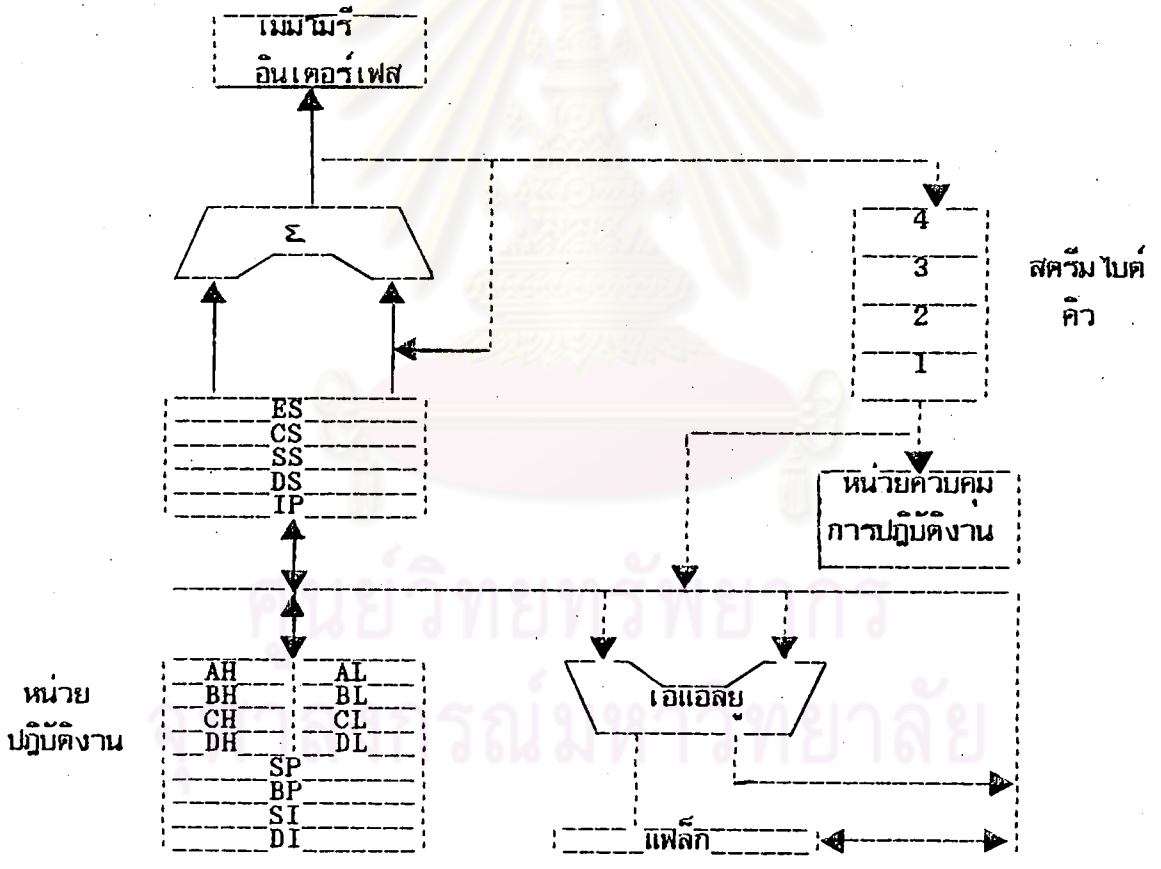
สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

บทที่ 3

ไมโครโปรเซสเซอร์เบอร์ 8088 และชุดคำสั่ง

8088 เป็นไมโครโปรเซสเซอร์ขนาด 16 บิต มีสายแอดเดรส 20 เส้น สามารถมีหน่วยความจำได้ถึงขนาด 1 เมกกะไบต์ โครงสร้างของ 8088 มีลักษณะพิเศษกว่าไมโครโปรเซสเซอร์อื่น ๆ คือมีส่วนที่เฟตคำสั่งจากหน่วยความจำ กับส่วนปฏิบัติงานทำงานแยกเป็นอิสระต่อกัน โครงสร้าง ของ 8088เป็นดังนี้



รูปที่ 3.1 โครงสร้างของ 8088

การทำงานของ 8088 จะเป็นดังนี้คือ ส่วนเฟต (Fetch) คำสั่งจะทำหน้าที่เฟตคำสั่งจากหน่วยความจำเข้ามาในซีพียูโดยผ่านทาง เมมโมรีอินเตอร์เฟส (Memory Interface)

013574

คำสั่งที่ได้มานั้นจะถูกเก็บไว้ในคิวที่เรียกว่าอินสตรัคชั่นสตรีมไบต์คิว (Instruction Stream Byte Queue) ซึ่งมีขนาด 4 ไบต์ ส่วนที่ควบคุมการปฏิบัติงาน (Execution Unit Control System) จะดึงคำสั่งจากคิวนี้ไปยังส่วนปฏิบัติงานเพื่อปฏิบัติตามคำสั่งต่อไป ดังได้กล่าวมาแล้วว่าการทำงานระหว่างส่วนเฟตคำสั่งกับส่วนปฏิบัติงานนั้นเป็นอิสระต่อกัน นั่นคือขณะที่ส่วนปฏิบัติงานกำลังปฏิบัติงาน ส่วนเฟตคำสั่งจะเฟตคำสั่งถัดไปในหน่วยความจำมาเก็บไว้ในคิว หลังจากที่ส่วนปฏิบัติงานปฏิบัติงานในคำสั่งปัจจุบันเรียบร้อยแล้ว ก็สามารถดึงคำสั่งถัดไปจากสตรีมไบต์คิวนี้มาปฏิบัติงานได้ทันที ทำให้ 8088 ปฏิบัติงานได้อย่างมีประสิทธิภาพมากกว่าไมโครโปรเซสเซอร์ทั่ว ๆ ไป

8088 รีจิสเตอร์

รีจิสเตอร์ของ 8088 เป็นรีจิสเตอร์ขนาด 16 บิต มีทั้งหมด 14 ตัว แบ่งได้เป็น 4 กลุ่มดังนี้

