บทที่ 2

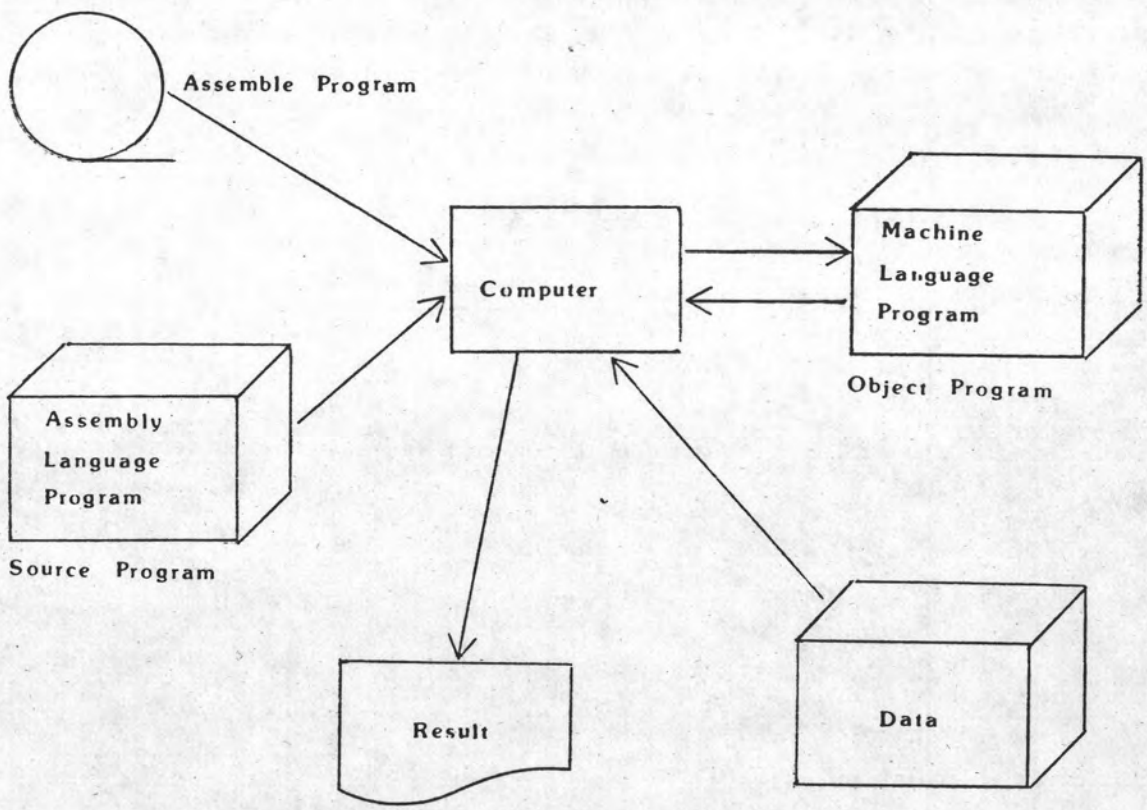
โครงสร้างของตัวแปลภาษาทั่วๆไป

2.1 บทนำ

ในยุคแรกของการนำคอมพิวเตอร์มาใช้การเขียนโปรแกรมให้คอมพิวเตอร์ทำงาน ผู้เขียนโปรแกรมต้องเขียนคำสั่งเป็น ภาษาเครื่อง ซึ่งเป็นภาษารหัสตัวเลขที่คอมพิวเตอร์สามารถตีความได้ การตั้งรหัสต่างๆจะขึ้นกับระบบการทำงานของเครื่องคอมพิวเตอร์แต่ละแบบ ดังนั้นผู้เขียนโปรแกรมจะต้องมีความเข้าใจเกี่ยวกับระบบคอมพิวเตอร์เป็นอย่างดี และเนื่องจากเป็นการสั่งงานโดยตรงจึงต้องสั่งงานให้ละเอียดทุกขั้นตอน และติดตามการทำงานของคอมพิวเตอร์ได้อย่างถูกต้อง ต่อมาได้มีการพัฒนาโปรแกรมโดยใช้สัญลักษณ์แทนรหัสตัวเลข เพื่อลดความยุ่งยากในการเขียนโปรแกรม เรียกว่า โปรแกรมภาษาแอสแซมบลี (Assembly Language) ดังรูป 2.1 ทำให้การเขียนโปรแกรมสะดวก และเข้าใจได้ง่ายขึ้นแต่จำเป็นที่จะต้องแปลภาษาแอสแซมบลีที่เขียนขึ้นให้เป็นภาษาเครื่องที่เครื่องคอมพิวเตอร์สามารถเข้าใจได้ก่อนทำงานตามคำสั่งนั้น โปรแกรมสำหรับแปลภาษาแอสแซมบลีให้เป็นภาษาเครื่อง เรียกโปรแกรมแอสแซมเบลอ (Assembler program) ดังรูป 2.2 โดยโปรแกรมที่เขียนด้วยภาษาแอสแซมบลีเรียกว่า โปรแกรมดิบ และโปรแกรมภาษาเครื่องที่จะนำไปใช้งาน เรียกโปรแกรมผล (Object progrm)

โปรแกรมภาษาแอสแซมบลี	โปรแกรมภาษาเครื่อง
MVC OUTAREA,BLANK	D2 28D49F 28D49E
UNPK A2,AREA	P3 28D5D8 28D5C8
PACK ADDR,ADD	F2 28D529 28D523
BNE ADDRESS	47 28D19C

รูป 2.1 เปรียบเทียบระหว่างภาษาแอสแซมบลีกับภาษาเครื่อง

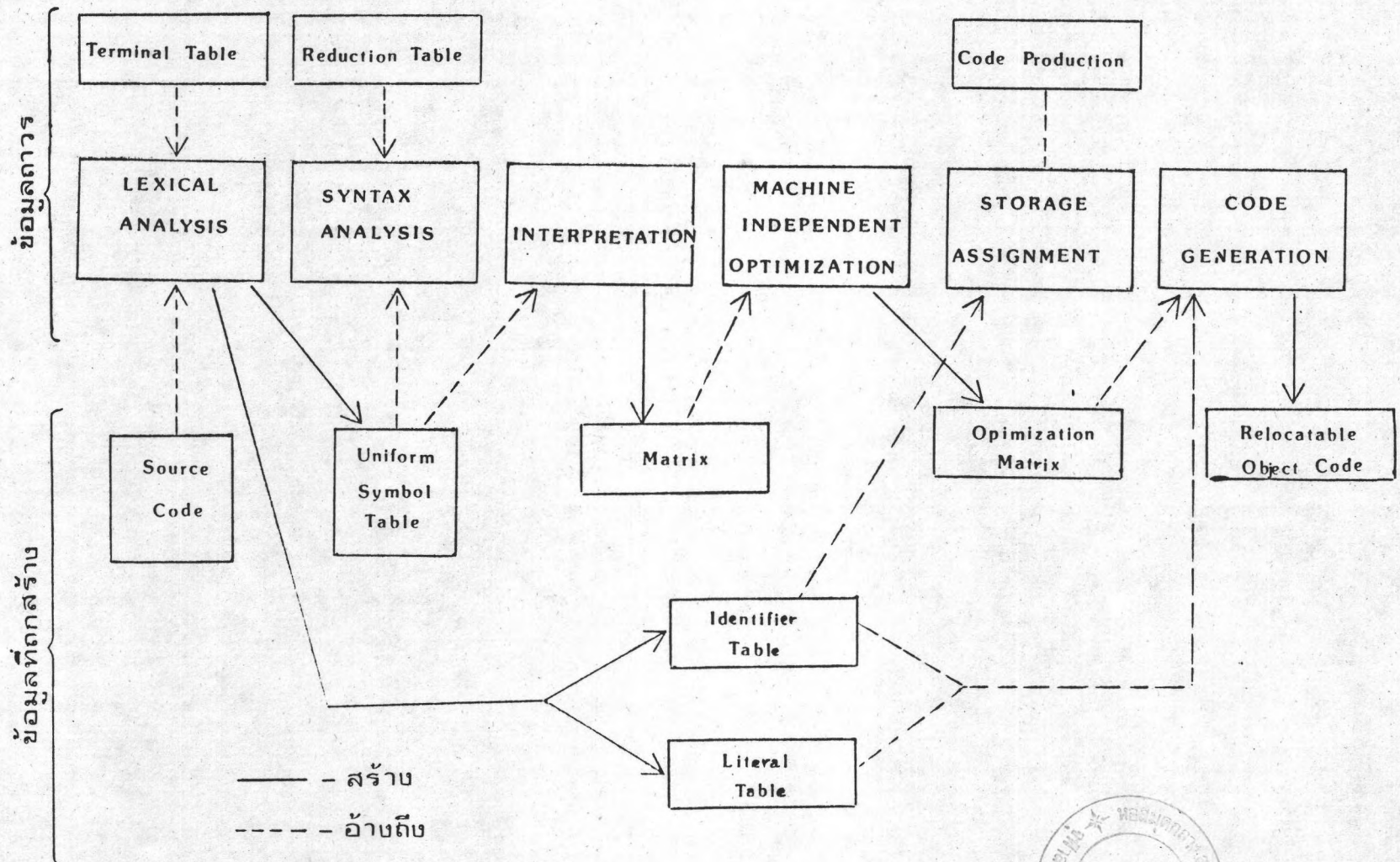


รูป 2.2 โปรแกรมแปลภาษาแอสแซมบลีและการทำงานตามโปรแกรมผล

เนื่องจากภาษาแอสแซมบลีกับภาษาเครื่องคล้ายกันมาก การเขียนคำสั่งยังคงจะต้องเขียนโดยละเอียดทุกขั้นตอน จึงได้มีการคิดค้นภาษาเขียนขึ้นมาใหม่โดยให้สามารถสั่งงานได้อย่างกว้างๆ แล้วให้โปรแกรมแปลแตกแยกคำสั่งเหล่านี้ให้เป็นขั้นตอนโดยละเอียด ทำให้ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมโดยคำนึงถึงแต่ความมุ่งหมายของโปรแกรมนั้น และไม่ต้องคอยพะวงกับรายละเอียดของขั้นตอนการทำงาน ภาษานี้เรียกว่าภาษาระดับสูง (High level language) ซึ่งมีหลายภาษา เช่น ฟอร์แทรน, โคบอล, พีแอล/1 ฯลฯ โปรแกรมพิเศษที่ใช้แปลภาษาเหล่านี้เรียกว่า ตัวแปลภาษา (Compiler)

