204433 การแปลภาษาโปรแกรม

บทนำ

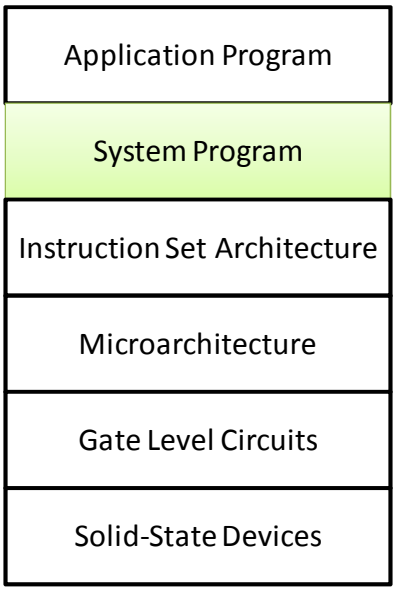
System Program

ก่อนที่จะมาถึงวิชานี้ เราได้เรียนรู้สาระสำคัญของการโปรแกรมในระดับล่างที่สุด นั่นคือการโปรแกรมภาษาแอสเซมบลีว่ามีปฏิบัติการหลักๆเพียงสามอย่างเท่านั้นนั่นคือ

- การย้ายข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่ง
- การคำนวณทางตรรกและคณิตศาสตร์พื้นฐาน
- การกระโดดข้ามการทำงานจากคำสั่งหนึ่งไปยังคำสั่งที่ไม่จำเป็นต้องอยู่ติดกัน

และเราได้รู้ซึ้งมาแล้วว่าโปรแกรมต่างๆที่มีอยู่นั้น สามารถเขียนโดยผสมผสานเพียงปฏิบัติการสามอย่างนี้เข้าด้วยกัน สำหรับนิสิตที่แม่นเรื่องแอสเซมบลีอาจจะบอกว่าภาษาเครื่องต่างหากที่อยู่ในระดับล่างที่สุด แต่นิสิตได้เรียนรู้ (หรือกำลังจะเรียนรู้) ในวิชาซอฟต์แวร์ระบบมาแล้วว่าคำสั่งใดคำสั่งหนึ่งในภาษาเครื่องนั้นก็คือคำสั่งภาษาแอสเซมบลี "ถอดหน้ากาก " ออกจนเหลือเพียงรูปแบบบิทศูนย์กับหนึ่งเท่านั้น การจับคู่จากคำสั่งใด คำสั่งหนึ่ง ในภาษาเครื่องไปยังคำสั่งเดียวกันในภาษาแอสเซมบลีนั้นแทบจะจับคู่กันได้แบบหนึ่งต่อหนึ่ง

พิจารณาแผนผังลำดับชั้นของระบบคอมพิวเตอร์ดังแสดงด้านล่างต่อไปนี้



จะเห็นได้ว่าจิ๊กซอชิ้นสุดท้ายที่เหลืออยู่ก่อนที่เราจะเข้าใจระบบคอมพิวเตอร์ทั้งหมดก็คือชิ้นส่วนของ system program ซึ่งเป็นโปรแกรมช่วยให้ application program สามารถรันโดยใช้ ISA ของ CPU ใดๆได้ ตัวอย่างของ system program เช่น

- แอสเซมเบลอร์
- ระบบปฏิบัติการ
- คอมไพเลอร์

ภาษาระดับสูงกับแอสเซมบลี

ในวิชานี้เราจะกล่าวถึงคอมไพเลอร์ซึ่งเป็นโปรแกรมที่ทำหน้าที่แปลจากภาษาระดับสูงมาเป็นภาษาแอสเซมบลีนั่นเอง วิชานี้เราจะกล่าวถึงกระบวนการนี้โดยละเอียด และเราไม่ต้องการเพียงการแปลที่ถูกต้องเท่านั้น เรายังต้องการให้โค๊ดแอสเซมบลีสุดท้ายที่ผลิตได้นี้มีประสิทธิภาพใกล้เคียงกับโค๊ดที่เขียนด้วยมือโดยโปรแกรมเมอร์ภาษาแอสเซมบลีอีกด้วย

