจากอุปกรณ์ฮาร์ดแวร์หรือไม่

คำสั่งเกี่ยวกับการกำหนดค่าของแฟล็ก

สามารถกำหนดค่าของแฟล็กทด แฟล็กทิศทาง และอินเตอร์รัพท์แฟล็กได้โดยใช้คำสั่งต่อไปนี้

ตารางที่ 6.1 คำสั่งสำหรับการกำหนดค่าแฟล็ก

แฟล็ก (FLAG)	คำสั่งในการเซต (SET)	คำสั่งในการเคลียร์ (CLEAR)
carry flag	STC	CLC
direction flag	STD	CLD
interrupt flag	STI	CLI

6.5 คำสั่งทางคณิตศาสตร์

คำสั่งทางคณิตศาสตร์ใน 8086 ที่เราจะศึกษาในวิชานี้แบ่งได้เป็นกลุ่มใหญ่ ๆ 3 กลุ่มดังนี้

6.5.1 กลุ่มคำสั่งบวกและลบ

คำสั่งเพิ่มและลดค่า : INC [Increment] และ DEC [Decrement]

คำสั่ง INC จะเพิ่มค่าของโอเปอร์แรนด์ขึ้นหนึ่ง ส่วนคำสั่ง DEC จะลดค่าของโอเปอร์แรนด์ลงหนึ่ง คำสั่งนี้มีผลกระทบกับแฟล็กทั้งหมด***ยกเว้นแฟล็กทด***

รูปแบบของทั่วไปของคำสั่ง INC และ DEC เป็นดังนี้

INC	*register*	DEC	*register*
INC	*memory*	DEC	*memory*

ตัวอย่างคำสั่ง

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

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

คำสั่งบวกจะนำค่าของตัวถูกดำเนินการ (Operand) ตัวที่สองมาบวกกับค่าของโอเปอร์แรนด์ตัวที่หนึ่งแล้วนำค่าที่ได้เก็บในโอเปอร์แรนด์อีกตัวที่หนึ่ง โดยที่คำสั่ง ADD และ ADC มีผลกระทบกับแฟล็กทางคณิตศาสตร์ทุกแฟล็ก เรานิยมใช้คำสั่ง ADC ในการบวกข้อมูลที่ต้องนำบิตที่ทดจากการบวกครั้งก่อนมารวมด้วย เช่น การบวกข้อมูลที่เก็บอยู่ในหลายรีจิสเตอร์ที่อยู่ต่อเนื่องกันเป็นต้น (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

รูปแบบของทั่วไปของคำสั่ง ADD และ ADC เป็นดังนี้

ADD	*register,number*	ADC	*register,number*
ADD	*memory,number*	ADC	*memory,number*
ADD	*register,register*	ADC	*register,register*
ADD	*register,memory*	ADC	*register,memory*
ADD	*memory,register*	ADC	*memory,register*

ตัวอย่างคำสั่ง

```
MOV     AL,10h          รีจิสเตอร์ AL = 30h
ADD     AL,20h
MOV     BX,200h         ข้อมูลแบบเวิร์ด ณ ตำแหน่ง [DS:BX] = 80h
MOV     WORD PTR [BX],10h   ทำการบวก 1234 5678h เข้ากับ 8765
ADD     WORD PTR [BX],70h   4321h โดยผลลัพธ์ที่ได้ 16 บิตบนจะเก็บที่
MOV     AX,5678h        รีจิสเตอร์ DX ส่วน 16 บิตล่างจะเก็บที่
MOV     DX,1234h        รีจิสเตอร์ AX และใช้คำสั่ง ADC ในการบวก
ADD     AX,4321h        ครั้งที่สองเพื่อนำตัวทดจากการบวกครั้งแรก
ADC     DX,8765h        มารวมด้วย
```

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	*register,number*	SBB	*register,number*
SUB	*memory,number*	SBB	*memory,number*
SUB	*register,register*	SBB	*register,register*
SUB	*register,memory*	SBB	*register,memory*
SUB	*memory,register*	SBB	*memory,register*

ตัวอย่างคำสั่ง

```
MOV     CX,10h              รีจิสเตอร์ CX จะถูกลดค่าลงเท่ากับข้อมูลแบบ
SUB     CX,[200h]           เวิร์ดในแอดเดรส [DS:200h].
MOV     BX,202h
MOV     DX,345h
SUB     WORD PTR [BX],DX    ข้อมูลแบบเวิร์ด ณ ตำแหน่ง [DS:BX] มีค่าลดลง
MOV     AX,0AAFFh           เป็น 345h
MOV     DX,0BBCCh
SUB     BX,AX               ทำการลบค่าของข้อมูลขนาด 32 บิตที่เก็บที่ CX
SBB     CX,DX               ด้วยค่าขนาด 32 บิตในรีจิสเตอร์ DX และ AX
```

คำสั่งเปรียบเทียบ : CMP [Compare]

คำสั่ง CMP จะมีการทำงานเหมือนกับคำสั่ง SUB ทุกประการ แต่จะไม่มีการเปลี่ยนค่าในโอเปอร์แรนด์ใด ๆ โดยผลลัพธ์ที่แท้จริงของคำสั่งนี้คือการเปลี่ยนค่าในแฟล็กต่าง ๆ เพื่อแสดงผลลัพธ์ของการลบ ซึ่งจะใช้คำสั่งนี้ในการเปรียบเทียบค่าต่าง ๆ และนำผลที่ได้ในแฟล็กไปใช้ในการกำหนดเงื่อนไขของการกระโดด

รูปแบบของคำสั่ง CMP จะเหมือนคำสั่ง SUB โดยมีรูปแบบทั่วไปเป็นดังนี้

```
CMP     register,number
CMP     memory,number
CMP     register,register
CMP     register,memory
CMP     memory,register
```

ตัวอย่างคำสั่ง

```
MOV     CX,10h              ค่าในริจีสเตอร์ CX จะคงค่าเดิม แต่ปรับค่าแฟล็กใหม่
CMP     CX,20h              โดย S-flag=1, C-flag=1, O-flag=0
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
```

6.5.2 กลุ่มคำสั่งคูณและหาร

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

การคูณใน 8086 นั้นค่าของตัวตั้งของการคูณ และค่าผลลัพธ์ของการคูณนั้นจะต้องเก็บในรีจิสเตอร์ที่กำหนดไว้ โดยขึ้นกับขนาดของการคูณ รีจิสเตอร์ที่กำหนดไว้เป็นดังนี้

การคูณข้อมูล 8บิต :ตัวตั้ง AL ผลลัพธ์ AX

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

โดยจะระบุตัวคูณและขนาดของการคูณที่โอเปอร์แรนด์ของคำสั่ง IMUL หรือ MUL ทั้งสองคำสั่งมีรูปแบบดังนี้

IMUL *register* MUL *register*

IMUL *memory* MUL *memory*

ตัวอย่างคำสั่ง

```
MOV     AL,17h          ทำการคูณ 17h ด้วย 13h แบบ 8 บิต
MOV     CL,13h
IMUL    CL              ผลลัพธ์จะมีขนาด 16 บิต เก็บในรีจิสเตอร์ AX
MOV     BX,1234h        นำผลลัพธ์ของ 17h คูณกับ 13h มาคูณด้วย 1234h
IMUL    BX              ค่าใน DX,AX มีค่าเท่ากับ (17h x 13h) x 1234h
```

คำสั่ง MUL และ IMUL จะมีผลกระทบกับแฟล็กทดและโอเวอร์โฟล์แฟล็กเท่านั้น

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	หาร 4022h ด้วย 1000h โดยการหารดังกล่าวเป็นการหาร
MOV	DX,0000h	แบบ 16 บิต จึงต้องกำหนดค่าใน DX,AX
MOV	CX,1000h	
DIV	CX	ผลลัพธ์จากการหารจะมีขนาด 16 บิต เก็บในรีจิสเตอร์ AX โดยเศษจะเก็บใน DX. AX = 4 และ DX = 22h
MOV	BL,3h	นำผลลัพธ์มาหารด้วย 3h
DIV	BL	ค่าใน AL จะเก็บผลหาร มีค่าเท่ากับ 1 และ AH จะเก็บเศษ โดยมีค่าเท่ากับ 1 เช่นกัน

คำสั่ง DIV และ IDIV จะไม่มีผลกระทบกับแฟล็กใด ๆ แต่ถ้าในการหารมีการหารด้วยศูนย์(0) หรือเกิดการหารที่ไม่สามารถเก็บผลลัพธ์ลงในรีจิสเตอร์ที่ต้องการได้ เช่น การหาร 1234 5678h ด้วย 2h หน่วยประมวลผลจะสร้างสัญญาณในการขัดจังหวะขึ้นเพื่อแจ้งกับโปรแกรมที่จัดการระบบต่อไป การเกิดการขัดจังหวะในลักษณะนี้เรียกว่าเกิด **Exception**

6.6 กลุ่มคำสั่งแปลงขนาดตัวเลข

จากข้อจำกัดของการใช้คำสั่งหารที่ตัวตั้งจะต้องมีขนาดมากกว่าตัวหาร ทำให้ในบางครั้งจะต้องกำหนดการหารข้อมูลให้มีขนาดเท่ากัน โดยจะต้องแปลงตัวตั้งให้มีขนาดที่เหมาะสมเสียก่อน สำหรับการแปลงขนาดของเลขไม่คิดเครื่องหมายนั้น เราสามารถกระทำได้โดยกำหนดให้ข้อมูลนัยสำคัญสูงที่ขยายเพิ่มมานั้นมีค่าเป็นศูนย์ ตัวอย่างเช่น การขยาย AL ที่เป็นตัวเลขแบบไม่คิดเครื่องหมายให้เป็นข้อมูล 16 บิตใน AX สามารถกระทำได้โดยกำหนดค่าศูนย์ (0) ให้กับ 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                     หลังจากแปลงแล้วค่า AX=0022h
MOV     AL,F0h          AL = F0h = –16
CBW                     หลังจากแปลงแล้วค่า AX=FFF0h = –16
```

คำสั่งแปลงจากเวิร์ดเป็นดับเบิลเวิร์ด : CWD [Convert word to doubleword]

คำสั่งนี้จะแปลงเลขแบบคิดเครื่องหมายขนาด 16 บิตใน AX เป็นเลขคิดเครื่องหมายขนาด 32 บิตใน DX,AX

รูปแบบของคำสั่ง

CWD

ตัวอย่างคำสั่ง

```
MOV     AX,3422h        AX = 3422h
CWD                     หลังจากแปลงแล้วค่า DX = 0000h , AX=3422h

MOV     AX,FFF0h        AX = FFF0h = –16
CWD                     หลังจากแปลงแล้วค่าา DX = FFFFh, AX=FFF0h
```

ตัวอย่างการใช้คำสั่งทางคณิตศาสตร์

ตัวอย่างที่ 1 คำนวณค่า $AL^2 + BL^2$ คิดตัวเลขแบบไม่คิดเครื่องหมายโดยให้ผลลัพธ์เป็นเลข 16 บิต เก็บที่รีจิสเตอร์ AX

```
MUL     AL              หาค่า AL*AL เสียก่อน
MOV     CX, AX
MOV     AL, BL          นำค่าไปเก็บไว้ใน CX เพราะรีจิสเตอร์ AL ต้องใช้ในการ
MUL     BL              คำนวณ BL*BL
ADD     AX, CX          ได้ผลลัพธ์แล้วนำค่า AL*AL เก็บใน CX มาบวกกัน
```

ตัวอย่างที่ 2 คำนวณค่า (CL*BL+DX)/SI โดยคิดเป็นเลขคิดเครื่องหมาย

```
MOV     AL,CL       หาค่าของ CL*BL โดยผลลัพธ์เก็บในรีจีสเตอร์ AX
MUL     BL
ADD     AX,DX       นำค่าใน DX มาบวกเข้ากับค่าใน AX
CWD                 ขยายข้อมูลใน AX เป็น 32 บิตใน DX,AX โดยนำ SI มาหาร
IDIV    SI          ผลลัพธ์จะเก็บไว้ใน AX เศษของการหารเก็บไว้ใน DX
```

ตัวอย่างที่ 3 คำนวณค่า (DX+AX*BX)/(DI–CX) โดยคิดเป็นเลขคิดเครื่องหมาย

```
MOV     SI,DX       นำค่าของ DX ไปเก็บที่ SI เสียก่อนเนื่องจาก DX จะถูก
MUL     BX          การคำนวณ AX*BX. เมื่อคำนวณ AX*BX แล้วนำ SI มา
ADD     AX,SI       โดยต้องนำตัวทดของการบวก SI กับ AX มาทดยัง DX
ADC     DX,0
SUB     DI,CX       ลบ DI ด้วย CX
IDIV    DI          นำไปหารด้วย DI
```

ผลกระทบของคำสั่งทางคณิตศาสตร์ต่อแฟล็ก

ตารางที่ 6.2 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*

การเปลี่ยนแปลงของแฟล็กในการทำงานของคำสั่งต่าง 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

คำสั่งภาษาแอสแซมบลีเบื้องต้น มีคำสั่งเกี่ยวกับการโอนย้ายข้อมูล อยู่ด้วยกัน 2 แบบ คือ

1. การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ เป็นการโอนย้ายข้อมูลระหว่างรีจิสเตอร์สามารถทำได้ถ้าขนาดของรีจิสเตอร์ของทั้งคู่เท่ากัน แต่ยังมีเซกเมนต์รีจิสเตอร์บางตัวซึ่งไม่ควรเข้าไปกำหนดค่าโดยตรง เช่น CS หรือ SS

2. การโอนย้ายข้อมูลกับหน่วยความจำ โดยปกติการโอนย้ายข้อมูลกับหน่วยความจำนั้น เราจะระบุเฉพาะออฟเซ็ตของหน่วยความจำที่เราต้องการจะโอนย้ายข้อมูลด้วย ออฟเซ็ตนั้นจะถูกนำมาประกอบกับเซกเมนต์รีจิสเตอร์ DS เพื่อเป็นตำแหน่งในหน่วยความจำที่แท้จริงที่จะอ่านเขียนข้อมูลด้วย

ดีบัก (Debug) คือ โปรแกรมที่พัฒนาเพื่อแก้ไขปัญหาพื้นฐานในระบบปฏิบัติการดอส (DOS = Disk Operation System) เป็นโปรแกรมสำหรับแก้ไขแฟ้มอย่างง่าย เป็นคำสั่งภายนอก (External Command) ซึ่งเป็นที่นิยมใช้งานในกลุ่มนักพัฒนาโปรแกรมคอมพิวเตอร์มาตั้งแต่ยุคระบบปฏิบัติการดอส (DOS) เครื่องมือในการทดลองการโปรแกรมภาษาแอสแซมบลีก็คือโปรแกรม DEBUG เป็นโปรแกรมสารพัดประโยชน์สำหรับการทดลองเกี่ยวกับการเขียนโปรแกรมภาษาแอสแซมบลี โปรแกรมนี้มีอยู่ทั้งในระบบปฏิบัติการ DOS และ Windows

แฟล็ก (Flags) เป็นรีจิสเตอร์ขนาด 16 บิตที่ใช้บ่งบอกผลลัพธ์ที่ได้จากการคำนวณ คำสั่งที่เกี่ยวข้องกับการคำนวณจะส่งผลกระทบให้แฟล็กเปลี่ยนแปลงค่าไป การคัดลอกข้อมูลจะไม่ทำให้แฟล็กเปลี่ยนแปลง ค่าแฟล็กเปรียบเสมือนรีจิสเตอร์ตัวหนึ่ง แต่แทนที่จะใช้เก็บค่าต่าง ๆ แฟล็กจะเก็บสถานะของการคำนวณทางคณิตศาสตร์ที่ผ่านมา

คำถามทบทวน

1. จงให้คำจำกัดความของคำว่าตัวถูกดำเนินการ (operand)
2. โปรแกรม DEBUG คืออะไรและมีหน้าที่อย่างไรในภาษาโปรแกรมภาษาแอสแซมบลี
3. รีจีสเตอร์ AX มีหน้าที่อย่างไรในคำสั่ง MOV
4. คำสั่งต่อไปนี้ทำงานอย่างไร
   - คำสั่ง G (go)
   - คำสั่ง D (dump)
   - คำสั่ง R (register)
5. คำสั่ง A (assemble) มีหน้าที่และทำงานอย่างไร
6. คำสั่งภาษาแอสแซมบลีที่ใช้สำหรับการโอนย้ายข้อมูลมีกี่แบบอะไรบ้าง
7. จงอธิบายความหมายและหน้าที่ของแฟล็ก (Flags) ต่าง ๆ ว่ามีหน้าที่อย่างไร
8. จงอธิบายความแตกต่างระหว่างคำสั่งลบ SUB และลบรวมตัวยืม SBB
9. จงอธิบายความแตกต่างระหว่างคำสั่งคูณแบบคิดเครื่องหมาย IMUL และไม่คิดเครื่องหมาย MUL
10. จงอธิบายความหมายของคำสั่งเปรียบเทียบ CMP และคำสั่งแปลงจากไบต์เป็นเวิร์ด CBW
11. จากตัวอย่างคำสั่งข้างล่างจงอธิบายความหมายของคำสั่งแต่ละบรรทัด

    ตัวอย่างคำสั่ง

```
MOV   AL,25h
MOV   CL,13h
IMUL  CL
MOV   BX,3456h
IMUL  BX
```

12. จากตัวอย่างคำสั่งข้างล่างจงอธิบายความหมายของคำสั่งแต่ละบรรทัด

    ตัวอย่างคำสั่ง

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

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 30 ธันวาคม 2556,จาก :http://rirs3.royin.go.th/coinages/

โปรแกรมดีบัก. (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 30 ธันวาคม 2556, จาก: http://th.wikipedia.org/wiki/

แฟล็ก. (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 30 ธันวาคม 2556, จาก: http://th.wikipe.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย–ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 7

หัวข้อเรื่อง โปรแกรมภาษาแอสแซมบลีเบื้องต้น
(Assembly Language Programming)

รายละเอียด

ศึกษาการเขียนโปรแกรมภาษาแอสแซมบลีแบบเต็มรูปแบบ นั่นคือจะเขียนโปรแกรมภาษาแอสแซมบลีที่เป็นโปรแกรมที่ทำงานได้จริง มีการกำหนดรูปแบบต่างๆ ครบถ้วน โปรแกรมภาษาแอสแซมบลีที่จะเขียนต่อไปนี้ไม่ได้ทำงานบนโปรแกรม DEBUG เท่านั้น จะต้องใช้ assembler แปลโปรแกรมที่เขียนขึ้นให้อยู่ในภาพที่คอมพิวเตอร์สามารถนำไปประมวลผลได้เสียก่อน

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 7 การจัดการเรียนการสอน จะเกี่ยวข้องกับรูปแบบของโปรแกรมภาษาแอสแซมบลีแบบเก่า เปรียบเทียบกับรูปแบบของโปรแกรมภาษาแอสแซมบลีแบบใหม่ ขั้นตอนการแปลโปรแกรม การเรียกใช้บริการของ DOS ตารางรหัสแอสกี ซึ่งการเขียนโปรแกรมภาษาแอสแซมบลีแบบเต็มรูปแบบ นั่นคือจะเขียนโปรแกรมภาษาแอสแซมบลีที่เป็นโปรแกรมที่ทำงานได้จริง มีการกำหนดรูปแบบต่างๆ ครบถ้วน โปรแกรมภาษาแอสแซมบลีที่จะเขียนต่อไปนี้ไม่ได้ทำงานบนโปรแกรม DEBUG เท่านั้น จะต้องใช้ assembler แปลโปรแกรมที่เขียนขึ้นให้อยู่ในภาพที่คอมพิวเตอร์สามารถนำไปประมวลผลได้เสียก่อน

7.1 รูปแบบของโปรแกรมภาษาแอสแซมบลีแบบเก่า

โปรแกรมที่ทำงานในเครื่องคอมพิวเตอร์ซึ่งใช้หน่วยประมวลผลตระกูล 80x86 นั้นจะมีการแบ่งโปรแกรมทั้งหมดเป็นเซกเมนต์ย่อย ๆ เช่น Code segment Data segment หรือ Stack segment ดังนั้นในโปรแกรมภาษาแอสแซมบลีที่เขียนจะประกอบไปด้วยเซกเมนต์ต่าง 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 เซกเมนต์ดังกล่าวนี้ถูกประกาศด้วย**คำสั่งเทียม segment** การที่เรียกคำสั่ง segment ว่าคำสั่งเทียมเพราะคำสั่งนี้เป็นคำสั่งที่ผู้เขียนโปรแกรมระบุให้โปรแกรม assembler แปลโปรแกรมตามลักษณะที่กำหนด โดยจะไม่มีคำสั่งภาษาเครื่องถูกสร้างจากคำสั่งกลุ่มนี้ ตัวอย่างอื่น ๆ ของคำสั่งเทียมคือ db assume และ org เป็นต้น

การประกาศเซกเมนต์

การประกาศเซกเมนต์ในโปรแกรมภาษาเอสเซมบลี ใช้คู่คำสั่งเทียม segment และ ends โดยมีลักษณะการประกาศดังนี้

segment_name segment

...

segment_name ends

จากตัวอย่างได้ประกาศเซกเมนต์ cseg dseg และ sseg **คำสั่งเทียม stack** ระบุให้ระบบใช้เซกเมนต์ sseg เป็นแสต็กของโปรแกรม **คำสั่งเทียม assume** เป็นการระบุให้ assembler ได้ทราบว่าเซกเมนต์ที่ประกาศนั้นจะให้ระบบพิจารณาว่าทำหน้าที่อะไรและมีเซกเมนต์รีจิสเตอร์ใดเป็นตัวเก็บค่าเซกเมนต์ จากตัวอย่างประกาศให้ assembler ทราบว่าเซกเมนต์ cseg จะชี้โดยรีจิสเตอร์ CS เซกมนต์ 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

โปรแกรมภาษาแอสแซมบลีจะประกอบไปด้วยการประกาศเซกเมนต์ต่าง ๆ และจะสิ้นสุดโปรแกรมที่**คำสั่งเทียม end** หลังคำสั่งเทียม end จะระบุจุดเริ่มต้นของโปรแกรม ในโปรแกรมตัวอย่างระบุจุดเริ่มต้นของโปรแกรมที่เลเบล start ที่ประกาศไว้ที่ตอนต้นของโปรแกรม การประกาศเลเบลสามารถทำได้ดังนี้

label_name:

ระบบจะจดจำตำแหน่งของเลเบลที่ประกาศไว้และจะนำแอดเดรสของเลเบลไปแทนที่ให้โดยอัตโนมัติ การที่โปรแกรม assembler จัดการเรื่องเกี่ยวกับเลเบลในโปรแกรมภาษาแอสแซมบลีนั้น นับเป็นการเพิ่มความสะดวกให้กับผู้เขียนโปรแกรมเป็นอย่างมาก

การประกาศข้อมูล

ภายในเซกเมนต์ข้อมูลสามารถประกาศข้อมูลต่าง ๆ ได้จากโปรแกรมตัวอย่างประกาศข้อมูลเป็นข้อความที่จะให้โปรแกรมพิมพ์ออกมา จะศึกษารูปแบบการประกาศข้อมูลในบทต่อไป

การใส่หมายเหตุ

หลังเครื่องหมาย **';'** assembler จะตีความว่าเป็นหมายเหตุ การใส่หมายเหตุจะช่วยทำให้โปรแกรมอ่านง่ายขึ้น จากตัวอย่างโปรแกรมข้างต้น 3 บรรทัดแรกจะเป็นหมายเหตุ

การสั่งให้โปรแกรมจบการทำงาน

โปรแกรมจะจบการทำงานเมื่อสั่งให้โปรแกรมจบการทำงานเท่านั้น ถ้าไม่ได้สั่งให้จบการทำงานเมื่อจบโปรแกรมแล้ว หน่วยประมวลผลจะทำงานคำสั่งอื่น ๆ ที่อยู่ในหน่วยความจำต่อจากโปรแกรมของไปเรื่อยๆ ในโปรแกรม DEBUG เรียกใช้คำสั่ง INT 20h เพื่อให้โปรแกรมจบการทำงาน แต่ในโปรแกรมภาษาแอสแซมบลีทั่วไปจะเรียกใช้บริการหมายเลข 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 จะกล่าวถึงในเนื้อหาสัปดาห์ถัดไป

ตัวอย่าง โครงร่างของโปรแกรมภาษาแอสแซมบลี

```
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
```

7.2 รูปแบบของโปรแกรมภาษาแอสแซมบลีแบบใหม่

รูปแบบของโปรแกรมภาษาแอสแซมบลีที่ใช้ในตอนต้นนั้นเป็นรูปแบบเก่า ในปัจจุบันโปรแกรมassembler ส่วนใหญ่มีรูปแบบในการประกาศเซกเมนต์ต่าง ๆ ให้ง่ายขึ้น โดยใช้คุณสมบัติของ MACRO ต่าง ๆโปรแกรมตัวอย่างแรกของเมื่อนำมาเขียนในรูปแบบใหม่จะได้เป็น

```
; 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
```

จะสังเกตได้ว่าโปรแกรมกระทัดรัดขึ้นมาก ข้อแตกต่างของโปรแกรมที่เขียนในรูปแบบใหม่คือชื่อของเซกเมนต์ต่าง ๆ จะถูกกำหนดให้โดยอัตโนมัติ จะสังเกตได้ว่าในส่วนของการกำหนดค่า DS ใช้ชื่อของเซกเมนต์ข้อมูลว่า @data เป็นต้น

7.3 การเรียกใช้บริการของ DOS

สามารถเรียกใช้บริการต่าง ๆ ของ DOS ได้โดยผ่านทางการขัดจังหวะหมายเลข 21h DOS ได้จัดสรรบริการ (function) ต่าง ๆ มากมายให้กับผู้เขียนโปรแกรม เมื่อเรียกใช้บริการจะต้องระบุว่าต้องการบริการใด 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 7.1 แสดงบริการของ DOS ที่สำคัญและพารามิเตอร์

หมายเลข	หน้าที่	พารามิเตอร์	หมายเหตุ
01h	รับค่าจากแป้นพิมพ์	Input : AH = 01h Output :AL = รหัสแอสกีของปุ่มที่กด	
02h	แสดงตัวอักษร	Input : AH = 02h DL = รหัสแอสกีของอักษรที่จะแสดง	
05h	พิมพ์ตัวอักษรทางเครื่องพิมพ์	Input : AH = 05h DL = รหัสแอสกีของอักษรที่จะพิมพ์	
07h	อ่านตัวอักษรจากแป้นพิมพ์ แต่ไม่แสดงผล (ไม่ตรวจสอบ Crtl–Break)	Input : AH = 07h Output :AL = รหัสแอสกีของอักษรที่อ่านได้	
08h	อ่านตัวอักษรจากแป้นพิมพ์ แต่ไม่แสดงผล (ตรวจสอบ Crtl–Break)	Input : AH = 07h Output :AL = รหัสแอสกีของอักษรที่อ่านได้	

หมายเลข	หน้าที่	พารามิเตอร์	หมายเหตุ
09h	พิมพ์ข้อความ	Input : AH = 09h DS:DX = ตำแหน่งของข้อความที่ต้องการพิมพ์ ข้อความจบด้วยอักษร '$'	การประกาศข้อมูลในหน่วยความจำจะอธิบายในบทถัดไป
0Ah	อ่านข้อความ	Input : AH = 0Ah DS:DX = ตำแหน่งของบัฟเฟอร์เก็บข้อมูล	รูปแบบของบัฟเฟอร์และการใช้บริการนี้จะอธิบายในบทถัดไป
4Ch	จบโปรแกรม	Input : AH = 4Ch AL = รหัสที่จะส่งคือสู่ระบบ	

7.4 ขั้นตอนการแปลโปรแกรม

ผู้เขียนโปรแกรมจะต้องแปลโปรแกรมที่เขียนขึ้นให้อยู่ในรูปแบบที่สามารถทำงานได้จริง โดยขั้นตอนการแปลโปรแกรมเป็นดังนี้

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

ตัวอย่างการแปลโปรแกรม

จากโปรแกรมตัวอย่าง สมมติว่าโปรแกรมให้เก็บในแฟ้มชื่อ 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 เมื่อแปลโปรแกรมภาษาแอสแซมบลีเรียบร้อยแล้ว จะได้แฟ้มเป้าหมายที่มีนามสกุลเป็น OBJ เช่น จากตัวอย่างจะได้ EX1.OBJ ซึ่งจะให้โปรแกรม 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.
```

จะได้แฟ้มที่มี**นามสกุล EXE** ซึ่งสามารถเรียกใช้ได้จาก DOS prompt

ตัวอย่างโปรแกรม

ตัวอย่างที่ 1

โปรแกรมนี้รับการกดปุ่มจากผู้ใช้โดยใช้บริการหมายเลข 01h แล้วแสดงอักขระที่อ่านได้โดยใช้บริการของ DOS หมายเลข 02h สังเกตว่าโปรแกรมนี้ไม่มีการใช้ข้อมูลในหน่วยความจำ ดังนั้นจึงไม่ต้องประกาศเซกเมนต์ข้อมูล

```
;Example 1
.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

```
; Example 2
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              ;increse DL (next char.)
        mov     ah,02h          ;display it (Function 02h)
        int     21h
        mov     ax,4C00h        ;Exit
        int     21h
cseg    ends
        end     start
```

ตัวอย่างที่ 3

โปรแกรมนี้รับตัวอักษรจากผู้ใช้ จากนั้นแปลงตัวอักษรเล็กให้เป็นตัวอักษรใหญ่โดยการลบค่ารหัสแอสกีด้วย 32 แล้วแสดงอักขระนั้นกับผู้ใช้