2.2 ลักษณะโดยทั่วไปของตัวแปลภาษา

ตัวแปลภาษาจะมีโปรแกรมดิบเป็นข้อมูลเข้า และได้ผลลัพธ์เป็นชุดคำสั่งเครื่อง (Machine instruction) ที่มีลำดับการทำงานของคำสั่งเทียบเท่ากันกับโปรแกรมดิบ โดยทั่วๆไปตัวแปลภาษาจะมีหลักการคล้ายกันจะแตกต่างกันที่การจัดระบบการทำงานของเครื่องคอมพิวเตอร์ ตัวแปลภาษาจึงมักจะแบ่งออกเป็นโปรแกรมย่อยหลายๆ ส่วน เรียกว่า เฟส ดังแสดงไว้ในรูป 2.3



รูปที่ 2.3 โครงสร้างของตัวแปลภาษาทั่วๆไป

โครงสร้างหลักของตัวแปลภาษาทั่วไปประกอบเฟสต่างๆดังต่อไปนี้

1. เฟสการวิเคราะห์เลคซิเคิล (Lexical Analysis Phase) : ทำหน้าที่ในการกระจายโปรแกรมดิบให้เป็นโทเคน (token) และสร้างตารางสัญญลักษณ์ (Symbol table) , ตารางตัวชี้เฉพาะ , ตารางค่าคงที่

2. เฟสการวิเคราะห์วากยสัมพันธ์ (Syntax Analysis Phase) : ทำหน้าที่ตรวจสอบหลักไวยากรณ์ของภาษา และจัดรูปแบบให้เป็นไปตามหลักไวยากรณ์นั้นๆ

3. เฟสการแปล (Interpretation Phase) : ทำหน้าที่สร้างรูปแบบตัวกลางของภาษาดิบและเพิ่มข้อมูลให้กับตารางตัวชี้เฉพาะ (Indentifier table)

4. เฟสออปติไมเซชั่น (Optimization Phase) : ทำหน้าที่เปลี่ยนแปลงแปลงข้อมูลในเมตริกซ์ (matrix) ให้กระชับขึ้น

5. เฟสการจัดแบ่งเนื้อที่ในหน่วยความจำ (Storage Assignment Phase): ทำหน้าที่จัดแบ่งเนื้อที่ในหน่วยความจำให้กับข้อมูลทั้งหมดที่ต้องใช้ และแก้ไขข้อมูลในตารางตัวชี้เฉพาะ และตารางค่าคงที่ (Literal table)

6. เฟสการสร้างรหัส (Code Generation Phase) : ทำหน้าที่สร้างรหัสภาษาเครื่อง

2.3 ขั้นตอนการทำงานของเฟสต่างๆ

ในการแสดงขั้นตอนการทำงานของเฟสต่างๆจะขอใช้ตัวอย่างโปรแกรม ดังแสดงในรูป 2.4 เพื่อประกอบคำอธิบายต่อไป

```
WCM : PROCEDURE (RATE,START,FINISH);
      DECLARE (COST,RATE,START,FINISH) FIXED BINARY(31)STATIC;
      COST = RATE * (START-FINISH) +2 * RATE * (START-FINISH-100);
      RETURN (COST);
END;
```

รูป 2.4 ตัวอย่างโปรแกรมดิบภาษา PL/I

2.3.1 เฟสการวิเคราะห์เลคซิเคิล

หน้าที่

- กระจายโปรแกรมดิบให้เป็นโทเคน (ตัวอย่างดังแสดงในรูป 2.5)
- สร้างตารางค่าคงที่ และตารางตัวชี้เฉพาะ
- สร้างตารางสัญญลักษณ์

□ = 1 โทเคน

```
[WCM] [:] [PROCEDURE] [(] [RATE] [,] [START] [,] [FINISH] [)] [;]
    [DECLARE] [(] [COST] [,] [RATE] [,] [START] [,] [FINISH] [)] [FIXED] [BINARY] [(]
      [31] [)] [STATIC] [;]
    [COST] [=] [RATE] [*] [(] [START] [-] [FINISH] [)] [+] [2] [*]
      [RATE] [*] [(] [START] [-] [FINISH] [-] [100] [)] [;]
    [RETURN] [(] [COST] [)] [;]
[END] [;]
```

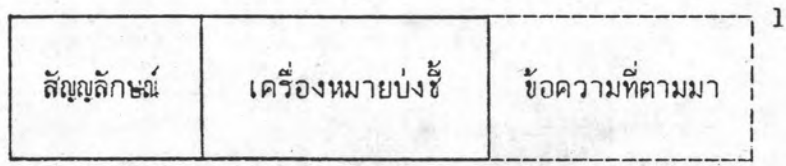
รูป 2.5 โทเคนของโปรแกรมตัวอย่าง

ข้อมูลที่ใช้

1. โปรแกรมดิบ - โปรแกรมภาษาต่างๆ (ตัวอย่างดังแสดงในรูป 2.4)

2. ตารางเทอร์มินอล (Terminal table) - เป็นส่วนหนึ่งของตัวแปลภาษา ประกอบด้วยระเบียนของสัญญลักษณ์เทอร์มินอลแต่ละตัว สัญญลักษณ์เทอร์มินอลต่างๆได้แก่ ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic operators) , คำหลัก (key words) , สัญญลักษณ์ที่ไม่ใช้ตัวอักษร (Nonalpharic symbol) เป็นต้น ในแต่ละระเบียน (entry) จะประกอบด้วย

- สัญญลักษณ์เทอร์มินอล (Terminal symbol)
- เครื่องหมายบ่งชี้รูปแบบ (Indicators) จะระบุว่าเป็นตัวดำเนินการหรือตัวหยุด (break character)
- ข้อความที่ตามมา (Precedence) ซึ่งจะถูกใช้ในเฟสที่ตามมา



รูป 2.6 ระเบียนของตารางเทอร์มินอล

3. ตารางค่าคงที่ - เป็นผลลัพธ์ของการทำงานของโปรแกรมในเฟสนี้ เพื่ออธิบายค่าคงที่ทั้งหมดที่ใช้ในโปรแกรมดิบ จะสร้างหนึ่งระเบียนสำหรับค่าคงที่แต่ละตัว ระเบียนหนึ่งจะประกอบด้วย

- ค่า (value)
- คุณลักษณะ (attribute)
- ตำแหน่งที่อยู่ (address)

- ข้อมูลอื่นๆ

ค่าคงที่	Base	Scale	Precision	ข้อมูลอื่นๆ	ตำแหน่งที่อยู่

1

รูป 2.7 ระเบียนของตารางค่าคงที่

4. ตารางตัวชี้เฉพาะ - สร้างขึ้นโดยเฟสนี้ เพื่ออธิบายตัวชี้เฉพาะทั้งหมดที่ใช้ในโปรแกรมดิบและจะสร้างหนึ่งระเบียนสำหรับตัวชี้เฉพาะแต่ละตัว ระเบียนหนึ่งๆจะประกอบด้วย

- ชื่อของตัวชี้เฉพาะ : ชื่อของตัวชี้เฉพาะในบางภาษาอาจจะมีความยาวตั้งแต่ 1-31 ตัวอักษรได้ ดังนั้น เฟสนี้อาจจะเก็บตัวชี้ (index)ไว้ในตารางนี้ โดยที่ตัวชี้จะชี้ไปที่ชื่อในตารางบรรจุชื่อ(table of names)
- คุณลักษณะ
- ตำแหน่งที่อยู่

ชื่อ	คุณลักษณะ	ตำแหน่งที่อยู่

1

รูป 2.8 ระเบียนของตารางตัวชี้เฉพาะ

---

1 ส่วนที่เป็นเส้นหนานั้นหมายถึงบริเวณที่จะถูกป้อนข้อมูลซึ่งกำลังกล่าวถึงตอนนี้ หรือได้กล่าวถึงแล้ว ส่วนที่เป็นรอยประนั้นหมายถึงส่วนที่จะถูกป้อนโดยเฟสอื่นที่ตามมา

5. ตารางสัญญลักษณ์ - สร้างขึ้นโดยเฟสนี้เพื่อเก็บโทเคนทั้งหมดในโปรแกรมคิบ และจะไม่เก็บที่ว่าง (spaces) หรือคำอธิบาย(comment) แต่ละระเบียนจะประกอบด้วย

- ตัวบ่งชี้ของตาราง : จะมีโทเคนนั้นๆเป็นสมาชิกอยู่
- ดัชนีของโทเคน ซึ่งอยู่ในตารางนั้น