กระบวนการแปลโดยคอมไพเลอร์นั้นจะมีความยุ่งยากมากกว่ากระบวนการแปลของแอสเซมเบลอร์ ทั้งนี้เพราะภาษาระดับสูงและภาษาแอสเซมบลีมีโครงสร้างและการสื่อความหมายที่แตกต่างกันมากดังแสดงดังตารางเปรียบเทียบด้านล่างต่อไปนี้

ภาษาระดับสูง	ภาษาแอสเซมบลี
มีโครงสร้างเป็น blocks หรือ modules	มีโครงสร้างเป็นแบบเชิงเส้น
ไม่ส่งเสริมการใช้ label และ goto	เต็มไปด้วย jump และ label ในลักษณะโค๊ดสปาเก็ตตี้
มีการบอกชนิดข้อมูลด้วย type	ไม่มีแนวคิดเรื่องของ type
มีปฏิบัติการให้ใช้ได้มากเพื่ออำนวยความสะดวกในการโปรแกรม	มีปฏิบัติการหลักเพียงสามกลุ่ม

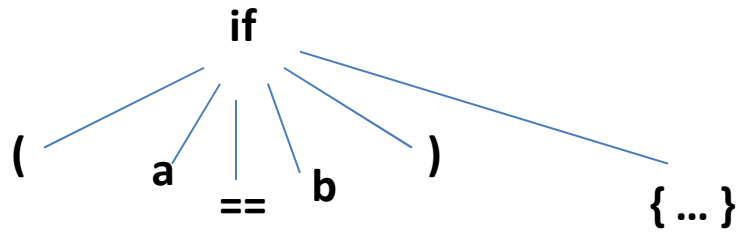
[ตัดไปที่การแนะนำโฮมเพจของวิชา รวมไปถึงเกณฑ์การให้คะแนนและรายละเอียดการจัดการอื่นๆในวิชานี้]

แนวคิดเบื้องต้นในการแปลจากภาษาระดับสูงเป็นแอสเซมบลี

ต่อไปนี้เราจะมาดูหลักการในการแปลภาษาในภาพรวมว่ามีกระบวนการอย่างไรบ้าง พิจารณาการแปลส่วนของโปรแกรมภาษาระดับสูง (ภาษาซี) ต่อไปนี้

```
if (a == b) {
        C[i] = 0;
}
```

ถ้าเราสามารถมองให้โค๊ดด้านบนนี้แทนอยู่ในรูปแบบของโครงสร้างข้อมูลต้นไม้ (tree) ตามด้านล่างนี้



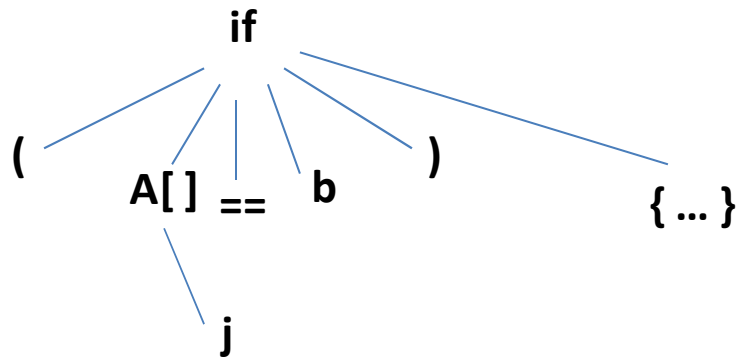
เราสามารถใช้หลักการอย่างง่ายและตรงไปตรงมาในการแปลโค๊ดส่วนนี้ได้ดังต่อไปนี้