```
.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 แต่ไม่แสดงอักษรที่ผู้ใช้ป้อนให้ผู้ใช้เห็น 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
```

7.5 ตารางแสดงรหัสแอสกี

ตารางที่ 7.2 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	l
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

ภาษาแอสแซมบลี (Assembly Language) เป็นภาษาที่ใช้สัญญาลักษณ์ในการสื่อสาร ความหมายภาษาแอสแซมบลีมีลักษณะคำสั่ง ที่ขึ้นกับเครื่องคอมพิวเตอร์ที่ใช้งานและมีการแปลคำสั่งให้เป็นภาษาเครื่องนอกจากภาษาเครื่อง และ ภาษาแอสแซมบลีแล้ว ก็ยังมีภาษาระดับสูง เช่น Basic Cobol Fortran ซึ่งเป็นภาษาที่มีคำสั่งใกล้เคียงกับภาษาอังกฤษมากทำให้ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมได้สะดวกและรวดเร็ว แต่ว่าโปรแกรมที่เขียนด้วยภาษาระดับสูงต้องใช้เนื้อที่เก็บในหน่วยความจำเป็นจำนวนมาก อีกทั้งทำงานได้ช้ากว่าภาษาแอสแซมบลี ดังนั้นภาษาระดับสูงจึงไม่นิยมนำมาประยุกต์ใช้กับการทำงานที่ระบบการควบคุมที่มีความสำคัญมาก

คำสั่งปฏิบัติการของภาษาแอสแซมบลี แบ่งออกเป็น 4 ชนิด คือ

1. Machine instruction เป็นคำสั่งที่ทำให้เกิดการปฏิบัติการ (execution) ชุดของคำสั่งอยู่ใน assembler's instruction

2. Assembler instruction เป็นคำสั่งที่บอกแอสเซมเบลร์ให้ทำการระหว่าง Assembly source program

3. Macro instruction เป็นคำสั่งที่บอกแอสสเซมเบลร์ให้ดำเนินการกับชุดของคำสั่งที่ได้บอกไว้ก่อนแล้ว ซึ่งจากชุดของคำสั่งนี้ แอสเซมเบลร์จะผลิตชุดของคำสั่งซึ่งต่อไปจะดำเนินการเหมือนหนึ่งว่าชุดของคำสั่งนี้เป็นส่วนหนึ่งของ source program แต่เริ่มแรก

4. Pseudo instruction เป็นคำสั่งที่บอกให้แอสเซมเบลร์รู้ว่า ควรปฏิบัติการเช่นไรกับข้อมูลการ branch อย่างมีข้อแม้ แมคโคและ listing ซึ่งปกติแล้วคำสั่งเหล่านี้จะไม่ผลิตคำสั่งภาษาเครื่องให้

คำถามทบทวน

1. โปรแกรมที่ทำงานในเครื่องคอมพิวเตอร์ซึ่งใช้หน่วยประมวลผลตระกูล 80x86 นั้นจะมีการแบ่งโปรแกรมทั้งหมดเป็นเซกเมนต์ย่อย ๆ อะไรบ้าง
2. จงอธิบายขั้นตอนการแปลโปรแกรมในภาษาแอสแซมบลี
3. จงแสดงรูปแบบวิธีการเขียนโปรแกรมภาษาแอสแซมบลีแบบใหม่
4. แฟ้มเป้าหมาย (Object file) คืออะไรและมีความสัมพันธ์กันอย่างไรกับการเขียนโปรแกรมภาษาแอสแซมบลี
5. จงเขียนโปรแกรมนี้รับตัวอักษรจากผู้ใช้ จากนั้นแปลงตัวอักษรใหญ่ให้เป็นตัวอักษรเล็กออกทางหน้าจอคอมพิวเตอร์ (Display on Screen)

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 5 มกราคม 2557,จาก :http://rirs3.royin.go.th/coinages/

รหัสแอสกี. (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 5 มกราคม 2557, จาก: http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย–ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 8

หัวข้อเรื่อง การประกาศข้อมูล คำสั่งกระโดดและการกระทำซ้ำ
(Data Declared, Jump and Iteration Loop)

รายละเอียด

ศึกษาเกี่ยวกับรูปแบบของการเขียนโปรแกรมภาษาแอสแซมบลีและการประกาศเซกเมนต์แล้ว ในบทนี้จะศึกษาเกี่ยวกับการประกาศข้อมูลภายในเซกเมนต์ข้อมูล และการใช้บริการหมายเลข 09h และ 0Ah ของระบบปฏิบัติการ DOS ในการแสดงผลข้อความและการอ่านข้อความจากผู้ใช้ คำสั่งกระโดดและคำสั่งเกี่ยวกับการทำซ้ำ การนำคำสั่งเหล่านี้ไปใช้ในการสร้างโครงสร้างควบคุม (Control Structures) แบบต่าง ๆ

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 8 การจัดการเรียนการสอน จะเกี่ยวข้องกับรูปแบบและวิธีการประกาศข้อมูล การอ้างใช้ข้อมูลที่ประกาศไว้ การอ้างตำแหน่งของข้อมูล การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah การใช้บริการของ DOS หมายเลข 0Ah : การอ่านข้อความ และหมายเลข 09h : การอ่านพิมพ์ข้อความ การใช้งานคำสั่งกระโดดแบบไม่มีเงื่อนไข คำสั่งกระโดดที่พิจารณาค่าจากแฟล็ก คำสั่งกระโดดที่พิจารณาค่าจากรีจิสเตอร์ มีความรู้และความเข้าใจเกี่ยวกับกลุ่มคำสั่งวนรอบ เช่น LOOP, LOOPZ และ LOOPNZ เป็นต้น

8.1 การประกาศข้อมูล

การประกาศข้อมูลหรือตัวแปรในโปรแกรมภาษาแอสแซมบลีนั้น ทำได้โดยประกาศจองเนื้อที่ในหน่วยความจำในเซกเมนต์ข้อมูล แล้วตั้งเลเบลของข้อมูลนั้นไว้ ในการอ้างถึงข้อมูลในหน่วยความจำตำแหน่งนั้น สามารถอ้างโดยใช้เลเบลที่ประกาศไว้ได้ ดังนั้นการประกาศตัวแปรหรือข้อมูลนั้นจะมีลักษณะเช่นเดียวกับการประกาศเลเบลนั่นเอง

คำสั่งเทียมในการประกาศข้อมูล

คำสั่งเทียม (Pseudo Instruction) ที่ใช้ในการประกาศข้อมูลมีหลายคำสั่ง ดังตารางที่ 8.1 แสดงคำสั่งเทียมที่ใช้ในระบุขนาดในการจองหน่วยความจำ (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

ตารางที่ 8.1 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 โดยจะกันเนื้อที่ในหน่วยความจำที่มีขนาดของแต่ละหน่วยตามขนาดที่ระบุในคำสั่ง

รูปแบบของการประกาศข้อมูล

ในการประกาศข้อมูล (ตัวแปร) มักประกาศในเซกเมนต์ข้อมูล โดยจะระบุชื่อของตัวแปรนั้น พร้อมทั้งคำสั่งเทียมที่ใช้ระบุขนาดของข้อมูล จากนั้นจะระบุข้อมูลต่าง ๆ ที่จะใช้ตำแหน่งที่จะจองนั้น รูปแบบในการระบุเป็นดังนี้

variable_name D*x* *data*

ส่วนของโปรแกรมที่ 8.1 การประกาศข้อมูล

ตัวอย่างการประกาศข้อมูล

จากการประกาศข้อมูลในส่วนของโปรแกรมที่ 8.1 จะมีการจัดสรรเนื้อที่ในหน่วยความจำ แสดงได้ดังภาพที่ 8.1 สังเกตว่าในการประกาศข้อมูล data1 กับ data2 นั้นการระบุข้อมูลเหมือนกันแต่ขนาดของข้อมูลต่างกัน ทำให้การจองเนื้อที่ในหน่วยความจำนั้นแตกต่างกันด้วย

```
dseg    segment
data1   db      1,2
data2   dw      1,2
data3   db      'Hi',10,13
```

```
data4   dd      1234h
dseg    ends
```

ส่วนของโปรแกรมที่ 8.1 ตัวอย่างการประกาศข้อมูล

DS:00h	01h	data1
DS:01h	02h	
DS:02h	01h	data2
DS:03h	00h	
DS:04h	02h	
DS:05h	00h	
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	

ภาพที่ 8.1 แสดงการจัดเรียงข้อมูลในหน่วยความจำจากการประกาศในส่วนของโปรแกรมที่ 8.1

การระบุไม่ระบุค่าของข้อมูลที่จองเนื้อที่

สามารถประกาศจองหน่วยความจำโดยไม่ระบุค่าเริ่มต้นได้โดยการระบุค่าเป็น '?' ดังเช่นในส่วนของโปรแกรมที่ 8.2 จะมีการจองเนื้อที่ไว้แต่ไม่มีการกำหนดค่าเริ่มต้น

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

ส่วนของโปรแกรมที่ 8.2 การใช้จองหน่วยความจำโดยไม่ระบุค่าเริ่มต้น

การประกาศข้อมูลที่ซ้ำกัน

สามารถใช้**คำสั่งเทียม dup** เพื่อบอกการซ้ำกันของข้อมูลได้ รูปแบบของคำสั่งเทียม dup มีดังนี้

count dup (*value*)

ตัวอย่าง การประกาศที่ใช้คำสั่งเทียม dup ดังเช่นในส่วนของโปรแกรมที่ 8.3

```
data7    db    10 dup (0)
data8    db    5 dup (4 dup (0))
data9    dw    5 dup (1, 2, 3 dup (4))
data10   db    20 dup (?)
```

ส่วนของโปรแกรมที่ 8.3 การใช้คำสั่งเทียม dup

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

8.2 การอ้างใช้ข้อมูลที่ประกาศไว้

ในการอ้างใช้ข้อมูลหรือตัวแปรที่ประกาศไว้ สามารถอ้างโดยใช้ชื่อของเลเบลที่ประกาศไว้ได้ Assembler จะจัดการนำตำแหน่งของข้อมูลนั้นมาแทนค่าให้โดยอัตโนมัติ ยังสามารถอ้างค่าในหน่วยความจำโดยอ้างสัมพัทธ์กับเลเบลที่กำหนดขึ้นได้ ส่วนของโปรแกรมที่ 8.4 เป็นโปรแกรมที่อ้างใช้ค่าของตัวแปรที่กำหนดในส่วนของโปรแกรมที่ 8.2 โดยหลังจากการทำงานของโปรแกรมค่าในหน่วยความจำจะเปลี่ยนไป แสดงได้ดังภาพที่ 8.2

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	

ภาพที่ 8.2 แสดงการเปลี่ยนแปลงค่าหลังการทำงานของโปรแกรมที่ 8.4

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

ส่วนของโปรแกรมที่ 8.4 ตัวอย่างการเรียกใช้ตัวแปร

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

8.3 การอ้างตำแหน่งของข้อมูล

สามารถอ้างถึงออฟเซ็ตของข้อมูลที่ประกาศไว้ได้โดยใช้คำสั่งเทียม OFFSET ดังส่วนของโปรแกรมที่ 8.5

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

ส่วนของโปรแกรมที่ 8.5 การอ้างตำแหน่งของข้อมูล

การอ้างตำแหน่งข้อมูลโดยคิดสัมพัทธ์กับรีจิสเตอร์ BX

นอกจากการระบุตำแหน่งสัมพัทธ์กับเลเบลโดยใช้ค่าคงที่แล้ว สามารถระบุตำแหน่งของข้อมูลสัมพันธ์กับเลเบลโดยใช้ค่าจากรีจิสเตอร์ BX ได้ ตัวอย่างเช่นส่วนของโปรแกรมที่ 8.6

```
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 8.6 การอ้างตำแหน่งของข้อมูลสัมพัทธ์กับเลเบลโดยใช้ค่าจากรีจิสเตอร์ BX

ในคำสั่ง mov ก่อนบรรทัดที่ 5 อ้างหน่วยความจำโดยสัมพันธ์กับ data3 และค่าใน BX แต่ในคำสั่ง mov บรรทัดสุดท้ายของส่วนของโปรแกรมที่ 8.6 อ้างหน่วยความจำสัมพันธ์กับ BX ซึ่งเก็บออฟเซ็ตของ data3

8.4 การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah

ฟังก์ชั่นหมายเลข 09h และ 0Ah ของ DOS เป็นฟังก์ชั่นที่ต้องมีการส่งแอดเดรสของข้อมูลในหน่วยความจำ การประกาศข้อมูลสำหรับฟังก์ชั่นหมายเลข 09h จะไม่มีความซับซ้อนมากนัก แต่สำหรับฟังก์ชั่นหมายเลข 0Ah การประกาศข้อมูลที่เหมาะสมจะทำให้เขียนโปรแกรมได้ง่ายมากขึ้น

8.5 การใช้บริการของ DOS หมายเลข 09h : การพิมพ์ข้อความ

ฟังก์ชั่นหมายเลข 09h นี้รับข้อมูลป้อนเข้าคือ

AH = 09h

DS : DX = ตำแหน่งของหน่วยความจำของข้อมูลที่จะแสดง โดยข้อมูลนี้จะต้องจบด้วยอักขระ '$'

ถ้าต้องการพิมพ์ข้อความ "Hello world" สามารถประกาศข้อมูลในหน่วยความจำได้ดังนี้

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

ส่วนของโปรแกรมที่ 8.7 ตัวอย่างการประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h

สามารถสั่งแสดงข้อมูลดังกล่าวได้โดย

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

ส่วนของโปรแกรมที่ 8.7 ตัวอย่าง การใช้บริการของ DOS หมายเลข 09h อักขระหมายเลข 10 (Line feed) และ 13 (Carriage Return) คือรหัสควบคุมใช้ในการสั่งให้ขึ้นบรรทัดใหม่

8.6 การใช้บริการของ DOS หมายเลข 0Ah : การอ่านข้อความ

ฟังก์ชั่นนี้จะอ่านข้อความจากผู้ใช้จนกระทั่งผู้ใช้กดปุ่ม Enter โดยข้อมูลป้อนเข้าจะต้องระบุตำแหน่งของหน่วยความจำที่ใช้เก็บข้อมูล (บัฟเฟอร์) ของข้อความ ฟังก์ชั่นหมายเลข 0Ah นี้รับข้อมูลป้อนเข้าคือ

AH = 0Ah

DS : DX = ตำแหน่งของหน่วยความจำที่จะใช้เก็บข้อความ (บัฟเฟอร์)

บัฟเฟอร์จะต้องมีรูปแบบดังนี้

1. ไบต์แรกของหน่วยความจำเก็บค่าความยาวสูงสุดของข้อความที่อ่านได้ ความยาวนี้จะรวมรหัสขึ้นบรรทัดใหม่ด้วย

2. DOS จะเขียนความยาวจริงของข้อความที่อ่านเข้ามาได้ในไบต์ที่สอง

3. สำหรับไบต์ถัด ๆ ไปจะเป็นรหัสแอสกีของข้อความที่อ่านเข้ามา

การประกาศข้อมูลสำหรับการเรียกใช้ฟังก์ชั่นนี้จะสามารถประกาศได้ดังส่วนของโปรแกรมที่ 8.9

```
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 8.9 การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 0Ah

เมื่อเรียกใช้บริการหมายเลข 0Ah จะส่งตำแหน่งของ maxlen ซึ่งเป็นตำแหน่งเริ่มต้นของบัฟเฟอร์ที่ประกาศไปให้กับ DOS จากนั้นสามารถอ่านความจริงของข้อความที่อ่าน

มาได้ทางตัวแปร msglen ตัวอย่างโปรแกรมที่ 8.10 แสดงการใช้งานบริการหมายเลข 0Ah ในการอ่านข้อความและแสดงข้อความนั้นออกมาโดยใช้บริการหมายเลข 09h ในการรับข้อความนั้นบริการหมายเลข 0Ah จะเก็บอักขระขึ้นบรรทัดใหม่ให้ด้วย ดังนั้นจะต้องเพื่อขนาดบัฟเฟอร์ที่จะให้เก็บข้อความไว้ 1 ไบต์ด้วย แต่ในการคืนค่าความยาวของข้อความมาให้ บริการหมายเลข 0Ah นี้จะใส่ความยาวที่ไม่รวมอักขระขึ้นบรรทัดใหม่นี้ เมื่อรับข้อความเสร็จแล้ว เคอร์เซอร์จะอยู่ที่ต้นบรรทัดที่ป้อนข้อความนั้น ดังนั้นถ้าพิมพ์ข้อความเดิมซ้ำไปอีกครั้งจะทำให้ข้อความทับกันและจะไม่ทราบว่ามีการพิมพ์ข้อความออกมาอย่างถูกต้องหรือไม่ ดังนั้นจึงใช้ฟังก์ชั่นหมายเลข 09h สั่งพิมพ์ชุดอักขระสำหรับการขึ้นบรรทัดใหม่ก่อนที่จะสั่งพิมพ์ข้อความที่รับมา ข้อความที่สั่งพิมพ์ด้วยบริการหมายเลข 09h จะต้องจบด้วยอักขระ ‘$’ ดังนั้นจึงต้องกำหนดค่าในไบต์สุดท้ายของข้อความที่รับมาด้วยอักขระ ‘$’ โปรแกรมนี้จะทำงานผิดพลาดถ้าภายในข้อความมีเครื่องหมาย ‘$’ อยู่ด้วย

```
;
; display string
;
dseg    segment
;string buffer
maxlen  db      30                      ;29 chars + 1 return
msglen  db      ?
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 8.10 แสดงการรับข้อความและแสดงข้อความนั้นกลับมา

คำสั่งกระโดด และคำสั่งเกี่ยวกับการทำซ้ำ คือการนำคำสั่งเหล่านี้ไปใช้ในการสร้างโครงสร้างควบคุม (Control Structures) แบบต่าง ๆ ซึ่งจะศึกษาในสัปดาห์ต่อไป

8.7 คำสั่งกระโดด (Jump Structures)

คำสั่งกระโดดเป็นคำสั่งที่สั่งให้หน่วยประมวลผลกระโดดไปทำงานที่ตำแหน่งอื่น รูปแบบทั่วไปของคำสั่งกระโดดคือ

Jxx label

คำสั่งกระโดดแบ่งได้เป็น 2 กลุ่ม คือ คำสั่งกระโดดแบบไม่มีเงื่อนไข และคำสั่งกระโดดแบบมีเงื่อนไข คำสั่งกระโดดแบบไม่มีเงื่อนไขคือคำสั่ง JMP ส่วนในกลุ่มของคำสั่งกระโดดแบบมีเงื่อนไขแบบคร่าว ๆ ออกเป็นสองกลุ่มคือกลุ่มซึ่งพิจารณาการกระโดดจากค่าในแฟล็ก และกลุ่มที่พิจารณาการกระโดดจากค่าในรีจิสเตอร์ ส่วนใหญ่คำสั่งกระโดดที่พิจารณาค่าในแฟล็กจะใช้ผลลัพธ์ที่ได้จากคำสั่ง CMP คำสั่งกระโดดต่าง ๆ สรุปได้ดังตารางที่ 8.3 แสดงคำสั่งกระโดดต่างๆ

ตารางที่ 8.3 แสดงคำสั่งกระโดดต่างๆ

คำสั่งกระโดด	ความหมาย	เงื่อนไข
คำสั่งกระโดดแบบไม่มีเงื่อนไข		
JMP	Jump	Always
คำสั่งกระโดดที่พิจารณาค่าจากแฟล็ก (Flag)		
Zero flag JZ (JE) JNZ (JNE)	 Jump if Zero (Jump if Equal) Jump if Not Zero (Jump if Not Equal)	 ZF = 1 ZF = 0
Overflow flag JO JNO	 Jump if Overflow Jump if Not Overflow	 OF = 1 OF = 0
Parity flag JPO JPE	 Jump if Parity Odd Jump if Parity Even	 PF = 0 PF = 1
Signs flag JS JNS	 Jump if Sign Jump if No Sign	 SF = 1 SF = 0
Carry flag JC JNC	 Jump if Carry Jump if No Carry	 CF = 1 CF = 0
Comparing unsigned numbers JA (JNBE) JB (JNAE) JAE (JNB) JBE (JNA)	 Jump if Above (Not Below or Equal) Jump if Below (Not Above or Equal) Jump if Above or Equal (Not Below) Jump if Below or Equal (Not Above)	 (CF and ZF) = 0 CF = 1 CF = 0 (CF or ZF) = 1
คำสั่งกระโดดที่พิจารณาค่าจากรีจิสเตอร์		
Testing for CX JCXZ	 Jump if CX is equal to zero	 CX = 0

คำสั่งต่าง ๆ เหล่านี้จะใช้ในการสร้างโครงสร้างควบคุมการทำงานของโปรแกรม โดยจะใช้ประกอบกับคำสั่ง CMP ดังที่ได้กล่าวมาแล้ว ยกเว้นคำสั่ง JCXZ จะนิยมใช้ประกอบกับกลุ่มคำสั่งประเภทการทำซ้ำ (LOOP) คำสั่งที่ใช้ควบคุมการกระโดดแบบมีเงื่อนไขที่ใช้การเปรียบเทียบระหว่างโอเปอร์แรนด์สองตัวของคำสั่ง CMP มีสองกลุ่มคือ กลุ่มที่คิดการเปรียบเทียบเป็นการเปรียบเทียบของเลขไม่คิดเครื่องหมาย (JA JB JAE JBE) และกลุ่มที่คิดเป็นเลขคิดเครื่องหมาย (JG JL JGE JLE) ในการใช้คำสั่งกระโดดทั้งสองกลุ่มนี้จะต้องพิจารณาข้อมูลที่เปรียบเทียบกันด้วย (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

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:
```

8.8 คำสั่งวนรอบ (Loop Statements)

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

คำสั่ง LOOP

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

```
LOOP  label
```

คำสั่ง LOOP จะลดค่าของรีจิสเตอร์ CX โดย**ไม่กระทบกับแฟล็ก** นั่นคือคำสั่ง LOOP มีผลเหมือนคำสั่งแต่จะไม่มีผลกระทบกับแฟล็ก

```
DEC   CX
JNZ   label
```

ตัวอย่างการใช้คำสั่ง LOOP

โปรแกรมตัวอย่างนี้คำนวณผลรวมของเลขตั้งแต่ 1 ถึง 20

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

คำสั่ง JCXZ

ในกรณีที่ CX มีค่าเท่ากับศูนย์ก่อนการกระทำซ้ำโดยใช้คำสั่ง LOOP ผลลัพธ์จากการลดค่าจะมีค่าเท่ากับ 0FFFFh ทำให้จำนวนรอบของการทำงานไม่ถูกต้อง นิยมใช้คำสั่ง JCXZ ในการป้องกันความผิดพลาดในกรณีที่ค่าของรีจิสเตอร์ CX มีค่าเท่ากับ 0 โดยปกติถ้า CX มีค่าเท่ากับศูนย์ (0) 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 (ผลลัพธ์ของคำสั่งก่อนหน้ามีค่าเท่ากับศูนย์) สังเกตว่า คำสั่ง LOOPZ จะทำงานเหมือนคำสั่ง LOOP แต่แฟล็กศูนย์จะต้องมีค่าเป็นหนึ่งด้วย คำสั่งนี้ถึงจะกระโดดไปที่เลเบลที่กำหนด ในทำนองกลับกันคำสั่ง LOOPNZ จะกระโดดไปทำงานเมื่อ CX มีค่าไม่เท่ากับศูนย์และแฟล็กศูนย์มีค่าเป็น 0 คำสั่งทั้งสองนิยมใช้ในการทำซ้ำที่ทราบจำนวนครั้งแต่มีเงื่อนไขในการทำซ้ำ

ตัวอย่างคำสั่ง LOOPZ และ LOOPNZ

ตัวอย่างนี้แสดงการใช้คำสั่ง LOOPNZ ในการค้นหาข้อมูลค่าหนึ่งจากชุดของข้อมูล ในตัวอย่างนี้จำนวนข้อมูลคือ 100 ค่า และข้อมูลที่ต้องการค้นหาเก็บที่รีจิสเตอร์ DX

```
.data
 datalistdw         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 ชี้ไปยังข้อมูลตัวถัดไป
        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 ควรใส่หมายเหตุให้ชัดเจนและโดยปกติยังสามารถใช้วิธีอื่นในการจัดการกับข้อมูลตัวแรกได้ 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 ชี้ไปยังอักษรตัวถัดไป
        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 ในการป้องกันกรณีที่ผู้ใช้ป้อนอักษรเพียงตัวเดียวหรือไม่ป้อนเลย เลือกที่จะทดสอบตัวอักษรตัวแรก แทนที่จะใช้วิธีลดค่าของ BX ในการจัดการเกี่ยวกับข้อมูลตัวแรก

สรุป

การประกาศข้อมูลหรือตัวแปรในโปรแกรมภาษาแอสแซมบลีนั้น ทำได้โดยประกาศจองเนื้อที่ในหน่วยความจำในเซกเมนต์ข้อมูล แล้วตั้งเลเบลของข้อมูลนั้นไว้ ในการอ้างถึงข้อมูลในหน่วยความจำตำแหน่งนั้น สามารถอ้างโดยใช้เลเบลที่ประกาศไว้ได้ ดังนั้นการประกาศตัวแปรหรือข้อมูลนั้นจะมีลักษณะเช่นเดียวกับการประกาศเลเบลนั่นเอง ส่วนในการอ้างใช้ข้อมูลหรือตัวแปรที่ประกาศไว้ สามารถอ้างโดยใช้ชื่อของเลเบลที่ประกาศไว้ได้ Assembler จะจัดการนำตำแหน่งของข้อมูลนั้นมาแทนค่าให้โดยอัตโนมัติ ยังสามารถอ้างค่าในหน่วยความจำโดยอ้างสัมพัทธ์กับเลเบลที่กำหนดขึ้นได้

การประกาศข้อมูลสำหรับการใช้บริการของ DOS หมายเลข 09h และ 0Ah ฟังก์ชั่นหมายเลข 09h และ 0Ah ของ DOS เป็นฟังก์ชั่นที่ต้องมีการส่งแอดเดรสของข้อมูลในหน่วยความจำ การประกาศข้อมูลสำหรับฟังก์ชั่นหมายเลข 09h จะไม่มีความซับซ้อนมากนัก แต่สำหรับฟังก์ชั่นหมายเลข 0Ah การประกาศข้อมูลที่เหมาะสมจะทำให้เขียนโปรแกรมได้ง่ายมากขึ้น การใช้บริการของ DOS ฟังก์ชั่นหมายเลข 0Ah จะเกี่ยวข้องกับการอ่านข้อความส่วนฟังก์ชั่นหมายเลข 09h จะเกี่ยวข้องกับการพิมพ์ข้อความ

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

คำถามทบทวน

1. จงอธิบายความหมายของคำสั่งเทียมสำหรับการระบุขนาดข้อมูลในการจองหน่วยความจำต่อไปนี้ DB, DW, DD, DQ, และ DT
2. จงแสดงวิธีการจัดเรียงข้อมูลในหน่วยความจำจากการประกาศในส่วนของโปรแกรมที่แสดงอยู่ข้างล่างนี้

```
dseg        segment
data1       dw    1,2
data2       dw    1,2
data3       db    'Cs',14,15
data4       dd    3456h
dseg        ends
```

2. คำสั่งเทียม dup มีไว้เพื่ออะไร
3. อักขระหมายเลข 10 (Line feed) และ 13 (Carriage Return) หมายถึงอะไร
4. จงอธิบายการประกาศข้อมูลแต่ละบรรทัดสำหรับการเรียกใช้ฟังก์ชั่นจากส่วนของโปรแกรมที่แสดงอยู่ข้างล่างนี้

```
dseg        segment
maxlen      db    60
msglen      db    ?
msg         db    40 dup (?)
dseg        ends
```

5. ถ้าต้องการพิมพ์ข้อความ “Computer Science SDU” สามารถประกาศข้อมูลในหน่วยความจำนี้ได้อย่างไร
6. ข้อความที่สั่งพิมพ์ด้วยบริการหมายเลข 09h จะต้องจบด้วยอักขระใดเสมอ
7. จงอธิบายพร้อมยกตัวอย่างคำสั่งกระโดดแบบไม่มีเงื่อนไขและคำสั่งกระโดดแบบมีเงื่อนไขว่ามีอะไรบ้าง
8. จงอธิบายพร้อมยกตัวอย่างคำสั่ง CMP กลุ่มที่คิดการเปรียบเทียบเป็นการเปรียบเทียบของเลขแบบไม่คิดเครื่องหมายว่ามีอะไรบ้าง
9. จงอธิบายพร้อมยกตัวอย่างคำสั่ง CMP กลุ่มที่กลุ่มที่คิดเป็นเลขคิดเครื่องหมายว่ามีอะไรบ้าง
10. คำสั่งกระโดดแบ่งออกเป็นกี่กลุ่ม อะไรบ้าง

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 28 มกราคม 2557,จาก :http://rirs3.royin.go.th/coinages/

คำสั่งเทียม. (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 28 มกราคม 2557, จาก: http://th.wiki

คำสั่งกระโดด. (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 28 มกราคม 2557, จาก: http://th.wikipedia.org/wiki/pedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย–ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 9

หัวข้อเนื้อหา โครงสร้างควบคุม (Control Structure)

รายละเอียด

ศึกษาเกี่ยวกับการใช้คำสั่งกระโดดมาสร้างเป็นโครงสร้างควบคุมในรูปแบบต่าง ๆ การใช้คำสั่งกระโดดในลักษณะต่างๆ จะทำให้ชุดคำสั่งมีความเป็นโครงสร้างมากขึ้น

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 9 การจัดการเรียนการสอน จะเกี่ยวข้องกับโครงสร้างการตัดสินใจแบบ if–then–else การสร้าง repeat until loop การสร้าง while loop และการสร้าง for loop ซึ่งคำสั่งกระโดด ทำให้สามารถเขียนโปรแกรมให้มีการทำงานที่ซับซ้อนขึ้นได้ แต่การใช้กลุ่มคำสั่งต่างๆ อย่างไม่เป็นระบบทำให้ชุดคำสั่งที่เขียนขึ้นมานั้นทำความเข้าใจได้ยากและมีลักษณะไม่เป็นระบบที่สมบูรณ์ ดังนั้นการใช้คำสั่งกระโดดมาสร้างเป็นโครงสร้างควบคุมในรูปแบบต่าง ๆ จะทำให้ชุดคำสั่งมีความเป็นโครงสร้างมากขึ้น (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

9.1 การสร้างโครงสร้างการตัดสินใจแบบ if–then–else

รูปแบบของโครงสร้างที่ง่ายที่สุดคือโครงสร้างแบบ if–then–else รูปแบบของโปรแกรมภาษาแอสแซมบลีให้มีโครงสร้างแบบ if–then–else มีลักษณะดังนี้

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

ตัวอย่าง โปรแกรมที่ 9.1 เปรียบเทียบการเขียนโปรแกรมด้วยโครงสร้างของภาษา pascal กับโครงสร้างภาษา Assembly

(a)

```
if AL<10 the

DL:=AL+'0'

        else
  DL:=AL+'A'-10;
```

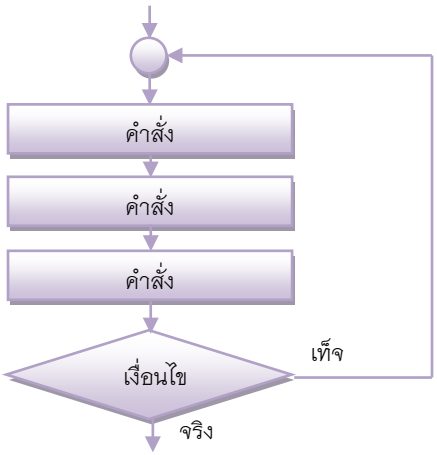
(b)

```
        cmp   al,10
        jae   abovenine
        mov   dl,al
        add   dl,'0'
        jmp   endif
abovenine:
        mov   dl,al
        add   al,'A'-10
endif:
```

โปรแกรมที่ 9.1 โปรแกรม (a) แสดงตัวอย่างของโปรแกรมที่เขียนด้วยโครงสร้างของภาษา pascal โปรแกรม (b) แสดงโปรแกรมภาษาแอสแซมบลี (Assembly) ที่เทียบเท่ากัน

9.2 การสร้าง repeat until loop

โครงสร้างของ repeat until loop ในภาษาระดับสูงทั่วไปมีลักษณะ แสดงได้ดังภาพที่ 9.1



ภาพที่ 9.1 แสดงโครงสร้างควบคุมแบบ repeat until

ตัวอย่าง โปรแกรมที่ 9.2 การเขียนโปรแกรมโดยใช้ภาษาแอสแซมบลีโดยมีโครงสร้างแบบ repeat until

(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
```

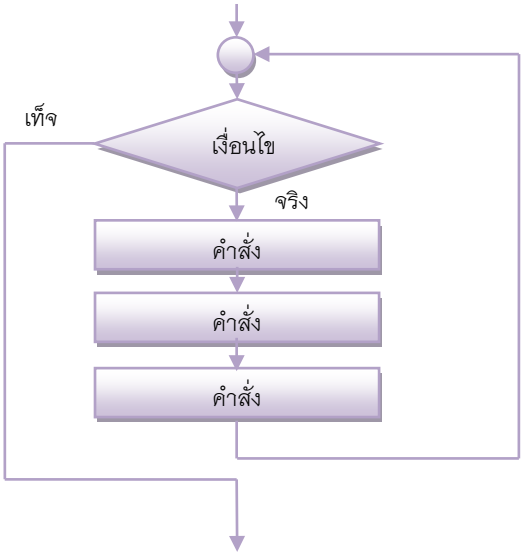
โปรแกรมที่ 9.2 โปรแกรม (a) แสดงตัวอย่างของโปรแกรมภาษา Pascal ที่ใช้โครงสร้างแบบ repeat until
โปรแกรม (b) แสดงโปรแกรมภาษาแอสแซมบลี (Assembly) ที่เทียบเท่ากัน

รูปแบบของโปรแกรมภาษาแอสแซมบลีที่เทียบเท่ากับ repeat until loop มีลักษณะเป็นดังนี้

```
startlabel:
        action;
        ...
        action;
if condition is false  then jump to startlabel
```

9.3 การสร้าง while loop

โครงสร้างของ while loop ในภาษาระดับสูงทั่วไป แสดงได้ดังภาพที่ 9.2



ภาพที่ 9.2 โครงสร้างควบคุมแบบ while loop

ตัวอย่าง โปรแกรมที่ 9.3 การเขียนโปรแกรมภาษาแอสแซมบลีโดยมีโครงสร้างแบบ while loop

(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:
```

โปรแกรม (a) แสดงตัวอย่างของโปรแกรมภาษา pascal ที่ใช้โครงสร้างแบบ while loop

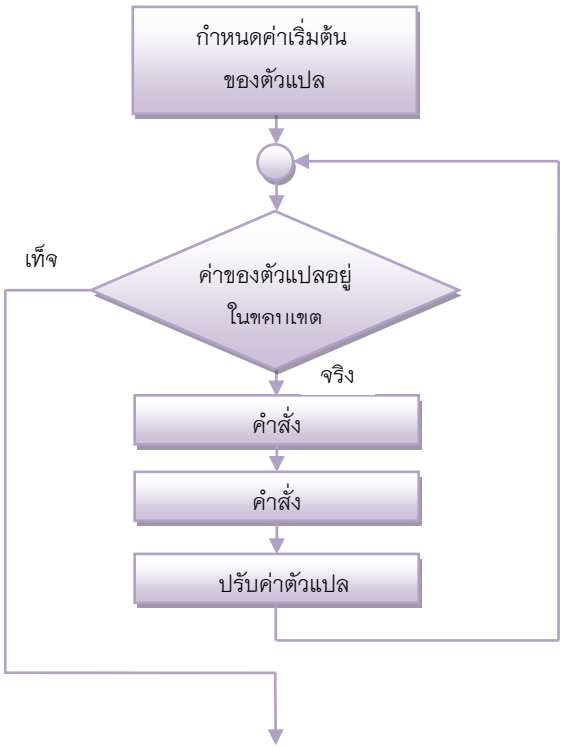
โปรแกรม (b) แสดงโปรแกรมภาษาแอสแซมบลีที่เทียบเท่ากัน

รูปแบบของโปรแกรมภาษาแอสแซมบลีที่เทียบเท่ากับ while loop มีลักษณะเป็นดังนี้

startlabel:
if *condition is false* then
    jump to endlabel
        action
        ...
        action
    jump to startlabel
endlabel:

9.4 การสร้าง for loop

เราสามารถใช้คำสั่งกระโดดในการสร้างโครงสร้างแบบ for loop ได้ นอกจากนั้นเรายังสามารถใช้คำสั่งกลุ่ม LOOP ในการสร้างโครงสร้างแบบ for loop ได้เช่นเดียวกัน โครงสร้างของ for loop มีลักษณะ แสดงได้ดังภาพที่ 9.3



ภาพที่ 9.3 โครงสร้างของ for loop

รูปแบบของโปรแกรมภาษาแอสแซมบลีที่มีโครงสร้างเป็นแบบ 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:
```

ตัวอย่าง โปรแกรมที่เขียนโดยใช้รูปแบบลักษณะนี้เป็นดังโปรแกรมที่ 9.4

(a)

```
CX:=0;

for DL:=1 to 100 do

  begin
 if DL mod 7=0 then

      CX:=CX+1;

end;
```

(b)

```
      mov   cx,0

      mov   dl,1
startloop:
      cmp   dl,100
      ja    endloop

      mov   al,dl
      mov   ah,0
      mov   bl,7
      div   bl

      cmp   ah,0
      jne   endif

      inc   cx
endif:

      inc   dl
      jmp   startloop
endloop:
```

โปรแกรมที่ 9.4 โปรแกรม (a) แสดงโปรแกรมที่เขียนด้วยภาษา pascal ที่ใช้โครงสร้างแบบ for loop โปรแกรม (b) แสดงโปรแกรมที่เขียนด้วยภาษาแอสแซมบลี

นอกจากนี้ยังสามารถใช้คำสั่ง LOOP ในการสร้างโครงสร้างแบบ for loop ได้เช่นเดียวกัน 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 9.5 โปรแกรม (a) แสดงโปรแกรมที่เขียนด้วยภาษา pascal
โปรแกรม (b) แสดงโปรแกรมที่เขียนด้วยภาษาแอสแซมบลี

สรุป

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

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 ที่ให้มาให้อยู่ในรูปของโปรแกรมภาษาแอสแซมบลีที่เทียบเท่ากัน

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

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 9 มีนาคม 2557,จาก :http://rirs3.royin.go.th/coinages/

โครงสร้างควบคุม. (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 9 มีนาคม 2557, จาก: http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย–ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 10

หัวข้อเนื้อหา โปรแกรมย่อยขั้นต้น (The initial Subprogram)

รายละเอียด

รูปแบบของโปรแกรมภาษาแอสแซมบลีที่เขียนในบทก่อน ๆ จะมีส่วนของโปรแกรมหลายส่วนที่ซ้ำซ้อนกัน สามารถที่จะแยกส่วนย่อยเหล่านั้นเป็นโปรแกรมย่อยที่มีความอิสระจากโปรแกรมหลักได้ การแยกโปรแกรมเป็นโปรแกรมย่อยนี้ ทำให้สามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวก และการตรวจสอบและแก้ไขโปรแกรมยังสามารถกระทำได้ง่ายขึ้นด้วย

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 10 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่งที่รองรับการเรียกโปรแกรมย่อย การประกาศโปรแกรมย่อย คำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแสต็กและตัวอย่างการใช้งานโปรแกรมย่อย ซึ่งรูปแบบของโปรแกรมภาษาแอสแซมบลีที่ได้ศึกษาก่อนหน้านี้จะมีส่วนของโปรแกรมหลายส่วนที่ซ้ำซ้อนกัน สามารถที่จะแยกส่วนย่อยเหล่านั้นเป็นโปรแกรมย่อยที่มีความอิสระจากโปรแกรมหลักได้การแยกโปรแกรมเป็นโปรแกรมย่อยนี้ ทำให้สามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวก อีกทั้งการตรวจสอบและแก้ไขโปรแกรมยังสามารถกระทำได้ง่ายขึ้นด้วย

10.1 คำสั่งที่รองรับการเรียกโปรแกรมย่อย

การเรียกโปรแกรมย่อยมีความแตกต่างกับการกระโดดทั่วไป เนื่องจากภายหลังที่โปรแกรมย่อยทำงานเสร็จ หน่วยประมวลผลจะต้องสามารถกระโดดกลับมาทำงานในโปรแกรมหลักต่อไปได้ ดังนั้นการเรียกใช้โปรแกรมย่อยนั้นจะต้องมีการเก็บค่า ณ ตำแหน่งของคำสั่งที่ทำงานอยู่เดิมด้วยและเมื่อจบโปรแกรมย่อยโปรแกรมจะต้องกระโดดกลับมาทำงาน ณ

ตำแหน่งเดิมโดยใช้ข้อมูลที่เก็บไว้ คำสั่งใน 8086 ที่รองรับการใช้งานโปรแกรมย่อยคือ **คำสั่ง CALL** และ **คำสั่ง RET** เมื่อผู้ใช้เรียกคำสั่ง CALL พร้อมทั้งระบุตำแหน่งของโปรแกรมย่อย หน่วยประมวลผลจะเก็บตำแหน่งของคำสั่งถัดไปที่จะกลับมาทำงานลงในแสต็กและจะกระโดดไปทำงานที่โปรแกรมย่อย เมื่อโปรแกรมย่อยทำงานเสร็จ โปรแกรมย่อยจะเรียกใช้คำสั่ง RET เพื่อกระโดดกลับมาทำงานในโปรแกรมหลักต่อไป (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

10.2 การประกาศโปรแกรมย่อย

ผู้เขียนโปรแกรมสามารถสร้างโปรแกรมย่อยโดยการประกาศเลเบลที่จุดเริ่มต้นของโปรแกรมย่อยเท่านั้นก็ได้ แต่โดยทั่วไปแล้วจะประกาศโปรแกรมย่อยโดยใช้คู่กับคำสั่งเทียม PROC และ ENDP

ตัวอย่าง โปรแกรมที่ 10.1 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:
; ...
; แต่ละส่วนอ่านการกดปุ่มและส่วนหาค่าของตัวเลขของแต่ละหลักไว้
; ให้ตัวเลขหลักหน้าเก็บใน bh และหลักหลังเก็บใน bl
; ...
                mov         al,bh
                call        printhexdigit
                mov         al,bl
                call        printhexdigit
                mov         ax,4c00h
                int         21h
end             start
```

โปรแกรมที่ 10.1 ส่วนของโปรแกรมพิมพ์ค่ารหัสแอสกีเป็นเลขฐานสิบหก

รูปแบบของการประกาศโปรแกรมย่อยมีลักษณะดังนี้

```
proc_name       PROC        NEAR
                actions
proc_name       ENDP
```

คำสั่งเทียม NEAR ที่ต่อจากคำสั่งเทียม PROC เป็นการระบุว่าโปรแกรมย่อยนี้เป็นโปรแกรมย่อยที่อยู่ในเซกเมนต์เดียวกันกับโปรแกรมหลัก และมีการเรียกใช้แบบใกล้ ซึ่งจะมีผลในขั้นตอนการเก็บตำแหน่งที่จะกระโดดกลับมาทำงานหลังการทำงานของโปรแกรมย่อย

การทำงานของโปรแกรมย่อยจากตัวอย่างมีการเปลี่ยนแลงค่าของรีจิสเตอร์ AH และรีจิสเตอร์ DL ซึ่งการเปลี่ยนค่าของรีจิสเตอร์ทั้งสองตัวอาจทำให้เกิดผลกระทบข้างเคียงกับโปรแกรมหลักได้ถ้าโปรแกรมหลักมีการใช้งานรีจิสเตอร์ทั้งสองด้วยเช่นเดียวกัน ควรจะลดการเกิดผลข้างเคียงนี้โดยให้โปรแกรมย่อยเก็บค่าของรีจิสเตอร์ต่าง ๆ ที่โปรแกรมย่อยใช้และคืนค่าเดิมให้กับรีจิสเตอร์เหล่านั้นหลังการทำงาน นิยมใช้แสต็กในการเก็บค่าของรีจิสเตอร์เป็นการชั่วคราวเนื่องจากสามารถเก็บข้อมูลลงในแสต็กได้โดยไม่ต้องจองเนื้อที่ล่วงหน้าและการเก็บข้อมูลในลักษณะของแสต็กมีความสอดคล้องกับการทำงานแบบโปรแกรมย่อย

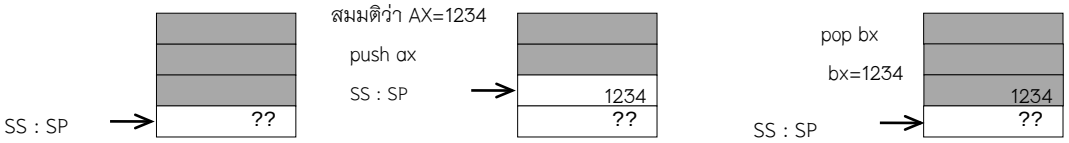
10.3 คำสั่งเก็บข้อมูลและดึงข้อมูลจากแสต็ก : คำสั่ง PUSH และคำสั่ง POP

สามารถใช้คำสั่ง PUSH ในการเก็บข้อมูลลงไปในแสต็ก และคำสั่ง POP สำหรับการเรียกข้อมูลออกมาจากแสต็ก คำสั่งทั้งสองมีรูปแบบดังนี้

push	*regs16*	push	*mem*
pop	*regs16*	pop	*mem*

ข้อมูลที่จะเก็บลงในแสต็กจะต้องมีขนาด 16 บิตเท่านั้น โดยการเก็บข้อมูลในแสต็กจะมีลักษณะเป็นแบบใส่ทีหลังดึงออกก่อน การเก็บข้อมูลลงในแสต็กจึงต้องระวังลำดับของการเก็บและการดึงข้อมูลออกไปด้วย

การทำงานของแสต็กจะมีรีจิสเตอร์ SS และ SP เป็นรีจิสเตอร์ที่ใช้เก็บตำแหน่งของข้อมูลที่เก็บลงไปอันล่าสุด โดย SS จะเก็บเซกเมนต์ของแสต็ก และ SP จะเก็บออฟเซ็ตของข้อมูลล่าสุด เมื่อมีการเก็บข้อมูลเพิ่มลงในแสต็ก หรือมีการดึงข้อมูลออกไปก็จะมีการปรับค่าของรีจิสเตอร์ทั้งสองนี้ การทำงานของแสต็ก แสดงได้ดังภาพที่ 10.1



ภาพที่ 10.1 แสดงการทำงานของแสต็ก

โปรแกรมย่อยที่มีการเก็บค่าของรีจิสเตอร์ต่าง ๆ

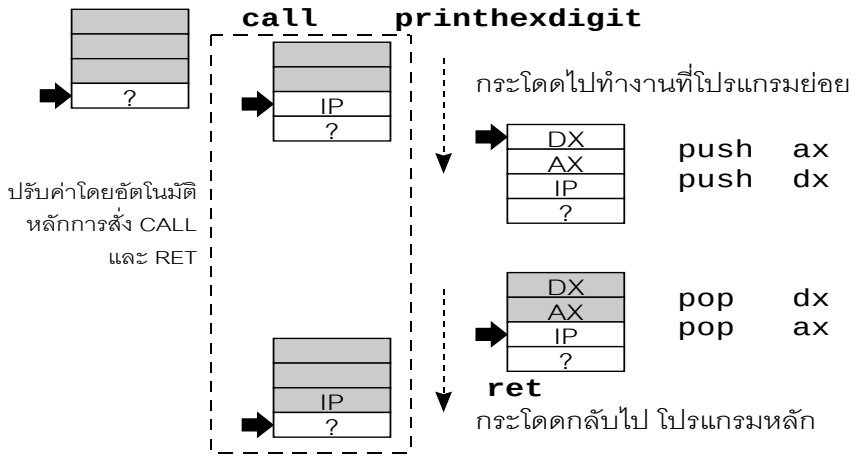
สามารถแก้โปรแกรมย่อยในตัวอย่างให้มีการรักษาค่าในรีจิสเตอร์ต่าง ๆ ได้ดังโปรแกรมที่ 10.2

```
;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 10.2 ตัวอย่างโปรแกรมย่อยที่รักษาค่าของรีจิสเตอร์

สังเกตลำดับในการ push และ pop ของโปรแกรมย่อย ลำดับในการ push จะตรงกันข้ามกับลำดับในการ pop เนื่องจากการทำงานของแสต็กเป็นแบบข้อมูลที่ใส่ทีหลังจะถูกดึงออกก่อน การเปลี่ยนแปลงของแสต็กในการทำงานดังกล่าว แสดงได้ดังภาพที่ 10.2



ภาพที่ 10.2 แสดงการเปลี่ยนแปลงของแสต็กในการเรียกใช้โปรแกรมย่อย

เมื่อมีการเรียกใช้โปรแกรมย่อยโดยคำสั่ง CALL ค่าของรีจิสเตอร์ IP ซึ่งเก็บตำแหน่งของคำสั่งถัดไปจะถูก PUSH ลงในแสต็กโดยอัตโนมัติ จากนั้นโปรแกรมจะกระโดดไปทำงานที่โปรแกรมย่อย ในโปรแกรมย่อยมีการเก็บค่าของรีจิสเตอร์ AX และรีจิสเตอร์ DX ลงในแสต็ก หลังการทำงานโปรแกรมย่อยได้คืนค่าของรีจิสเตอร์ทั้งสองโดยการดึงค่าจากแสต็ก และเรียกใช้คำสั่ง RET เพื่อกระโดดกลับไปทำงานยังโปรแกรมหลักต่อ สังเกตว่าถ้าเก็บและดึงค่าออกจากแสต็กได้ไม่ถูกต้องการกระโดดกลับไปทำงานยังโปรแกรมหลักอาจมีการผิดพลาดได้

10.4 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 เนื่องจากภายหลังที่โปรแกรมย่อยทำงานเสร็จ หน่วยประมวลผลจะต้องสามารถกระโดดกลับมาทำงานในโปรแกรมหลักต่อไปได้ ดังนั้นการเรียกใช้โปรแกรมย่อยนั้นจะต้องมีการเก็บตำแหน่งของคำสั่งที่ทำงานอยู่เดิมด้วย และเมื่อจบโปรแกรมย่อยโปรแกรมจะต้องกระโดดกลับมาทำงานที่เดิม โดยใช้ข้อมูลที่เก็บไว้ คำสั่งของ 8086 ที่รองรับการใช้งานโปรแกรมย่อยคือคำสั่ง CALL และคำสั่ง RET เมื่อผู้ใช้เรียกคำสั่ง CALL พร้อมทั้งระบุตำแหน่งของโปรแกรมย่อย หน่วยประมวลผลจะเก็บตำแหน่งของคำสั่งถัดไปที่จะกลับมาทำงานลงในแสต็ก และจะกระโดดไปทำงานที่โปรแกรมย่อย เมื่อโปรแกรมย่อยทำงานเสร็จ โปรแกรมย่อยจะเรียกใช้คำสั่ง RET เพื่อกระโดดกลับมาทำงานในโปรแกรมหลักต่อไป เมื่อหน่วยประมวลผลประมวลผลคำสั่ง RET หน่วยประมวลผลจะดึงค่าตำแหน่งที่โปรแกรมจะกระโดดกลับไปทำงานจากแสต็ก และกระโดดกลับไปทำงานต่อยังโปรแกรมหลักต่อไป

คำถามทบทวน

1. จงอธิบายการทำงานของคำสั่ง CALL และคำสั่ง RET
2. การประกาศโปรแกรมย่อยในภาษาแอสแซมบลีโดยทั่วไปมักจะใช้คู่ของคำสั่งเทียมอะไร
3. คำสั่งเก็บข้อมูลและดึงข้อมูลจากแสต็กคำสั่ง PUSH และคำสั่ง POP มีขั้นตอนการงานอย่างไร
4. การทำงานของแสต็กจะมีรีจิสเตอร์ที่เกี่ยวข้องอะไรบ้างและแต่ละรีจิสเตอร์มีหน้าที่และทำงานอย่างไร
5. จงอธิบายเหตุการณ์ที่เกิดขึ้นหลังจากมีการเรียกใช้โปรแกรมย่อยโดยคำสั่ง CALL

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 23 พฤษภาคม 2557, จาก :http://rirs3.royin.go.th/coinages/

โปรแกรมย่อย. (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 23 พฤษภาคม 2557, จาก :http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย–ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 11

หัวข้อเนื้อหา การกระทำระดับบิต (The Bit Level)

รายละเอียด

การจัดการกับข้อมูลที่ได้ศึกษาในบทก่อน ๆ นั้นมีหน่วยย่อยในการจัดการเป็นไบต์ โดยจะไม่สามารถจัดการข้อมูลที่มีขนาดย่อยกว่านั้นได้ แต่ในการทำงานจริงในบางครั้งรูปแบบของข้อมูลที่นำไปใช้ จำเป็นต้องจัดการให้อยู่ในรูปของบิต ดังนั้นการใช้คำสั่งเกี่ยวกับการจัดการระดับบิตในการประมวลผลข้อมูลกลุ่ม ในบางกรณีคำสั่งกระทำระดับบิตสามารถช่วยให้การคำนวณต่าง ๆ ทำได้ง่ายและมีประสิทธิภาพมาขึ้นตามไปด้วย

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 11 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่งทางตรรกศาสตร์ การประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์ การใช้คำสั่งเลื่อนบิต ตัวอย่างการใช้งานคำสั่งเลื่อนบิต การประยุกต์ใช้งานคำสั่งเลื่อนบิต คำสั่งหมุนบิต คำสั่งหมุนบิตที่ผ่านแฟล็กทด ตัวอย่างการใช้งานคำสั่งเกี่ยวกับการประมวลผลระดับบิต ซึ่งปัญหาของการจัดการกับข้อมูลจะกระทำในระดับไบต์ โดยจะไม่สามารถจัดการข้อมูลที่มีขนาดย่อยกว่านั้นได้ แต่ในการทำงานจริงในบางครั้งรูปแบบของข้อมูลที่นำไปใช้ จำเป็นต้องจัดการให้อยู่ในรูปของบิต ดังนั้นการใช้คำสั่งเกี่ยวกับการจัดการระดับบิตในการประมวลผลข้อมูลกลุ่ม ในบางกรณีคำสั่งกระทำระดับบิตสามารถช่วยให้การคำนวณต่าง ๆ ทำได้ง่ายและมีประสิทธิภาพมาขึ้นตามไปด้วย

11.1 คำสั่งทางตรรกศาสตร์

คำสั่งในกลุ่มนี้เป็นคำสั่งประมวลผลข้อมูลระดับบิต โดยจะนำค่าในแต่ละบิตของข้อมูลมาประมวลผลทางตรรกศาสตร์ คำสั่งในกลุ่มนี้ได้แก่ คำสั่ง AND คำสั่ง OR คำสั่ง XOR และคำสั่ง NOT ซึ่งรูปแบบการใช้งานของกลุ่มคำสั่งดังกล่าวจะมีลักษณะเหมือนกัน คือจะมีการ

รับโอเปอร์แรนด์สองตัว และจะนำข้อมูลในโอเปอร์แรนด์ตัวแรกมากระทำกับข้อมูลตัวที่สอง และจะเก็บผลลัพธ์ของการกระทำนั้นในโอเปอร์แรนด์ตัวแรก ส่วนในกรณีของคำสั่ง NOT จะรับโอเปอร์แรนด์ตัวเดียวและจะทำการกลับค่าในบิตแล้วเก็บผลลัพธ์ลงในโอเปอร์แรนด์ตัวนั้นเลย ตารางค่าความจริงของการกระทำทางตรรกศาสตร์ แสดงได้ดังตารางที่ 11.1

ตารางที่ 11.1 แสดงค่าของการกระทำทางตรรกศาสตร์

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	
1	1	1	1	0	

คำสั่ง AND

ผลลัพธ์ของคำสั่ง AND จะมีบิตที่เป็น 1 เมื่อบิตของข้อมูลตัวตั้งทั้งสองตัวมีค่าเป็น 1 แสดงได้ (ตารางที่ 11.1)

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 แสดงได้ (ตารางที่ 11.1)

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 (ตารางที่ 11.1) 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 จะสลับบิตของโอเปอร์แรนด์จาก 0 เป็น 1 และ 1 เป็น 0 แสดงได้ (ตารางที่ 11.1)

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 นิยมใช้คำสั่งนี้ในการทดสอบว่าข้อมูลในบิตที่ต้องการมีค่าเป็นหนึ่งหรือไม่ 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

การนำคำสั่งทางตรรกศาสตร์มาใช้ในการประมวลผลข้อมูลระดับบิตได้ จากตารางที่ 11.1 สามารถนำมาสร้างตารางที่ 11.2 ซึ่งแสดงผลของการใช้คำสั่งทางตรรกศาสตร์กับข้อมูลได้

ตารางที่ 11.2 แสดงผลของการใช้คำสั่งทางตรรกศาสตร์กับข้อมูล

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

จากตารางเราจะพบว่าถ้าเราต้องการให้บิตใดของข้อมูลมีค่าเป็นหนึ่งโดยที่บิตอื่นมีค่าคงเดิม เราสามารถใช้คำสั่ง AND ได้ และถ้าเราต้องการจะทำให้บิตใดของข้อมูลมีค่าเป็นศูนย์โดยไม่มีผลกระทบกับแฟล็กอื่น 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 การทำงานควรจะมี แสดงได้ดังภาพที่ 11.1

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

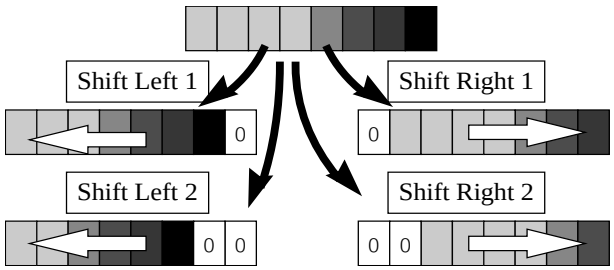
ภาพที่ 11.1 แสดงขั้นตอนการแปลงค่าของ AL

โปรแกรมจะมีลักษณะดังนี้

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

11.2 คำสั่งเลื่อนบิต

การประมวลผลอีกรูปแบบที่เราสามารถกระทำกับข้อมูลในระดับขั้นของบิตได้แก่ การเลื่อนบิต ลักษณะการเลื่อนบิต แสดงได้ดังภาพที่ 11.2 ในการเลื่อนบิตเราสามารถเลื่อนได้ทั้งทางซ้ายและทางขวา โดยคำสั่งสำหรับการเลื่อนบิตไปทางซ้ายได้แก่ **คำสั่ง SHL** (Shift Left) คำสั่งสำหรับการเลื่อนบิตไปทางขวาได้แก่ **คำสั่ง SHR** (Shift Right) เรานิยมใช้การเลื่อนบิตในการประมวลผลที่ต้องการประมวลผลข้อมูลทีละบิต และมีการประมวลผลเป็นแบบวงรอบ



ภาพที่ 11.2 แสดงลักษณะของการเลื่อนบิต

รูปแบบของคำสั่งเลื่อนบิตมีลักษณะดังนี้

SHR	*regs*,1	SHR	*mem*,1
SHR	*regs*,CL	SHR	*mem*,CL
SHR	*regs,number*	SHR	*mem,number*

โดยรูปแบบของคำสั่ง SHL จะมีลักษณะเหมือนคำสั่ง SHR รูปแบบที่สามจะใช้ได้กับหน่วยประมวลผล 80286 ขึ้นไปเท่านั้นโดยในการที่เราจะใช้รูปแบบของคำสั่งของ 80286 ในโปรแกรมเราจะต้องระบุ **คำสั่งเทียม 286** ลงในโปรแกรมด้วย โดยใส่คำสั่งนี้ก่อนหน้าการใช้งานคำสั่งครั้งแรก (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

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 11.3 แสดงผลลัพธ์ของการเลื่อนบิตของข้อมูลต่าง ๆ

จากตารางจะสังเกตได้ว่านอกจากการเลื่อนบิตจะมีความหมายโดยตรงคือการเลื่อนบิตไปทางซ้ายหรือทางขวาแล้ว การเลื่อนบิตยังมีความหมายทางคณิตศาสตร์อีกด้วย

ข้อมูล (ค่า)	เลื่อนบิตไปทาง	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 ข้อสังเกตคือจะต้องพิจารณากรณีที่ข้อมูลอยู่ในขอบเขตด้วย เช่น กรณีของการเลื่อน 0010 1110 ไปทางซ้าย 3 บิต (คูณด้วย 8) ผลลัพธ์ที่ได้จะมีความผิดพลาด การเลื่อนบิตไปทางขวาจะให้ผลลัพธ์ตรงกันข้ามกับการเลื่อนบิตไปทางขวา นั่นคือจะเสมือนการหารด้วยกำลังสอง (สังเกตว่าผลลัพธ์ที่ได้จะมีการปัดเศษเนื่องจากบิตที่เลื่อนจะหายไป เช่นในตัวอย่างที่เลื่อนบิตทางขวา 3 บิต)

คำสั่งเลื่อนบิตแบบคิดเครื่องหมาย : คำสั่ง SAL และคำสั่ง SAR

ถ้าใช้การเลื่อนบิตแทนการคูณหรือหารด้วยกำลังของสองกับตัวเลขแบบคิดเครื่องหมาย เราจะพบว่าการเลื่อนบิตไปทางซ้ายที่แสดงถึงการคูณนั้นยังสามารถใช้กับตัวเลขแบบคิดเครื่องหมายได้ เนื่องจากหลักที่เลื่อนเข้ามาแทนนั้นยังคงเป็นเลขศูนย์เหมือนในกรณีของเลขไม่คิดเครื่องหมาย แต่ในกรณีของการเลื่อนบิตไปทางขวาที่ใช้สำหรับการหารด้วยกำลังของสองนั้น บิตที่เลื่อนเข้ามาแทนอาจมีค่าเป็น 0 หรือ 1 ก็ได้ขึ้นกับเครื่องหมายของตัวเลขนั้น จึงมีคำสั่งเลื่อนบิตที่ใช้สำหรับเลขที่มองเป็นเลขคิดเครื่องหมาย คือ **คำสั่ง SAL** (Shift Arithmetic Left) และ **คำสั่ง SAR** (Shift Arithmetic Right) คำสั่ง SAL จะทำงานเหมือนคำสั่ง SHL ทุกประการ ตัวอย่างการใช้งานคำสั่งเป็นดังตารางที่ 11.4

ตารางที่ 11.4 แสดงตัวอย่างผลลัพธ์ของการเลื่อนบิตแบบคิดเครื่องหมาย

ข้อมูล (ค่า)	เลื่อนบิตไปทาง	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

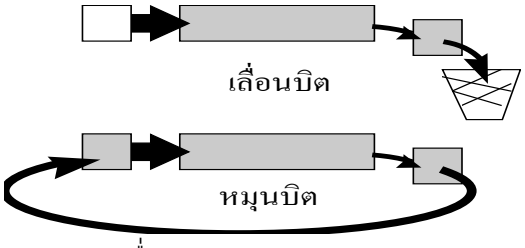
การนำคำสั่งเลื่อนบิตไปใช้ในการคูณและหารข้อมูลได้ โดยการใช้คำสั่งเลื่อนบิตแทนการใช้คำสั่ง MUL ทำให้การคูณทำงานได้เร็วขึ้นและในบางกรณีทำให้เขียนโปรแกรมได้ง่ายขึ้นด้วย

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
```

11.3 คำสั่งหมุนบิต

คำสั่งหมุนบิตมีความแตกต่างกับคำสั่งเลื่อนบิตในจุดที่ว่า บิตที่เลื่อนไปแล้วไม่ได้ถูกทิ้งหายไป แต่จะถูกนำมาใส่แทนบิตที่เลื่อนไป แสดงได้ดังภาพที่ 11.3



ภาพที่ 11.3 แสดงลักษณะของการเลื่อนบิต และ การหมุนบิต

เช่นเดียวกับคำสั่งเลื่อนบิต คำสั่งหมุนบิตมีลักษณะการหมุนสองแบบคือ หมุนไปทางซ้าย (**คำสั่ง ROL** : Rotate Left) และ หมุนไปทางขวา (**คำสั่ง ROR** : Rotate Right) รูปแบบของคำสั่งทั้งสองจะมีลักษณะเหมือนคำสั่งเลื่อนบิต การทำงานของคำสั่งทั้งสอง แสดงได้ดังภาพที่ 11.4

ภาพที่ 11.4 แสดงลักษณะการทำงานของคำสั่งหมุนบิต

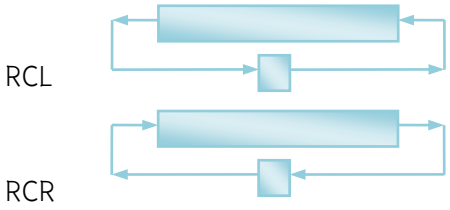
นิยมใช้คำสั่งหมุนบิตแทนคำสั่งเลื่อนบิตในกรณีที่ต้องการให้ค่าของข้อมูลกลับเหมือนเดิมหลังประมวลผลครบรอบ

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
```

คำสั่งหมุนบิตที่ผ่านแฟล็กทด

คำสั่งหมุนบิตอีกกลุ่มหนึ่งจะเป็นการหมุนโดยนำบิตไปผ่านแฟล็กทด ลักษณะการทำงาน แสดงได้ดังภาพที่ 11.5 ซึ่งจะเห็นว่าบิตที่เข้ามาแทนบิตที่หมุนไปจะนำมาจากแฟล็กทด และบิตที่ถูกหมุนออกไปจะเข้าไปแทนค่าในแฟล็กทด โดยคำสั่งหมุนบิตผ่านแฟล็กทดคือ**คำสั่ง RCL** (Rotate Carry Left) และ**คำสั่ง RCR** (Rotate Carry Right)



ภาพที่ 11.5 แสดงลักษณะการทำงานของคำสั่งหมุนบิตที่ผ่านแฟล็กทด

สังเกตว่าบิตที่ล้นออกมาจะถูกนำไปพักที่แฟล็กทด ก่อนที่จะนำมาแทนที่ในข้อมูล นิยมใช้คำสั่งหมุนบิตผ่านแฟล็กทดในการเลื่อนบิตข้อมูลที่เก็บต่อเนื่องอยู่ในหลายรีจิสเตอร์ ในการใช้งานคำสั่งนี้เราจะต้องกำหนดค่าให้กับแฟล็กทดเสียก่อน โดยใช้คำสั่ง STC และคำสั่ง CLC

ตัวอย่างการใช้งานคำสั่งหมุนบิตที่ผ่านแฟล็กทด

แสดงการเขียนคำสั่งแสดงการเลื่อนบิตของข้อมูลขนาด 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)
        dispbin  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 ส่วนการประมวลผลอีกรูปแบบที่เราสามารถกระทำกับข้อมูลในระดับขั้นของบิตได้แก่การเลื่อนบิต ในการเลื่อนบิตเราสามารถเลื่อนได้ทั้งทางซ้ายและทางขวา โดยคำสั่งสำหรับการเลื่อนบิตไปทางซ้ายได้แก่ คำสั่ง SHL (Shift Left) คำสั่งสำหรับการเลื่อนบิตไปทางขวาได้แก่ คำสั่ง SHR (Shift Right) เรานิยมใช้การเลื่อนบิตในการประมวลผลที่ต้องการประมวลผลข้อมูลทีละบิต และมีการประมวลผลเป็นแบบวงรอบ

คำถามทบทวน

1. จงแสดงผลลัพธ์ของคำสั่งจากคำสั่งข้างล่างที่ให้มาเมื่อบิตของข้อมูลตัวตั้งทั้งสองตัวมีค่าเป็น 1