ตัวบ่งชี้	ดัชนี

รูป 2.9 ระเบียนของตารางสัญญลักษณ์

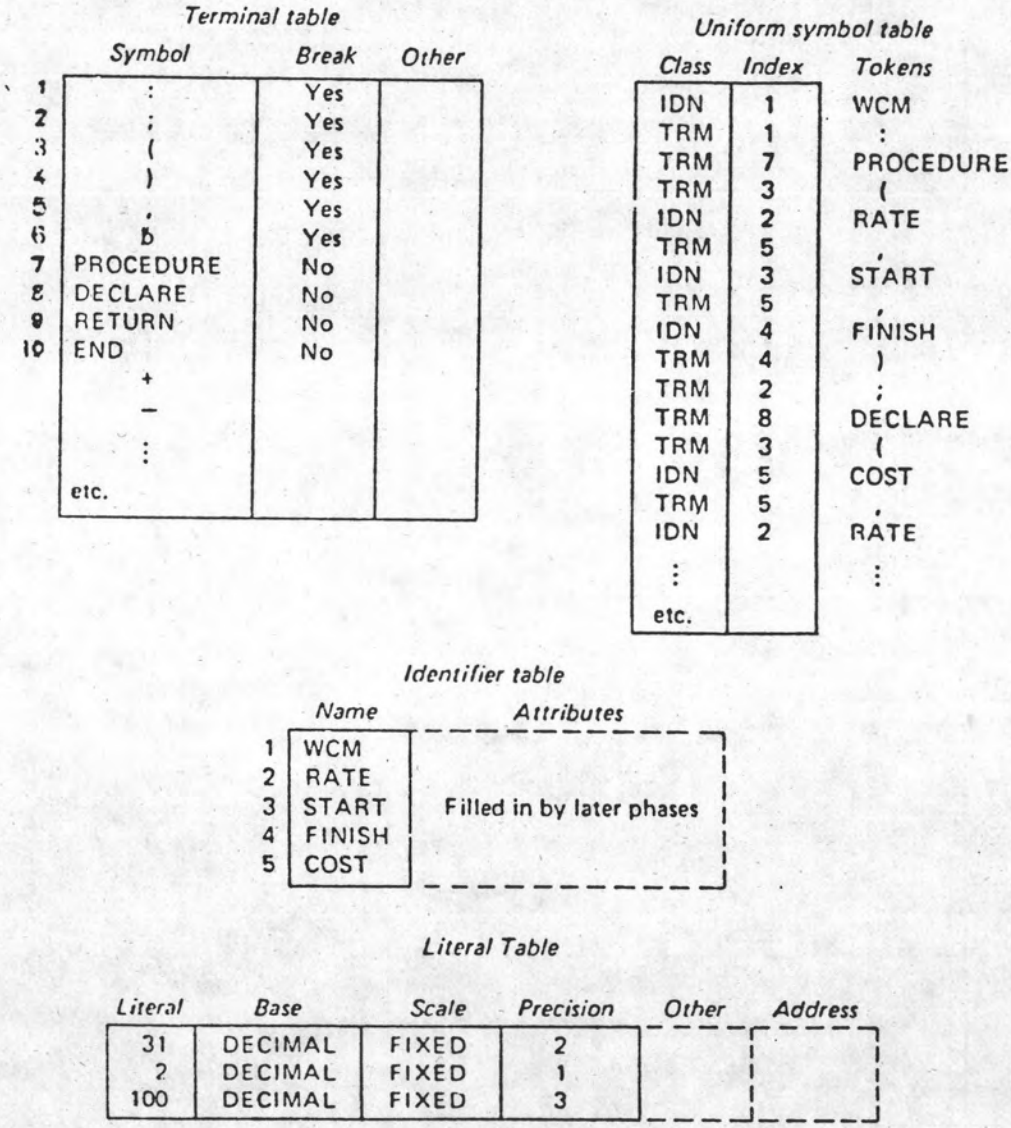
การทำงาน

1. กระจายสตริง (string) ของข้อมูลนำเข้าออกเป็น โทเคน
2. สร้างระเบียนต่างๆที่เหมาะสมเข้าไปในตารางต่างๆ

ตัวแปลภาษาจะถือว่าโทเคนเป็นหน่วยที่เล็กที่สุด เพราะโทเคน คือสตริงย่อยของสตริงที่ถูกนำเข้า อาจจะประกอบด้วยตัวอักษรหรือตัวเลข ตัวหยุดจะเป็นตัวแยกสตริงของข้อมูลนำเข้าออกเป็นโทเคนย่อยๆ ซึ่ง ได้แก่ สัญญลักษณ์ต่างๆที่บรรจุอยู่ในตารางเทอร์มินอล , ตัวชี้เฉพาะ และค่าคงที่

ในเฟสนี้จะใช้โทเคน 3 ชนิดด้วยกันคือ สัญญลักษณ์เทอร์มินอล, ตัวชี้เฉพาะ และค่าคงที่ที่จะตรวจสอบโทเคนทั้งหมด โดยการนำไปเปรียบเทียบกับสมาชิกต่างๆในตารางเทอร์มินอล ถ้าพบว่ามีคู่เหมือน (match) โทเคนตัวนี้จะถูกจัดเป็นสัญญลักษณ์เทอร์มินอล และสร้างระเบียนขึ้นในตารางสัญญลักษณ์หนึ่งระเบียน โดยที่ตัวบ่งชี้จะเป็น "TRM" แต่ถ้าโทเคนนั้นไม่ได้เป็นสัญญลักษณ์เทอร์มินอลก็จะตรวจสอบต่อไปว่า โทเคนนั้นเป็นตัวชี้เฉพาะหรือเป็นค่าคงที่ ถ้าโทเคนมีลักษณะถูกต้องตามข้อกำหนดของตัวชี้เฉพาะ ก็จะถูกจัดเป็นตัวชี้เฉพาะ แต่ถ้าโทเคนไม่ใช่ทั้ง 3 ประเภท จะถือว่าเป็น ข้อผิดพลาด(error) เมื่อโทเคนถูกจัดให้เป็นตัวชี้เฉพาะ

แล้วก็จะนำไปตรวจสอบเทียบในตารางตัวชี้เฉพาะ ถ้าไม่มีอยู่ในตารางก็จะมีการสร้างระเบียนใหม่เพิ่มเข้าไปในตารางตัวชี้เฉพาะ เนื่องจากคุณลักษณะอย่างเดียวของตัวชี้เฉพาะที่รู้ได้คือ ชื่อ เพราะฉะนั้นข้อมูลที่ป้อนเข้าในตารางตัวชี้เฉพาะในขณะนี้ก็คือชื่อของตัวชี้เฉพาะเท่านั้น ส่วนข้อมูลอื่นๆ ที่เกี่ยวข้องก็ถูกป้อนโดยเฟสที่ตามมา และจะสร้างระเบียนใหม่ขึ้นในตารางสัญญลักษณ์ โดยตัวบ่งชี้จะเป็น "IDN" ส่วนตัวเลขต่างๆ,กลุ่มคำ(character strings)ที่อยู่ในเครื่องหมายคำพูด และข้อมูลที่มีความหมายในตัวเอง (data self-defining) จะถูกจัดเป็นค่าคงที่ และจะมีการตรวจสอบในตารางค่าคงที่ ถ้าไม่พบ จะสร้างระเบียนใหม่ , กำหนดลักษณะต่างๆ และตัวแทนของค่าคงที่นั้น (internal representation) โดยพิจารณาจากตัวอักษร (character) ดังนั้นระเบียนใหม่นี้จะประกอบด้วยตัวชี้เฉพาะและคุณลักษณ์ของตัวชี้เฉพาะนั้น ซึ่งประกอบด้วย base, scale, pricision และจะสร้างระเบียนใหม่ในตารางค่าคงที่ โดยตัวบ่งชี้จะเป็น "LIT" การสร้างตารางต่างๆของเฟสนี้มีหลายวิธี แต่จะขอยกตัวอย่างดังแสดงในรูป 2.10



รูป 2.10 ตารางต่างๆที่ถูกสร้างขึ้นโดยเฟสการวิเคราะห์เลคซิเคิล

2.3.2 เฟสการวิเคราะห์วากยสัมพันธ์

หน้าที่

- ตรวจสอบหลักไวยากรณ์ในการสร้างประโยคของภาษา

- จัดรูปแบบของประโยคให้ตรงตามหลักไวยากรณ์

ตัวแปลภาษาบางชนิดจะใช้โปรแกรมขนาดใหญ่หนึ่งโปรแกรมเพื่อตรวจสอบรูปแบบ (construct) แต่ละแบบแยกกัน แต่เฟสนี้จะเป็นการแปลสำหรับกฎหรือข้อกำหนดทั่วไป (rules or reduction) ที่จะกำหนดรูปแบบของภาษา และของโปรแกรมที่จะใช้กับโครงสร้างแต่ละแบบ ข้อบังคับเหล่านี้ขึ้นอยู่กับหลักไวยากรณ์ของภาษาดิบ ถ้าไวยากรณ์ของภาษาดิบเปลี่ยน ข้อบังคับเหล่านี้ก็จะต้องถูกแก้ไขด้วย

ข้อมูล

1. ตารางสัญญลักษณ์ : - สร้างขึ้นโดยเฟสการวิเคราะห์เลคซิเคิล และเก็บสัญญลักษณ์ทั้งหมดในตารางสัญญลักษณ์ เพื่อเป็นข้อมูลนำเข้าของสแต็ค (stack)

2. สแต็ค : - แหล่งรวบรวมสัญญลักษณ์ทั้งหมดที่ถูกใช้งานโดยเฟสนี้ และเฟสการแปล เพื่อเพิ่มหรือลบข้อมูลออกจากสแต็ค การดำเนินการของข้อมูลในสแต็คจะมีลักษณะ "เข้าที่หลังออกก่อน" (Last In - First Out : LIFO) โดย "ตัวบนสุดของสแต็ค" (Top of stack) หมายถึงสมาชิกตัวล่าสุด และ "ตัวล่างสุดของสแต็ค" (Bottom of Stack) หมายถึงสมาชิกตัวเก่าที่สุด

3. ตารางกำหนดรูปแบบ (Reduction table): - เก็บหลักไวยากรณ์ต่างๆ ของภาษาดิบ เฟสนี้ใช้ข้อบังคับดังนี้คือ

ชื่อ : ตัวบนสุดเก่าของสแต็ค/โปรแกรม/ตัวบนสุดใหม่ของสแต็ค/ข้อกำหนดต่อไป

(Label : Old Top of Stack/Action/New Top of Stack/Next Reduction)

-ชื่อ : จะมีหรือไม่มีก็ได้

-ตัวบนสุดเก่าของสแต็ค : จะต้องเทียบกับตัวบนสุดของสแต็ค

ก. ที่ว่างหรือตัวที่ใช้ไม่ได้ (null) - จะมีคู่จับ(match)เสมอ โดยไม่คำนึงว่าจะมีอะไรในสแต็ค

ข. ไม่ใช่ที่ว่าง (nonblank) - จะต้องเป็นสมาชิกอย่างน้อยหนึ่งหนึ่งประเภทในประเภทต่อไปนี้

1) <ตัวชี้เฉพาะ หรือ ค่าคงที่> - จะจับคู่กับสัญลักษณ์ตัวไหนก็ได้ในชนิดนี้

2) <อื่นๆ> (any) - จะจับคู่กับสัญลักษณ์ชนิดไหนก็ได้

3) คำหลัก เช่น คำว่า "PROCEDURE" หรือ "IF" หรือ ":" จะจับคู่กับสัญลักษณ์ที่เป็นคำหลักด้วยกันเท่านั้น