"เราจะท่องไปในต้นไม้ (tree traversal) ที่แทนโค๊ดในภาษาระดับสูง จากนั้นเมื่อมีโหนด (node) ใดของต้นไม้ที่สามารถรับกับรูปแบบการแปลเป็นภาษาแอสเซมบลีได้ เราจะทำการแปลที่ node นั้นและผลิตโค๊ดแอสเซมบลีออกมา"

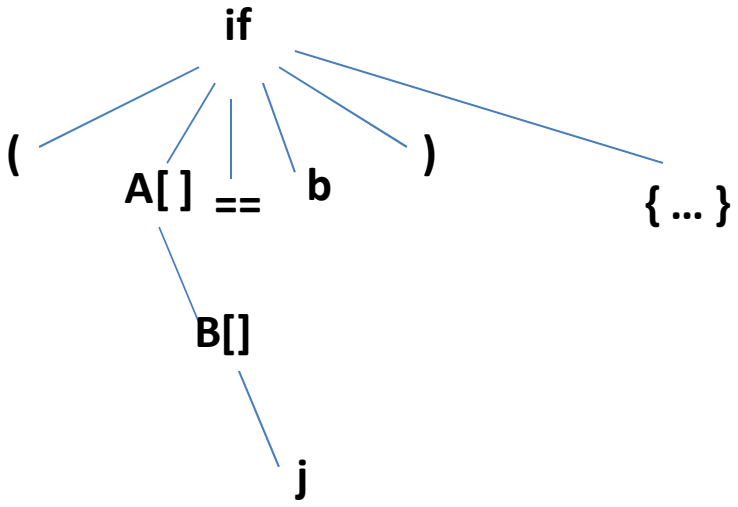
เช่นเมื่อเราเจอ node if เราทำการเช็คว่าnodeลูกของ if มีแบบแผนที่ถูกต้อง จากนั้นเราทำการแปลในส่วนของ condition และโค๊ดบล๊อกที่ตามมาดังแสดงด้านล่างนี้

```
bne     $t0, $t1, L1      # if (a != b) skip C[i] = 0

sll     $t3, $t4, 2       # i * 4
add     $t3, $t5, $t3     # &C[0] + i*4
sw      $zero, ($t3)      # C[i] = 0
L1:
```

ในการรูปแบบแอสเซมบลีที่จะมาจับคู่ (match) กับnode if ดังตัวอย่างในข้างต้นนั้น บางครั้งเราจะต้อง recurse แบบ top-down จากลูกของ node if เพื่อผลิตโค๊ดของ node ลูกนั้นเสียก่อนดังแสดงในตัวอย่างทั้งสองด้านล่างนี้




```
if (A[j] == b) {
        C[i] = 0;
}

sll     $t0, $t6, 2       #
add     $t0, $t7, $t0     # translation to get the value of A[j]
lw      $t0, ($t0)        #

bne     $t0, $t1, L1      # if (a != b) skip C[i] = 0
sll     $t3, $t4, 2       # i * 4
add     $t3, $t5, $t3     # &C[0] + i*4
sw      $zero, ($t3)      # C[i] = 0
L1:
```