1. รีจิสเตอร์สำหรับใช้งานทั่วไป

เป็นรีจิสเตอร์ที่สามารถใช้งานได้ทั้ง 8 บิตหรือ 16 บิตมีด้วยกัน 4 ตัวคือ

AX ถ้าใช้ในลักษณะ 8 บิตจะใช้ชื่อ AH กับ AL

BX ถ้าใช้ในลักษณะ 8 บิตจะใช้ชื่อ BH กับ BL

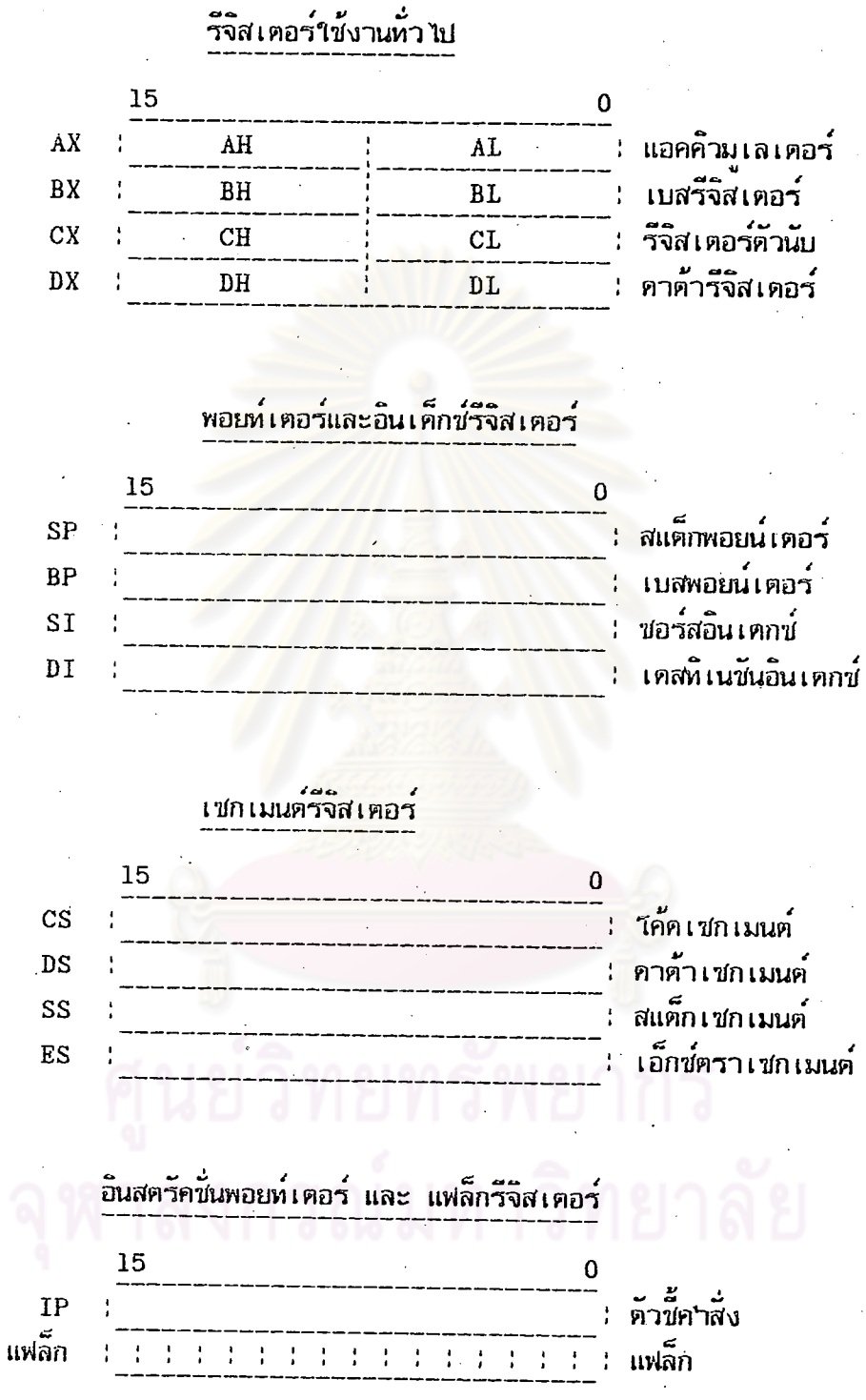
CX ถ้าใช้ในลักษณะ 8 บิตจะใช้ชื่อ CH กับ CL

DX ถ้าใช้ในลักษณะ 8 บิตจะใช้ชื่อ DH กับ DL

รีจิสเตอร์เหล่านี้นอกจากจะใช้เป็น คาต้ารีจิสเตอร์ในการใช้งานทั่วไปแล้วยังใช้ในกรณีพิเศษอีกดังนี้

AX - จะใช้เป็นแอคคิวมูเลเตอร์ (Accumulater) ใช้ในคำสั่งคูณ, หาร คำสั่งเกี่ยวกับอินพุท เอ้าท์พุท คำสั่งเกี่ยวกับสตริง (String)

BX - ใช้เป็นเบสรีจิสเตอร์ (Base Register) ใช้มากในการอ้างถึงตำแหน่งในหน่วยความจำ



รูปที่ 3.2 8088 รีจิสเตอร์

CX - ใช้เป็นตัวนับ (Counter) ใช้มากในการทำงานเป็นวัฏจักร (Loop) หรือการทำงานเกี่ยวกับสตริง

DX - ใช้เป็นตาต้ารีจิสเตอร์ (Data Register) ใช้ในคำสั่งคูณ, หาร คำสั่งเกี่ยวกับอินพุท เอ้าท์พุท โดยจะเก็บหมายเลขของพอร์ทไว้

2. เซกเมนต์รีจิสเตอร์ (Segment Register)

8088 ไมโครโปรเซสเซอร์มีลักษณะพิเศษกว่าไมโครโปรเซสเซอร์อื่นๆ คือ จะจัดแบ่งหน่วยความจำออกเป็นส่วนๆ เรียกว่า "เซกเมนต์" โดยจะมีเซกเมนต์รีจิสเตอร์เก็บตำแหน่งเริ่มต้นของแต่ละเซกเมนต์ไว้ เซกเมนต์รีจิสเตอร์มีอยู่ด้วยกัน 4 ตัวคือ