-โปรแกรม (Action Routine) : จะถูกเรียกออกมาเมื่อตัวบนสุดเก่าเก่าของสแต็คจับคู่กับตัวบนสุดของสแต็ค

ก. ที่ว่าง หรือ ตัวที่ใช้ไม่ได้ - ไม่มีการเรียกโปรแกรม

ข. ชื่อของโปรแกรม - เรียกโปรแกรม

-ตัวบนสุดใหม่ของสแต็ค : จะมีการเปลี่ยนแปลงในตัวบนสุดของสแต็คหลังจากโปรแกรมถูกเรียกมาใช้งานแล้ว

ก. ที่ว่าง หรือ ตัวที่ใช้ไม่ได้ - ไม่มีการเปลี่ยนแปลง

ข. " - ลบตัวบนสุดของสแต็ค

ค. สัญลักษณ์ที่เป็น ชนิดของไวยากรณ์ (syntactic type), คำหลัก หรือ สมาชิกตัวเก่าที่สุดของสแต็ค - ลบตัวบนสุดเก่าของสแต็คออก และแทนที่ด้วยตัวนี้

ง. * - ดึงสัญลักษณ์ตัวต่อไปจากตารางสัญลักษณ์ แล้วใส่ไว้ส่วนบนสุดของสแต็ค

---

1 สัญลักษณ์นี้หมายถึงการลบออกเฉพาะสมาชิกที่จับคู่แล้ว

-ข้อกำหนดต่อไป

ก. ที่ว่าง หรือ ตัวที่ใช้ไม่ได้ - เลื่อนไปข้อกำหนดข้อต่อไป

ข. n - เลื่อนไปข้อกำหนดที่ n

การทำงาน

1. ข้อกำหนดรูปแบบต่างๆจะได้รับการตรวจสอบตามลำดับ เพื่อจับคู่ระหว่างตัวบนสุดเก่าของสแต็คกับตัวบนสุดของสแต็ค จนกว่าจะได้คู่

2. เมื่อพบคู่แล้ว, โปรแกรมที่ได้กำหนดไว้แล้วจะถูกเรียกออกมาใช้ตามลำดับจากซ้ายไปขวา

3. ส่งการควบคุมกลับ, ก็จะแก้ตัวบนสุดของสแต็คให้เข้ากันได้กับ ตัวบนสุดใหม่ของสแต็ค

4. จะย้อนกลับไปทำข้อ 1 โดยเริ่มด้วยข้อบังคับถัดไป

ตัวอย่าง :

```
//***/
  2:  <idn> : PROCEDURE / bgn-proc / S1 **** / 4
      <any> <any> <any> / ERROR / S2 S1 * / 2
```

1. เริ่มด้วยการดึงเอาสัญญาลักษณ์ 3 ตัวแรกจากตารางสัญญาลักษณ์ มาใส่ในสแต็ค

2. ทดสอบหาว่า สมาชิกทั้ง 3 ตัวนั้นเป็น <idn> : PROCEDURE ใช่หรือไม่ ถ้าใช่เรียกโปรแกรม begin procedure (bgn-proc) มา, ลบชื่อและเครื่องหมาย ":" ออก และดึงสัญญลักษณ์ 4 ตัวต่อมา จากตารางสัญญลักษณ์มาใส่ในสแต็ค และเลื่อนไปที่ reduction 4 ถ้าไม่ใช่ เรียกโปรแกรม ERROR ย้ายเอาสัญญลักษณ์ตัวที่สามออกจากสแต็ค แล้วดึงเอาอีกตัวจากตารางสัญญลักษณ์มาเพิ่ม แล้วเลื่อนไปที่ reduction 2

ปกตินั้น ข้อบังคับต่างๆจะระบุว่าโปรแกรมทุกโปรแกรมต้องเริ่มด้วย คำว่า " <label> : PROCEDURE " หลังจากตรวจดูประโยคนี้แล้ว โปรแกรม (bgn-proc) จะ

เตรียมข้อมูลของตารางตัวชี้เฉพาะที่เป็นสัญญลักษณ์ <ชื่อ> นี้ เพื่อใส่ในส่วนที่เป็นชื่อ จากนั้นเฟสนี้จะลบชื่อ และเครื่องหมาย ':' ออก

```
 1: //***/
 2: <idn> : PROCEDURE / bgn_proc/ S1 **** / 4        } INITIALIZE STACK
    <any> <any> <any> / error / S2 S1 * / 2          }   PROCEDURE?
 4: <idn> , <any> / arg / S4 S1 ** 4                 } ARGUMENT LIST
 5: <idn> ; / arg / " ** / 7                         }
    <any> <any> <any> <any> / error / S2 S1 / 7      }
 7: DECLARE <any> / / S1 ** / 14                     }
    RETURN <any> / return / S1 *** / 5               } DECLARE?
    <idn> = /op_prec// 12                            } RETURN?
    PROCEDURE END; /end_proc/ " *** / 2              } ASSIGNMENT?
    <any> <any> / error / S1 * / 7                   } END?
12: <idn> = <any> ; / assign / " ** / 7              }
    <any> <any> <any> <any> / error / S2 S1 / 7      } ASSIGNMENT
14: <idn> FIXED BINARY / var_name base_scale/"*****/ 18   }
15: (<idn>, /var_name / S3 ** / 15                  DECLARE   }
    (<idn>) / / S2 ** / 14                          name base scale }
    <any> <any> <any> / error / S2 S1 / 7                     }
18: (<lit>) STATIC , / precision_class / " *** / 14        DECLARE   }
    (<lit>) STATIC ; / precision_class / " ** / 7  precision and scale }
    <any> <any> <any> <any> <any> / error / S2 S1 / 7                  }
```

รูป 2.11 ตัวอย่างข้อกำหนดรูปแบบต่างๆในเฟสการวิเคราะห์วากยสัมพันธ์

```
Action routines:

   1 arg - place ".arg<idn>" into matrix

   2 assign - place "=<idn><any>" into matrix

   3 base_scale - place "base = binary" into proper identifier table
                  entry(ies)

   4 bgn_proc - (note identifier is a label in IDN table)
                place ".procbgn <idn>" into matrix

   5 end_proc - place ".procend" into matrix

   6 error - print out error msg and top of stack

   7 op_prec - parse arithmetic expression and generate matrix
               entries using rules of operator precedence, leave
               (result); on stack

   8 precision_class - place proper precision an "STATIC" into
                       proper idn table entry(ies)

   9 return - put "rtn" into matrix

  10 var_name - put name on temporary list
```

รูป 2.12 ตัวอย่างโปรแกรมในเฟสการวิเคราะห์วากยสัมพันธ์

2.3.3 เฟสการแปล

หน้าที่

- สร้างรูปแบบตัวกลางของภาษาดิบ
- เพิ่มข้อมูลให้กับตารางตัวชี้เฉพาะ

ข้อมูล

1. สแต็ค - เก็บโทเคนทั้งหลายที่ถูกกระจายโดยเฟสที่ผ่านมาแล้ว
2. ตารางตัวชี้เฉพาะ - เพิ่มรายละเอียดของข้อมูลในตารางของแต่ละระเบียนดังรูป 2.13

Name	Base	Scale	Prec.	Stor. Class	Array bound	Struc. infor.	Litr. value	Block infor.	Other	Addr.

รูป 2.13 ระเบียนของตารางตัวชี้เฉพาะ

3. เมตริกซ์ (matrix) - เก็บรูปแบบตัวกลางของภาษาดิบขั้นต้น(primary intermediate form) ของโปรแกรมดิบ ดังตัวอย่างในรูปที่ 2.15 และ 2.16

สมาชิกในเมตริกซ์จะประกอบ

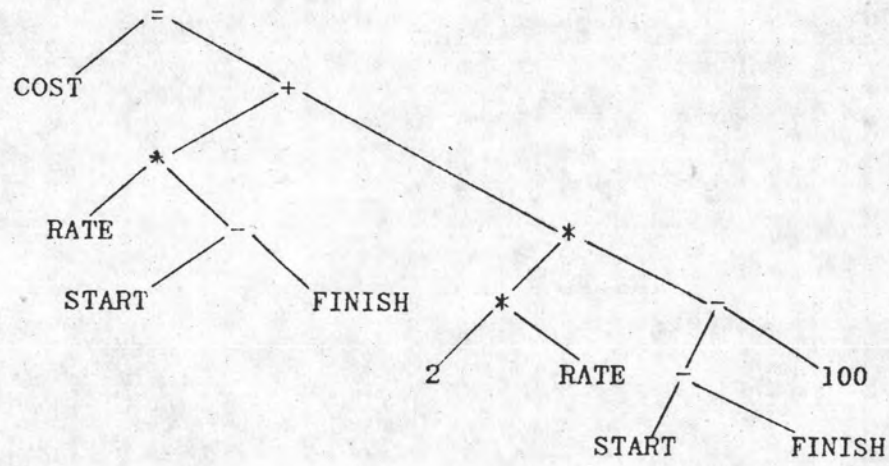
- อาร์กิวเมนต์ (argument)
- สัญลักษณ์เทอร์มินอล หรือ ตัวดำเนินการ (operator)

ตัวดำเนินการ	ตัวถูกกระทำ 1	ตัวถูกกระทำ 2	chaining

รูป 2.14 ระเบียนของเมตริกซ์

COST	=	RATE	+	(	START	-	FINISH	)	+	2	*	RATE	*	(	START	-	FINISH

-	100	)	;



รูป 2.15 Tree - intermediate form of example arithmetic statement

COST	=	RATE	*	(	START	-	FINISH	)	+	2	*	RATE	*	(	START	-	FINISH

-	100	)	;

Matrix line no.	Operator	Operand 1	Operand 2
1	-	START	FINISH
2	*	RATE	M1
3	*	2	RATE
4	-	START	FINISH
5	-	M4	100
6	*	M3	M5
7	+	M2	M6
8	=	COST	M7

รูป 2.16 Matrix for example arithmetic statement

ตัวถูกกระทำของแมททริกซ์เป็นสัญลักษณ์ประเภท 'IDN', 'LIT' หรือ 'TRM' และ 'MTX' | MTX | n | หมายถึง ผลลัพธ์ของระเบียนของแมททริกซ์ตัวที่ n จะชี้ไปยังระเบียนตัวที่ตรงกันในที่เก็บข้อมูลชั่วคราวของตารางตัวชี้เฉพาะ