```
1.1     mov     ax,2345h
        and     ax,3456h        ; ax = ? and ?
                                ; ax = ?
1.2     mov     ax,2345h
        or      ax,3456h        ; ax = ? and ?
                                ; ax = ?
1.3     mov     ax,2345h
        xor     ax,3456h        ; ax = ? and ?
                                ; ax = ?
```

2.จงแสดงผลลัพธ์ของการเลื่อนบิตของข้อมูลต่าง ๆ จากตารางที่ให้มา

ข้อมูล (ค่า)	เลื่อนบิตไปทาง	จำนวน(บิต)	ผลลัพธ์ (ค่า)
0011 1110 (63)	ซ้าย	1	?
0011 1110 (63)	ซ้าย	2	?
0011 1110 (63)	ซ้าย	3	?
0111 0100 (116)	ขวา	1	?
0111 0100 (116)	ขวา	2	?
0111 0100 (116)	ขวา	3	?

3.จงแสดงผลลัพธ์ของการเลื่อนบิตแบบคิดเครื่องหมายของข้อมูลต่าง ๆ จากตารางที่ให้มา

ข้อมูล (ค่า)	เลื่อนบิตไปทาง	จำนวน(บิต)	ผลลัพธ์ (ค่า)
1100 0010 (–63)	ซ้าย	1	?
1101 1000 (–40)	ซ้าย	2	?
1100 0010 (–63)	ขวา	1	?
1000 1100 (–116)	ขวา	2	?
1000 1100 (–116)	ขวา	3	?

4.จงอธิบายพร้อมยกตัวอย่างคำสั่งหมุนบิตผ่านแฟล็กทดต่อไปนี้

4.1 คำสั่ง RCL (Rotate Carry Left)

4.2 คำสั่ง RCR (Rotate Carry Right)

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 15 มิถุนายน 2557,จาก :http://rirs3.royin.go.th/coinages/

การเลื่อนบิตของข้อมูล (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ มิถุนายน 2557, จาก: http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย–ญี่ปุ่น), 2537.

.

แผนการสอนประจำสัปดาห์ที่ 12

หัวข้อเนื้อหา การอ้างแอดเดรสและการขัดจังหวะ
(Addressing Mode and Interrupted)

รายละเอียด

8086 จะมองลักษณะของหน่วยความจำเป็นกลุ่ม ๆ เก็บในรูปแบบของเซกเมนต์ (64 Kbyte/ 1 Segment) ได้แก่ CS, DS, SS และ ES โดยแสดงแอดเดรสของหน่วยความจำที่จะติดต่อด้วยซึ่ง CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรมส่วน DS จะเก็บค่าดาตาเซกเมนต์ขณะนั้น SS ก็จะเก็บค่าสแต็กเซกเมนต์ขณะนั้น และ ES จะกำหนดเซกเมนต์ของข้อมูลรวมที่เรียกว่า global data segment โดยการทำงานของคอมพิวเตอร์จะประกอบด้วยการทำงานร่วมกันของหน่วยประมวลผลกลางกับอุปกรณ์รอบข้าง ซึ่งจำเป็นต้องมีการส่งผ่านข้อมูลระหว่างหน่วยประมวลผลกลางและอุปกรณ์อื่น ๆ อยู่เสมอ ดังนั้นรูปแบบการเชื่อมต่อจึงต้องถูกออกแบบมาให้เหมาะสมกับการทำงานร่วมกันของอุปกรณ์ต่าง ๆ

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 12 การจัดการเรียนการสอน จะเกี่ยวข้องกับการอ้างแอดเดรสในเซกเมนต์ทั้งสี่ คือ CS, DS, SS และ ES การประยุกต์ใช้คำสั่งในการอ้างแอดเดรสแบบต่าง ๆ กระบวนการขัดจังหวะและการใช้คำสั่งติดต่อกับอุปกรณ์ใน

12.1 การอ้างแอดเดรสใน 8086

ใน 8086 จะมองลักษณะของหน่วยความจำ โดยแบ่งหน่วยความจำเป็นกลุ่มๆในรูปแบบของ เซกเมนต์ ในหนึ่งเซกเมนต์จะชี้ได้ถึง 64 กิโลไบต์ เซกเมนต์ทั้งสี่ได้แก่ CS, DS, SS และ ES จะแสดงแอดเดรสของหน่วยความจำที่จะติดต่อด้วย CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรม DS จะเก็บค่าดาตาเซกเมนต์ขณะนั้น SS ก็จะเก็บค่าสแต็กเซกเมนต์ขณะนั้น และ ES จะกำหนดเซกเมนต์ของข้อมูลรวมที่เรียกว่า global data segment

โดยสิ่งที่สำคัญคือ เซกเมนต์จะแสดงตำแหน่งเหมือนกับเป็นพารากราฟ (paragraph) โดยจะเลื่อนไปทางซ้าย 4 บิต เพื่อที่จะกำหนดหรืออ้างแอดเดรสให้ครบ 20 เส้น โดยจุดเริ่มต้นของพารากราฟจะต้องมี 4 บิต หลังสุดเป็น 0 เช่น เป็น 00000H, 00010H, 00020H เป็นต้น เพื่อที่จะทำการติดต่อกับข้อมูลหนึ่งไบต์หรือหนึ่งเวิร์ดนั้น 8086 ได้เตรียมค่า

ออฟเซต เพื่อใช้อ้างตำแหน่งตั้งแต่จุดเริ่มต้นของเซกเมนต์แอดเดรส ตำแหน่งใดๆจะได้มาจากการบวกค่าเซกเมนต์รีจิสเตอร์กับค่าของออฟเซต 16 บิต เช่นถ้าเซกเมนต์มีค่า E89F และให้ออฟเซตมีค่า 0003H ทำการอ้างแอดเดรสไปที่ E89F3H (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544) ลักษณะในการอ้างแอดเดรส โดยระบุตำแหน่งด้วย เซกเมนต์และออฟเซต เช่น

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

การอ้างข้อมูลทั่วๆไปโดยไม่ระบุเซกเมนต์ ออฟเซตที่ระบุจะคิดเทียบกับ DS เช่น

```
mov     ax, B800h
mov     es, ax
```

12.2 แบบการอ้างแอดเดรส

อ้างแบบรีจิสเตอร์ (Register addressing) ข้อมูลอยู่ในรีจิสเตอร์ เช่น

```
mov     ax, bx
```

อ้างแบบค่าคงที่ (Immediate addressing) ข้อมูลอยู่ในคำสั่ง เช่น

```
mov     ax, 10
```

การอ้างตำแหน่งโดยตรง (Direct addressing) เป็นการอ้างแอดเดรสแบบที่ระบุค่าออฟเซตโดยตรง เช่น

```
mov     ax, [100h]
mov     [200h], cl
mov     cl, total
mov     sum, ax
```

การอ้างตำแหน่งทางอ้อมโดยใช้รีจิสเตอร์ (Register indirect addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลอยู่ในรีจิสเตอร์ เช่น

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

การอ้างตำแหน่งแบบดัชนีโดยตรง (Direct indexed addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลได้จากการนำค่ารีจิสเตอร์ดัชนี (SI หรือ DI) มาบวกกับเลขคิดเครื่องหมายขนาดแปดบิต หรือเลขไม่คิดเครื่องหมายขนาดสิบหกบิต

```
.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
```

การอ้างตำแหน่งแบบสัมพัทธ์กับฐาน (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
```

การอ้างตำแหน่งแบบดัชนีกับฐาน (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
```

ถ้าคิดสัมพัทธ์กับ BP ออฟเซตที่ได้จะคิดเทียบกับรีจิสเตอร์ SS (Stack Segment)

ตัวอย่าง โปรแกรมแสดงตัวอักษรแบบใหญ่

การแสดงตัวอักษรใหญ่ๆนั้น 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 )
```

ผู้เขียนโปรแกรมสามารถเปลี่ยนจากการเก็บ 0 และ 1 เป็น ช่องว่าง และ # แทนตามลำดับ

โปรแกรมย่อยสำหรับแสดงตัวอักษร หาตำแหน่งเริ่มต้นของตัวอักษร

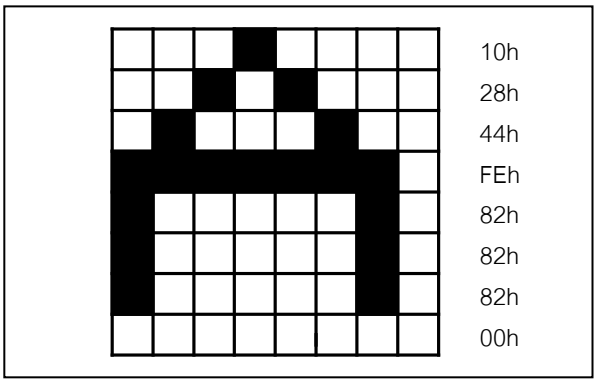
```
        mov     bx, offset fontbuf
        mov     dh, 0
        sub     dx, 'A'
        mov     cl, 6
        shl     dx, cl              ; dx = dx*16
        add     bx, dx              ; bx = buf addr
```

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
```

ผู้เขียนโปรแกรมสามารถลดขนาดการเก็บจากไบต์เป็นบิตได้โดย



ซึ่งจากวิธีนี้ทำให้ใช้ 8 ไบต์เท่านั้นในการเก็บสำหรับหนึ่งตัวอักษร

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

การแก้ไขส่วนของโปรแกรมแสดงผลใหม่ 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
```

การทำงานของคอมพิวเตอร์จะประกอบด้วยการทำงานร่วมกันของหน่วยประมวลผลกลางกับอุปกรณ์รอบข้าง ซึ่งการทำงานร่วมกันนี้จำเป็นต้องมีการส่งผ่านข้อมูลระหว่างหน่วยประมวลผลกลางและอุปกรณ์อื่น ๆ อยู่เสมอ ดังนั้นรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับ

อุปกรณ์รอบข้างจึงต้องถูกออกแบบมาอย่างเหมาะสมกับการทำงานของอุปกรณ์ต่าง ๆ ซึ่งจะแบ่งรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับอุปกรณ์รอบข้างได้ดังนี้

1. Programmed I/O หน่วยประมวลผลมีคำสั่งโดยเฉพาะสำหรับการส่งรับข้อมูลกับอุปกรณ์

2. Memory-mapped I/O ซึ่งการติดต่อแบบนี้นิยมใช้ในอุปกรณ์ที่ไม่มีคำสั่งพิเศษในการติดต่อกับอุปกรณ์ I/Oการติดต่อจะเหมือนกับการเขียนและอ่านกับหน่วยความจำ

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 หน่วยประมวลผลสั่งงาน I/O และระบุตำแหน่งในหน่วยความจำที่จะให้ I/O เขียนผลลัพธ์หรืออ่านข้อมูล จากนั้น CPU จะไปทำงานอื่น

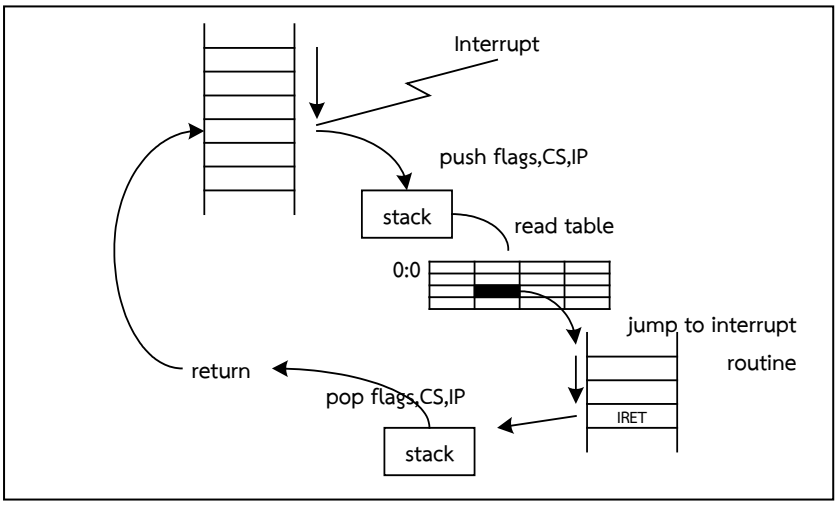
4.2 I/O จะเขียนผลลัพธ์ที่ได้หรืออ่านข้อมูลจากหน่วยความจำโดยตรงโดยไม่ผ่าน CPU

4.3 เมื่อ I/O ทำงานเสร็จจะแจ้ง CPU โดยการสร้างการขัดจังหวะ

12.3 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 จากภายนอกได้หรือไม่ ขึ้นอยู่กับค่านี้ ถ้า IF มีค่าเป็น 0 CPU จะไม่ยอมรับสัญญาณจาก maskable interrupt ขณะที่กำลัง execute interrupt service routine อยู่เมื่อ 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 แสดงได้ดังภาพ 12.1 (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)



ภาพที่ 12.1 แสดงการเก็บตำแหน่งที่อยู่ของคำสั่งถัดไป (return address) บน stack

ที่มาของการขัดจังหวะ

การขัดจังหวะของ CPU มีที่มาจาก 2 ลักษณะได้แก่

1. การขัดจังหวะที่มาจากระบบฮาร์ดแวร์ อุปกรณ์ต่างๆที่ต้องการสร้างการขัดจังหวะจะส่งสัญญาณมาที่อุปกรณ์ควบคุมการขัดจังหวะ (Interrupt Controller) โดยสัญญาณที่เข้ามาที่อุปกรณ์ควบคุมนี้มีทั้งสิ้น 16 สัญญาณ (IRQ0 – IRQ16) จากนั้นอุปกรณ์ควบคุมการขัดจังหวะส่งสัญญาณมาที่ CPU พร้อมทั้งส่งหมายเลขของการขัดจังหวะมาพร้อมกันด้วย โดยถ้ามีการขัดจังหวะหลายอันที่ร้องขอพร้อมกันอุปกรณ์นี้จะจัดลำดับความสำคัญของอุปกรณ์และส่ง

สัญญาณที่มีความสำคัญสูงสุดก่อน อุปกรณ์ที่สร้างการขัดจังหวะกลุ่มนี้ได้แก่ นาฬิกาของระบบ แป้นพิมพ์ ฮาร์ดดิสก์ การ์ดเสียง ฯลฯ

2. การขัดจังหวะที่มาจากระบบซอฟต์แวร์ หรืออาจจะเรียกได้ว่าเป็นการขัดจังหวะที่เกิดขึ้นจากอุปกรณ์ภายในตัว CPU เองได้แก่ การเกิดความผิดพลาดในการทำงาน และการสั่งคำสั่ง 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

ตัวอย่าง การรับตัวอักษรจากแป้นพิมพ์และแสดงผล

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 ;แสดงสายตัวอักษร(string)บนจอภาพจนกว่าจะพบตัวอักษร
        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
```

12.4 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) เราสามารถระบุค่านี้โดยผ่าน
        ทาง register DX ได้ หมายเลขพอร์ตนี้มีค่าตั้งแต่ 0 ถึง 65535
```

ตัวอย่าง การติดต่อกับอุปกรณ์รอบข้าง : การติดต่อกับลำโพง

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
```

การสั่งงานลำโพงอย่างง่ายๆสามารถทำได้โดยสั่งงานบิตที่ 1 โดยการกำหนดค่าลงในบิตนี้เปลี่ยนค่าไปมาระหว่าง 0 และ 1 ซึ่งจะทำให้ลำโพงเกิดการสั่นขึ้น การสั่งงานนั้นต้องสั่งงานโดยไม่ให้กระทบกับบิตอื่นๆ เช่น

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

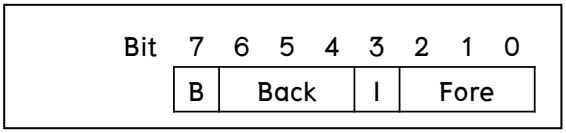
ในการสร้างเสียงต้องมีการหน่วงเวลาในการเปลี่ยนค่าบิตควบคุมลำโพง (บิตที่ 1) เพราะความถี่สูงเกินไปลำโพงจะไม่สามารถเคลื่อนที่ทันอีกทั้งให้ผู้ใช้สามารถได้ยินเสียงนั้นด้วย

ตัวอย่าง โปรแกรมย่อยสำหรับสร้างเสียงอย่างง่าย

```
gensoundproc    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
gensoundendp
```

ตัวอย่าง การติดต่อกับอุปกรณ์รอบข้าง : ระบบการแสดงผลแบบตัวอักษร

การแสดงแบบตัวอักษรใน IBMPC นั้น ในแต่ละตัวอักษรที่แสดงจะมีค่าที่เกี่ยวข้อง 2 ค่าคือ รหัสแอสกีของตัวอักษรนั้น และแสดงค่า display attribute ของตัวอักษร ซึ่งเป็นค่าที่บอกสีของตัวอักษร และลักษณะการแสดงผล โดยบิตที่ 0–3 แสดงสีของตัวอักษร และบิตที่ 4–7 แสดงสีของพื้นหลัง



ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมจัดการการแสดงผลที่หน้าจอได้สองวิธี

1. ใช้บริการของ Video BIOS [Interrupt 10H]

ตัวอย่าง แสดงรูปบนจอโดยใช้ 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 โดยการเก็บขอมูลจะมีลักษณะเก็บทีละตัวอักษรเรียงกันไปตามลำดับ ซึ่งการเก็บข้อมูลจะเก็บตัวอักษรละ 2 ไบต์ ไบต์แรก (ไบต์ต่ำ) จะเก็บค่ารหัสแอสกีไบต์ที่สองจะเก็บค่า display attribute การเก็บจะเก็บเรียงทีละบรรทัด บรรทัดละ 80 ตัวอักษร

ตัวอย่าง การคำนวณข้อมูลของตัวอักษรที่บรรทัดที่ 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         EX, 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
END             START
```

ตัวอย่าง โปรแกรมย่อยเขียนตัวอักษรลงบนหน้าจอโดยตรง

```
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

8086 จะมองลักษณะของหน่วยความจำ โดยแบ่งหน่วยความจำเป็นกลุ่มๆในรูปแบบของเซกเมนต์ ในหนึ่งเซกเมนต์จะชี้ได้ถึง 64 กิโลไบต์ เซกเมนต์ทั้งสี่ได้แก่ CS, DS, SS และ ES จะแสดงแอดเดรสของหน่วยความจำที่จะติดต่อด้วย CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรม DS จะเก็บค่าดาตาเซกเมนต์ขณะนั้น SS ก็จะเก็บค่าสแต็กเซกเมนต์ขณะนั้น และ ES จะกำหนดเซกเมนต์ของข้อมูลรวมที่เรียกว่า global data segment

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. 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 อ้างแบบสัมพัทธ์กับฐาน (Base relative addressing)
   2.7 อ้างแบบดัชนีกับฐาน (Base indexed addressing)
3. จงเขียนโปรแกรมแสดงตัวอักษร B และมีการเก็บตัวอักษรที่จะพิมพ์ไว้ในหน่วยความจำโดยเก็บเป็นตารางขนาด 8*8
4. จงอธิบายรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับอุปกรณ์รอบข้างต่อไปนี้
   4.1 Programmed I/O
   4.2 Memory-mapped I/O
   4.3 Interrupt I/O
   4.4 Direct Memory Access I/O
5. การขัดจังหวะของ CPU มีกี่ลักษณะอะไรบ้าง
6. เมื่อคำสั่ง INT ถูก execute CPU จะดำเนินการโดยมีขั้นตอนการทำงานอย่างไร
7. จงอธิบายหน่วยประมวลผลของ 8086 สามารถติดต่อกับอุปกรณ์ในแบบต่างๆ ต่อไปนี้ได้อย่างไร
   7.1 Programmed I/O
   7.2 Interrupt I/O
   7.3 Direct Memory Access
8. จงเขียนโปรแกรมแสดงตัวอักษรที่มีคำว่า “Computer Science Suan Dusit” บนจอภาพ

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 18 มิถุนายน 2557,จาก :http://rirs3.royin.go.th/coinages/

การอ้างอิงแอดเดรสของหน่วยความจำ (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 18 มิถุนายน 2557, จาก:http://th.wikipedia.org/wiki/

การขัดจังหวะ (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 18 มิถุนายน 2557, จาก: http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย–ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 13

หัวข้อเนื้อหา คำสั่งจัดการกับสายข้อมูล (Character Instruction)

รายละเอียด

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

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

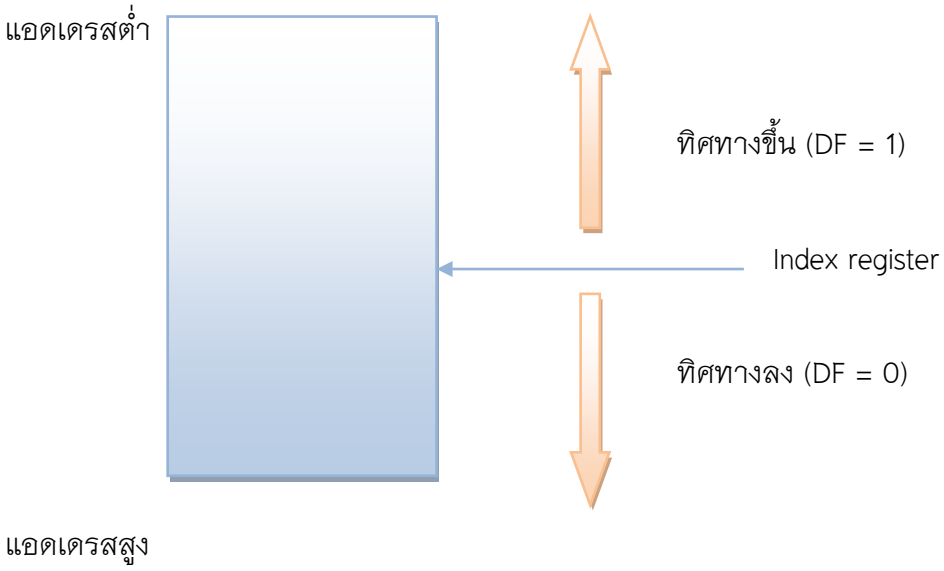
3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 13 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่งจัดการสายข้อมูลต่างๆ เช่น คำสั่ง MOVS, คำสั่ง REP, คำสั่ง STOS, คำสั่ง LODS, คำสั่ง REPZ, คำสั่ง CMPS, คำสั่ง REPNZ และคำสั่ง SCAS เป็นต้น

คำสั่งจัดการกับสายข้อมูล เป็นคำสั่งที่ใช้จัดการกับข้อมูลเป็นที่เป็นชุดเรียงต่อกัน คำสั่งในกลุ่มนี้จะระบุตำแหน่งของข้อมูลด้วย Index register คือ SI (Source Index) และ DI (Destination Index) ประกอบกับ DS และ EX โดย ข้อมูลต้นทางจะอยู่ที่ตำแหน่ง DS:SI และข้อมูลปลายทางจะอยู่ที่ ES:DI การทำงานคำสั่งในกลุ่มนี้จะมีการปรับค่าของรีจิสเตอร์ที่เก็บตำแหน่งของข้อมูลให้โดยอัตโนมัติ เพื่อว่าเราจะสามารถใช้คำสั่งนั้นกระทำกับข้อมูลที่ตำแหน่งติดกันถัดไปได้โดยไม่ต้องปรับค่าของรีจิสเตอร์และทิศทางในการปรับค่าจะขึ้นอยู่กับค่าที่กำหนดในแฟล็กทิศทาง (DF : Direction flag) (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

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



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

คำสั่งต่างๆ ในกลุ่มนี้ ประกอบไปด้วย

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

หมายเหตุ ตัวแปร x แทนค่าของ B (byte) หรือ W (word) หรือ D (double) ซึ่งหมายถึงขนาดของข้อมูลที่ถูกกระทำด้วยคำสั่งนั้นๆ

13.1 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
  ...
```

13.2 คำสั่ง REP

คำสั่งการกระทำกับสายข้อมูลนั้นมักจะต้องทำงานซ้ำเป็นวงรอบเพราะในการเรียกคำสั่งแต่ละครั้งนั้นจะเป็นการกระทำกับข้อมูลทีละ 1 ตัวเท่านั้น ดังเช่นในตัวอย่างข้างต้นเป็นการใช้คำสั่ง MOVSW ร่วมกับคำสั่ง LOOP แต่เราก็อาจจะนำคำสั่ง REP มาใช้แทนได้

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

ตัวอย่างคำสั่ง

```
        mov     cx,100
copy:   movsw
        loop    copy
```

ผู้เขียนโปรแกรมสามารถแทนบรรทัดข้างต้นทั้ง 3 ได้ด้วย

```
mov     cx,500
rep     movsw
```

13.3 คำสั่ง STOS

คำสั่ง STOS เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลขนาดไบท์จากรีจิสเตอร์ 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
    ...
```

13.4 คำสั่ง LODS

คำสั่ง LODS เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลขนาดไบท์จากหน่วยความจำตำแหน่ง 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
  ...
```

โดยปกติจะไม่พบการใช้คำสั่ง 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
  ...
```

จากโปรแกรมข้างต้นเป็นตัวอย่างการใช้คำสั่ง 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
  ...
```

13.5 คำสั่ง REPZ

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

13.6 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 เราจะต้องทำการเปรียบเทียบข้อมูลไปทีละตัวจนกว่าจะหมดข้อมูลหรือพบข้อมูลที่ไม่เท่ากันเป็นตัวแรก 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:   ...
        ...
```

13.7 คำสั่ง REPNZ

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

13.8 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 REPNZ แทนคำสั่ง REP เพราะคำสั่ง REPNZ 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) ประกอบกับ DS และ EX โดย ข้อมูลต้นทางจะอยู่ที่ตำแหน่ง DS:SI และข้อมูลปลายทางจะอยู่ที่ ES:DI การทำงานคำสั่งในกลุ่มนี้จะมีการปรับค่าของรีจิสเตอร์ที่เก็บตำแหน่งของข้อมูลให้โดยอัตโนมัติ เพื่อว่าเราจะสามารถใช้คำสั่งนั้นกระทำกับข้อมูลที่ตำแหน่งติดกันถัดไปได้โดยไม่ต้องปรับค่าของรีจิสเตอร์เอง ทิศทางในการปรับจะขึ้นกับค่าที่กำหนดในแฟล็กทิศทาง (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

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 8 กรกฎาคม 2557,จาก :http://rirs3.royin.go.th/coinages/

คำสั่งจัดการกับสายข้อมูล (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 8 กรกฎาคม 2557,จาก :http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 14

หัวข้อเนื้อหา คำสั่งตารางและการสร้างแมคโคร (Macro and Instruction Table)

รายละเอียด

ศึกษาเกี่ยวกับแมคโคร ซึ่งเป็นส่วนของโปรแกรมที่ได้กำหนดชื่อไว้ เมื่อใดก็ตามที่มีการเรียกใช้ชื่อนั้นในโปรแกรม ส่วนของโปรแกรมที่มีชื่อดังกล่าวก็จะถูกนำไปแทนที่แมคโครมีรูปแบบการประกาศดังนี้

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 14 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่ง XLAT ความหมาย ข้อดีและข้อเสียของการใช้แมคโคร พารามิเตอร์ของแมคโครการใช้ LABEL ในแมคโครและแมคโครกับโปรแกรมย่อย

14.1 คำสั่ง XLAT

คำสั่งด้านล่างนี้เป็นคำสั่งที่ผิดรูปแบบไม่สามารถใช้ได้

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

โดยที่ค่าใน AL เป็นหมายเลขช่องของตารางและค่าใน BX เป็นแอดเดรสของตาราง คำสั่งนี้มีการนำเอาค่าในตารางที่ตำแหน่งด้วย BX ช่องที่ AL มาใส่ใน Al นั่นเอง ซึ่งในหลาย ๆ ครั้งมีความต้องการจะทำงานในลักษณะนี้ ซึ่งคำสั่งข้างต้นสามารถใช้งานได้ด้วยคำสั่ง XLAT แทน

ตัวอย่าง โปรแกรมแปลงค่าเลขฐานสิบหกหนึ่งหลักใน AL เป็นอักขระฐานสิบหก

```
.data
hextab  db      '0123456789ABCDEF'
.code
  mov   bx,offset hextab
  xlat
```

14.2 แมคโคร

แมคโครคือส่วนของโปรแกรมที่ได้กำหนดชื่อไว้ เมื่อใดก็ตามที่มีการเรียกใช้ชื่อนั้นในโปรแกรม ส่วนของโปรแกรมที่มีชื่อดังกล่าวก็จะถูกนำไปแทนที่แมคโครมีรูปแบบการประกาศดังนี้

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

ตัวอย่างคำสั่ง

```
print     macro
  mov     bx,offset msg
  mov     ah,09h
  int     21h
  endm
.data
msg       db      'TEST$'
.code
  ...
  print
  ...
```

เมื่อโปรแกรมข้างต้นถูกขยายจะมีลักษณะดังนี้

```
.data
msg       db      'TEST$'
.code
  ...
```

```
;print
mov     bx,offset msg
mov     ah,09h
int     21h
...
```

14.3 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
```

เมื่อโปรแกรมข้างต้นถูกขยายจะมีลักษณะดังนี้

```
.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
```

14.4 การใช้ LABEL ในแมคโคร

ตัวอย่างคำสั่ง

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

เมื่อโปรแกรมข้างต้นถูกขยายจะมีลักษณะดังนี้

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

จะเห็นได้ว่ามีการซ้ำกันของ LABEL สามารถกำหนดให้ MASM สร้าง LABEL ที่มีชื่อไม่ซ้ำกันได้โดยจะต้องประกาศให้เป็น LABEL แบบ LOCAL

ตัวอย่างคำสั่ง

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

เมื่อโปรแกรมข้างต้นถูกขยายจะมีลักษณะดังนี้

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

14.5 แมคโครกับโปรแกรมย่อย

คำสั่งในส่วนของแมคโครจะถูกนำไปแทนที่ในตำแหน่งที่มีการเรียกใช้ ในการทำงานจริงจะไม่มีการกระโดดไปทำงาน แต่ถ้ามีการเรียนใช้แมคโครหลายครั้ง โปรแกรมส่วนนั้นก็จะถูกขยายออกมาหลายครั้ง ส่วนโปรแกรมย่อยจะเป็นส่วนของโปรแกรมที่มีเพียงชุดเดียว การเรียกใช้โปรแกรมย่อยจะเป็นการกระโดดไปทำงานในโปรแกรมย่อยนั้น

แมคโครมีข้อดี คือ ลดเวลาในการกระโดดไปทำงานสำหรับส่วนของคำสั่งที่มีขนาดเล็ก แต่แมคโครก็มีข้อเสียคือถ้าส่วนของแมคโครมีขนาดใหญ่หรือต้องเรียกใช้หลายครั้ง จะทำให้โปรแกรมที่ได้จากการแปลมีขนาดใหญ่

สรุป

แมคโครคือส่วนของโปรแกรมที่ได้กำหนดชื่อไว้ เมื่อใดก็ตามที่มีการเรียกใช้ชื่อนั้นในโปรแกรม แมคโครมีข้อดี คือ ลดเวลาในการกระโดดไปทำงานสำหรับส่วนของคำสั่งที่มีขนาดเล็ก แต่แมคโครก็มีข้อเสียคือถ้าส่วนของแมคโครมีขนาดใหญ่หรือต้องเรียกใช้หลายครั้งจะทำให้โปรแกรมที่ได้จากการแปลมีขนาดใหญ่ไปด้วย

คำถามทบทวน

1. จงอธิบายการทำงานของคำสั่ง XLAT
2. จงแสดงวิธีการใช้งานการใช้ LABEL ในแมคโคร
3. พารามิเตอร์ของแมคโครมีหน้าที่อะไร
4. การซ้ำกันของ LABEL สามารถกำหนดให้ MASM สร้าง LABEL ที่มีชื่อไม่ซ้ำกันได้โดยจะต้องประกาศให้เป็นแบบใด
5. แมคโครกับโปรแกรมย่อยคืออะไร มีข้อดีและข้อเสียอย่างไร

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 25 กรกฎาคม 2557, จาก:http://rirs3.royin.go.th/coinages/

แมคโครกับโปรแกรมย่อย (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 25 กรกฎาคม 2557,จาก :http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 15

หัวข้อเนื้อหา การฝึกทักษะการเขียนโปรแกรมภาษาแอสแซมบลีเบื้องต้น
(Skill Training Programs in a Primary Assembly)

รายละเอียด

ศึกษาเกี่ยวข้องกับการเรียนรู้และการฝึกทักษะให้กับนักพัฒนาโปรแกรม ที่ต้องการเขียนโปรแกรมภาษาระดับต่ำ (Low Level Language) หรือบางครั้งอาจเรียกว่าการเขียนโปรแกรมภาษาเครื่อง โดยเฉพาะการเขียนโปรแกรมภาษาแอสแซมบลีเบื้องต้นซึ่งเหมาะสำหรับนักศึกษาที่ยังไม่เคยเขียนโปรแกรมหรือเขียนโปรแกรมแล้วแต่ยังไม่ค่อยเข้าใจเท่าทีควร ซึ่งเขียนโปรแกรมภาษาแอสแซมบลีนี้จัดเป็นภาษาระดับต่ำ ถ้าเทียบกับภาษาระดับสูง เช่น C++, JAVA , VISUAL BASIC เป็นต้น แต่ภาษา assembly ก็จะมีคุณสมบัติที่ดีกว่าภาษาอื่นตรงที่จะใช้เวลาในการทำงานของเครื่องน้อยกว่า

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง

1.2 สังเกตจากการตอบคำถาม

1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

2.1 ตรวจผลงานภาคปฏิบัติ

2.2 ตรวจรายงาน

2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

3.1 ใบงานที่นักศึกษาทำมาส่ง

3.2 คะแนนเก็บในชั้นเรียน

3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 15 การจัดการเรียนการสอน จะเกี่ยวข้องกับวิธีการดาวน์โหลดและติดตั้งโปรแกรม MASM32 รวมถึงวิธีการดาวน์โหลดและติดตั้งโปรแกรม EditPlus การประยุกต์ใช้งานคำสั่งและฝึกทักษะการเขียนโปรแกรมภาษาแอสแซมบลีและตรวจสอนความถูกต้องของโปรแกรมที่เขียนขึ้นมาเบื้องต้นได้ การฝึกทักษะให้กับนักพัฒนาโปรแกรมที่ต้องการเขียนโปรแกรมภาษาระดับต่ำ (Low Level Language) หรือบางครั้งอาจเรียกว่าการเขียนโปรแกรมภาษาเครื่อง โดยเฉพาะการเขียนโปรแกรมภาษาแอสแซมบลีเบื้องต้นซึ่งเหมาะสำหรับนักศึกษาที่ยังไม่เคยเขียนโปรแกรมหรือเขียนโปรแกรมแล้วแต่ยังไม่ค่อยเข้าใจเท่าที่ควรซึ่งเขียนโปรแกรมภาษาแอสแซมบลีนี้จัดเป็นภาษาระดับต่ำ ถ้าเทียบกับภาษาระดับสูง เช่น C++, JAVA , VISUAL BASIC เป็นต้น แต่ภาษา assembly ก็จะมีคุณสมบัติที่ดีกว่าภาษาอื่นตรงที่จะใช้เวลาในการทำงานของเครื่องน้อยกว่า