CS - เก็บตำแหน่งเริ่มต้นของโปรแกรมที่กำลังทำงานอยู่ในขณะนั้น

SS - เก็บตำแหน่งเริ่มต้นของส่วนของสแต็กในหน่วยความจำ

DS - เก็บตำแหน่งเริ่มต้นของส่วนของข้อมูลของโปรแกรม

ES - เก็บตำแหน่งของข้อมูลเช่นเดียวกันกับ DS

3. พอยท์เตอร์และอินเด็กซ์รีจิสเตอร์ (Pointer And Index Register) มี 4 ตัวคือ

BP - ใช้อ้างถึงตำแหน่งของสแต็กเพื่อประโยชน์ในการอ่านค่าในสแต็กมาใช้งาน

SP - จะเก็บตำแหน่งบนสุดของสแต็ก ค่าของ SP จะเปลี่ยนไปเมื่อใช้คำสั่ง PUSH หรือ POP

SI - จะเก็บตำแหน่งของกลุ่มข้อมูลต้นทาง มักจะใช้ในกลุ่มของคำสั่งเกี่ยวกับสตริง

DI - จะเก็บตำแหน่งปลายทางของกลุ่มข้อมูล มักจะใช้ในกลุ่มของคำสั่งเกี่ยวกับสตริง

4. อินสตรัคชั่นพอยท์เตอร์ และ แฟล็กรีจิสเตอร์ (Instruction Pointer AND Flag Register)

IP - จะเก็บตำแหน่งของคำสั่งถัดไปที่จะถูกนำมาทำงาน ค่าของ IP จะเปลี่ยนไปใน 2 กรณี คือ

1. เมื่อมีการนำคำสั่ง ณ.ตำแหน่งที่เก็บไว้ใน IP ในขณะนั้นมาทำงาน ค่าของ IP จะถูกเพิ่มขึ้นเท่ากับความยาวของคำสั่งนั้น

2. กรณีที่เป็นคำสั่งกระโดดไปทำงาน ณ.จุดใดจุดหนึ่งของโปรแกรม หรือ มีการเรียกโปรแกรมย่อยมาใช้

แฟล็กรีจิสเตอร์ - มี 16 บิต แต่จะใช้งานทั้งหมด 9 บิต คือ แฟล็กสถานะ 6 บิต แฟล็กควบคุม 3 บิต

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

รูปที่ 3.3 แฟล็กรีกิสเตอร์

CF - จะถูกเซตเมื่อมีการทดหรือการขอยืมของบิตสูงสุดของผลลัพธ์

PF - แสดงจำนวนคู่หรือคี่ของตัวเลขที่เป็น 1 ในรีจิสเตอร์ผลลัพธ์ ถ้าเป็นคู่จะถูกเซตเป็น 1

AF - จะถูกเซตเมื่อเกิดการทดหรือขอยืมที่บิตกึ่งกลางส่วนล่างของผลลัพธ์

ZF - จะถูกเซตเมื่อผลลัพธ์เป็น 0

SF - จะถูกเซตเมื่อผลลัพธ์เป็นเลขลบ

OF - จะถูกเซตเมื่อเกิดการผิดพลาดทางการคำนวณ

DF - แฟล็กควบคุมทิศทาง กรณีที่ใช้คำสั่งเกี่ยวกับสตริงโดยบอกทิศทางว่าจะกระทำจากตำแหน่งต่ำไปยังตำแหน่งสูง หรือจากตำแหน่งสูงมายังตำแหน่งต่ำ

IF - ถ้าถูกเซต จะหมายถึงยอมรับการเกิดอินเตอร์รัพจากภายนอก

TF - จะถูกเซตเมื่อต้องการให้ทำงานทีละคำสั่ง

การคำนวณหาตำแหน่งในหน่วยความจำ

ดังได้กล่าวมาแล้วข้างต้นว่า ไมโครโปรเซสเซอร์ 8088 สามารถอ้างตำแหน่งในหน่วย

ความจำได้ 1 เมกกะไบต์ แต่รีจิสเตอร์ที่ใช้ในการทำงานมี 16 บิต ซึ่งสามารถอ้างตัวเลขสูงสุดได้ 65,536 หรือ 64 กิโลไบต์ เท่านั้น ดังนั้นไมโครโปรเซสเซอร์ 8088 จึงใช้วิธีการแบ่งหน่วยความจำเป็นส่วน ๆ เรียกว่าเซกเมนต์ แต่ละเซกเมนต์มีขนาดสูงสุด 64 กิโลไบต์ โดยจะใช้เซกเมนต์รีจิสเตอร์ในการอ้างถึงตำแหน่งเริ่มต้นของแต่ละเซกเมนต์ ในการคำนวณหาตำแหน่งในหน่วยความจำ ค่าในเซกเมนต์รีจิสเตอร์จะถูกเลื่อนไปทางซ้าย 4 บิต แล้วรวมเข้ากับค่าของออฟเซต (Offset) ขนาด 16 บิต ค่าของออฟเซตอาจจะเก็บไว้ในเบสรีจิสเตอร์ หรืออินเดกซ์รีจิสเตอร์ ที่สัมพันธ์กับเซกเมนต์รีจิสเตอร์นั้นดังนี้

CS	ใช้กับ	IP	–	การเฟตคำสั่ง
SS	ใช้กับ	SP,BP	–	การทำสแต็ก
DS,ES	ใช้กับ	ค่าของแอดเดรส	–	การกำหนดตัวแปร
DS	ใช้กับ	SI	–	สตริงต้นทาง
ES	ใช้กับ	DI	–	สตริงปลายทาง

รูปที่ 3.4 การคำนวณในการอ้างแอดเดรส

การอ้างแอดเดรส (Addressing Mode)

การอ้างแอดเดรสของ ไมโครโปรเซสเซอร์เบอร์ 8088 สามารถแบ่งออกได้เป็น 7 ลักษณะคือ

1. รีจิสเตอร์แอดเดรส (Register Addressing)