4. ตารางที่เก็บข้อมูลชั่วคราว (Temporary storage table) - เฟสนี้จะป้อนข้อมูลที่เกี่ยวกับค่าของสัญลักษณ์ MTX เข้าไปเก็บในตารางนี้ ตัวอย่างข้อมูลที่เก็บ ได้แก่ ลักษณะของการคำนวณชั่วคราว (Temporary computation) ซึ่งเป็นผลลัพธ์จากแมททริกซ์ เช่น ชนิดของข้อมูล (data type), ความแน่นอน (precision) ฯลฯ ข้อมูลนี้จะถูกใช้ในเฟสการจัดแบ่งเนื้อที่ในหน่วยความจำ และการสร้างรหัส

MTXN	Base	Scale	Precision	Storage Class	Other	Address

รูป 2.17 ระเบียนของตารางที่เก็บข้อมูลชั่วคราว

การทำงาน

1. ทำหน้าที่เพิ่มหรือลบสัญลักษณ์ต่างๆในสแต็คได้

2. สร้างระเบียนใหม่ในแมททริกซ์ หรือ เพิ่มลักษณะข้อมูลเข้าไปในตารางตัวชี้เฉพาะ โดยเฟสนี้จะต้อง

- เลือกตัวดำเนินการและตัวถูกกระทำที่เหมาะสมและแทรกข้อมูลนี้เข้าไปในแมททริกซ์
- เลือกคุณลักษณะและใส่ลงในตารางตัวชี้เฉพาะ

ตัวอย่าง

ตัวอย่างของการทำงานของเฟสการวิเคราะห์วากยสัมพันธ์ และ เฟสการแปลในขณะที่ทั้งสองเฟสทำการแปล โปรแกรมตัวอย่างจากรูป 2.4 และเพื่อให้ตัวอย่างนี้เข้าใจได้

ง่ายขึ้น ได้สร้างตัวแปลภาษาซึ่งจะดูแลเฉพาะเซทย่อยของ PL/I (MINI-PL/I) เพียงส่วนเดียวเท่านั้น

จะให้คำจำกัดความของ MINI-PL/I ได้ดังต่อไปนี้ จะมีประโยคทั้งหมดเพียง 5 ชนิดเท่านั้น รูปแบบของประโยคเหล่านี้ได้แก่

(1) label : PROCEDURE (list of arguments);

(2) DECLARE followed by a string AB where

A) <idn> or (list of <idn>s)

B) FIXED BINARY (precision) STATIC

Where AB's phrases can be further seperated by commas and the statement is ended by a;

(3) <idn> = legal arithmetic expression;

(4) RETURN (argument);

(5) END;

ข้อมูลใดที่ไม่สอดคล้องกับข้อกำหนดถือว่าไม่ถูกต้อง(illegal) MINI-PL/I โปรแกรมที่ถูกต้องจะต้องเริ่มด้วยประโยคในฟอร์ม (1) และจบด้วยประโยค (5) และมีเพียงประโยค (2) - (4) อยู่ในระหว่างกลางเท่านั้น

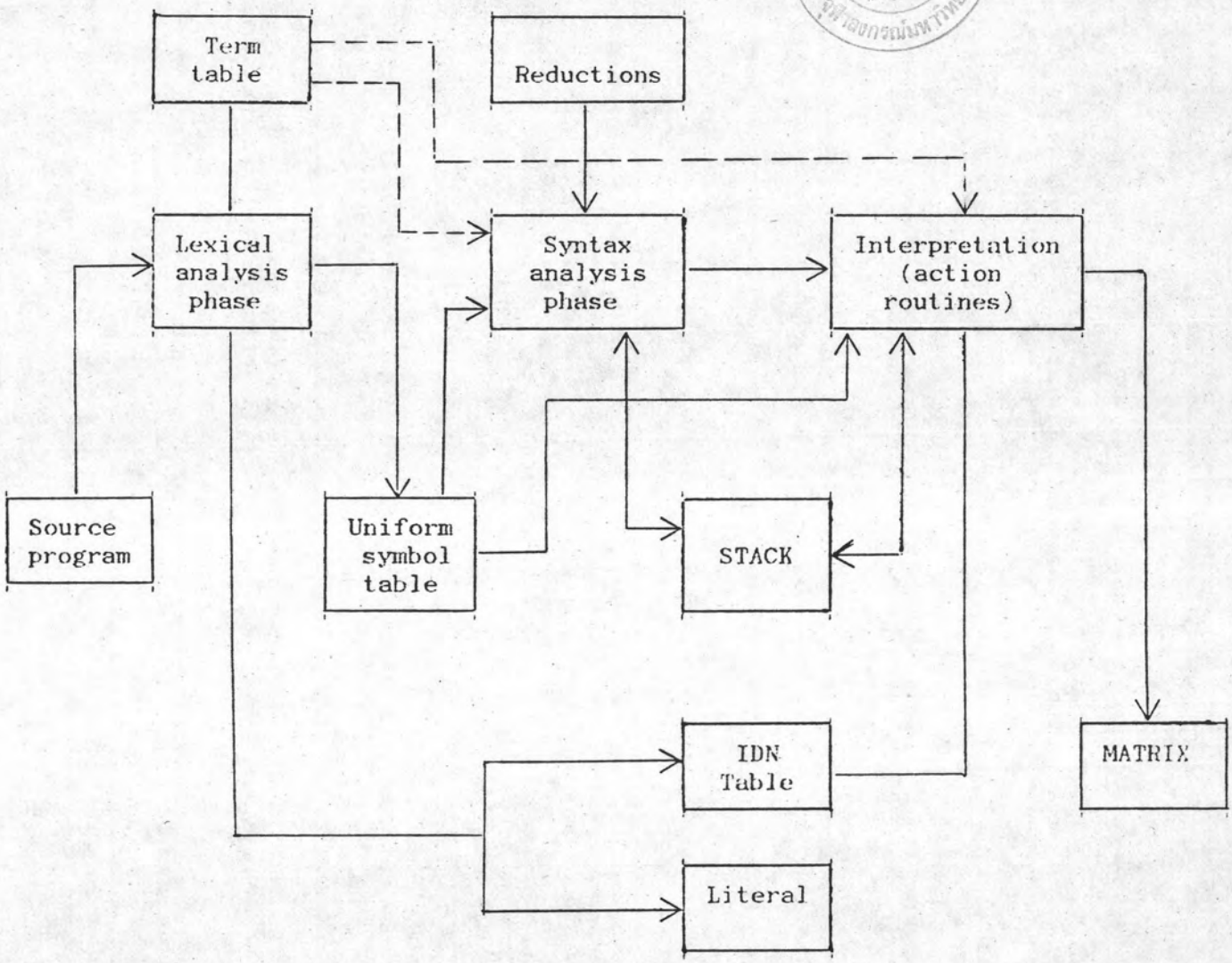
เป็นเรื่องที่ค่อนข้างยากที่จะเขียนตัวแปลภาษาจากการให้คำจำกัดความเป็นคำพูด (verbal definition) เพราะไม่เฉพาะเจาะจงพอ (imprecise) ตัวอย่างเช่น ถ้าประโยค PROCEDURE ไม่มี arguments ใดๆ จะถือว่าถูกต้อง (legal) ใช้ได้หรือไม่ ดังนั้นจึงกำหนด MINI-PL/I ขึ้นอย่างเป็นทางการในแบบของ BNF ดังรูป 2.18

```
<procedure>         :: = <proc> <body> <end>                             Procedure
<proc>              :: = <idn> : PROCEDURE (<var>);                      Procedure statement
<end>               :: = END;                                            END statement
<var>               :: = <idn> ¦ <idn> , <var>
<body>              :: = <stmnt> ¦ <stmnt> <body>                        Body of a procedure
<stmnt>             :: = <assign> ¦ <return> ¦ <dc¦>                     Statement
<assign>            :: = <idn> = <enp> ;                                 Assignment
<exp>               :: = <lit> ¦ <idn> ¦ <exp> <op> <exp> ¦ (<exp>)      Statement
<return>            :: = RETURN (<idn>);                                 Return statement
<dc¦>               :: = DECLARE <dc¦ element>;
<dc¦ element>       :: = <var list> <attribute list> ¦ <var list>
                         <attribute list>, <dc¦ element>                 Declare statement
<attribute list>    :: = FIXED BINARY (<lit>) STATIC
<var list>          :: = <idn> ¦ (<var>)
Terminal classes :       (for syntax phase - these are defined in the lexical phase)
                                                            30
<idn>               ::=  <letter> ¦ <letter> <letter-digit>
                                                            1
                               15
<lit>               :: = <digit>
                               1
<op>                :: = + ¦ - ¦ * ¦ - ¦ * * ¦ =
```

รูปที่ 2.18 ไวยากรณ์ BNF ของ MINI-PL/I

Reduction # of Fig. 2.11	Match?	Action routine-action	Matrix line #	Matrix			Stack
1	Yes	none					MAIN PROCEDURE
2	Yes	bgn_proc-(note MAIN is a label in IDN table) enter procedure name into matrix	M1	.proclbl	MAIN		PROC (RATE,START
3	Yes	arg-enter first argument into matrix	M2	.arg	RATE		PROC (START,FINISH
4	Yes	arg-enter second argument into matrix	M3	.arg	START		PROC (FINISH);
4	No						
5	Yes	arg-enter third argument into matrix	M4	.arg	FINISH		PROC DECLARE (
7	Yes	none					PROC (COST,
14	No						
15	Yes	var_name-put COST on list of entries to be updated					PROC (RATE,
15	Yes	var_name-put RATE on list of entries to be updated					PROC (START,
15	Yes	var_name-put START on list of entries to be updated					PROC (FINISH)
15	No						
17	Yes	assgn-enter=COST expression value into matrix	M12	=	COST	MTX11	PROC RETURN (
7	No						
8	Yes	return-enter .rtn onto matrix	M13	.rtn			PROC (COST):
5	Yes		M14	.arg	COST		PROC END;
16,9	No						
10	Yes	end_proc-enter .procend into matrix	M15	.procend			End of internal symbol table
11	Yes	none					PROC FINISH FIXED BINARY
14	Yes	var_name-put FINISH on list of entries to be updated					
14 cont'd		base_scale-set scale = FIXED and base = BINARY for all entries in IDN_TBL with flags turned on					PROC COST =
13	No						
2	Yes	op_prec-parse arithmetic expression (until ;) and generate the proper matrix entries, using the rules of operator precedence. This will generate a right and left precedence associated with each operator in the terminal table.	M5 " " " " " " "21	- + + - - * * (	START RATE 2 START MTX8 MTX7 MTX6	FINISH MTX5 RATE FINISH JOB MTX9 MTX10	PROC COST = MTX11;