Microsoft Macro Assembler (MASM) เป็นโปรแกรมสำหรับแปลภาษาแอสเซมบลี ที่เรียกว่า แอสเซมเบลอร์ (assembler) ใช้กับเครื่องคอมพิวเตอร์ที่ใช้ซีพียูตระกูล x86 แต่เดิมผลิตโดยบริษัทไมโครซอฟต์สำหรับใช้กับระบบปฏิบัติการดอส (MS-DOS) และเป็นโปรแกรมสำหรับแปลภาษาแอสแซมบลีที่นิยมใช้มากที่สุด (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

การฝึกทักษะและการเขียนโปรแกรมภาษาแอสแซมบลีเบื้องต้นประกอบด้วย

15.1 การดาวน์โหลดและติดตั้งโปรแกรม MASM32

วิธีการดาวน์โหลดและติดตั้งโปรแกรม MASM32

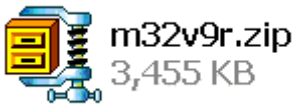
ขั้นตอนที่ 1. เข้าเว็บไซต์ไปที่ http://dusithost.dusit.ac.th/~juthawut_cha/home.htm

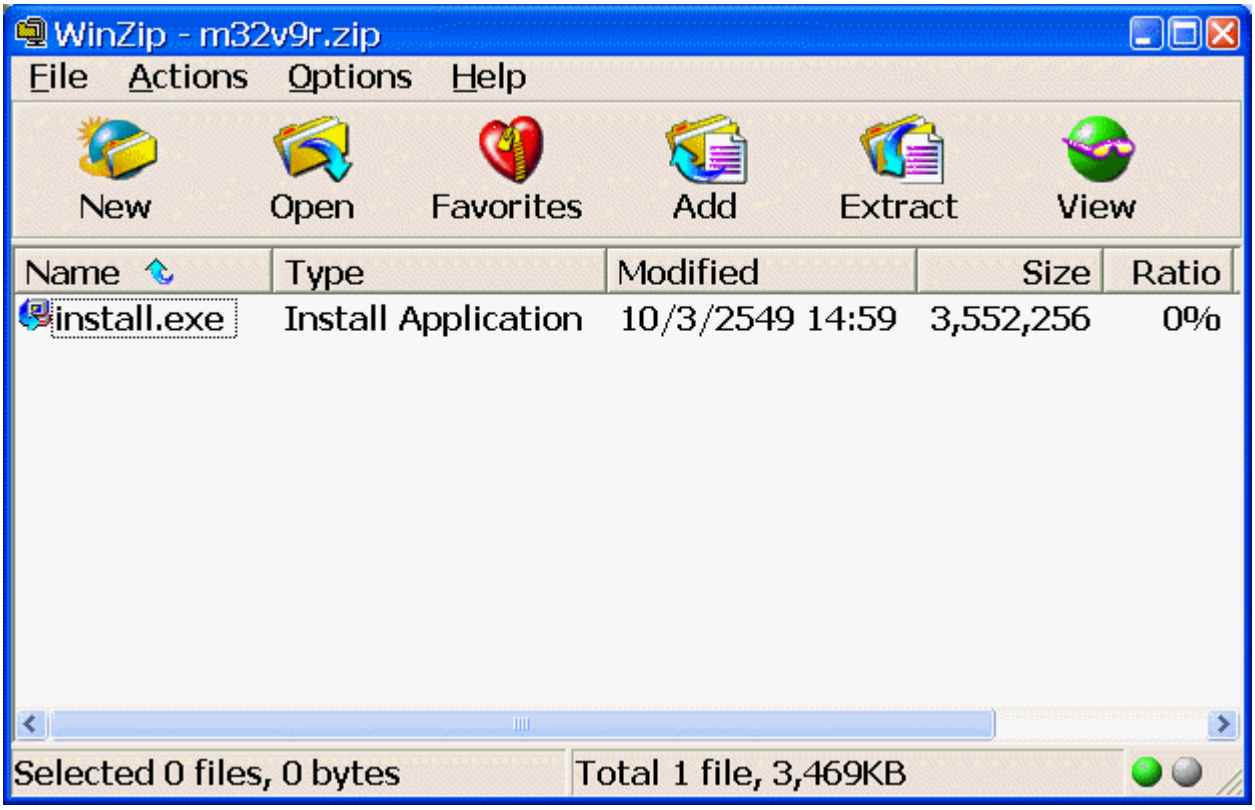
เลือก การฝึกทักษะการเขียนโปรแกรมเบื้องต้น (Programming Skills)

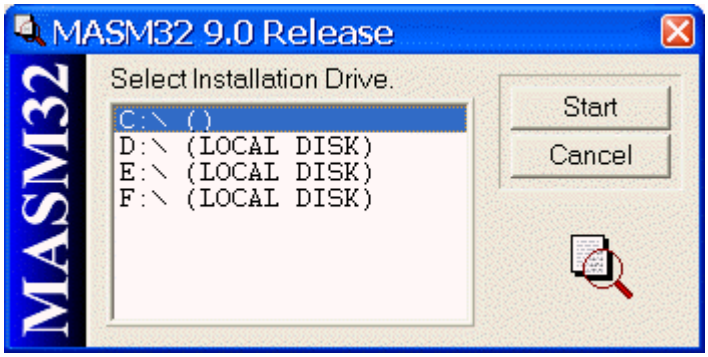
1. ไปที่ ดาว์นโหลด โปรแกรมสำหรับฝึกทักษะการเรียนรู้การเขียนโปรแกรมภาษาแอสแซมบลี คลิก **Download MASM32 Version 9**, ดาวน์โหลดไฟล์ **m32v9r.zip** ขนาด 3.37 MB

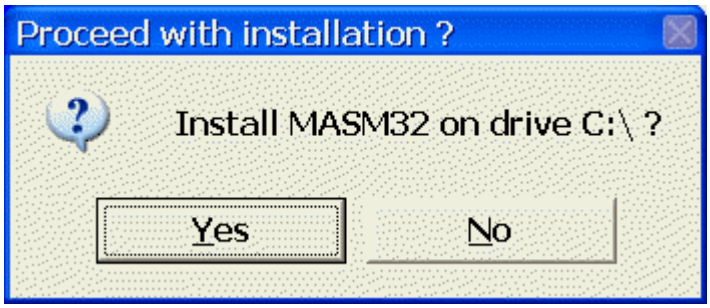
2. Uncompress the file **m32v9r.zip** and run the installer file **install.exe** to start the installation process until finish.

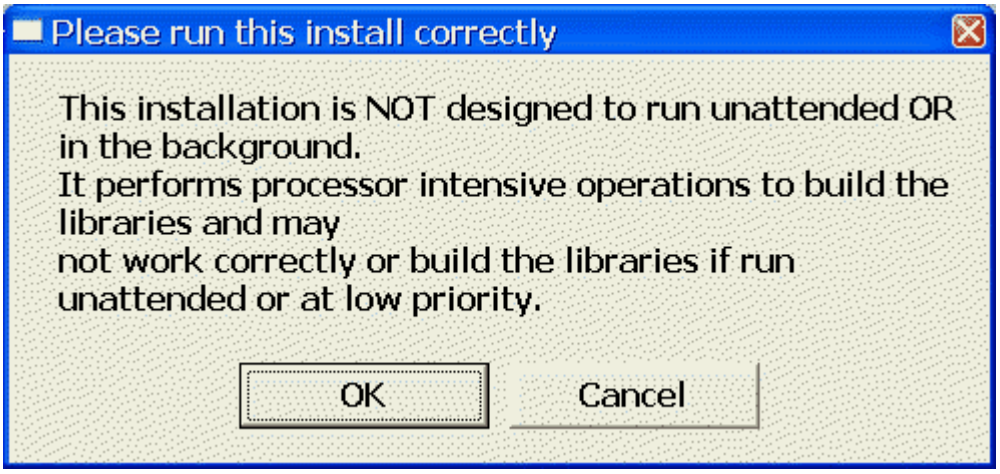
2. ขยายไฟล์ m32v9r.zip และรันตัวติดตั้ง install.exe เพื่อเริ่มกระบวนการติดตั้งไปจนเสร็จ



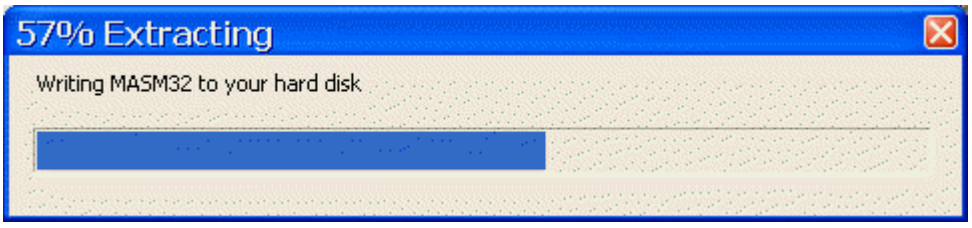


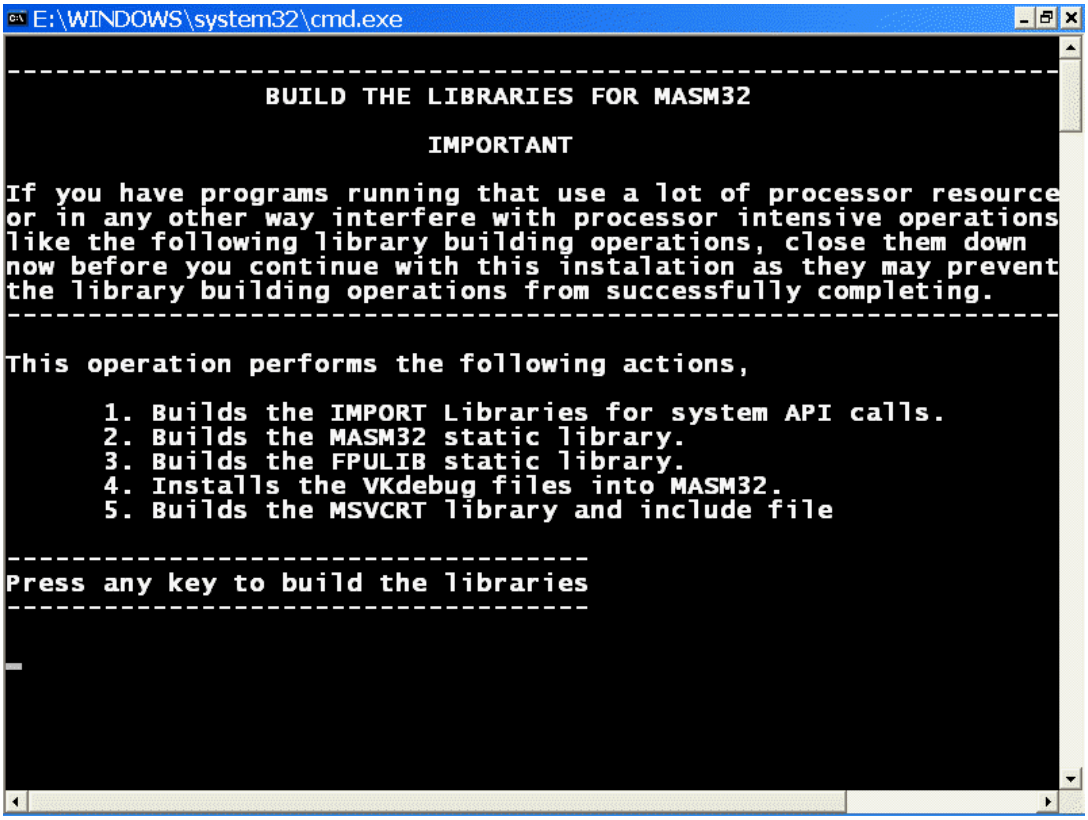


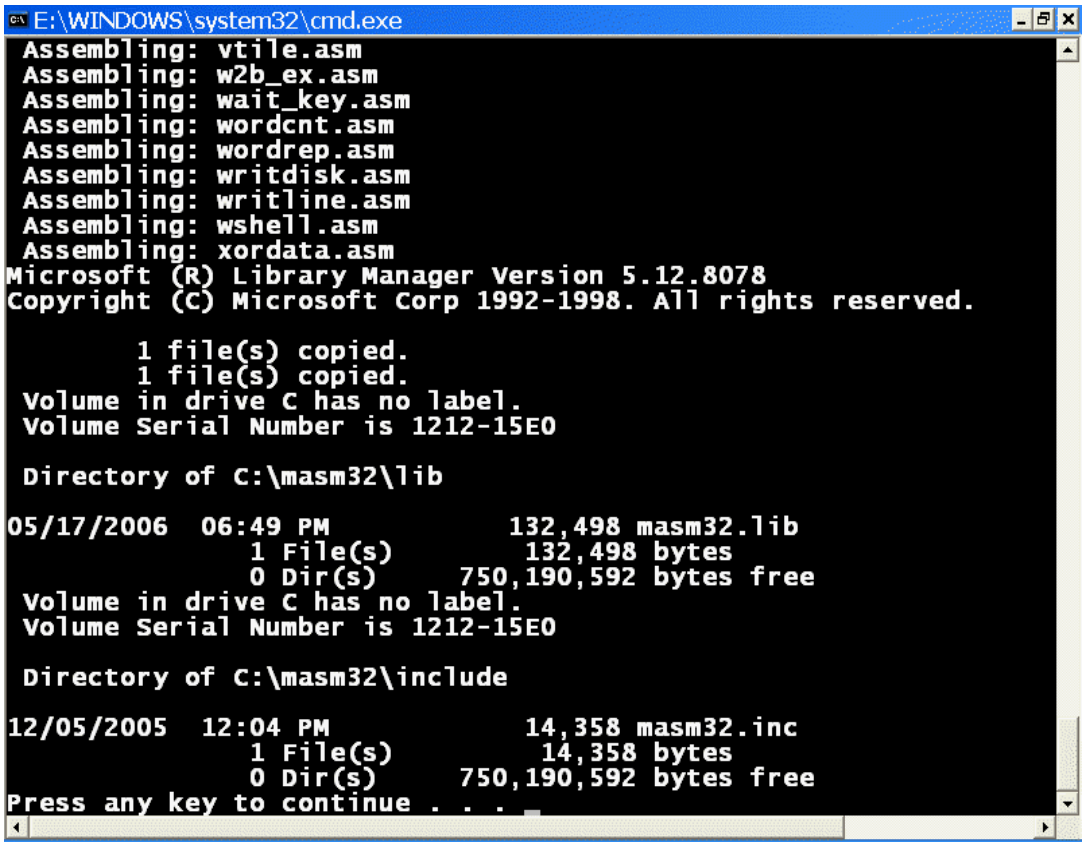

Proceed with installation ?
Install MASM32 on drive C:\ ?
Yes
No

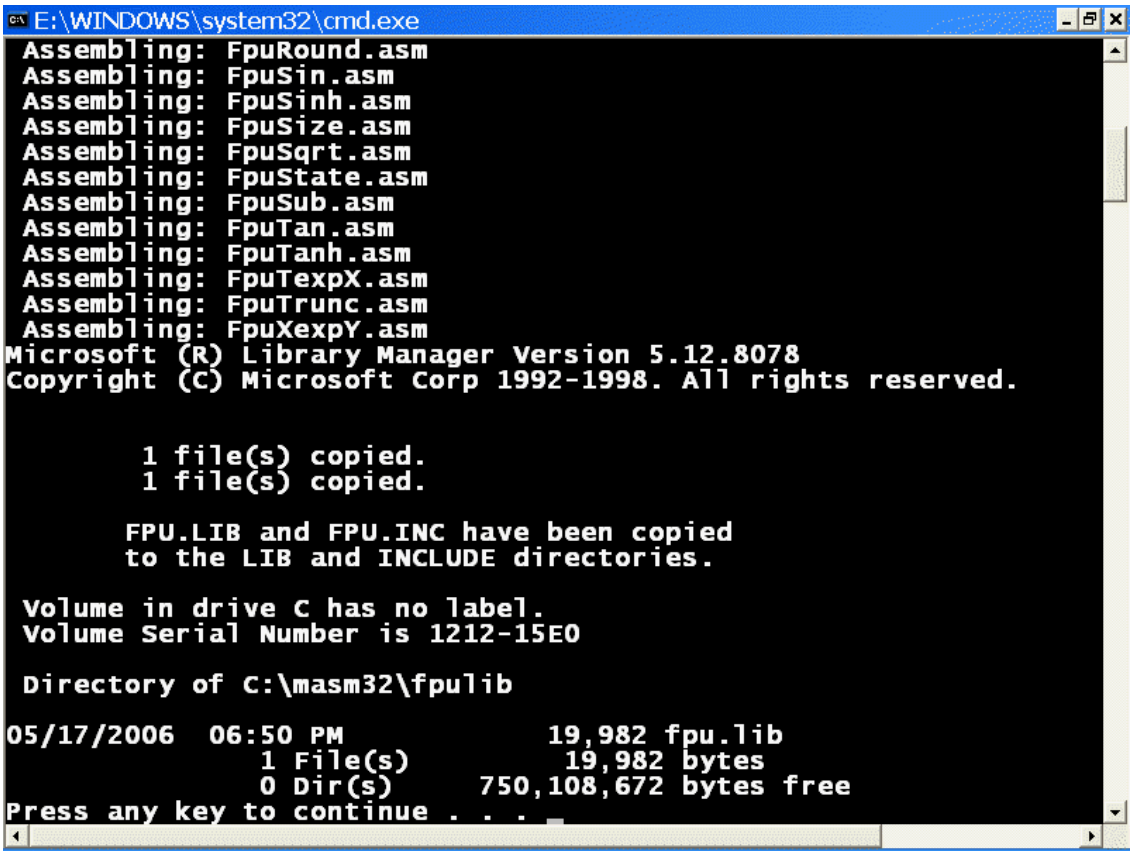

Please run this install correctly
This installation is NOT designed to run unattended OR in the background.
It performs processor intensive operations to build the libraries and may
not work correctly or build the libraries if run unattended or at low priority.
OK
Cancel

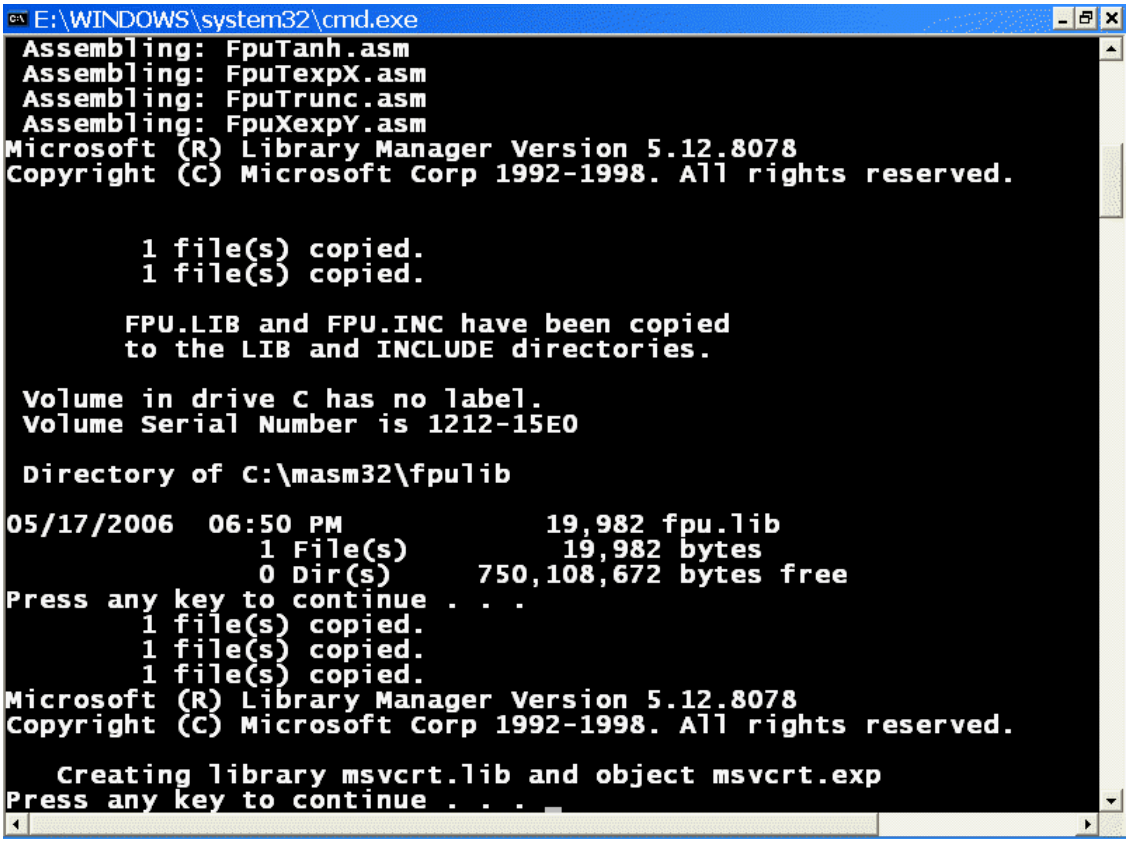

MASM32 Version 9.00 Release
Click Extract to Proceed
Extract

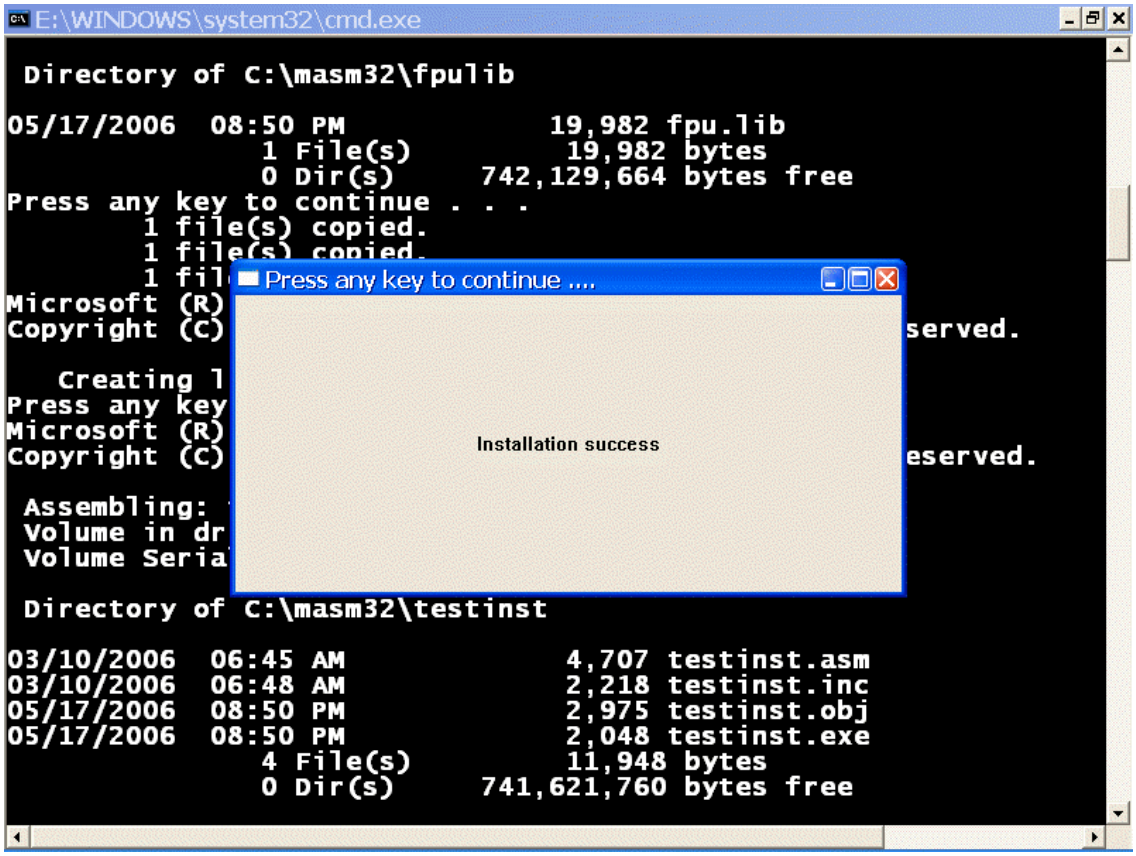

57% Extracting
Writing MASM32 to your hard disk

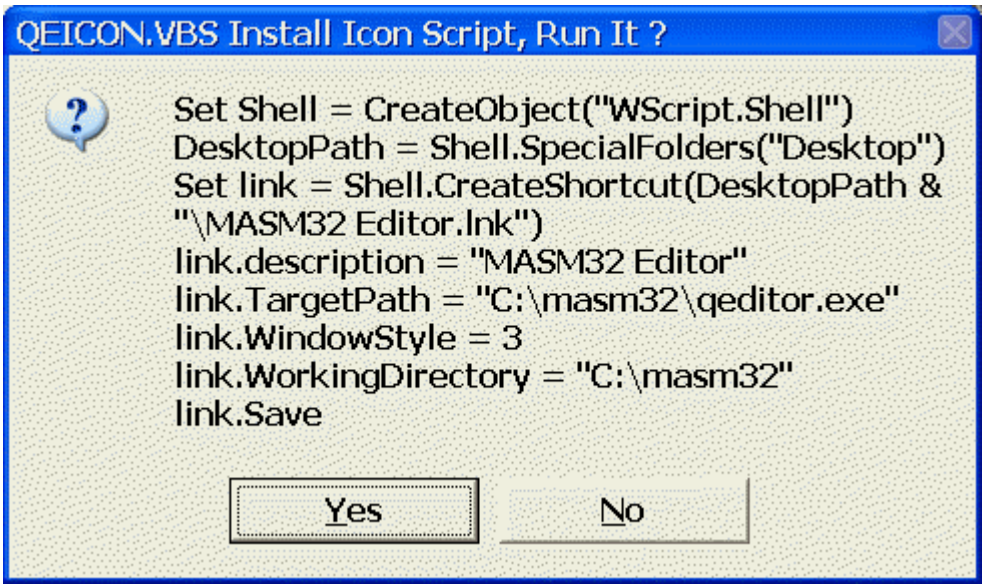

E:\WINDOWS\system32\cmd.exe
BUILD THE LIBRARIES FOR MASM32
IMPORTANT
If you have programs running that use a lot of processor resource
or in any other way interfere with processor intensive operations
like the following library building operations, close them down
now before you continue with this instalation 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

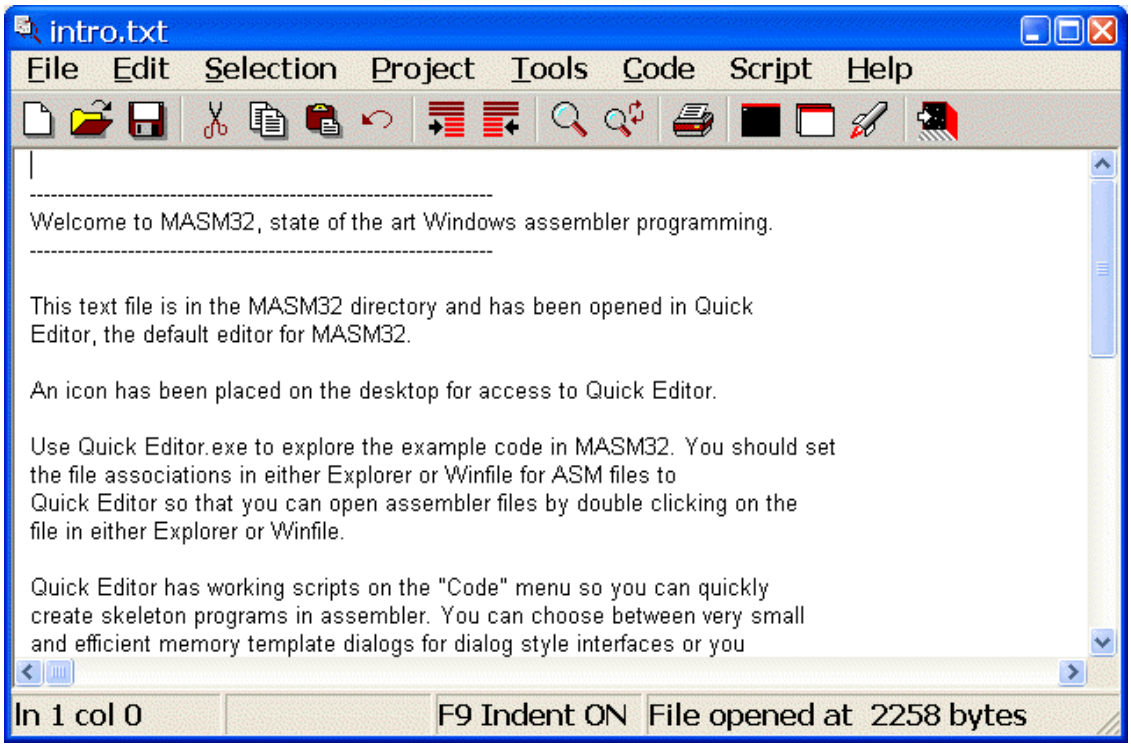

E:\WINDOWS\system32\cmd.exe
Assembling: vtile.asm
Assembling: w2b_ex.asm
Assembling: wait_key.asm
Assembling: wordcnt.asm
Assembling: wordrep.asm
Assembling: writdisk.asm
Assembling: writline.asm
Assembling: wshell.asm
Assembling: xordata.asm
Microsoft (R) Library Manager Version 5.12.8078
Copyright (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 . . .