2. อิมมีเดียทแอดเดรส (Immediate Addressing)

3. ไดเรคท์แอดเดรส (Direct Addressing)

4. รีจิสเตอร์อินไดเร็กซ์แอดเดรส (Register Indirect Addressing)

5. เบสรีเลทีพแอดเดรส (Base Relative Addressing)

6. ไดเรคท์อินเด็กซ์แอดเดรส (Direct Index Addressing)

7. เบสอินเด็กซ์แอดเดรส (Base Index Addressing)

การอ้างแอดเดรสใน 7 ลักษณะนี้ ใน 2 ลักษณะแรกจะเป็นวิธีการที่เร็วที่สุดเพราะว่าส่วนปฏิบัติงานสามารถดึงค่าโอเปอร์แรนด์ได้จากรีจิสเตอร์ หรือจากสตรีมไบต์คิวได้ทันที สำหรับใน 5 ลักษณะหลังเมื่อส่วนปฏิบัติงานต้องการอ่านค่า หรือเก็บค่าในหน่วยความจำ ส่วนปฏิบัติงานจะส่งค่าออฟเซตของโอเปอร์แรนด์ มายังส่วนเฟตคำสั่ง ค่าออฟเซตนี้เรียกว่า แอฟแฟคทีฟแอดเดรส หรือ อีเอ (Effective Address or EA) เมื่อได้ค่าอีเอ ส่วนเฟตคำสั่งจะรวมค่าของอีเอนั้นเข้ากับค่าของเซกเมนต์รีจิสเตอร์ จึงจะได้ตำแหน่งของโอเปอร์แรนด์ในหน่วยความจำ

1. รีจิสเตอร์แอดเดรส

ค่าของโอเปอร์แรนด์ได้จากรีจิสเตอร์ เช่น

```
MOV     AX,BX
```

2. อิมมีเดียทแอดเดรส

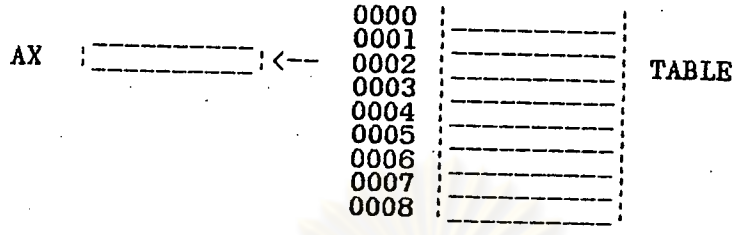
สามารถกำหนดค่าให้กับโอเปอร์แรนด์ได้โดยตรง เช่น

```
MOV     CX,10
```

3. ไดเรคท์แอดเดรส

ตำแหน่งของโอเปอร์แรนด์จะได้จากค่าแอฟแฟคทีฟแอดเดรสในคำสั่งรวมกับค่าในเซกเมนต์รีจิสเตอร์ DS เช่น

```
        MOV     AX,TABLE
```

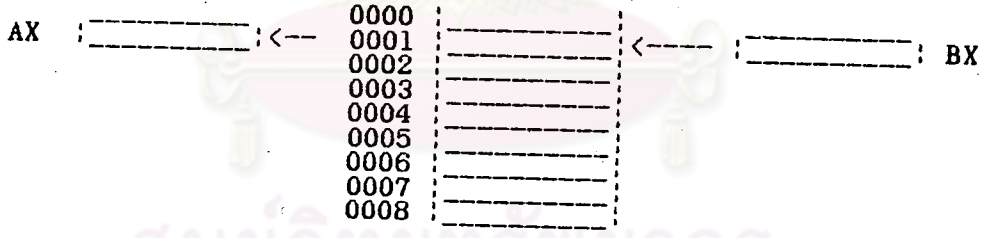


รูปที่ 3.5 การอ้างแอดเดรสแบบไดเรกท์แอดเดรส

4. รีจิสเตอร์อินไดเรกท์แอดเดรส

ค่าเอฟเฟคทีฟแอดเดรสอยู่ในเบสรีจิสเตอร์ BX,BP หรืออยู่ในอินเดกซ์รีจิสเตอร์ DI,SI ค่าของโอเปอร์แรนด์ได้จากการใส่เครื่องหมาย "[ ]" ในเบสหรืออินเดกซ์รีจิสเตอร์นั้น เช่น

```
        LEA     BX,TABLE
        MOV     AX,[BX]
```



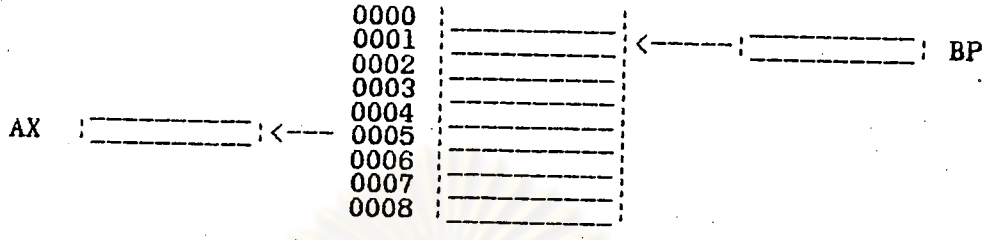
รูปที่ 3.6 การอ้างแอดเดรสแบบอินไดเรกท์แอดเดรส

5. เบสรีเลทีฟแอดเดรส

ค่าเอฟเฟคทีฟแอดเดรสได้จากผลรวมระหว่างเบสพอยน์เตอร์กับดิสเพลสเมนต์ (Displacement) รูปแบบของการใช้การแอดเดรสแบบเบสรีเลทีฟแอดเดรสทำได้ 3 ลักษณะที่แตกต่างกันคือ

```
        MOV     AX,[BP]+4       ; รูปแบบโดยทั่วไป
        MOV     AX,4[BP]        ; ดิสเพลสเมนต์อยู่ข้างหน้า
```

```
MOV     AX,[BP+4]       ; ดิสเพลสเมนต์อยู่ภายในวงเล็บ
```