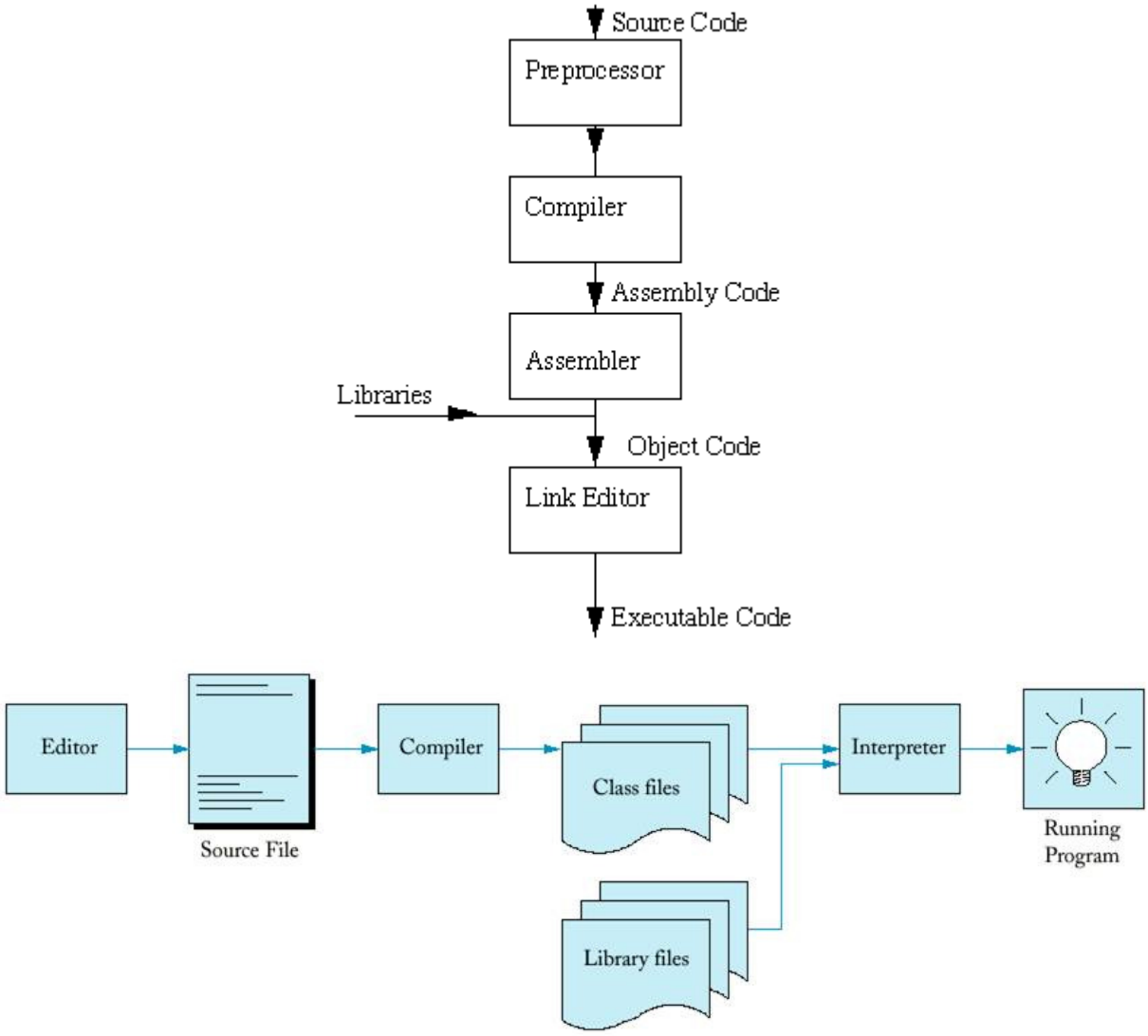



```
if (A[B[j]] == b) {

        C[i] = 0;

}

sll     $t0, $t8, 2     #
add     $t0, $t7, $t0   # translation to get the value of B[j]
lw      $t6, ($t0)      #

sll     $t0, $t6, 2     #
add     $t0, $t7, $t0   # translation to get the value of A[B[j]]
lw      $t0, ($t0)      #

bne     $t0, $t1, L1    # if (a != b) skip C[i] = 0
sll     $t3, $t4, 2     # i * 4
add     $t3, $t5, $t3   # &C[0] + i*4
sw      $zero, ($t3)    # C[i] = 0

L1:
```

ต่อไปเราจะมาคุยกันถึงหลักการการแปลภาษาอย่างจริงจัง เราจะได้พูดถึงทฤษฎีทางคอมพิวเตอร์และอัลกอริทึมที่อยู่เบื้องหลังคอมไพเลอร์ นิสิตจะรู้สึกดีใจที่เมื่อได้เรียนรู้และเข้าใจคอมไพเลอร์อย่างถ่องแท้แล้ว เราจะเข้าใจเรื่องราวหลายๆอย่างที่เกี่ยวข้องกับทางด้านวิทยาการและวิศวกรรมคอมพิวเตอร์มากขึ้น