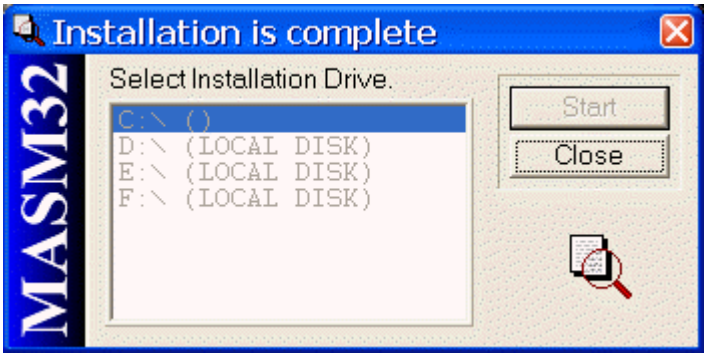

E:\WINDOWS\system32\cmd.exe
Assembling: FpuRound.asm
Assembling: FpuSin.asm
Assembling: FpuSinh.asm
Assembling: FpuSize.asm
Assembling: FpuSqrt.asm
Assembling: FpuState.asm
Assembling: FpuSub.asm
Assembling: FpuTan.asm
Assembling: FpuTanh.asm
Assembling: FpuTexpX.asm
Assembling: FpuTrunc.asm
Assembling: 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.
Press any key to continue ....
Installation success
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


QEICON.VBS Install Icon Script, Run It ?
Set Shell = CreateObject("WScript.Shell")
DesktopPath = Shell.SpecialFolders("Desktop")
Set link = Shell.CreateShortcut(DesktopPath &
"\MASM32 Editor.lnk")
link.description = "MASM32 Editor"
link.TargetPath = "C:\masm32\qeditor.exe"
link.WindowStyle = 3
link.WorkingDirectory = "C:\masm32"
link.Save
Yes
No





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) ที่อยู่ในไดเรคทอรี 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 ดาวน์โหลดไฟล์ **lnk563.exe** ขนาด 274 KB จากเส้นทางดังนี้:

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

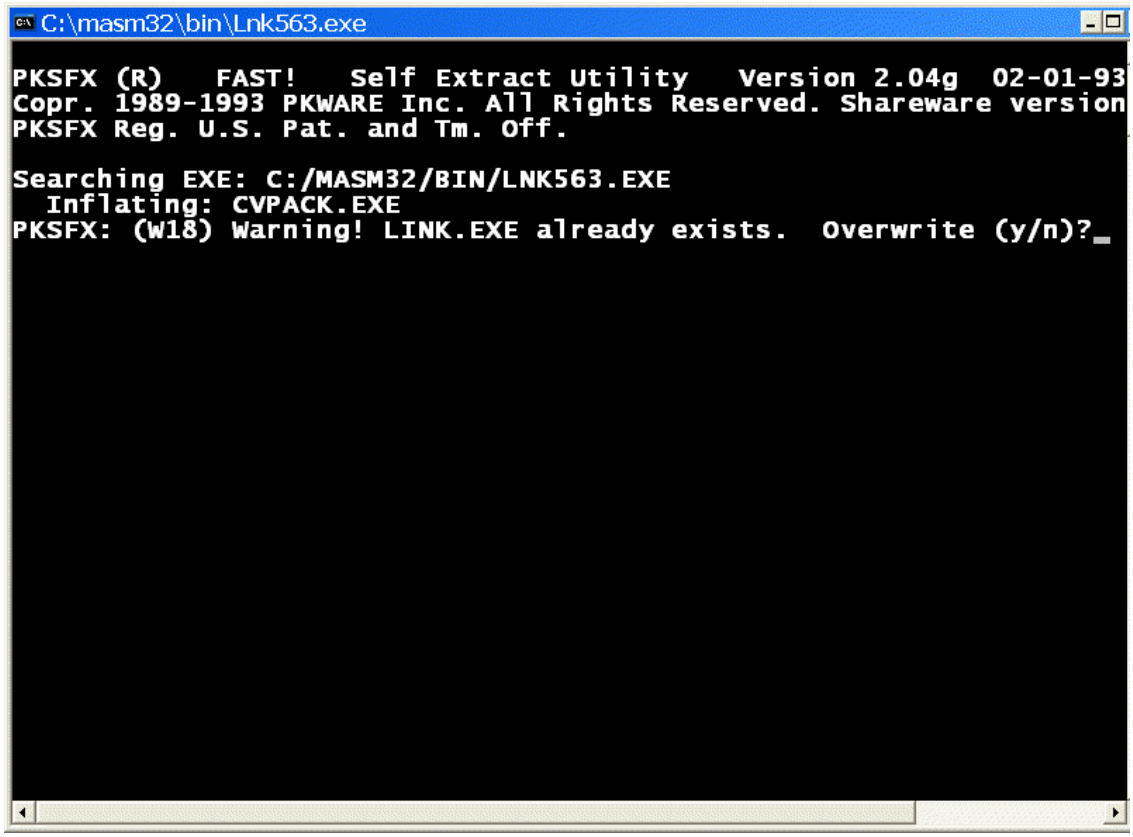
http://www.scs.carleton.ca/~sivarama/asm_book/source/win_nasm_readme.pdf

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

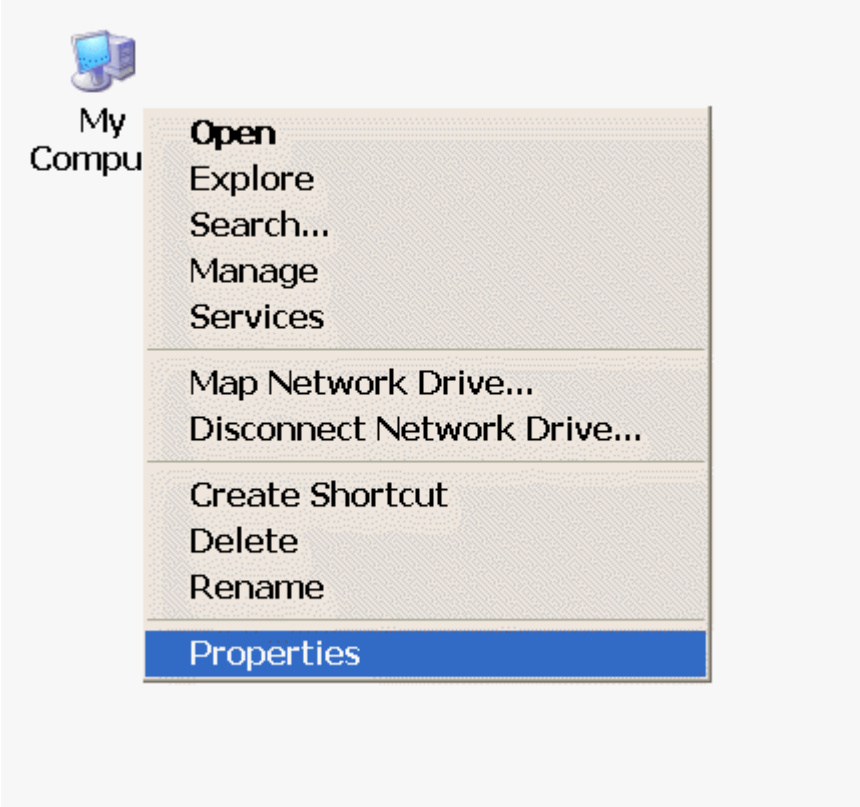
4. คัดลอกไฟล์นี้ไปไว้ที่ **C:\MASM32\BIN** และรันไฟล์ ให้ตอบใช่ (**y**) เมื่อโปรแกรมถามว่าจะเขียนทับหรือไม่

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



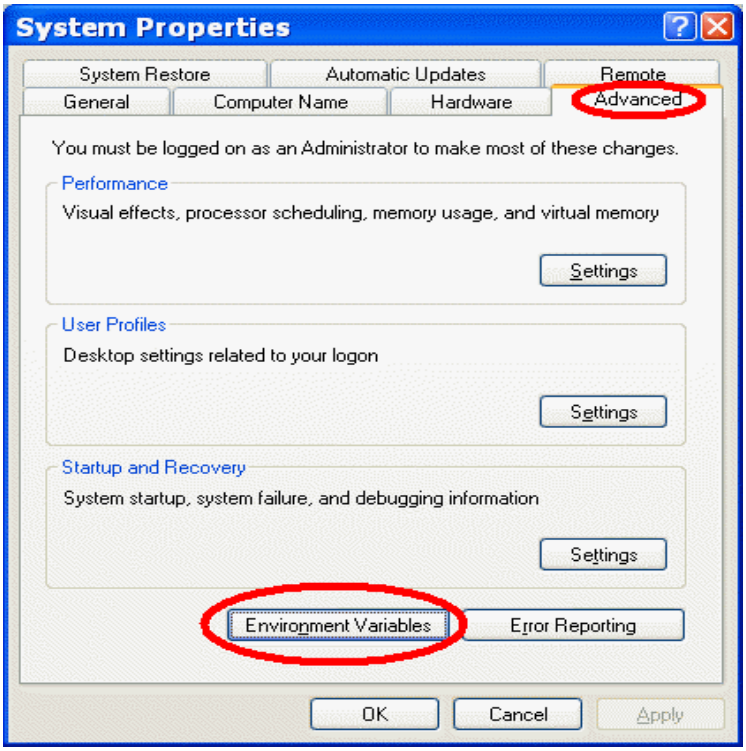
5. Right–click **my Computer** and click on **Poperties**.

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



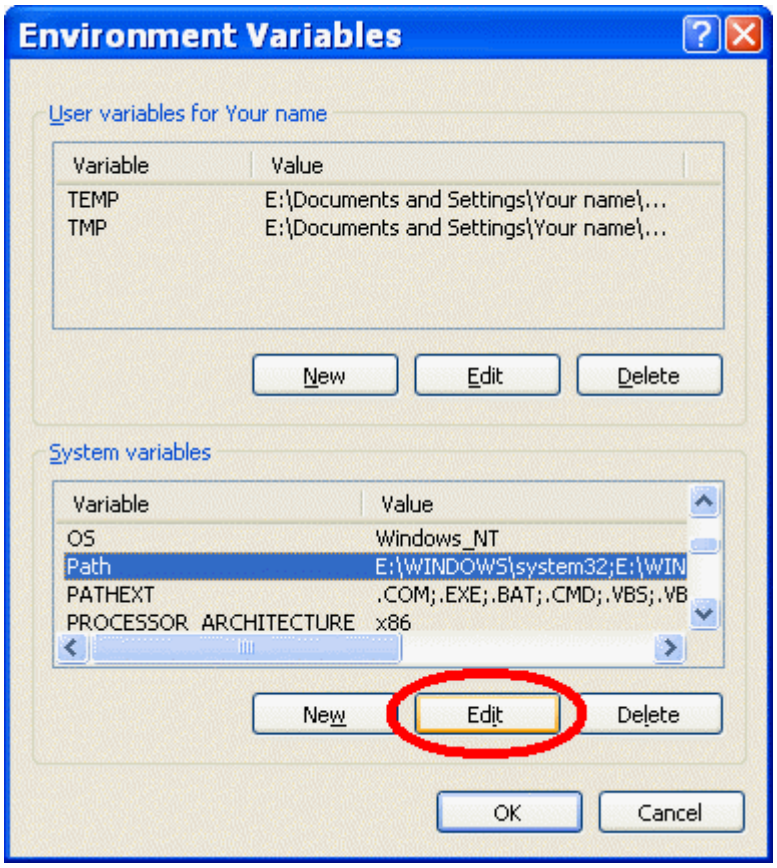
6. On the **Advanced** tab, click **Environment Variables**.

6. ที่แท็บ **Advanced**, คลิก **Environment Variables**



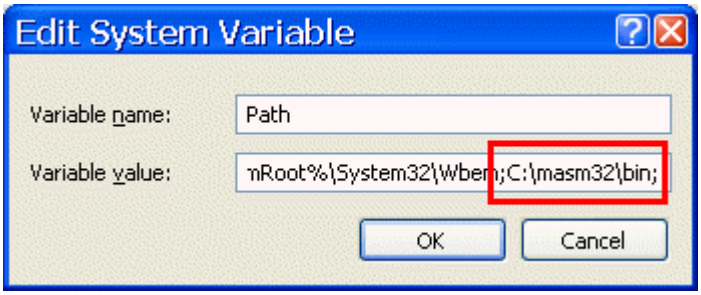
7. Under **System variables** box, select variable **Path** and click on **Edit**.

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



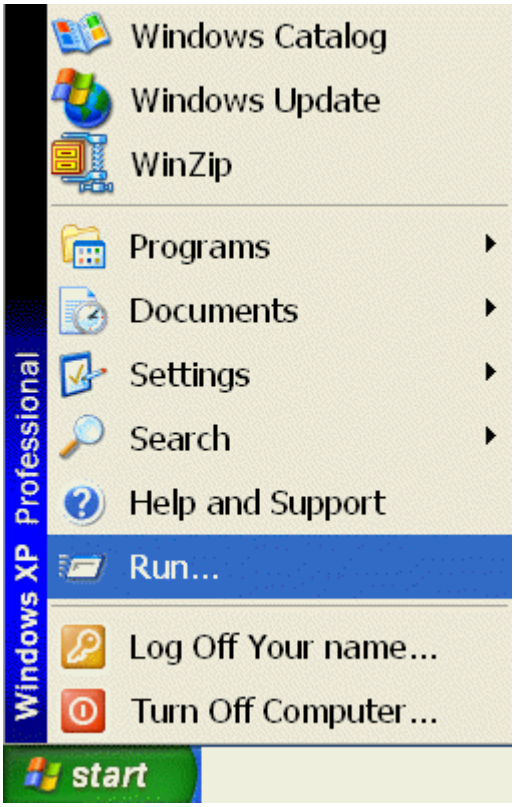
8. In **Variable value** edit box add the new paths separated by semicolons (**;**) **C:\masm32\bin**, click OK, click OK and click Ok again.

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



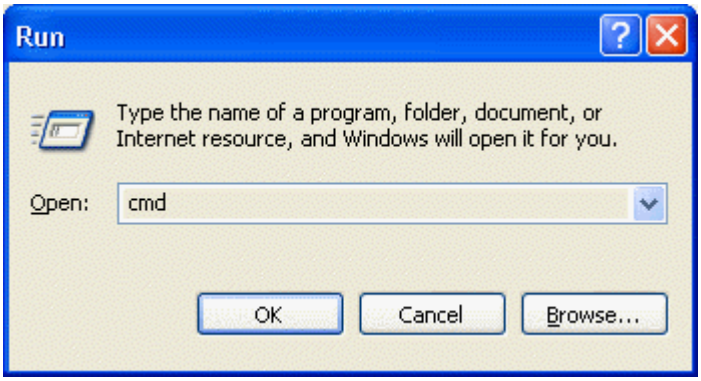
9. Click **Start**, and then click **Run**.

9. คลิก **Start**, และคลิก **Run**



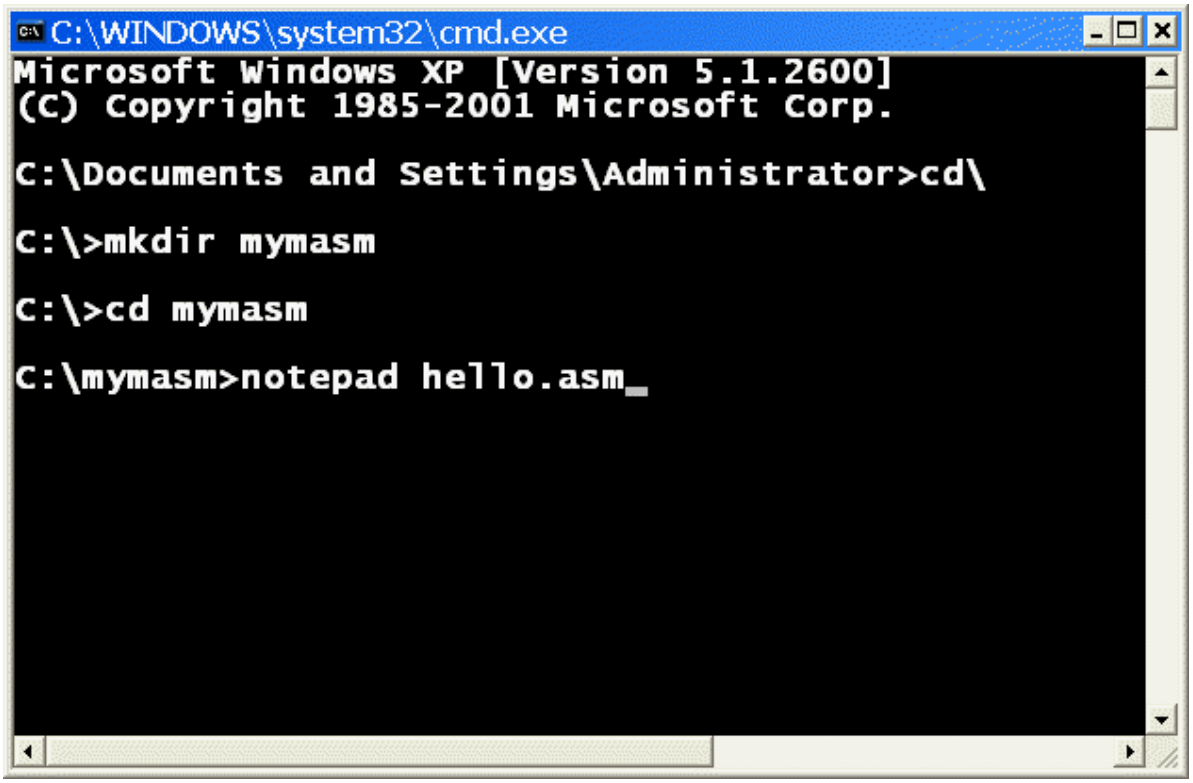
10. In the Open box, type **cmd**, and then click OK.

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



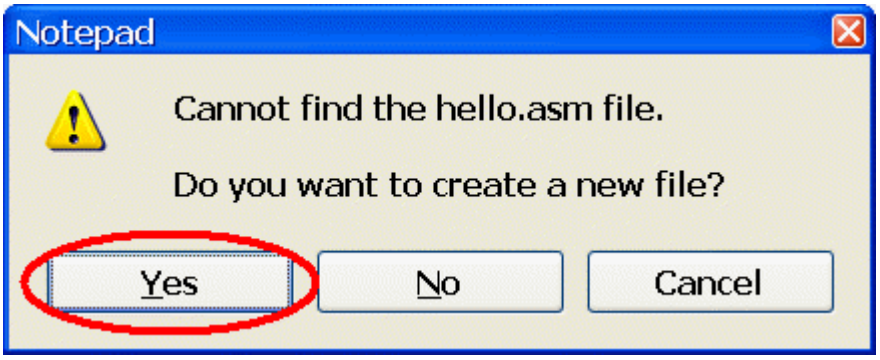
11. Create a folder to hold your Assembly **source code** file, using the **mkdir** command, and then change into the working directory, using the **cd** command. Finally, create your Assembly source code file by typing **notepad hello.asm**.

11. สร้างโฟลเดอร์เพื่อเก็บโค้ดโปรแกรมภาษาแอสแซมบลี, ใช้คำสั่ง **mkdir**, และใช้คำสั่ง **cd** เพื่อเข้าไปในโฟลเดอร์ที่สร้างไว้ ขั้นสุดท้ายสร้างโค้ดภาษาแอสแซมบลีโดยพิมพ์คำสั่งดังนี้ **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

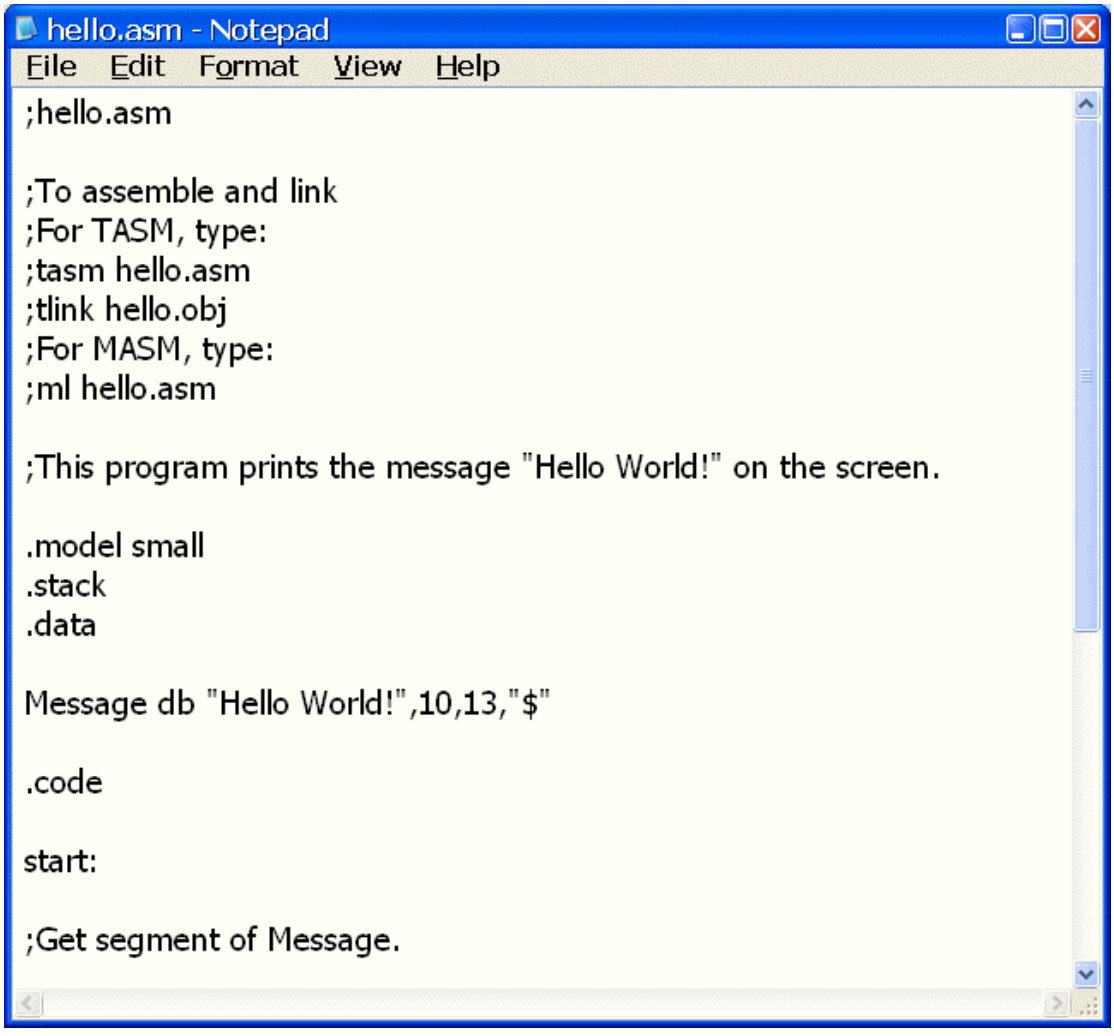


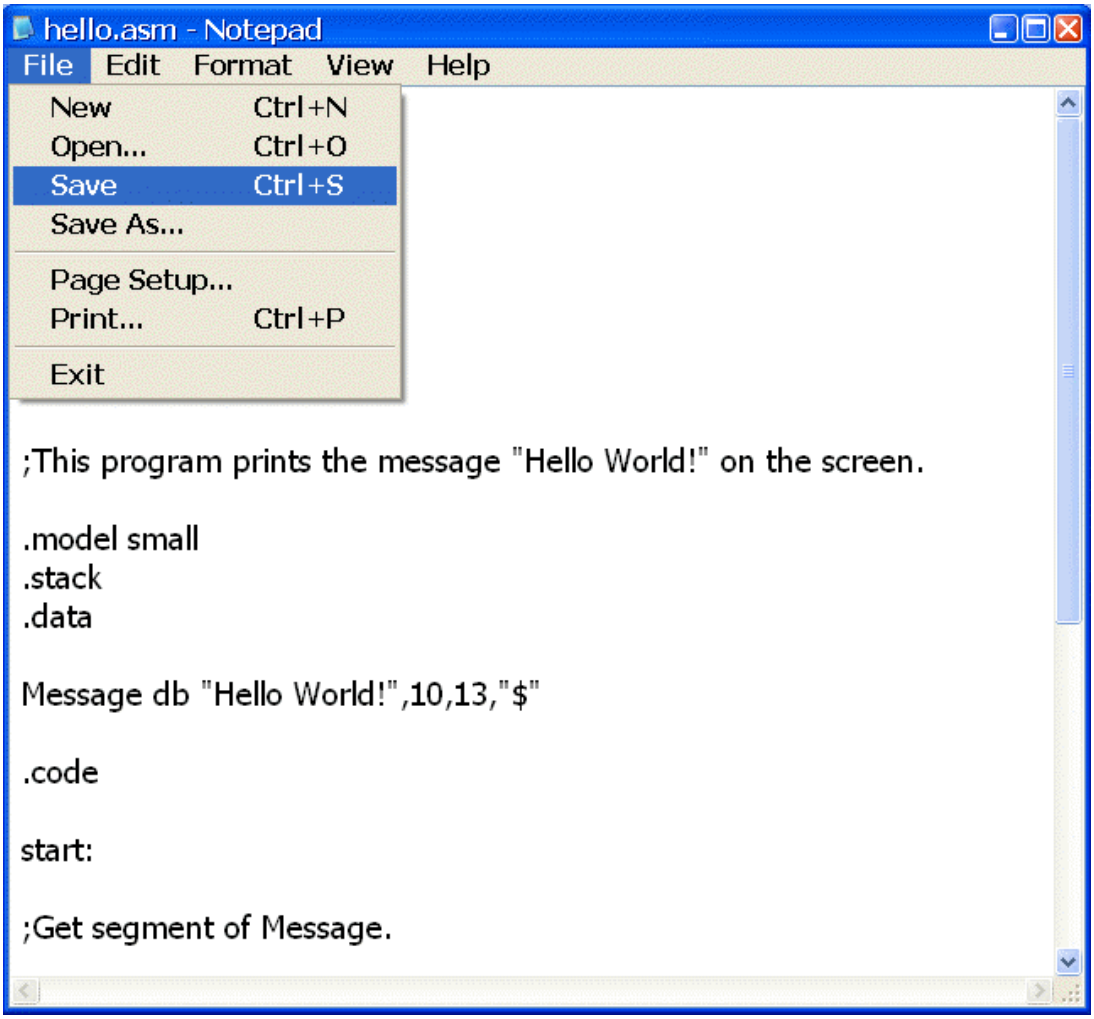
13. In **hello.asm** type the following Assembly source code, and then save it:

13. ในไฟล์ **hello.asm** พิมพ์โค้ดภาษาแอสแซมบลี, 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
```





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

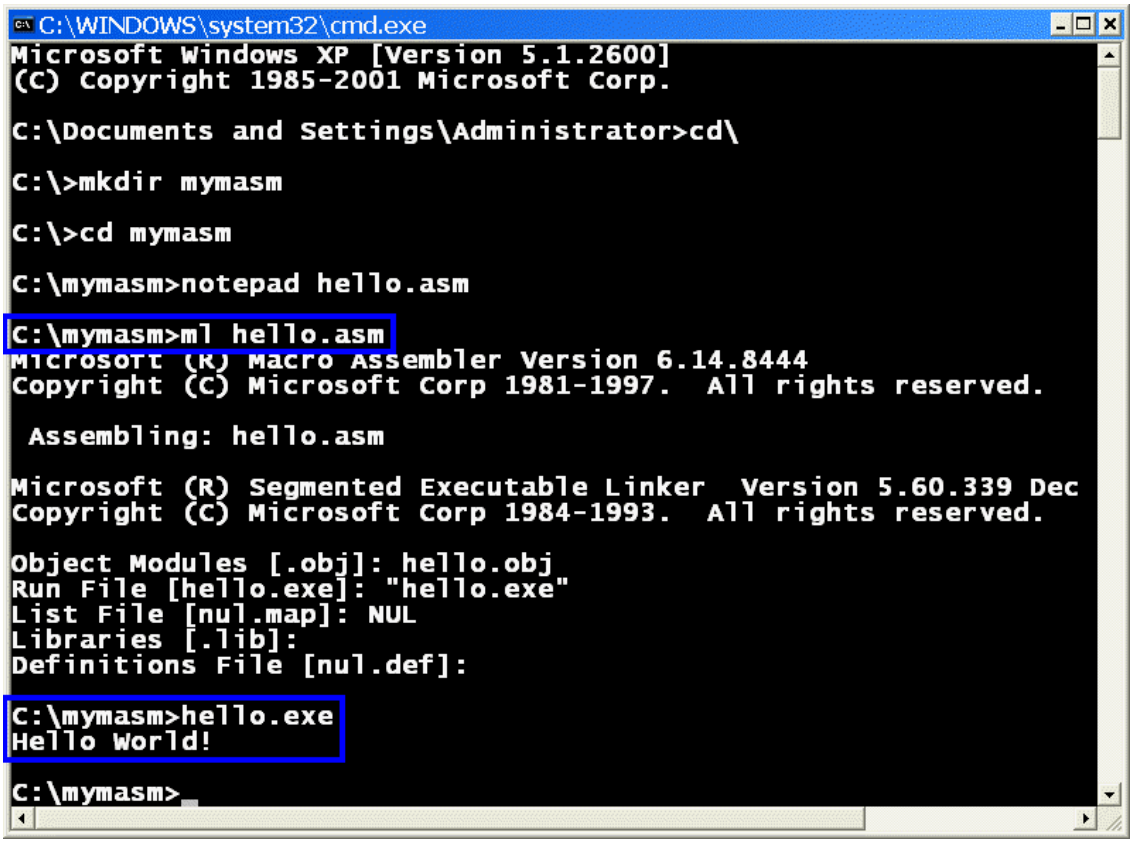
14. การแปลโค้ดและเชื่อม, ใช้คำสั่ง: **ml hello.asm**

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. โปรแกรมจะแสดงผลลัพธ์ข้อความ **Hello World!** ที่บรรทัดถัดไป

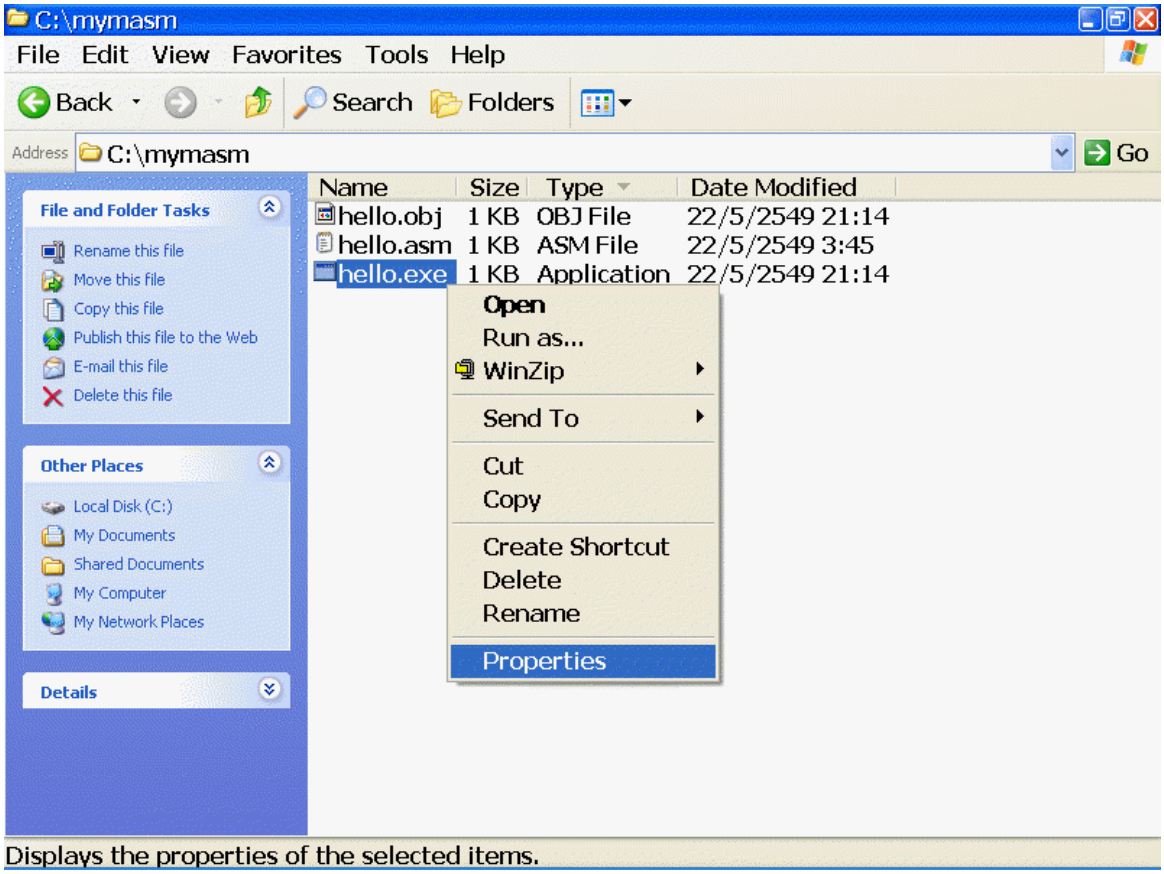


Running from icon

รันโปรแกรมโดยไอคอน

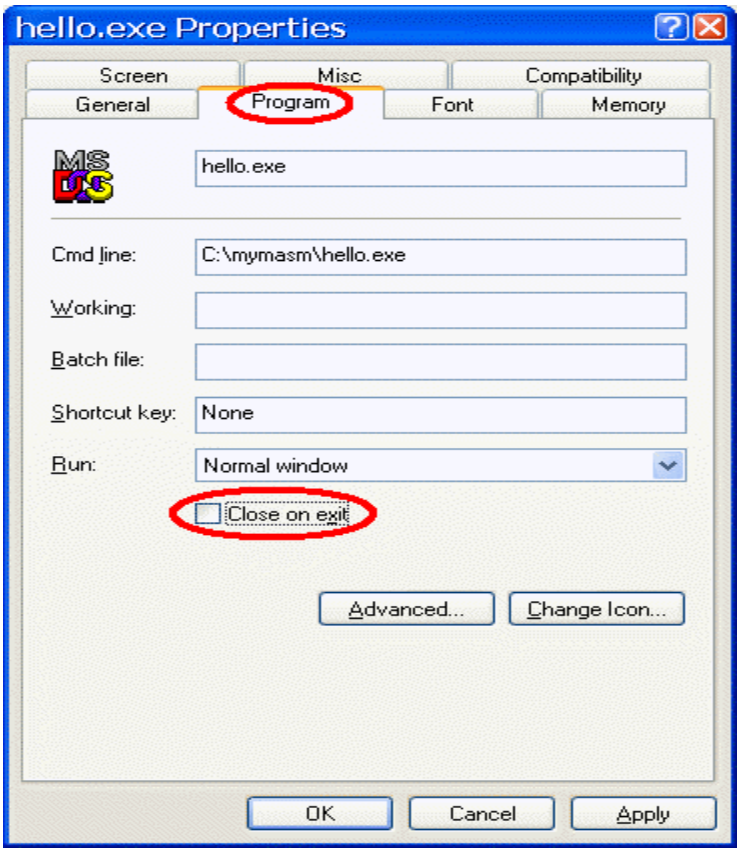
17. In Windows Explorer in the folder you saved the Pascal source code right click on the executable file **hello.exe**, and choose **Properties**.

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



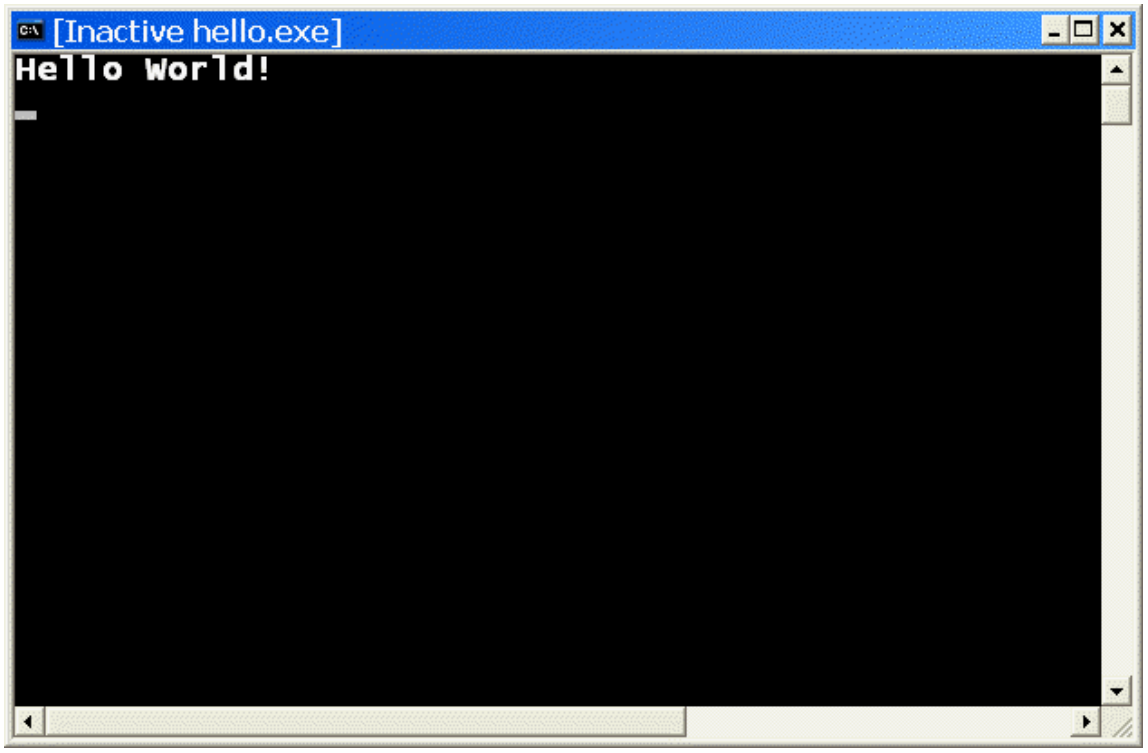
18. Click the **Program** tab, and uncheck the box marked **Close on exit** and then click "OK" to close the box.

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



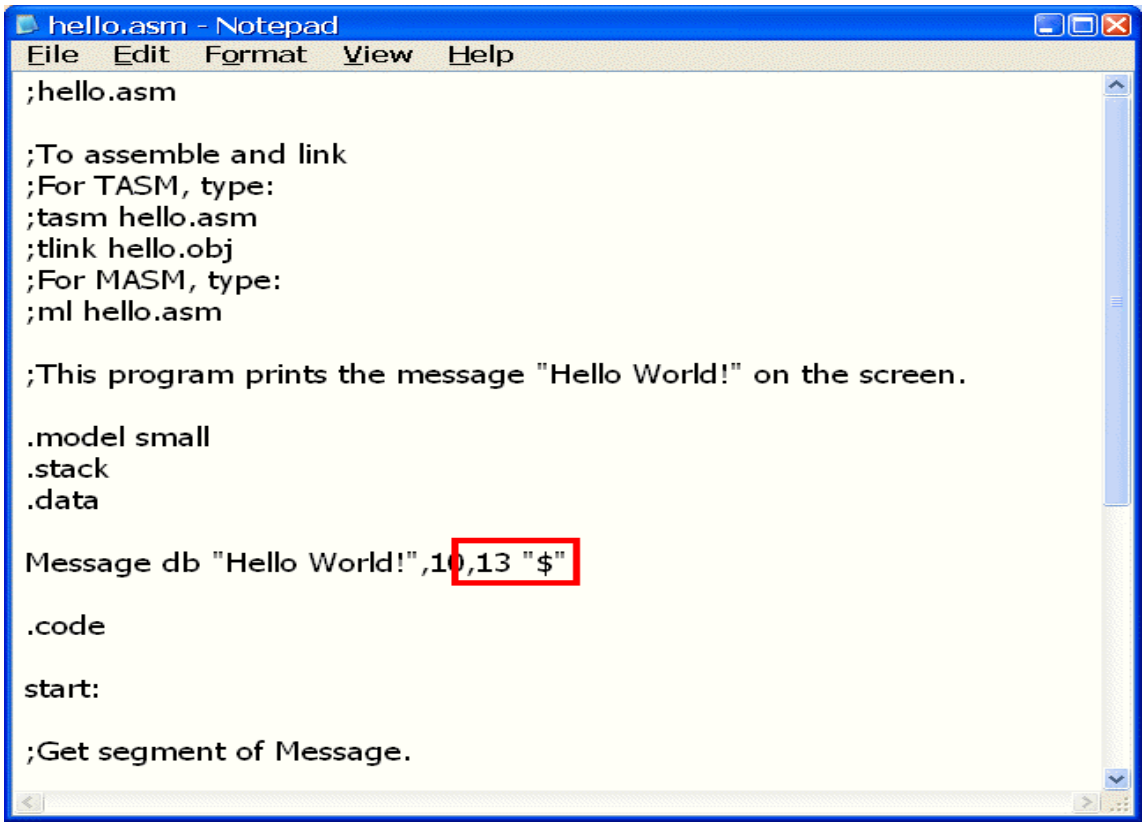
19. This will stop the DOS window from closing automatically when the program runs by **DOS program icon.**

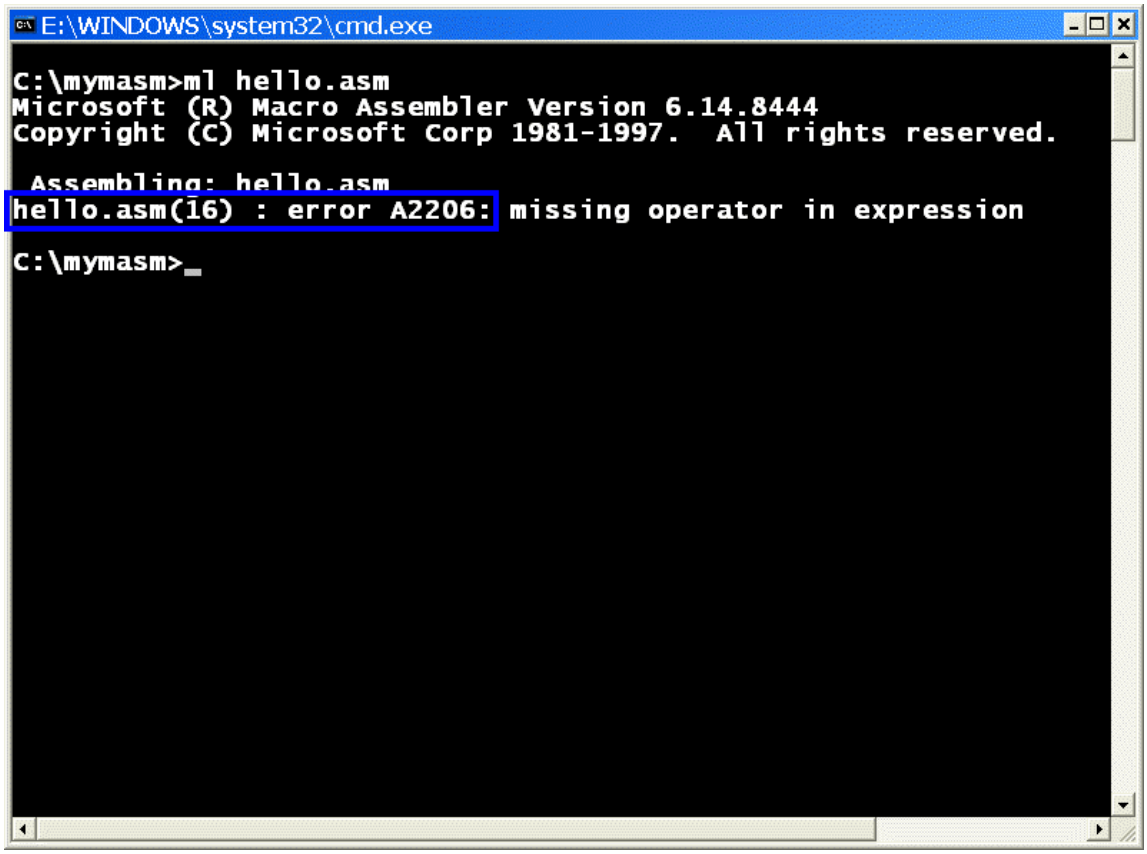
19. โปรแกรมจะรันค้างไว้โดยไม่ปิดเองถ้ารันโดยไอคอนของดอส



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>

15.2 การดาวน์โหลดและติดตั้งโปรแกรม EditPlus

EditPlus คือโปรแกรมพิมพ์ข้อความรวมทั้งโค้ดภาษาโปรแกรมและโค้ดภาษาเวบใช้ในวินโดว์ มีคุณสมบัติมากมายเหมาะสำหรับเขียนเวบภาษา HTML และเขียนโปรแกรมภาษาต่างๆ

วิธีการดาวน์โหลดและติดตั้ง โปรแกรม EditPlus

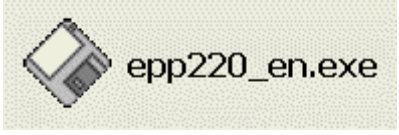
ขั้นตอนที่ 1. เข้าเว็บไซต์ไปที่ http://dusithost.dusit.ac.th/~juthawut_cha/home.htm

เลือก การฝึกทักษะการเขียนโปรแกรมเบื้องต้น (Programming Skills)

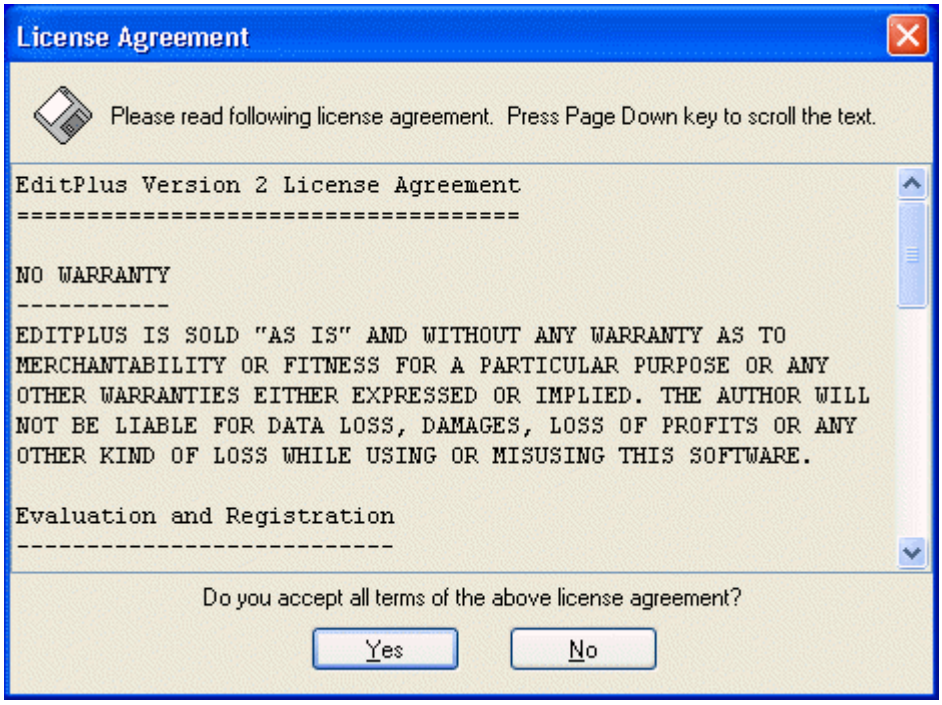
1. ไปที่ ดาว์นโหลด โปรแกรมสำหรับฝึกทักษะการเรียนรู้การเขียนโปรแกรมภาษาแอสแซมบลี คลิก **Download** ดาวน์โหลดไฟล์ **epp220_en.exe** ขนาด 939 KB

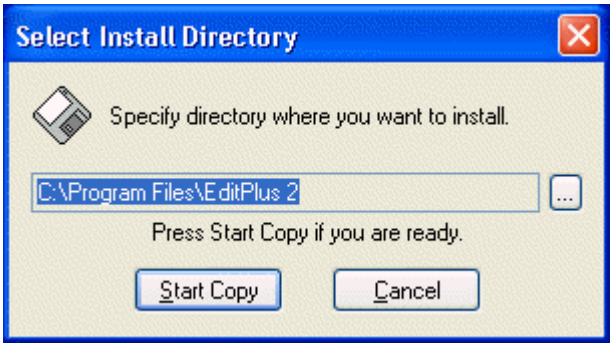
2. Double-click on the saved file icon **epp220_en.exe** to start the installation process.

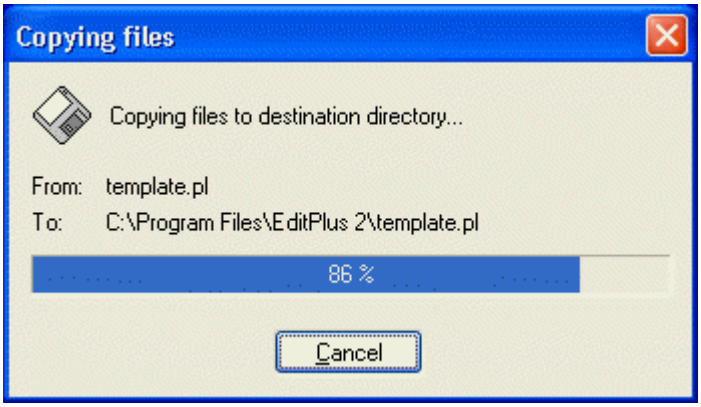
2. ดับเบิ้ลคลิกไอคอนของไฟล์ **epp220_en.exe** เพื่อเริ่มกระบวนการติดตั้งโปรแกรม

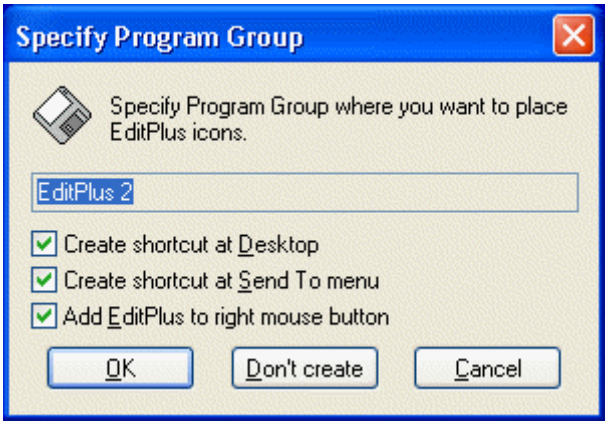



WinZip Self-Extractor - epp220_en.exe
EditPlus Text Editor v2.20 Setup
Press the "Setup" button to proceed.
http://www.editplus.com/
Setup
Cancel
About


License Agreement
Please read following license agreement. Press Page Down key to scroll the text.
EditPlus Version 2 License Agreement
NO WARRANTY
EDITPLUS IS SOLD "AS IS" AND WITHOUT ANY WARRANTY AS TO MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR ANY OTHER WARRANTIES EITHER EXPRESSED OR IMPLIED. THE AUTHOR WILL NOT BE LIABLE FOR DATA LOSS, DAMAGES, LOSS OF PROFITS OR ANY OTHER KIND OF LOSS WHILE USING OR MISUSING THIS SOFTWARE.
Evaluation and Registration
Do you accept all terms of the above license agreement?
Yes
No

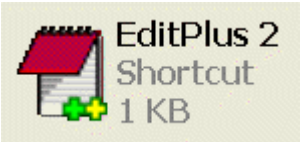

Select Install Directory
Specify directory where you want to install.
C:\Program Files\EditPlus 2
...
Press Start Copy if you are ready.
Start Copy
Cancel

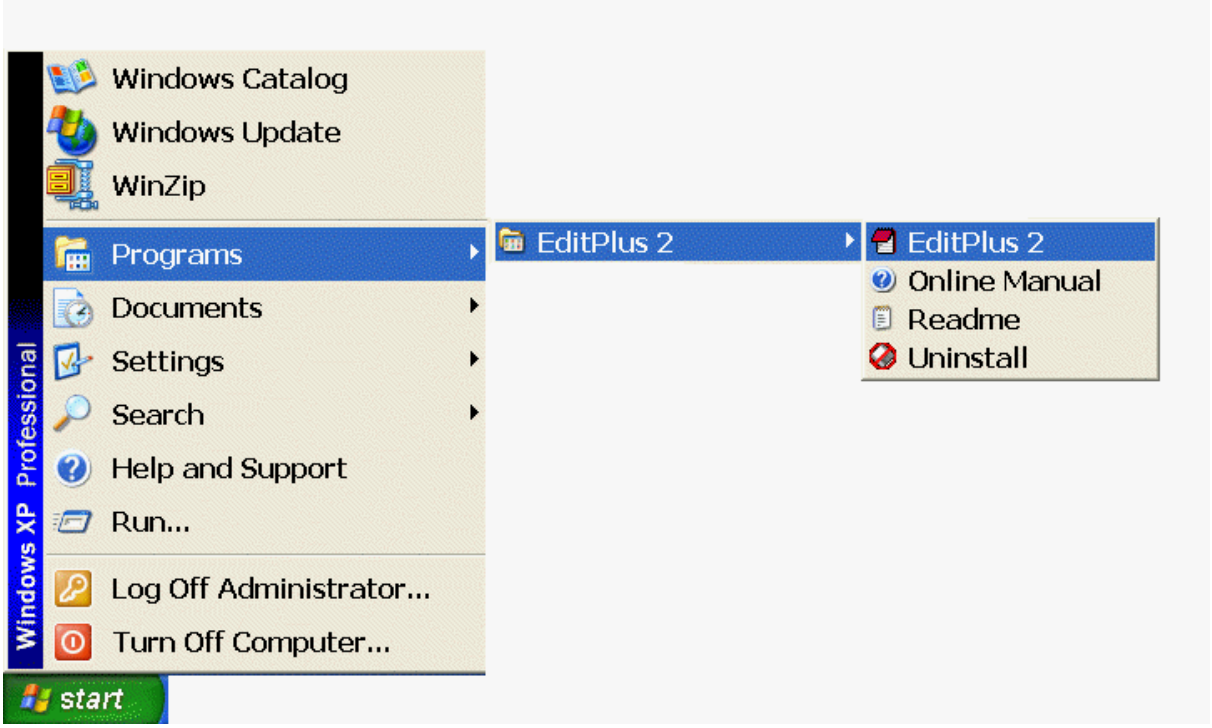




3. Click the program **icon** on your desktop or the shortcut on the **Start menu** to launch the program.

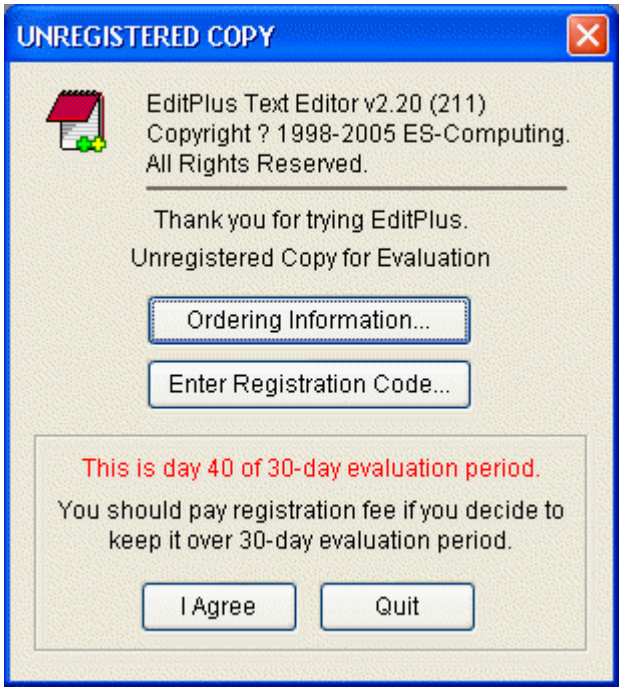
3. คลิกไอคอนโปรแกรมบนหน้าจอหรือคลิกชื่อโปรแกรมที่เมนู **Start** เพื่อรันโปรแกรม





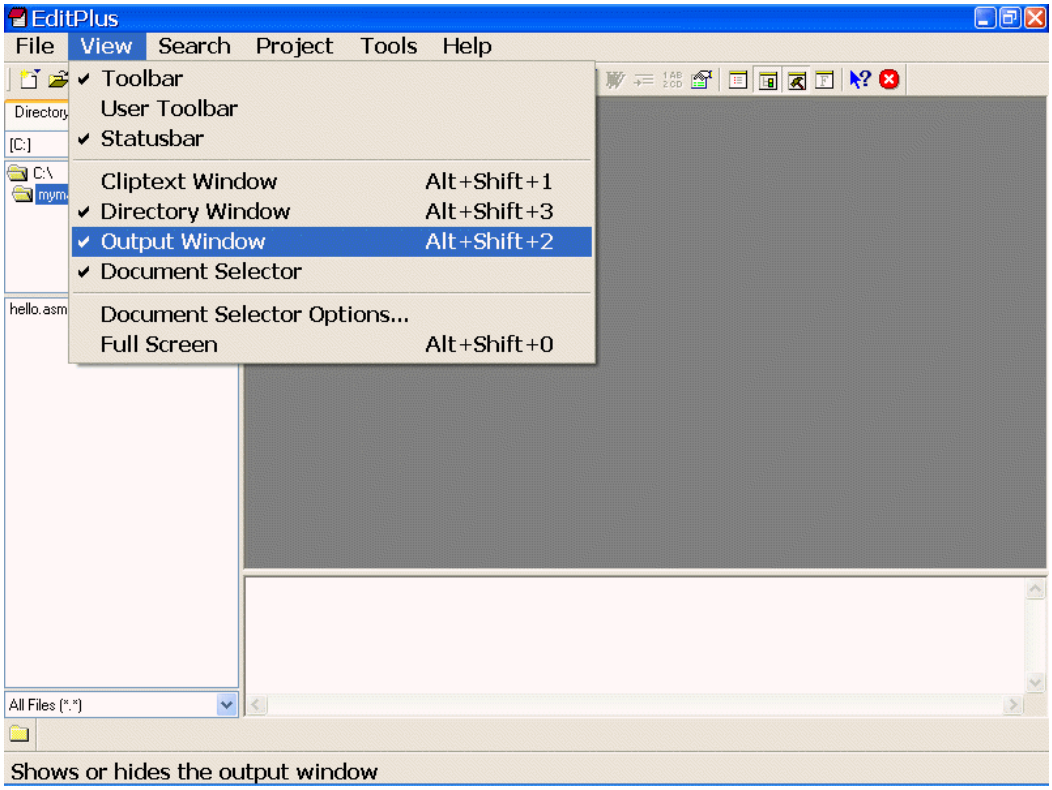
4. It will bring up a nag screen each time it is run until you register the program.

4. โปรแกรมจะแสดงข้อความเตือนว่ายังไม่รีจิสเตอร์ในกรอบสี่เหลี่ยม



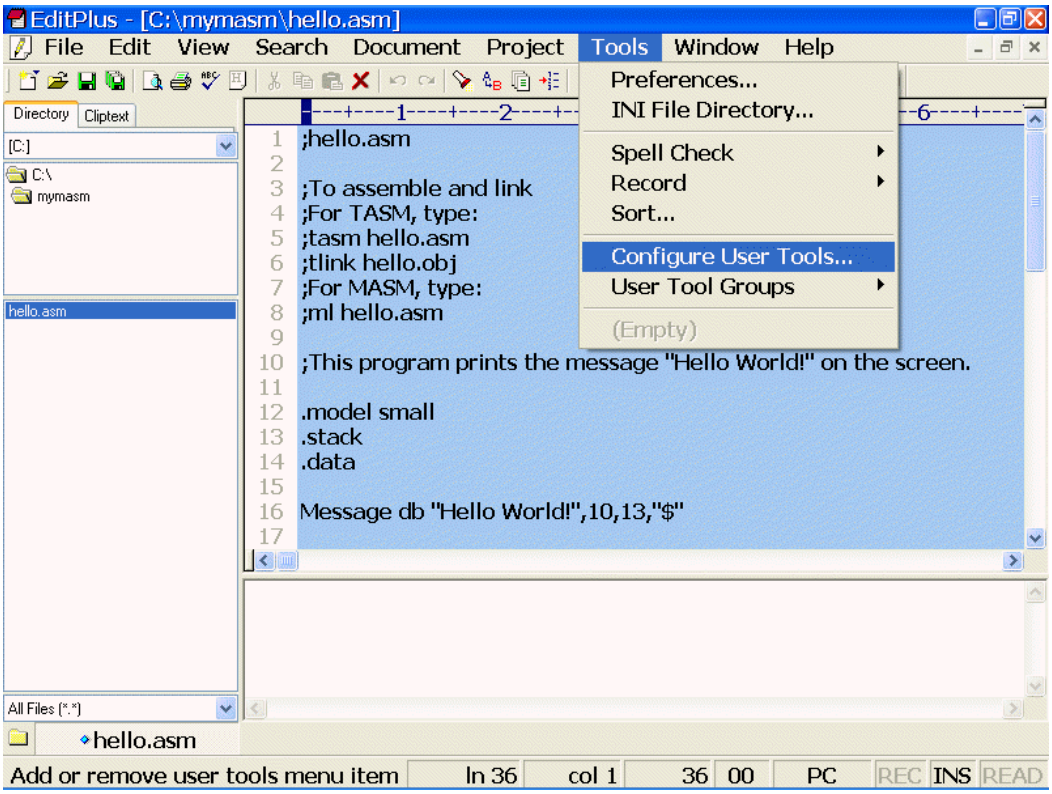
5. From the **View** menu, select **Output Window** to show it.

5. ที่เมนู View, เลือก Output Window เพื่อแสดงให้เห็น



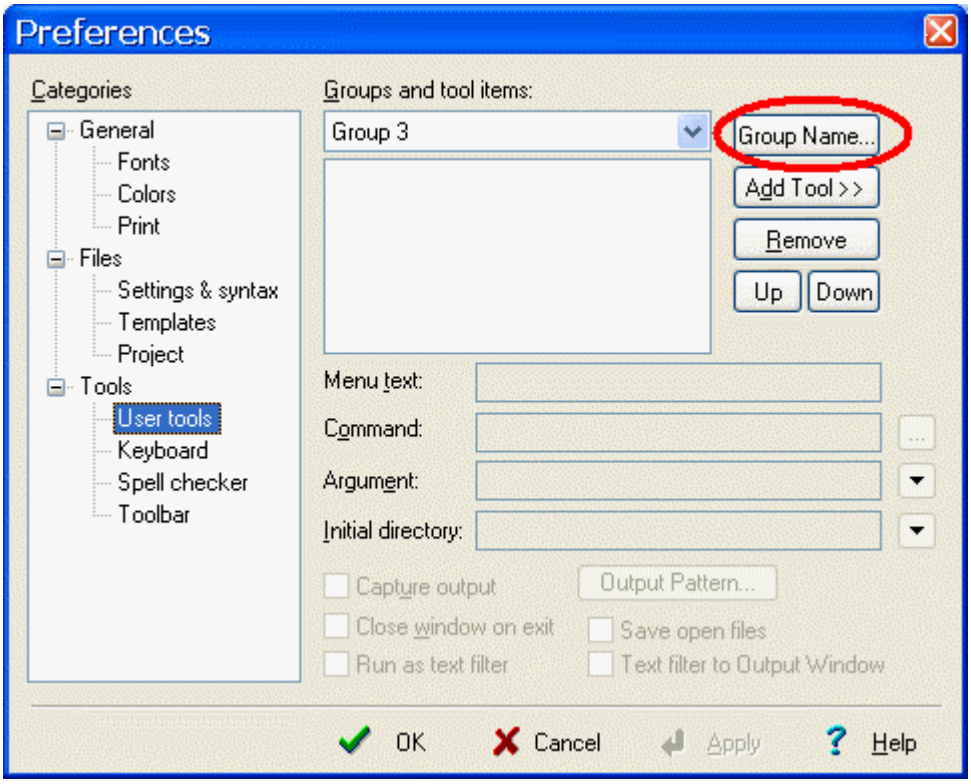
6. From the **Tools** menu, select **Configure User Tools**.

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



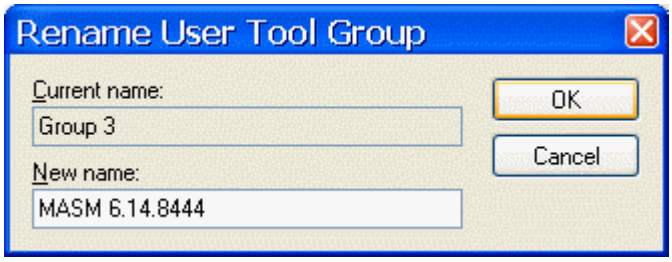
7. In **Preferences** dialog box On **User tools** page click on the **Group Name** button to change the default group name Group 3.

7. ในกรอบ **Preferences** ที่หน้า **User tools** คลิกปุ่ม **Group Name** เพื่อเปลี่ยนชื่อ



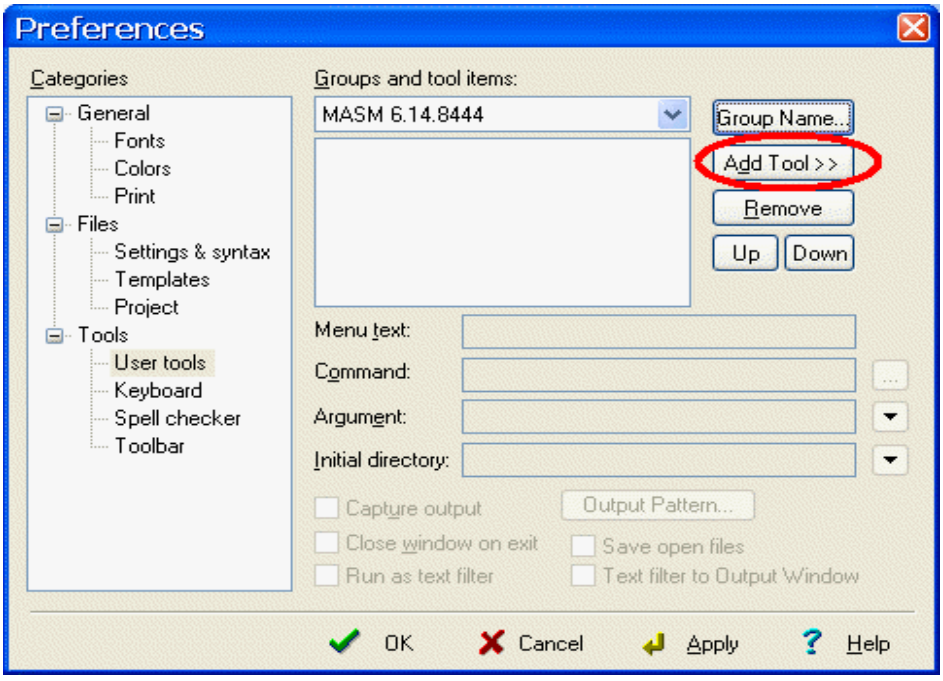
8. In the New Name text box, type **MASM 6.14.8444**

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



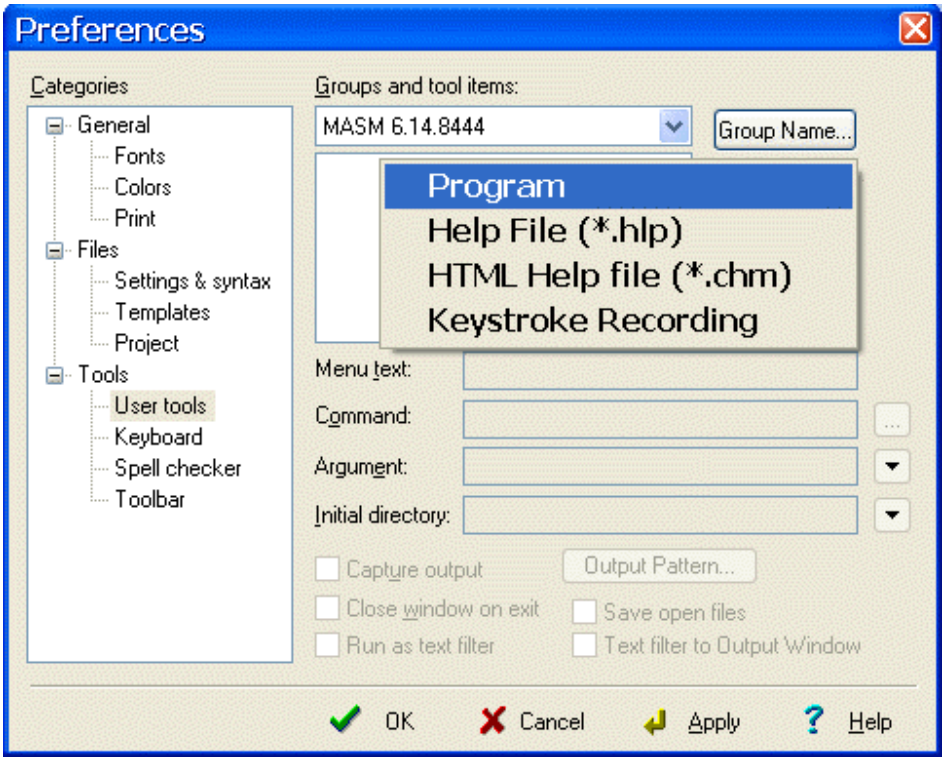
9. Select the **Add Tool** button.

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**

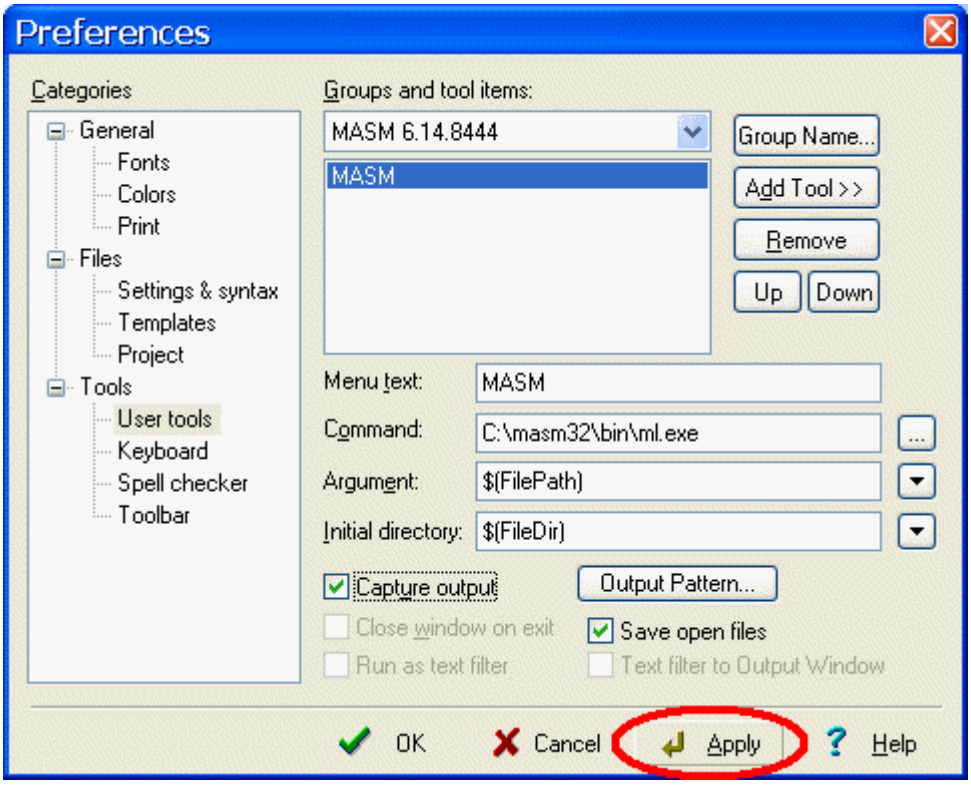
Command: **C:\masm32\bin\ml.exe**

Argument: **$(FilePath)**

Initial directory: **$(FileDir)**

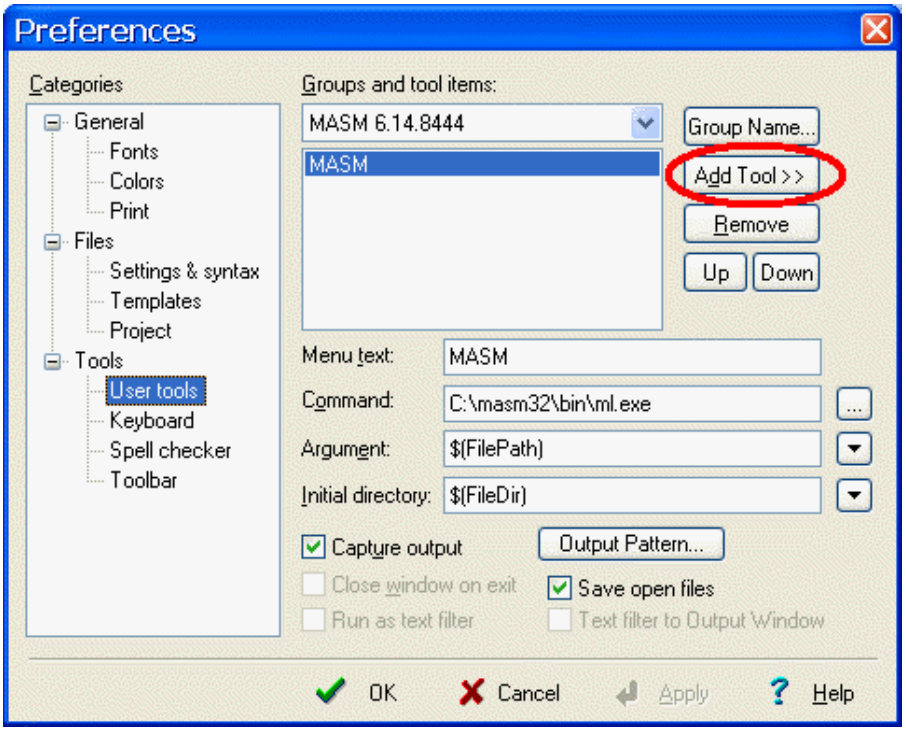
Capture output: **ON**

หลังจากนั้นคลิกปุ่ม **Apply**



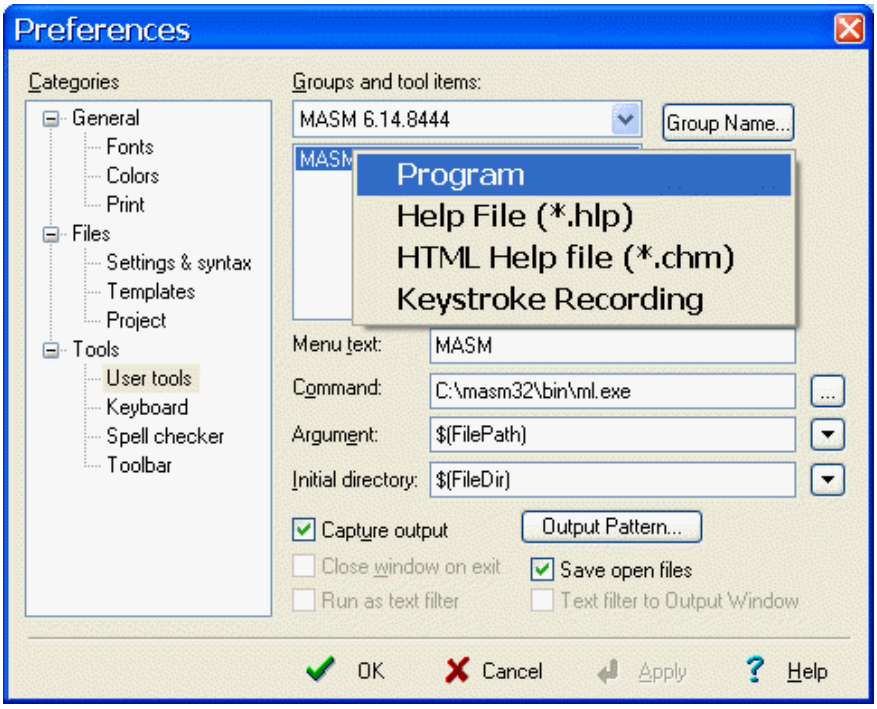
12. Select the **Add Tool** button.

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**

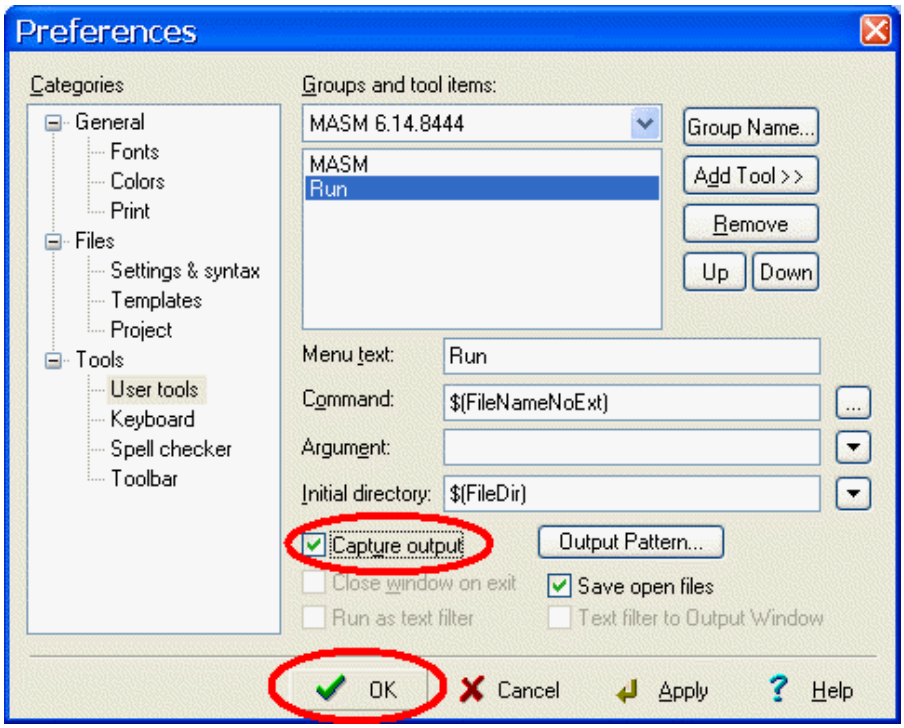
Command: **$(FileNameNoExt)**

Argument:

Initial directory: **$(FileDir)**

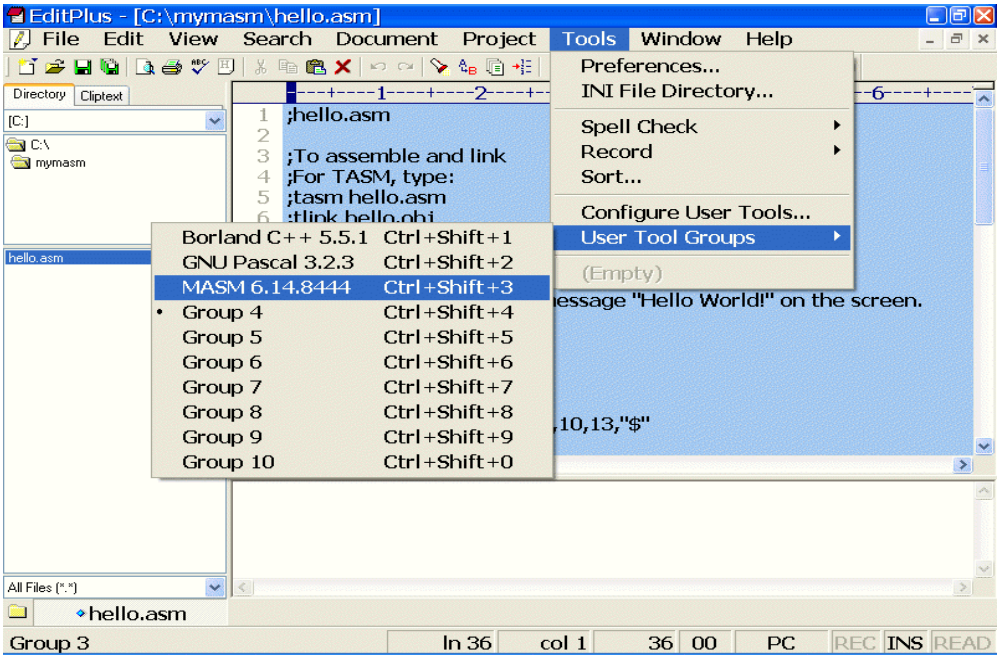
Capture output: **ON**

หลังจากนั้นคลิกปุ่ม **OK**



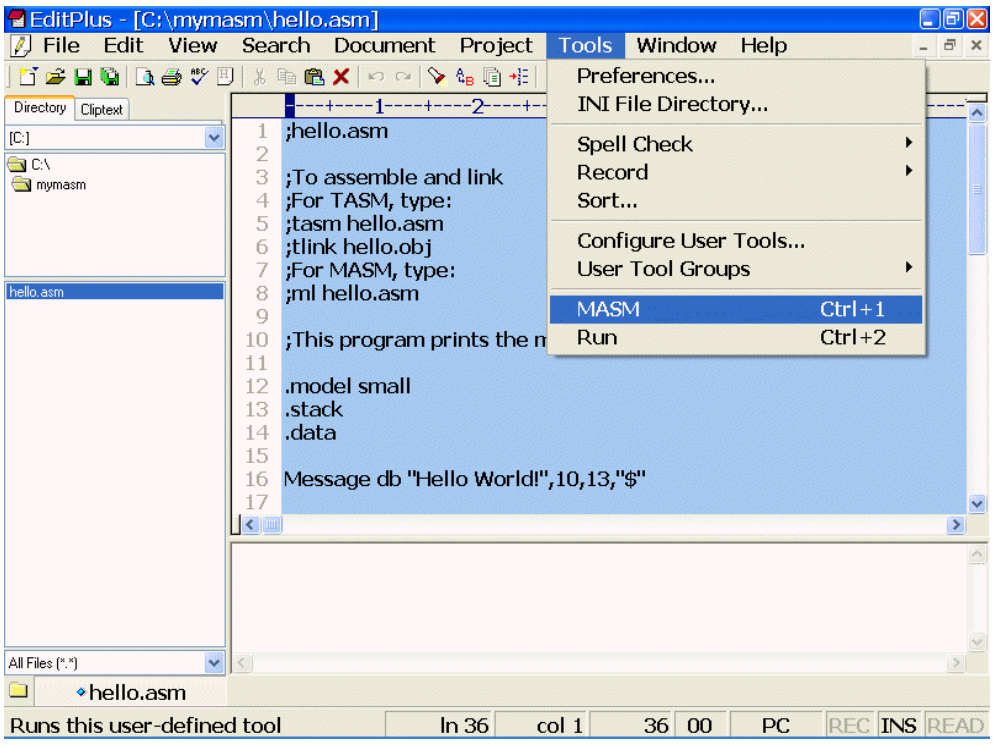