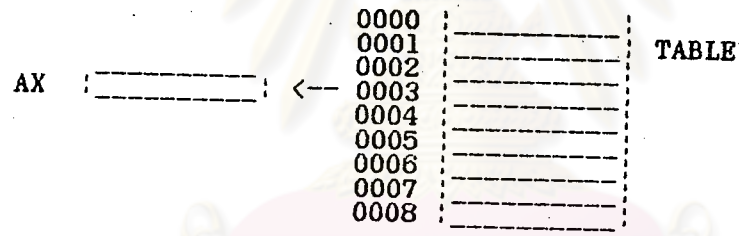


รูปที่ 3.7 การอ้างแอดเดรสแบบเบสรีเลทีฟแอดเดรส

6. ไดเรกท์อินเด็กซ์แอดเดรส

ค่าเอฟเฟคทีฟแอดเดรสได้จากผลรวมระหว่างอินเดกซ์รีจิสเตอร์กับดิสเพลสเมนต์

```
MOV     DI,2
MOV     AX,TABLE[DI]
```

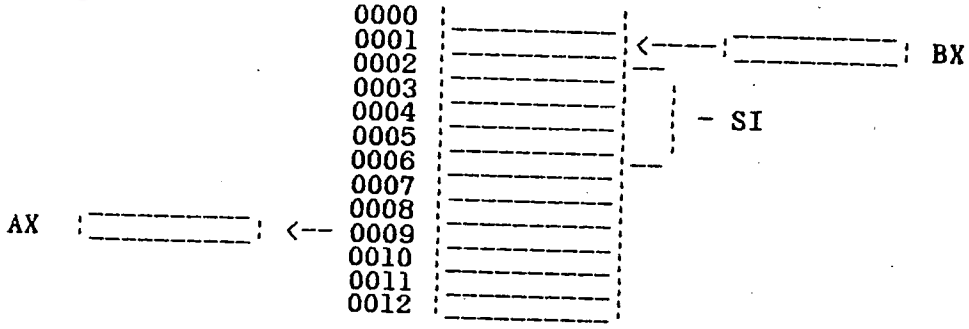


รูปที่ 3.8 การอ้างแอดเดรสแบบไดเรกท์อินเด็กซ์แอดเดรส

7. เบสอินเด็กซ์แอดเดรส

ค่าเอฟเฟคทีฟแอดเดรสได้จากผลบวกของเบสรีจิสเตอร์รวมกับอินเด็กซ์รีจิสเตอร์รวมกับดิสเพลทเมนต์

```
MOV     AX,[BX+SI+3]
```



รูปที่ 3.9 การอ้างแอดเดรสแบบเบสอินเด็กซ์แอดเดรส

ชุดคำสั่งของ 8088

คำสั่งของ 8088 แบ่งได้เป็น 7 กลุ่มดังนี้

1. คำสั่งเกี่ยวกับการเคลื่อนย้ายข้อมูล
2. คำสั่งทางคณิตศาสตร์
3. คำสั่งทางลอจิก
4. คำสั่งทำงานกับสตริง
5. คำสั่งเปลี่ยนทิศทางการทำงาน
6. อินเตอร์รัพ
7. คำสั่งควบคุมโปรเซสเซอร์

1. คำสั่งเกี่ยวกับการเคลื่อนย้ายข้อมูล

MOV - MOV dest,source

ทำการเคลื่อนย้ายข้อมูลจากต้นทางไปยังปลายทางโดยไม่ทำให้ค่าต้นทางเปลี่ยน เช่น

```
MOV     BX,AX
MOV     TABLE,AX
MOV     ES:[BX],AX
```

PUSH - PUSH source

เก็บข้อมูล 1 เวิร์ดลงสแต็กค่าของ SP จะถูกลดลง 2 ก่อนที่ข้อมูลจะถูกเก็บ

```
PUSH    SI
PUSH    COUNTER
PUSH    TABLE[BX][DI]
```

POP - POP dest

ดึงข้อมูลขนาด 1 เวิร์ดในสแต็กมาเก็บในรีจิสเตอร์หรือหน่วยความจำ แล้วเพิ่มค่า SP อีก 2

```
        POP     SI
        POP     COUNTER
        POP     TABLE[BX][DI]
```

XCHG - XCHG dest,source

สลับข้อมูลระหว่างรีจิสเตอร์หรือระหว่างหน่วยความจำกับรีจิสเตอร์

```
        XCHG    AX,BX
        XCHG    WORD_LOC,DX
```

XLAT - XLAT source-table

นำค่าในตารางตำแหน่งที่ AL ขนาด 1 ไบต์ ซึ่งตำแหน่งเริ่มต้นของตาราง เก็บใน BX มาเก็บใน AL

```
        MOV     AL,10
        MOV     BX,OFFSET TABLE
        XLAT    TABLE
```

IN - IN acc,port

อ่านข้อมูลจากพอร์ทมาเก็บไว้ใน AL หรือ AX ตำแหน่งของพอร์ทมักจะเก็บไว้ใน DX

```
        IN      AL,200
        IN      AX,DX
```

OUT - OUT port,acc

ส่งข้อมูล 1 ไบต์หรือ 1 เวิร์ด จาก AL หรือ AX ไปยังพอร์ทที่กำหนด

```
        OUT     200,AL
        OUT     DX,AX
```

LEA - LEA reg16,mem16

เก็บค่าตำแหน่งของหน่วยความจำไว้ในรีจิสเตอร์ที่กำหนด

```
        LEA     BX,TABLE
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

LDS - LDS reg16,mem32

นำค่าตำแหน่งของหน่วยความจำขนาด 2 เวิร์ด ไปเก็บในรีจิสเตอร์ที่กำหนด และ DS โดย 1 เวิร์ดแรกเก็บไว้ในรีจิสเตอร์ที่กำหนด 1 เวิร์ดหลังเก็บไว้ใน DS