คอมไพเลอร์คืออะไร

โปรแกรมที่แปลงโปรแกรมที่เขียนด้วยสัญญลักษณ์ในรูปแบบหนึ่งไปเป็นโปรแกรมที่เขียนด้วยสัญญลักษณ์ในอีกรูปแบบหนึ่ง โดยทั่วไปเราจะหมายถึงโปรแกรมที่แปลงโปรแกรมที่เขียนในภาษาระดับสูงเช่น ซี หรือ จาวา ไปเป็นโปรแกรมภาษาเครื่องของซีพียูเช่น MIPS หรือ X86



รูปเปรียบเทียบการแปลงจากภาษาระดับสูงเป็นภาษาระดับล่างที่คอมพิวเตอร์เข้าใจ รูปบนแสดงการแปลงจากภาษาระดับสูงเป็นภาษาเครื่องที่ซีพียูรันได้โดยตรง (เช่น ภาษาซี) รูปล่างแสดงการแปลงจากภาษาระดับสูงเป็นภาษาระดับกลางที่ต้องใช้ interpreter ที่เป็นตัวแทนของซีพียูในการรัน (เช่น ภาษาจาวา)

Source code VS assembly code VS machine code

Source code โปรแกรมที่เขียนด้วยภาษาระดับสูง ง่ายต่อมนุษย์ที่จะเข้าใจ

Machine code โปรแกรมที่อยู่ในรูปบิท 0 1 ง่ายต่อคอมพิวเตอร์ที่จะเข้าใจ

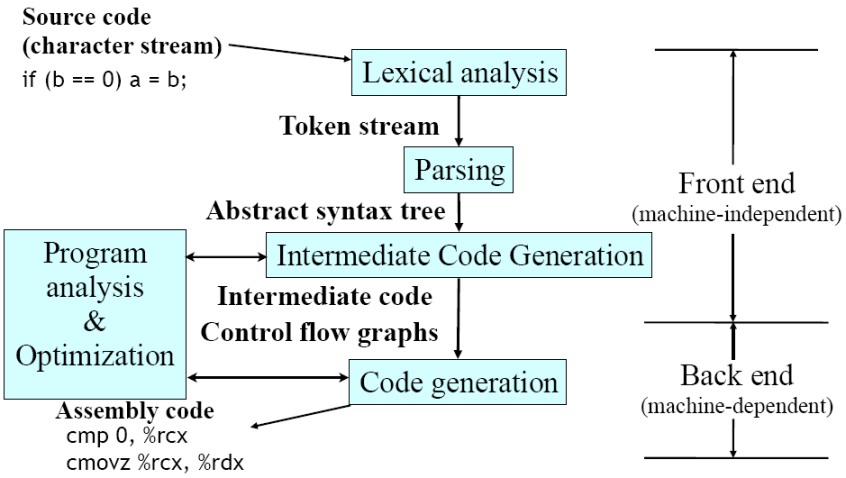
Assembly code โปรแกรมที่ใช้สัญญลักษณ์ที่เป็นตัวอักษรแทนคำสั่งภาษาเครื่อง การแปลงจาก assembly code ไปเป็น machine code ทำได้อย่างตรงไปตรงมาเพราะการจับคู่ระหว่างคำสั่งในทั้งสองรูปแบบเกือบจะเป็นแบบหนึ่งต่อหนึ่ง

```
int expr(int n)
{
    int d;
    d = 4 * n * n * (n + 1) * (n + 1);
    return d;
}
```