15. From the **Tools** menu, select **User Tool Groups**, and then **MASM 6.14.8444**

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



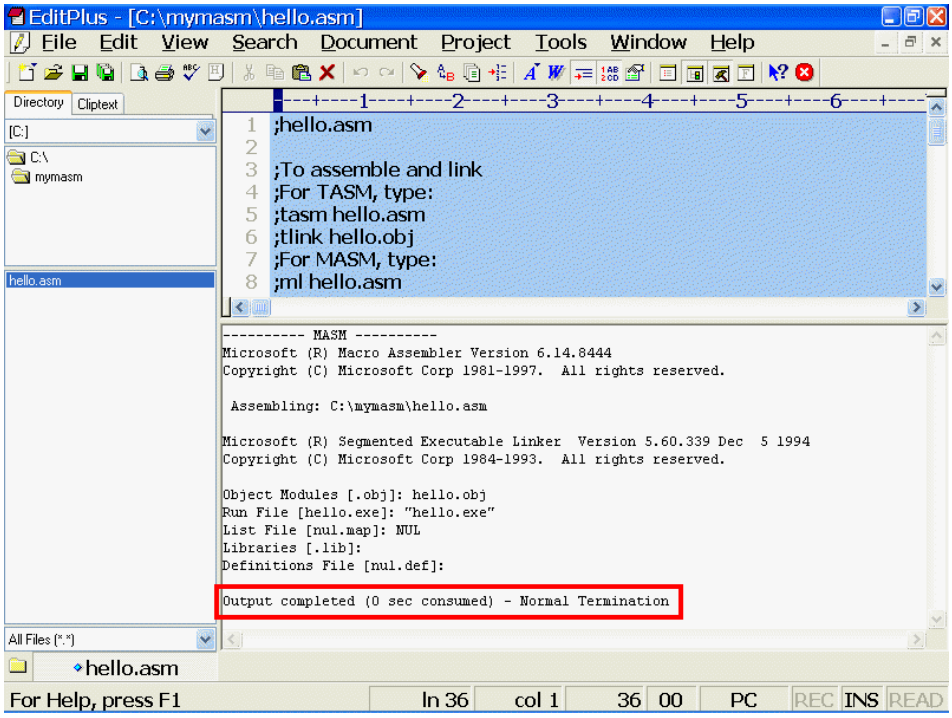
16. To compile, open the **source file**, select **MASM** from the **Tools** menu.

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



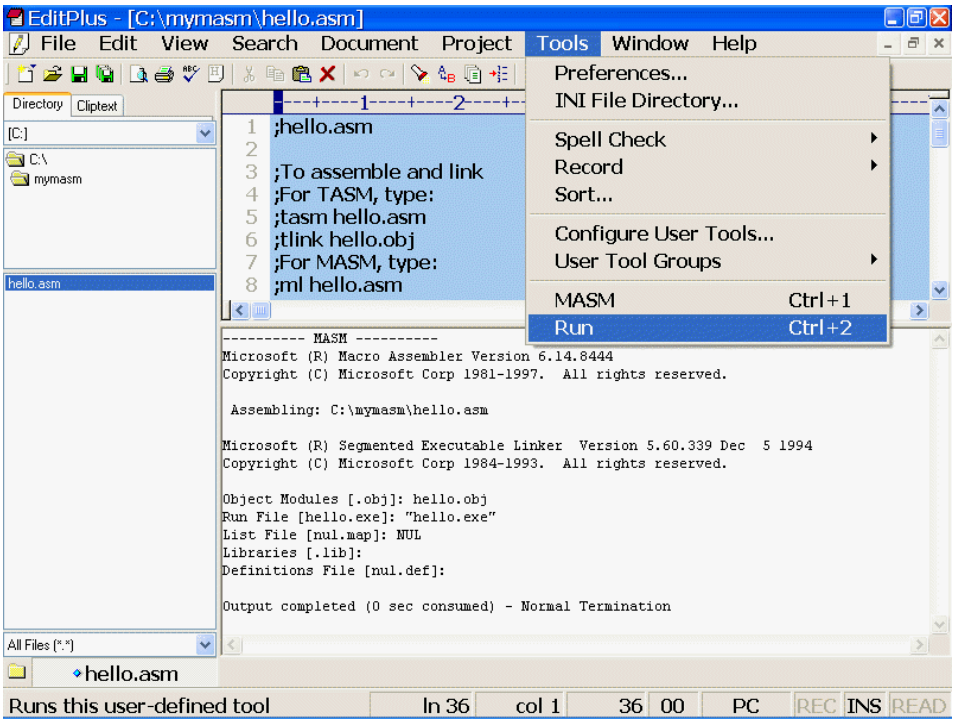
17. The result will be shown in the Output Window at the bottom.

17. โปรแกรมจะแสดงผลลัพธ์เป็นข้อความที่หน้าต่าง Output ด้านล่าง



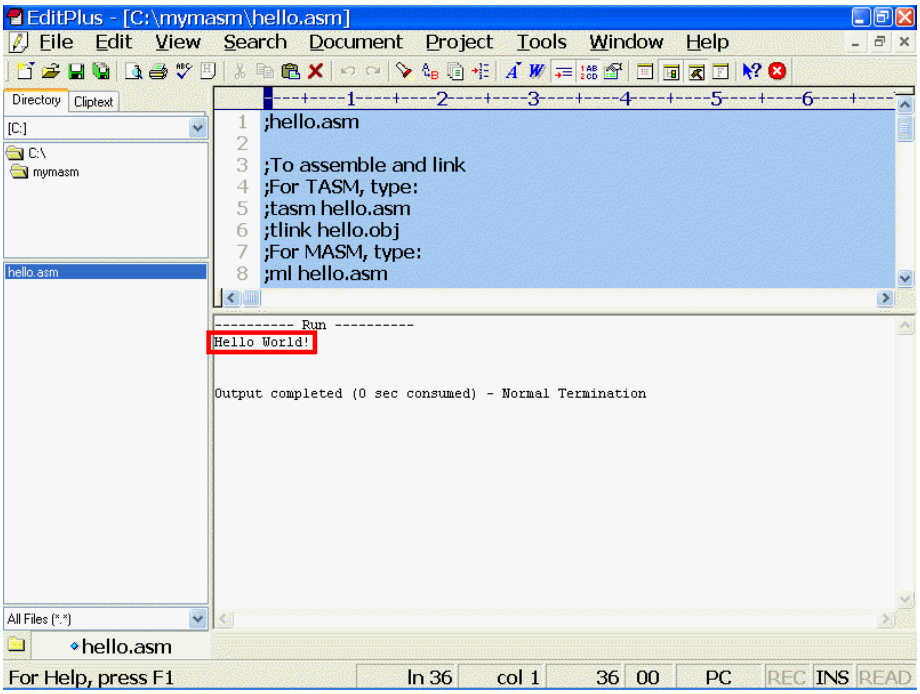
18. To run, select **Run** from the **Tools** menu.

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



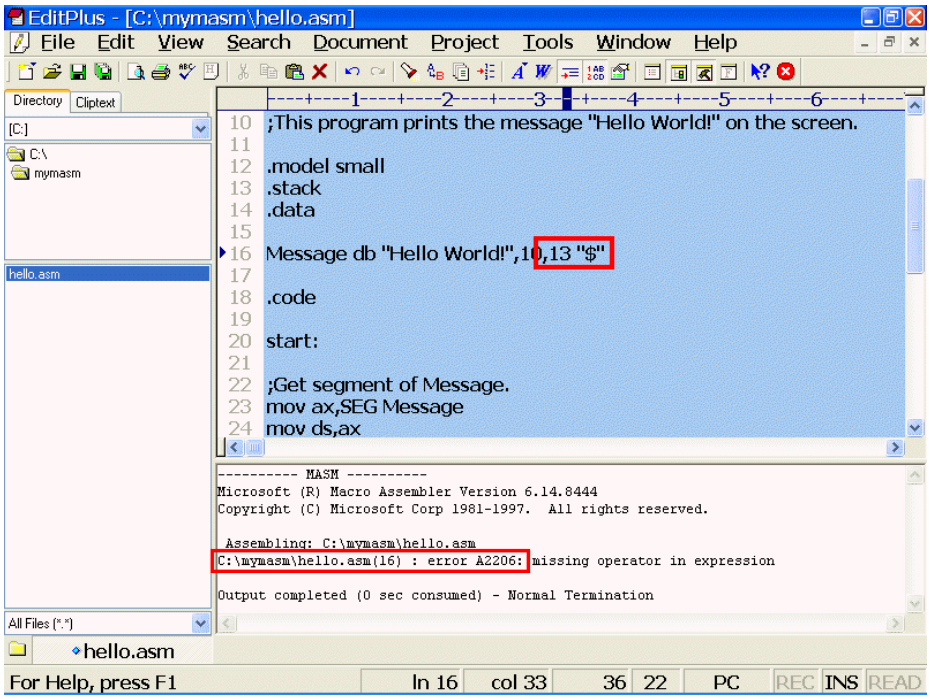
19. The result will be shown in the Output Window at the bottom.

19. โปรแกรมจะแสดงผลลัพธ์ในหน้าต่าง Output ด้านล่าง



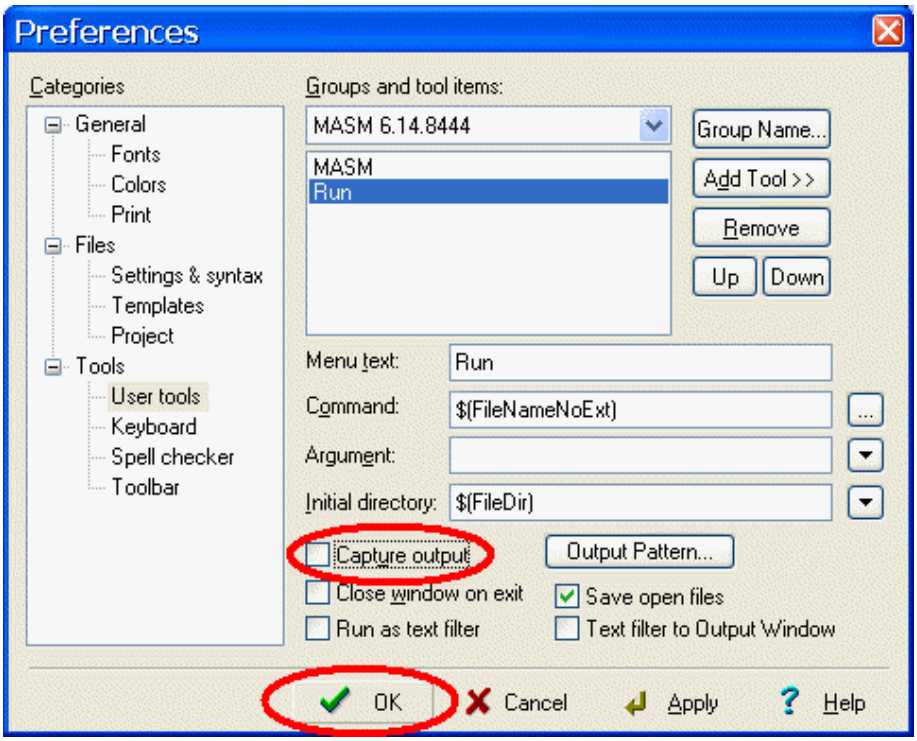
20. If you forget the comma, the compiler will give you an error message when you attempt to compile the program.

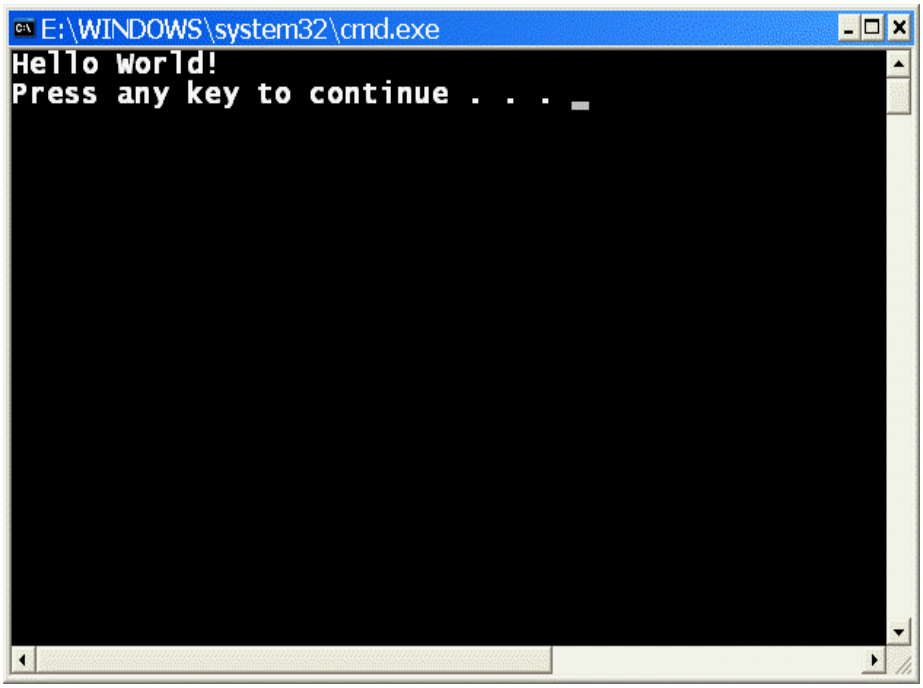
20. ถ้าท่านลืมเครื่องหมายจุลภาค (,), ตัวแปลภาษาจะแสดงข้อความแจ้งความผิดพลาดเมื่อท่านแปลโค้ด



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** โปรแกรมจะแสดงผลลัพธ์ในหน้าต่างแบบ command-prompt





4.int mouse ใช้ int 33h

สรุป

Microsoft Macro Assembler เป็นโปรแกรมสำหรับแปลภาษาแอสแซมบลี (assembler) ใช้กับเครื่องคอมพิวเตอร์ที่ใช้ซีพียูตระกูล x86 สำหรับใช้กับระบบปฏิบัติการดอส (MS-DOS) และเป็นโปรแกรมสำหรับแปลภาษาแอสแซมบลีที่นิยมใช้มากที่สุด ดังนั้นนักพัฒนาภาษาโปรแกรมภาษาแอสแซมบลีเบื้องต้นจำเป็นที่จะต้องเรียนรู้และฝึกทักษะการเขียนภาษาโปรแกรมและตรวจสอบความถูกต้องอย่างต่อเนื่อง เพื่อทำความเข้าใจและทดสอบโปรแกรมที่ต้นเขียนขึ้นมาซึ่งจะทำให้เข้าใจรูปแบบและวิธีการทำงานอย่างเป็นขั้นตอน รวมถึงช่วยให้สามารถพัฒนาโปรแกรมภาษาชั้นสูงที่เกี่ยวข้องต่อไป

คำถามทบทวน

1. จงยกตัวอย่างโปรแกรมที่ใช้ในการเขียนโปรแกรมภาษาแอสแซมบลีในปัจจุบันนอกเหนือจากโปรแกรม MASM32 หรือ EditPlus ว่ามีวิธีการดาว์นโหลดและติดตั้งเพื่อใช้งานได้อย่างไร
2. จงเขียนโปรแกรมภาษาแอสแซมบลีอ่านค่าจากคีย์บอร์ด (Keyboard) ตัวอักษรภาษาอังกฤษพิมพ์ใหญ่และให้แสดงผลเป็นจากแป้นพิมพ์ออกทางหน้าจอเป็นตัวอักษรภาษาอังกฤษพิมพ์เล็ก
3. จงเขียนโปรแกรมภาษาแอสแซมบลีที่พิมพ์ String โดยใช้ฟังก์ชันเบอร์ 9 ของขัดจังหวะเบอร์ 21H
4. จงเขียนโปรแกรมภาษาแอสแซมบลีอ่านค่าจากคีย์บอร์ด (Keyboard) โดยใช้ฟังก์ชั่นเบอร์ 1 ของขัดจังหวะเบอร์ 21H แล้วเปลี่ยนเป็นเลขฐาน 10 และพิมพ์ค่าออกทางจอภาพ (Monitor) โดยใช้ฟังก์ชันเบอร์ 2 ของขัดจังหวะเบอร์ 21H
5. จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 2 เป็นเลขฐาน 8 และแสดงผลออกทางทางจอภาพ (Monitor)
6. จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 2 เป็นเลขฐาน 10 และแสดงผลออกทางทางจอภาพ (Monitor)

7 จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 2 เป็นเลขฐาน 16 และแสดงผลออกทางทางจอภาพ (Monitor)

8. จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 8 เป็นเลขฐาน 2 และแสดงผลออกทางทางจอภาพ (Monitor)
9. จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 8 เป็นเลขฐาน 10 และแสดงผลออกทางทางจอภาพ (Monitor)

10 จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 8 เป็นเลขฐาน 16 และแสดงผลออกทางทางจอภาพ (Monitor)

11. จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 10 เป็นเลขฐาน 2 และแสดงผลออกทางทางจอภาพ (Monitor)

12. จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 10 เป็นเลขฐาน 8 และแสดงผลออกทางทางจอภาพ (Monitor)

13 จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 10 เป็นเลขฐาน 16 และแสดงผลออกทางทางจอภาพ (Monitor

14. จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 16 เป็นเลขฐาน 2 และแสดงผลออกทางทางจอภาพ (Monitor)

15. จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 16 เป็นเลขฐาน 8 และแสดงผลออกทางทางจอภาพ (Monitor)

16 จงเขียนโปรแกรมภาษาแอสแซมบลีที่เปลี่ยนค่าจากเลขฐาน 16 เป็นเลขฐาน 10 และแสดงผลออกทางทางจอภาพ (Monitor

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 27 กุมภาพันธ์ 2557, จาก:http://rirs3.royin.go.th/coinages/

ภาษาระดับต่ำ (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 27 กุมภาพันธ์ 2557,จาก:http://th.wikipedia.org/wiki/

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386(PC).* กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย–ญี่ปุ่น), 2537.

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 Publi shing Co Inc, 1982.

Thorne M. *Computer Organization and Assembly Language Programming 2nd Edition.*Benja min Cumming Publishing Company : USA, 1991.

Williams Dave. *The Programming Technical Reference MS–DOS, IBM PC & Compatibles.* Singapore : John Wiley & Sons (SEA), 1990.

ฉวีวรรณ โสภาจารีย์. *โครงสร้างคอมพิวเตอร์และการเขียนโปรแกรมภาษาแอสแซมบลี.* กรุงเท พฯ:ภาควิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์ สถาบัน เทคโนโลยีพระจอมเกล้าพระนครเหนือ, 2548.

ชูชัย ธนสารตั้งเจริญ, กำธร พานิชปฐมพงษ์. *ภาษาแอสแซมบลี 80286/80386 (PC).*กรุงเทพฯ : สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ, 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาแอสแซมบลี.* กรุงเทพฯ : สำนักพิมพ์สงเสริ มเทคโนโลยี (ไทย– ญี่ปุ่น), 2537.

ศิริวรรณ ฉันทาดิสัย. *หลักการเขียนโปรแกรมภาษาแอสแซมบลี 8088.* กรุงเทพฯ : สำนักพิมพ์ ประการพรึก 2534.