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
newlinedb       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.