รูป 2.19 ขั้นตอนต่างๆของเฟสการวิเคราะห์วากยสัมพันธ์และการแปล



รูป 2.19ก ผลกระทบซึ่งกันและกันของเฟสการวิเคราะห์เลคซิเคล,
การวิเคราะห์วากยสัมพันธ์และการแปล

2.3.4 เฟสออปติไมเซชั่น

หน้าที่

ส่วนการตัดสินว่าจะใช้ Optimization อันใดอันหนึ่งในตัวแปลภาษาหรือไม่นั้น เป็นเรื่องจำเป็นที่จะต้องชั่งน้ำหนักระหว่างผลตอบแทนที่จะได้ในแง่การเพิ่มประสิทธิภาพสำหรับโปรแกรมเป้าหมาย เทียบกับต้นทุนที่จะสูงขึ้นเนื่องจากเวลาที่ต้องใช้ในงานการแปล และและความซับซ้อนที่เพิ่มขึ้น ผลของการตัดสินใจนั้นส่วนใหญ่จะขึ้นอยู่กับสถานการณ์ในช่วงที่กำลังสร้างตัวแปลนั้นอยู่ รวมทั้งจุดมุ่งหมายในการใช้งานของตัวแปลนั้นด้วย

ข้อมูล

1. เมตริกซ์ - เป็นข้อมูลตัวที่ใช้ในเฟสนี้ เพื่อลด (elimination) หรือเพิ่มระเบียนเข้าไปในเมตริกซ์ และเพิ่มดัชนีไปข้างหน้าและถอยหลัง (forward และ backward pointer) เข้าไปในแต่ละระเบียน วิธีนี้จะช่วยให้สามารถหลีกเลี่ยงความจำเป็นในการจัดลำดับ และการจองระเบียนเมื่อมีการเพิ่มหรือลบระเบียนอันใดอันหนึ่งออกไป สำหรับดัชนีไปข้างหน้านั้นจะช่วยให้เฟสสร้างรหัสอ่านเมตริกซ์ในลำดับที่ถูกต้อง ส่วนดัชนีไปข้างหลังจะช่วยในการจัดลำดับถอยหลังของเมตริกซ์ ซึ่งจำเป็นต้องใช้ในออกไปนอก DO_LOOPS.

ระเบียนของเมตริกซ์จะมีลักษณะดังนี้:-

Operator	Operand 1	Operand 2	Forward pointer	Backward pointer

ตำแหน่งของดัชนีไปข้างหน้านั้นคือ ดัชนีของระเบียนต่อไปในโปรแกรม และดัชนีถอยหลังนั้นเป็นดัชนีของระเบียนที่ผ่านมาแล้ว

2. ตารางตัวชี้เฉพาะ - เพื่อลบส่วนที่เก็บข้อมูลชั่วคราวที่ไม่จำเป็นออก และ

เพื่อรับข้อมูลเกี่ยวกับตัวชี้เฉพาะ

3. ตารางค่าคงที่ - มีคำศัพท์บางประเภทใหม่ที่อาจจะสร้างขึ้นโดยเฟสนี้

การทำงาน

เทคนิคการออปติไมเซชั่นมี 4 แบบ ดังนี้:-

1. การกำจัดนิพจน์ที่ซ้ำกัน (Elimination of common expression) :

การลบระเบียนของแมททริกซ์ที่ซ้ำกันออกนั้นสามารถจะช่วยให้ได้โปรแกรมผลที่มีประสิทธิภาพมากขึ้น นิพจน์ย่อยธรรมดานั้นจะต้องเหมือนกันและอยู่ในประโยคเดียวกัน ตัวอย่างเช่น ในโปรแกรมต่อไปนี้ เราไม่สามารถกำจัดระเบียนของเมตริกซ์ที่ซ้ำกัน ซึ่งอยู่ใน START - FINISH ครั้งที่สองได้ เพราะค่าของ START นั้นเปลี่ยนไปในระหว่างการประเมินผลของนิพจน์ทั้งสอง

```
COST    =   RATE (START-FINISH)
START   =   START/2 ;
COST 2  =   RATE  (START-FINISH-100) ;
```

ในบางภาษาเช่น PL/I ซึ่งการขัดจังหวะอาจจะเปลี่ยนค่าของตัวแปรได้ในทุกขณะ การกำจัดนิพจน์ที่ซ้ำกันจะทำได้ก็ต่อเมื่อผู้เขียนโปรแกรมได้กำหนดไว้ว่าจะไม่มีการขัดจังหวะ ตัวใดเลยที่จะเปลี่ยนค่าของตัวแปรใดๆ ตัวอย่างเช่น ใน PL/I นั้น โปรแกรมเมอร์อาจจะอ้างถึง ON OVERFLOW (FINISH = 0) เช่นนี้สามารถจะทำให้ นิพจน์ย่อยของเราสองอันไม่เท่ากัน แม้ว่าทั้งสองจะเกิดขึ้นในประโยคเดียวกัน

ขั้นตอนในการกำจัด

1. จัดวาง เมตริกซ์ไว้ในฟอร์มแบบหนึ่งเพื่อให้สังเกตได้ถึงนิพจน์ย่อย
2. ตรวจหา นิพจน์ย่อยสองอันซึ่งเท่ากัน
3. กำจัด (ลบ) ออกตัวหนึ่ง
4. เปลี่ยนแปลง เมตริกซ์ที่เหลือเพื่อให้เห็นการกำจัดออกของเอ็นทรีนี้

เมื่อได้แก้ไขเอ็นทรีไว้ในประโยคที่ให้มาแล้ว วิธีนี้อาจจะต้องการตรวจสอบดูว่าการแก้ไขนั้นก่อให้เกิดนิพจน์ซ้ำกันเพิ่มขึ้นหรือไม่

ตัวอย่างเช่นรูป 2.19 นั้น แสดงให้เห็นเมตริกซ์ที่สร้างขึ้นสำหรับ Statement ต่างๆในโปรแกรม และขั้นตอนที่เป็นไปได้สำหรับการกำจัดนิพจน์ซ้ำกันนั้นจะทำงานดังนี้

1. เรียงลำดับ ตัวถูกกระทำของตัวดำเนินการ และ "*" ตามตัวอักษร ตรวจสอบเอ็นทรี แต่ละตัว และเปลี่ยนเอ็นทรี พวกที่อ่านถอยหลัง (backward) (มีเพียง M3 และ M4 เท่านั้นที่ต้องมีการสลับที่)

2. พิจารณาขอบเขตของ Statement ในกรณีนี้เพียงแต่แยกออกโดยใช้ ตัวดำเนินการ

3. หานิพจน์ย่อยที่ซ้ำกันเปรียบเทียบ - M2 กับ M3, M4, M5 M3 เป็นคู่เดียวที่มีจึงให้ลบ M3 โดยการเปลี่ยนดัชนี ของ M2 และ M4

4. โดยเปลี่ยนแปลงแมททริกซ์ที่เหลือเพียงให้เห็นเอ็นทรีที่ถูกลบออกไปตัวอย่างเช่น เปลี่ยนตัวอ้างอิงของ M3 ทั้งหมดเป็น M2

5. ตรวจสอบเมตริกซ์อีกครั้ง เพื่อสำรวจ นิพจน์ย่อยที่ซ้ำกัน ที่อาจจะเกิดขึ้นได้ ตรวจซ้ำ 1 ถึง 5 จนกระทั่งไม่มีการเปลี่ยนแปลงใดๆเกิดขึ้น

6. กำจัด MTX entries ใดๆที่ไม่จำเป็นอีกต่อไปออกจากตาราง temporary storage

Source code

```
B = A
A = C * D * (D * C + B)
```

Matrix before optimization

M1	=	B	A	0	2
M2	*	C	D	1	3
M3	*	D	C	2	4
M4	+	M3	B	3	5
M5	*	M2	M4	4	6
M6	=	A	M5	5	7

Matrix after step 1 and 2

M1	=	B	A	0	2	Statement
M2	*	C	D	1	3	
M3	*	C	D	2	4	
M4	+	B	M3	3	5	Statement
M5	*	M2	M4	4	6	
M6	=	A	M5	5	?	

Matrix after step 4

M1	=	B	A	0	2
M2	*	C	D	1	4
M3	*	C	D	2	4
M4	+	B	M2	2	5
M5	*	M2	M4	4	6
M6	=	A	M5	5	7

รูป 2.19 ตัวอย่างการกำจัดนิพจน์ย่อยที่ซ้ำกัน

2. การคำนวณในระหว่างเวลาทำการแปล (Compile time compute) ในการคำนวณนั้นจะต้องเกี่ยวพันกับเวลาของการแปล เพื่อให้ประหยัดทั้งเนื้อที่ และเวลาทำงานของโปรแกรมเป้าหมาย และจะเป็นการช่วยเหลืออย่างมาก ถ้ามีการคำนวณทางคณิตศาสตร์เช่นนั้นภายใน Loop ตัวอย่างเช่น นิพจน์

```
A = 2 * 276 / 92 * B
```

ตัวแปลภาษาสามารถทำการคูณและหารอย่างที่ระบุไว้ และแทนที่ 6 * B ด้วยประโยคอันเดิม เช่นนี้จะช่วยให้สามารถลบแมททริกซ์เอ็นทรีออกได้สองอัน และสามารถลดจำนวนของ รหัสภาษาเครื่อง ที่ถูกสร้างขึ้น ตัวอย่างดังรูป 2.20

Before

Matrix	optimization		
M1	*	2	276
M2	/	M1	92
M3	*	M2	B
M4	=	A	M3

After

Matrix	optimization		
M1			
M2			
M3	*	6	B
M4	=	A	M3

รูป 2.20 Compile-time-compute

การทำงานสำหรับเทคนิคแบบนี้จะทำงานโดยเริ่มที่การสำรวจแมททริกซ์ค้นหา ตัวดำเนินการ ซึ่งมีตัวถูกกระทำ ทั้งคู่เป็นค่าคงที่ เมื่อใดที่พบการดำเนินการแบบนั้น มันจะทำการประเมินผลและสร้างค่าคงที่ตัวใหม่ขึ้นลบบรรทัดเก่า และแทนที่ด้วยค่าคงที่ใหม่ด้วย และจากนั้นจะทำการสำรวจแมททริกซ์ต่อไปอีกเพื่อหาการคำนวณซึ่งอาจจะเกิดขึ้นอีก

3. การออปติมเซชั่นของนิพจน์ (Boolean expression optimization) เราอาจใช้คุณสมบัติของนิพจน์ตรรกะ (Boolean expression) เพื่อลดการคำนวณให้สั้นลง ตัวอย่างเช่น ในประโยค IF a or b or c, THEN... เมื่อ a,b,c เป็นนิพจน์แทนที่จะทำการสร้างโค็ต ซึ่งจะทำหน้าที่ทดสอบนิพจน์ a,b,c ทีละตัวเสมอ ไปวิธีสร้างโค็ตแบบนี้คือ ถ้า a ถูกคำนวณว่าเป็นจริงแล้ว b or c ไม่ต้องถูกคำนวณ และสำหรับ b ก็เช่นเดียวกัน

4. การย้ายการคำนวณที่ไม่มีการเปลี่ยนแปลงออกนอก loops (Move invariant computation outside of loops) ถ้าการคำนวณภายใน loop ขึ้นอยู่กับตัวแปรซึ่งไม่มีการเปลี่ยนแปลงภายใน loop การคำนวณนั้นอาจจะได้รับการเคลื่อนออกมานอก loop

การกระทำแบบนี้จะต้องมีการจัดลำดับบางส่วนของเมตริกซ์เสียใหม่ ซึ่งจะมีปัญหาสามประการ ซึ่งจำเป็นจะต้องได้รับการแก้ไขในการทำงานของเทคนิคแบบนี้

1. เลือกการคำนวณที่ไม่มีการเปลี่ยนแปลง (Recognition of invariant computations)

ตัวอย่างเช่น ในโปรแกรมต่อไปนี้อะไรคือส่วนที่เราควรจะเลื่อนออกไปนอก loop?

```
          DO    I  =  1 TO 10 ;
OUTLOOP:  -
          DO    J  =  1 TO 10 ;
LA :      A  =  10
LB:       B  =  Y(1+2) ;
LC:       C  =  C + 10 ;
          END ;
          END ;
```

ประโยค LA จะให้ผลลัพธ์เช่นเดิมเสมอ ดังนั้นจึงสามารถเลื่อนออกมานอก loop ได้ นอกจากนั้นจะต้องตรวจสอบว่าค่าของ Y นั้น เปลี่ยนแปลงหรือไม่เวลาที่อยู่ใน inner loop ถ้าไม่เปลี่ยน ข้อนี้ก็จัดอยู่ในประเภทการคำนวณที่ไม่เปลี่ยนแปลง สามารถย้ายออกมานอก loop ได้ LC เป็นการคำนวณที่เกี่ยวข้องกับ C ซึ่งเป็น variable ที่มีการเปลี่ยนแปลงภายใน inner loop ดังนั้นจึงไม่สามารถนำออกไปนอก loop ได้ ถ้ามี statement ใดๆ (ภายใน loop) อยู่ก่อนหน้า LA และอ้างอิงถึง A หรือ B เราจะไม่

สามารถเคลื่อนย้าย statement ทั้งหมดออกจาก inner loop ได้

2. หาที่สำหรับการเคลื่อนย้ายการคำนวณที่ไม่เปลี่ยนแปลง (Invariant computation) :

จะต้องย้ายการคำนวณนี้ (computation) ไปยังตำแหน่งที่อยู่ข้างหน้าของ loop นั้น ดังนั้นจึงต้องหาจุดเริ่มต้นของ DO_LOOPS โดยการตรวจสอบเมตริกซ์ ในเฟสการแปล จะต้องจัดวางระเบียนของเมตริกซ์ที่สร้างไว้โดยเฉพาะหนึ่งตัว ตรงจุดเริ่มต้นของ DO_LOOP ทุกประการ และต้องป้อนดัชนีถอยหลังเข้าไปในเมตริกซ์ เพื่อให้เฟสนี้สามารถทำงานย้อนกลับไปยัง loop ภายนอกทั้งหลายได้

3. การย้ายการคำนวณที่ไม่เปลี่ยนแปลง (Moving the invariant computation)

ใช้วิธีการ chaining แบบต่างๆ เช่นที่ได้กล่าวแล้วในข้อ (1) สามารถลบการคำนวณที่ไม่เปลี่ยนแปลงออกจากตำแหน่งที่อยู่เดิม แล้วแทรกเข้าไปไว้ในตำแหน่งที่เหมาะสม

2.3.5 การจัดแบ่งเนื้อที่ในหน่วยความจำ

หน้าที่

จุดประสงค์ของเฟสนี้ได้แก่

1. จัดแบ่ง (assign) เนื้อที่สำหรับตัวแปรทั้งหมดที่ถูกอ้างถึงในโปรแกรมดิบ

2. จัดแบ่งเนื้อที่สำหรับผลลัพธ์ชั่วคราว (intermediate results) เช่น ผลลัพธ์ของเมตริกซ์ เนื้อที่เหล่านี้จะถูกสำรองไว้โดยเฟสการแปล

3. จัดแบ่งเนื้อที่สำหรับค่าคงที่

4. ตรวจสอบเนื้อที่ต่างๆที่ได้รับการจัดสรรเพื่อใช้งาน และมีการกำหนดจุดเริ่มต้นของตารางข้อมูลต่างๆ

และเพื่อให้สามารถทำงานข้างต้นได้ จะต้องมีการเพิ่มข้อมูลสำหรับเฟสสร้างรหัสเข้าไปในตารางตัวชี้เฉพาะ, ตารางค่าคงที่ และในเมตริกซ์

ข้อมูล

1. ตารางตัวชี้เฉพาะ - สร้างขึ้นโดยเฟสเลคซิเคิล และได้รับการเพิ่มเติมข้อมูลเกี่ยวกับคุณลักษณะต่างๆ ทั้งหลายโดยเฟสการแปล ส่วนเฟสนี้จะกำหนดตำแหน่งที่อยู่ให้กับค่าคงที่ทั้งหมดที่ใช้แทนความหมายของข้อมูล โดยอีกสองเฟสต่อมา (code generation และ assembly phase) จะใช้ข้อมูลนี้ สำหรับการเข้าไปสู่ค่าคงที่นั้นๆ

Name	Base	Scale	Precision	Storage class	Array bounds	Structure into	Other	Address

รูป 2.21 ระเบียนของตารางค่าคงที่

2. ตารางที่เก็บข้อมูลชั่วคราว - (เป็นส่วนหนึ่งของ identifier table) สร้างขึ้นโดยเฟสการแปล เพื่อใช้ในการอธิบายผลลัพธ์ชั่วคราวของการคำนวณในเมตริกซ์ (temporary results of computations) ตารางอันนี้จะใช้เป็นส่วนหนึ่งของตารางตัวชี้เฉพาะได้ เพราะข้อมูลส่วนใหญ่จะออกมาในรูปแบบ (format) เดียวกัน และเนื้อที่ทั้งหมดในตารางนี้จะถูกใช้งาน

Mi	Base	Scale	Precision	Automatic	Other	Address

รูป 2.22 ระเบียนของตารางที่เก็บข้อมูลชั่วคราว

3. ตารางค่าคงที่ - จะกำหนดตำแหน่งที่อยู่ให้กับค่าคงที่ทั้งหมด

4. เมตริกซ์ - เฟสนี้จะป้อนข้อมูลเข้าไว้ในเมตริกซ์เพื่อให้การสร้างรหัสเตรียมเนื้อที่ได้เพียงพอสำหรับตัวชี้เฉพาะ ที่เก็บข้อมูลชั่วคราว และค่าคงที่ ดังรูป 2.23

Storage class	Size	
operator	operand	operand

Initialize	Identifier or literal	
operator	operand	operand

รูป 2.23 ระเบียนต่างๆของเมตริกซ์ที่สร้างโดยเฟสนี้

การทำงาน

จะ scan ผ่านไปทั่วตารางตัวชี้เฉพาะ และแจกแจงตำแหน่งที่อยู่ สำหรับแต่ละระเบียนโดยใช้ตัวนับ (location counter) เริ่มต้นที่ศูนย์ เพื่อจะได้ติดตามและตรวจสอบได้ว่ามีการจัดแบ่งเนื้อที่ไปมากน้อยเพียงใดแล้ว จะทำงานตามขั้นตอน 4 อย่างต่อไปนี้ :-

1. เปลี่ยนแปลงข้อมูลในตัวนับ โดยวิธี boundary alignment
2. กำหนดค่าปัจจุบันของตัวนับให้เป็นตำแหน่งที่อยู่ของตัวแปรนั้น
3. คำนวณหาความยาวของเนื้อที่ ที่ตัวแปรนั้นๆต้องการ (โดยการตรวจสอบคุณลักษณะของมัน)

4. เปลี่ยนแปลงส่วนของตัวนับ (location counter) โดยการเพิ่มความยาวในข้อ 3 เข้าไป

เมื่อได้มีการกำหนดตำแหน่งที่อยู่ต่างๆ ที่เกี่ยวข้องให้กับตัวชี้เฉพาะทั้งหมดที่ต้องการ หน่วยความจำคงที่ (static storage) เฟสนี้จะสร้าง ระเบียนของเมตริกซ์ขึ้นในลักษณะต่อไปนี้