```
HERE_FAR DD  HERE
          :
          :
         LDS    BX,HERE_FAR
```

LES - LES reg16,mem32

นำค่าตำแหน่งของหน่วยความจำขนาด 2 เวิร์ด ไปเก็บในรีจิสเตอร์ที่กำหนด และ ES โดย 1 เวิร์ดแรกเก็บไว้ในรีจิสเตอร์ที่กำหนด 1 เวิร์ดหลังเก็บไว้ใน ES

```
HERE_FAR DD  HERE
          :
          :
         LES    BX,HERE_FAR
```

LAHF - LAHF

นำค่า 8 บิตต่ำสุดของแฟล็กรีจิสเตอร์มาเก็บไว้ใน AH

SAHF - SAHF

เก็บค่า 8 บิตของ AH ไว้ใน 8 บิตท้ายของแฟล็กรีจิสเตอร์

PUSHF - PUSHF

เก็บค่า 16 บิตของแฟล็กรีจิสเตอร์ลงสแต็ก

POPF - POPF

อ่านค่า 16 บิตจากสแต็กเก็บไว้ในแฟล็กรีจิสเตอร์

2. คำสั่งทางคณิตศาสตร์

ADD - ADD dest,source

บวกค่า 1 ไบต์หรือ 1 เวิร์ด

```
ADD     AX,CX
ADD     AL,10
ADD     AX,MEM_WORD
```

ADC - ADC dest,source

บวกค่า 1 ไบต์หรือ 1 เวิร์ดและ CF แฟล็ก โดยที่ถ้า CF แฟล็กถูกเซต จะบวก 1 เข้ากับผลลัพธ์

```
ADC     AX,CX
ADC     AX,MEM_WORD
```

SUB - SUB dest,source

ลบค่า 1 ไบต์หรือ 1 เวิร์ด

```
SUB     AX,CX
SUB     AX,MEM_WORD
```

SBB - SBB dest,source

ลบค่า 1 ไบต์หรือ 1 เวิร์ดและ CF แฟล็ก โดยที่ถ้า CF แฟล็กถูกเซต จะลบ 1 ออกจากผลลัพธ์

```
SBB     AX,CX
SBB     AX,MEM_WORD
```

INC - INC dest

เพิ่มค่าในรีจิสเตอร์หรือในหน่วยความจำอีก 1

```
INC     CX
INC     WORD PTR [BX]
```

DEC - DEC dest

ลดค่าในรีจิสเตอร์หรือในหน่วยความจำลงอีก 1

```
DEC     CX
DEC     WORD PTR [BX]
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

NEG - NEG dest

ลบ 0 ด้วยค่าของโอเปอร์แรนด์

```
NEG     AL
```

CMP - CMP dest,source

ทำงานเหมือนคำสั่ง SUB แต่จะไม่มีการเก็บค่าของผลลัพธ์ไว้ นอกจากมีผลต่อแฟล็กรีจิสเตอร์ ใช้ในการเปรียบเทียบข้อมูล

```
CMP     AX,DX
CMP     MEM_WORD,SI
```

MUL - MUL source

คูณเลข 1 ไบต์หรือ 1 เวิร์ด กับ AL หรือ AX โดยไม่คิดเครื่องหมาย ถ้าเป็นการคูณ 1 ไบต์ ผลลัพธ์จะเก็บใน AX ถ้าเป็นการคูณ 1 เวิร์ดผลลัพธ์จะเก็บใน DX และ AX

```
MUL     BX
MUL     MEM_WORD
```

IMUL - IMUL source

คูณเลข 1 ไบต์หรือ 1 เวิร์ด กับ AL หรือ AX โดยคิดเครื่องหมาย

```
IMUL    BX
IMUL    MEM_WORD
```

DIV - DIV source

หารเลข 1 ไบต์หรือ 1 เวิร์ด กับ AL หรือ AX โดยไม่คิดเครื่องหมาย ถ้าเป็น 1 ไบต์ ผลลัพธ์จะเก็บใน AL เศษของการหารจะเก็บใน AH ถ้าเป็น 1 เวิร์ด ผลลัพธ์จะเก็บใน AX และเศษของการหารจะเก็บใน DX

```
DIV     BX
DIV     MEM_WORD
```

IDIV - IDIV source

หารเลข 1 ไบต์หรือ 1 เวิร์ด กับ AL หรือ AX โดยคิดเครื่องหมาย

```
IDIV    BX
IDIV    MEM_WORD
```

CBW - CBW

แปลงข้อมูล 1 ไบต์ใน AL เป็นข้อมูล 1 เวิร์ดใน AX

CWD - CWD

แปลงข้อมูล 1 เวิร์ดใน AX เป็นข้อมูล 2 เวิร์ดใน DX และ AX

3. คำสั่งทางลอจิก

AND - AND dest,source

"AND" ไบต์ หรือ เวิร์ด

```
AND     AX,BX
AND     BL,0110B
```

OR - OR dest,source

"OR" ไบต์ หรือ เวิร์ด

```
OR      AX,BX
OR      AX,MEM_WORD
```

XOR - XOR dest,source

"XOR" ไบต์ หรือ เวิร์ด

```
XOR     SI,DX
XOR     BETA[BX][DI],AX
```

TEST - TEST dest,source

"TEST" ไบต์ หรือ เวิร์ด ทำเหมือนคำสั่ง AND แต่ไม่เก็บค่าของผลลัพธ์ไว้นอกจากเซตค่าของแฟล็กรีจิสเตอร์

```
        TEST    AL,6
        TEST    BETA[BX][DI],AX
```

NOT - NOT dest

"NOT" ไบต์ หรือ เวิร์ด

```
        NOT     DX
        NOT     MEM_BYTE
```

SAL - SAL dest,count

เลื่อนบิตของไบต์หรือเวิร์ดไปทางซ้าย เท่ากับจำนวนครั้งที่กำหนดโดยคิดเครื่องหมาย

```
        SAL     AL,1
        SAL     AX,CL
```

SHL - SHL dest,count

เลื่อนบิตของไบต์หรือเวิร์ดไปทางซ้าย เท่ากับจำนวนครั้งที่กำหนดโดยไม่คิดเครื่องหมาย

```
        SHL     CX,1
        SHL     BETA[BX][DI],CL
```