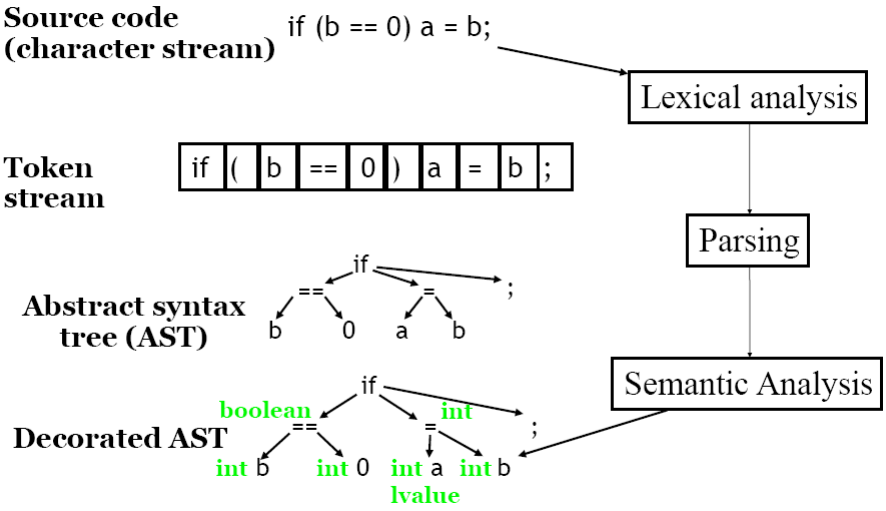
```
expr:
        pushl   %ebp                    55
        movl    %esp, %ebp              89 e5
        subl    $4, %esp                83 ec 04
        movl    8(%ebp), %eax           8b 45 08
        movl    %eax, %edx              89 c2
        imull   8(%ebp), %edx           0f af 55 08
        movl    8(%ebp), %eax           8b 45 08
        incl    %eax                    40
        imull   %eax, %edx              0f af d0
        movl    8(%ebp), %eax           8b 45 08
        incl    %eax                    40
        imull   %edx, %eax              0f af c2
        sall    $2, %eax                c1 e0 02
        movl    %eax, -4(%ebp)          89 45 fc
        movl    -4(%ebp), %eax          8b 45 fc
        leave                           c9
        ret                             c3
```

รูปเปรียบเทียบ source code (รูปบน) และ assembly กับ machine code (รูปล่าง)

โครงสร้างของคอมไพเลอร์



การแปลงจาก source code เป็น assembly code โดยผ่านเฟสต่างๆ



ขยายความในเฟสช่วง front-end

กระบวนการ bootstrapping

ในวิชานี้เราจะเขียนคอมไพเลอร์โดยใช้ภาษาระดับสูงคือภาษาซีเพื่อคอมไพล์โปรแกรมภาษาซี เกิดคำถามว่าแล้วตอนเริ่มแรกที่สุดเลย ใครเป็นคนเขียน machine code ที่จะทำหน้าที่คอมไพล์ตัวคอมไพเลอร์ของเรา

แนะนำกระบวนการที่เรียกว่า bootstrapping

- แน่นอนว่าตอนแรกสุดจะต้องมีคนที่เสียสละเขียน machine code ดังกล่าวขึ้นมา
- แทนที่จะเขียน machine code ที่แปลงจากภาษาระดับสูงเป็นภาษาเครื่องโดยตรง เขียน machine code ที่แปลงจาก assembly code เป็น machine code
- จากนั้นใช้ภาษา assembly ในการเขียนคอมไพเลอร์ที่มีความสามารถในระดับพื้นฐานที่สุด
- ณ ตอนนี้เรามี machine code ที่สามารถคอมไพล์ภาษาระดับสูงได้แล้ว แต่ machine code อันนี้ยังไม่ได้ถูก optimized
- ต่อไปเราสามารถใช้ภาษาระดับสูงในการเขียนคอมไพเลอร์ที่มีความสามารถสูงขึ้น ใช้ unoptimized machine code ในการคอมไพล์ให้ได้ machine code ที่ optimized มากขึ้น

[ตัดไปที่สไลด์เรื่อง Tombstone Diagram]