STATIC	Size	

ดังนั้นการสร้างรหัสจะสามารถสร้างรหัส (generate) ที่พอเหมาะสำหรับเนื้อที่นั้นๆ และสำหรับตัวแปรแต่ละตัวที่ต้องการค่าเริ่มต้น เฟสนี้จะสร้างระเบียนต่อไปนี้ขึ้น :

Initialize	Variable	

ระเบียนนี้จะช่วยให้เฟสการสร้างรหัสสามารถใส่ ค่าเริ่มต้น ซึ่งโปรแกรม เก็บ (save) ไว้ในตารางตัวชี้เฉพาะนั้นเข้าไปยังตำแหน่งที่เหมาะสมได้

วิธีการ scan ผ่านตารางตัวชี้เฉพาะ เช่นเดียวกันนี้จะถูกใช้อีกครั้งสำหรับการสร้างหน่วยความจำออโตเมติค (automatic storage) และหน่วยความจำควบคุม (control storage) การ scan นี้จะเข้าไป (entry) ยัง ตำแหน่งที่อยู่ที่เกี่ยวข้องเพื่อหาเอ็นทรี แต่ละตัว ทั้งเอ็นทรีออโตเมติค (automatic entry) และ เอ็นทรีควบคุม (controlled entry) จะต้องสร้างขึ้นในเมตริกซ์ เช่นเดียวกับ การสร้างรหัส จะใช้

ตำแหน่งที่อยู่ของเอ็นทรี ที่เกี่ยวข้องสำหรับการสร้าง คือตำแหน่งที่อยู่ของคำสั่ง จะไม่มีการสร้างหน่วยความจำใดๆ ในช่วงการแปล ทั้งสำหรับหน่วยความจำออโตเมติคและหน่วยความจำควบคุม แต่เอ็นทรีของเมตริกซ์จะทำให้เกิดมีการสร้างรหัส เพื่อบรรจุไว้ในหน่วยความจำนี้โดยอัตโนมัติในช่วงของการทำงานจริง เช่น เมื่อมีรหัสที่ได้สร้างขึ้นถูกนำไปใช้งาน (is executed) มันจะถูกบรรจุลงในหน่วยความจำออโตเมติคทันที

ในตารางค่าคงที่ ก็จะมีการ scan แบบเดียวกัน และจะมีการแจกแจงตำแหน่งที่อยู่สำหรับค่าคงที่แต่ละตัว และมีการสร้างระเบียนของเมตริกซ์ แบบ | LIT | Size | | ขึ้น เฟสการสร้างรหัสจะสร้างหน่วยความจำสำหรับค่าคงที่ทุกตัว ในหน่วยความจำคงที่ และหน่วยความจำเริ่มต้น ด้วยมูลค่าของค่าคงที่

ในส่วนของหน่วยความจำชั่วคราว (temporary storage) นั้นจะใช้วิธีการที่แตกต่างออกไป เพราะประโยคของภาษาดิบแต่ละอันนั้น อาจจะต้องการใช้หน่วยความจำชั่วคราว (ในส่วนของ intermediate matrix result area) ของประโยคของภาษาดิบอันที่อยู่ก่อนหน้าอีกก็เป็นได้ จะมีการคำนวณหาจำนวนของหน่วยความจำชั่วคราว ที่แต่ละประโยคต้องการใช้ ประโยค (statement) ตัวที่ต้องการใช้ (required) หน่วยความจำชั่วคราวในจำนวนสูงสุด จะเป็นตัวตัดสินว่าจะต้องการใช้หน่วยความจำชั่วคราว เป็นจำนวนเท่าใด ทั้งในโปรแกรมจะมีการสร้างระเบียนของเมตริกซ์ ในแบบต่อไปนี้ | AUTOMATIC | | Size | | วิธีนี้จะช่วยให้ เฟสการสร้างรหัสสามารถสร้างรหัส ซึ่งจะใช้สร้างจำนวนที่เหมาะสมของหน่วยความจำชั่วคราว หน่วยความจำชั่วคราวนั้นจะเป็นแบบอัตโนมัติ เพราะมันจะถูกอ้างถึงโดยโปรแกรมดิบอย่างเดียวเท่านั้น และจะถูกใช้เฉพาะในขณะที่โปรแกรมดิบกำลังทำงาน (active) เท่านั้น ตัวอย่างดังรูป 2.24

Identifier table

	Name	Base	Scale	Precision	Storage class	Other	Address
1	WCM	proc name					
2	RATE	Binary	Fixed	31	Static		0
3	START	Binary	Fixed	31	Static		4
4	FINISH	Binary	Fixed	31	Static		8
5	COST	Binary	Fixed	31	Static		12
	MTX5	Binary	Fixed	31	Automatic		0
	MTX6	Binary	Fixed	31	Automatic		4
	MTX7	Binary	Fixed	31	Automatic		8
	MTX8	Binary	Fixed	31	Automatic		12
	MTX9	Binary	Fixed	31	Automatic		16
	MTX10	Binary	Fixed	31	Automatic		20
	MTX11	Binary	Fixed	31	Automatic		24

Literal table

Literal	Base	Scale	Precision	Other	Address
31	Decimal	Fixed	2		
2	Decimal	Fixed	1		0
100	Decimal	Fixed	3		1

Matrix

Operator	Operand 1	Operand 2	Chaining
STATIC	16		
AUTOMATIC	28		
LITERALS	3		

รูป 2.24 ตารางตัวชี้เฉพาะ, ค่าคงที่, เมตริกซ์ หลังจากการจัดแบ่งหน่วยความจำ

2.3.6 การสร้างรหัส

หน้าที่

- ผลิตรหัสภาษาเครื่องที่เหมาะสม

ข้อมูล

1. เมตริกซ์ - เฟสนี้จะตรวจสอบระเบียนแต่ละตัวกับแหล่งรวมรหัสมาตรฐาน (code pattern data base) เพื่อดึงรูปแบบ (pattern) นั้นๆ มาใช้งาน

2. ตารางตัวชี้เฉพาะ, ตารางค่าคงที่ - ใช้เพื่อดึงข้อมูลบางอย่างเช่น ชนิดของข้อมูล และตำแหน่งของตัวแปร มาช่วยในการสร้างรหัสที่เหมาะสมในตำแหน่งที่ถูกต้อง

3. แหล่งรวมรหัสมาตรฐาน (code production) - เป็นข้อมูลถาวรที่จะกำหนดการทำงานของการสร้างรหัสที่เป็นไปได้ทั้งหมด ดังรูป 2.25

การทำงาน

เปลี่ยนรูปแบบตัวกลางให้เป็นรหัสภาษาเครื่อง โดยการดึงแมกโคร (macro) มากระจาย ซึ่งจะทำให้เกิดมีตัวดำเนินการ LOAD และ STORE ที่ซ้ำซ้อนกัน เป็นจำนวนมาก ดังนั้นเฟสนี้จะต้องลดตัวดำเนินการ LOAD และ STORE ที่ซ้ำซ้อนกัน ดังตัวอย่างในรูป 2.26 ถึง 2.28

Standard code definitions for - , * , + , =

```
-   L     1,&OPERAND1
    S     1,&OPERAND2
    ST    1,M&N
*   L     1,&OPERAND1
    M     0,&OPERAND2
    ST    1,M&N
+   L     1,&OPERAND1
    A     1,&OPERAND2
    ST    1,M&N
=   L     1,&OPERAND2
    ST    1,&OPERAND1
```

&OPERAND1 : ตัวถูกกระทำตัวที่ 1 ของเมตริกซ์

&OPERAND2 : ตัวถูกกระทำตัวที่ 2 ของเมตริกซ์

&N : บรรทัดที่ของเมตริกซ์

รูป 2.25 ตัวอย่างกฎการแปลงรหัสสำหรับตัวดำเนินการ - , * , + , =

เมตริกซ์				รหัสที่ถูกสร้าง	
1	-	START	FINISH	L	1,START
				S	1,FINISH
				ST	1,M1
2	*	RATE	M1	L	1,RATE
				M	0,M1
				ST	1,M2
3	*	2	RATE	L	1,=F'2'
				M	0,RATE
				ST	1,M3
4	-	START	FINISH	L	1,START
				S	1,FINISH
				ST	1,M4
5	-	M4	100	L	1,M4
				S	1,=F'100'
				ST	1,M5
6	*	M3	M5	L	1,M3
				M	0,M5
				ST	1,M6
7	+	M2	M6	L	1,M2
				A	1,M6
				ST	1,M7
8	=	COST	M7	L	1,M7
				ST	1,COST

รูป 2.26 ตัวอย่างการสร้างรหัส

เมตริกซ์ที่มีนิพจน์ซ้ำๆ

1	-	START	FINISH
2	*	RATE	M1
3	*	2	RATE
4	-	START	FINISH
5	-	M4	100
6	*	M3	M5
7	+	M2	M6
8	=	COST	M7

เมตริกซ์หลังจากการลดนิพจน์ซ้ำๆ

1	-	START	FINISH
2	*	RATE	M1
3	*	2	RATE
4			
5	-	M1	100
6	*	M3	M5
7	*	M2	M6
8	=	COST	M7

รูป 2.27 ตัวอย่างการลดนิพจน์ที่ซ้ำๆ

เมตริกซ์		การลดครั้งแรก		รหัสที่ได้		
1 - START	FINISH	L	1,START	L	1,START	
		S	1,FINISH	S	1,FINISH	M1 -> R1
		ST	1,M1			
2 * RATE	M1	L	1,RATE	L	3,RATE	
		M	0,M1	MR	2,1	M2 -> R3
		ST	1,M2			
3 * 2	RATE	L	1,=F'2'	L	5,=F'2'	
		M	0,RATE	M	4,RATE	M3 -> R5
		ST	1,M3			
4						
5 - M1	100	L	1,M1			
		S	1,=F'100'	S	1,=F'100'	M5 -> R1
		ST	1,M5			
6 * M3	M5	L	1,M3	LR	7,5	
		M	0,M5	MR	6,1	M6 -> R7
		ST	1,M6			
7 + M2	M6	L	1,M2			
		A	1,M6	AR	3,7	M7 -> R3
		ST	1,M7			
8 = M7	COST	L	1,M7			
		ST	1,COST	ST	3,COST	
		(80 bytes)		(36 bytes)		

รูป 2.28 การสร้างรหัสให้มีประสิทธิภาพ