SAR - SAR dest,count

เลื่อนบิตของไบต์หรือเวิร์ดไปทางขวา เท่ากับจำนวนครั้งที่กำหนดโดยคิดเครื่องหมาย

```
        SAR     AL,1
        SAR     AX,CL
```

SHR - SHR dest,count

เลื่อนบิตของไบต์หรือเวิร์ดไปทางขวา เท่ากับจำนวนครั้งที่กำหนดโดยไม่คิดเครื่องหมาย

```
SHR     AL,1
SHR     AX,CL
```

ROL - ROL dest,count

หมุนบิตของไบต์หรือเวิร์ดไปทางซ้าย เท่ากับจำนวนครั้งที่กำหนด

```
ROL     AH,1
ROL     BETA[BX][DI],CL
```

RCL - RCL dest,count

หมุนบิตของไบต์หรือเวิร์ดไปทางซ้าย โดยผ่าน CF แฟล็ก เท่ากับจำนวนครั้งที่กำหนด

```
RCL     AH,1
RCL     BETA[BX][DI],CL
```

ROR - ROR dest,count

หมุนบิตของไบต์หรือเวิร์ดไปทางขวา เท่ากับจำนวนครั้งที่กำหนด

```
ROR     CX,1
ROR     AX,CL
```

RCR - RCR dest,count

หมุนบิตของไบต์หรือเวิร์ดไปทางขวา โดยผ่าน CF แฟล็กเท่ากับจำนวนครั้งที่กำหนด

```
RCR     CX,1
RCR     BETA[BX][DI],CL
```

4. คำสั่งทำงานกับสตริง

REP - REP

จะกระทำซ้ำเท่ากับจำนวนครั้งที่เก็บในรีจิสเตอร์ CX

REPE - REPE

จะกระทำซ้ำเมื่อเท่ากับ เท่ากับจำนวนครั้งที่เก็บในรีจิสเตอร์ CX

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

REPNE - REPNE

จะกระทำซ้ำเมื่อไม่เท่ากับ เท่ากับจำนวนครั้งที่เก็บในรีจิสเตอร์ CX

REPZ - REPZ

จะกระทำซ้ำเมื่อ ZF แฟล็กเป็น 1 และ CX ไม่เป็น 0

REPNZ - REPNZ

จะกระทำซ้ำเมื่อ ZF แฟล็กเป็น 0 และ CX ไม่เป็น 0

MOVSB/ MOVSW - MOVSB
or
MOVSW

จะทำการอ่านข้อมูลทีละไบต์หรือเวิร์ด จากแหล่งข้อมูลต้นทางที่กำหนดโดย DS และ SI ไปเก็บยังแหล่งข้อมูลปลายทางที่กำหนดโดย ES และ DI ค่าของ SI,DI จะเพิ่มหรือลดขึ้นอยู่กับ DF แฟล็ก

```
MOV     SI,OFFSET SOURCE
MOV     DI,OFFSET DEST
MOV     CX,LENGTH_SOURCE
CLD
REP     MOVSB
```

CMPSB/ CMPSW - CMPSB
or
CMPSW

จะทำการเปรียบเทียบข้อมูลทีละไบต์หรือเวิร์ด จากแหล่งข้อมูลต้นทางที่กำหนดโดย DS และ SI กับแหล่งข้อมูลปลายทางที่กำหนดโดย ES และ DI ค่าของ SI,DI จะเพิ่มหรือลดขึ้นอยู่กับ DF แฟล็ก

```
MOV     SI,OFFSET SOURCE
MOV     DI,OFFSET DEST
MOV     CX,LENGTH_SOURCE
STD
REPE    CMPSW
```

SCASB/ SCASW - SCASB
or
SCASW

จะทำการค้นหาข้อมูลทีละ ไบต์หรือเวิร์ด ของแหล่งข้อมูลที่กำหนดโดย ES และ DI กับค่าใน AL หรือ AX

```
        MOV     AL,'A'
        MOV     DI,OFFSET DEST
        MOV     CX,LENGTH_DEST
        STD
        REPE    SCASB
```

LOSB/ LODSW - LODSB
or
LODSW

จะทำการอ่านข้อมูล 1 ไบต์หรือ 1 เวิร์ด จากแหล่งข้อมูลที่กำหนดโดย DS และ SI เก็บไว้ใน AL หรือ AX

```
        CLD
        LEA     SI,SOURCE
        LEA     DI,DEST
        MOV     CX,200
        REPE    CMPSB
        JCXZ    MATCH
        DEC     SI
        LODSB
        .
        .
        .
MATCH:  .
        .
        .
```

STOSB/ STOSW - STOSB
or
STOSW

จะทำการเก็บข้อมูล 1 ไบต์หรือ 1 เวิร์ด จาก AL หรือ AX ไว้ในแหล่งข้อมูลที่กำหนดโดย ES และ DI

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

```
        CLD
        LEA     DI,DEST
        MOV     CX,200
        MOV     AL,'A'
        REPE    SCASB
        JCXZ    ALLOW
        DEC     DI
        STOSB
        :
ALLOW:  :
        :
```

5. คำสั่งเปลี่ยนทิศทางการทำงาน

5.1 คำสั่งเปลี่ยนทิศทางการทำงานโดยไม่มีเงื่อนไข

CALL - CALL target

คำสั่งเรียกโปรแกรมย่อยทำงาน โปรแกรมย่อยแบ่งเป็น 2 ลักษณะคือ

- โปรแกรมที่อยู่คนละเซกเมนต์กับโปรแกรมหลัก กำหนดดังนี้

```
prog-name   PROC    FAR
```

- โปรแกรมย่อยที่อยู่ภายใน เซกเมนต์เดียวกับโปรแกรมหลักกำหนดได้ดังนี้

```
prog-name   PROC    NEAR
```

กรณีโปรแกรมย่อยอยู่คนละเซกเมนต์กับโปรแกรมหลัก ค่าของ CS และ IP จะถูกเก็บลงในสแต็ก และจะถูกเปลี่ยนไปเป็นเซกเมนต์และออฟเซตของโปรแกรมย่อย แต่ถ้าอยู่ในเซกเมนต์เดียวกับโปรแกรมหลัก ค่าของ IP เท่านั้นที่จะถูกเก็บในสแต็ก และเปลี่ยนไปเป็นออฟเซตของโปรแกรมย่อย

```
CALL    MY_PROG
CALL    WORD PTR [BX]
```

RET - RET

คำสั่งกลับจากโปรแกรมย่อย ค่าของ IP และ/หรือ CS จะถูกนำออกมาจากสแต็ก ขึ้นอยู่กับลักษณะของโปรแกรมย่อยนั้น

JMP - JMP target

กระโดดไปทำงาน ณ.ตำแหน่งที่กำหนดโดยไม่มีเงื่อนไข

```
        JMP     THERE
        :
        :
THERE:  :
        :
```

5.2 คำสั่งเปลี่ยนทิศทางการทำงานโดยมีเงื่อนไข

กระโดดไปทำงาน ณ.ตำแหน่งที่กำหนด ตามเงื่อนไขที่กำหนด คำสั่งเหล่านี้ได้แก่

JG/JNLE - JG or JNE - กระโดดเมื่อมากกว่า

JGE/JNL - JGE or JNL - กระโดดเมื่อมากกว่าหรือเท่ากับ

JL/JNG - JL or JNG - กระโดดเมื่อน้อยกว่าหรือเท่ากับ

JE/JZ - JE or JZ - กระโดดเมื่อเท่ากับ หรือเมื่อ ZF แฟล็กเป็น 1

JNE/JNZ - JNE or JNZ - กระโดดเมื่อไม่เท่ากับ หรือเมื่อ ZF แฟล็กเป็น 0

JO - กระโดดเมื่อ OF แฟล็กเป็น 1

JNO - กระโดดเมื่อ OF แฟล็กเป็น 0

JS - กระโดดเมื่อ SF แฟล็กเป็น 1

JNS - กระโดดเมื่อ SF แฟล็กเป็น 0

I16390805

JC - กระโดดเมื่อ CF แฟล็กเป็น 1

JNC - กระโดดเมื่อ CF แฟล็กเป็น 0

JP - กระโดดเมื่อ PF แฟล็กเป็น 1

JNP - กระโดดเมื่อ PF แฟล็กเป็น 0

JCXZ - JCXZ target

กระโดดเมื่อ CX เป็น 0

LOOP - LOOP target

ลดค่าใน CX ลง 1 ถ้า CX ไม่เป็น 0 จะกระโดดไปทำงาน ณ.ตำแหน่งที่กำหนด

```
        MOV     CX,10
START:  :
        :
        LOOP    START
```

LOOPE - LOOPE target

ลดค่าใน CX ลง 1 ถ้า CX ไม่เป็น 0 และ ZF แฟล็กเป็น 1 จะกระโดดไปทำงาน ณ.ตำแหน่ง ที่กำหนด

```
        MOV     CX,10
START:  :
        :
        LOOP    START
```

LOOPNE - LOOPNE target

ลดค่าใน CX ลง 1 ถ้า CX ไม่เป็น 0 และ ZF แฟล็กเป็น 0 จะกระโดดไปทำงาน ณ.ตำแหน่ง ที่กำหนด

```
        MOV     CX,10
START:  :
        :
        LOOPNE  START
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

6. อินเตอร์รัพ

อินเตอร์รัพเกิดได้ 2 กรณีคือ ฮาร์ดแวร์อินเตอร์รัพ และซอฟท์แวร์อินเตอร์รัพ ฮาร์แวร์อินเตอร์รัพเกิดจากอุปกรณ์ภายนอกของระบบเช่น เมื่อมีการกดแป้นคีย์บอร์ด สำหรับซอฟท์แวร์อินเตอร์รัพเกิดจากโปรแกรม คำสั่งเกี่ยวกับอินเตอร์รัพ มีดังนี้

INT - INT interrupt-type

เมื่อเกิดการอินเตอร์รัพ 8088 จะทำดังนี้

1. เก็บค่าของแฟล็กรีจิสเตอร์ลงสแต็ก

2. เคลียร์ค่าของ TF แฟล็ก และ IF แฟล็ก เป็น 0

3. เก็บค่าของ CS รีจิสเตอร์ลงสแต็ก

4. คำนวณหาตำแหน่งของอินเตอร์รัพเวคเตอร์ โดยคูณหมายเลขอินเตอร์รัพด้วย 4 จะได้ตำแหน่งของโปรแกรมที่ตอบสนองอินเตอร์รัพที่เกิดขึ้น

5. เปลี่ยนค่าของ CS เป็นค่าของ 1 เวิร์ดหลังของค่าในอินเตอร์รัพเวคเตอร์ที่คำนวณได้

6. เก็บค่าของ IP รีจิสเตอร์ลงสแต็ก

7. เปลี่ยนค่าของ IP เป็นค่าของ 1 เวิร์ดแรกของค่าในอินเตอร์รัพเวคเตอร์ที่คำนวณได้

INT 16H
INT 10H

INTO - INTO

เป็นการอินเตอร์รัพแบบมีเงื่อนไข คือเมื่อ OF แฟล็กมีค่าเป็น 1

IRET - IRET

เป็นคำสั่งกลับจากโปรแกรมที่ทำงานตอบสนองอินเตอร์รัพ มายังโปรแกรมหลักโดยการพอพค่าจากสแต็กมาไว้ใน IP , CS และ แฟล็กรีจิสเตอร์ คำสั่ง IRET มักเป็นคำสั่ง

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

สุดท้ายของโปรแกรมที่ตอบสนองการเกิดอินเตอร์รัพ

7. คำสั่งควบคุมโปรเซสเซอร์

STC - เซตค่า CF แฟล็กเป็น 1

STD - เซตค่า DF แฟล็กเป็น 1

STI - เซตค่า IF แฟล็กเป็น 1

CLC - เซตค่า CF แฟล็กเป็น 0

CLD - เซตค่า DF แฟล็กเป็น 0

CLI - เซตค่า IF แฟล็กเป็น 0

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย