ก่อนจะเรียนรู้คำสั่งในกลุ่มสุดท้ายของ MIPS assembly เราลองดูว่าเราจะแปลงโค้ดการเข้าถึง array ที่มี index เป็นตัวแปรเช่นส่วนของโค้ดด้านล่างนี้ได้อย่างไร

```
int A[10];
int i, temp;

temp = A[i+1];

A[i+1] = A[i];

A[i] = temp;
```

กำหนดให้ `$11 = &A[0]` และ `$8 = i` และ `$9 = temp`

ต่อไปนี้เป็นเป็นโค้ด `MIPS assembly` ที่แปลมาจากโค้ดด้านบน

```
sll $10, $8, 2 # i * 4

addu $10, $10, $11 # &A[i] = $A[0] + i*4

lw $9, 4($10) # temp = A[i+1]

lw $12, ($10) # get value of A[i]

sw $12, 4($10) # A[i+1] = A[i]

sw $9, ($10) # A[i] = temp
```

กลุ่มคำสั่ง control flow (branch/jump)

มีคุณสมบัติดังต่อไปนี้

- เป็นกลุ่มคำสั่งที่ไม่ได้เปลี่ยนแปลงค่าใน register หรือข้อมูลใน memory
- จะเปลี่ยนแปลงค่า register พิเศษที่เรียกว่า PC (Program Counter) โดย PC คือ register ที่เก็บค่า address ของคำสั่งที่กำลังทำการประมวลผล

การเปลี่ยนแปลง PC กระทำได้ใน 2 ลักษณะคือ

1) อัตโนมัติ ณ เวลาที่โปรแกรมทำงาน โดย PC จะเลื่อนจากคำสั่ง 1 ไป 2 ไป 3 ตามลำดับ
2) จากผลของการประมวลคำสั่ง branch หรือ jump โดยจะมีคำสั่งอยู่ 2 ลักษณะคือ
    - Conditional เปลี่ยนแปลง PC อย่างมีเงื่อนไข
    - Unconditional เปลี่ยนแปลง PC อย่างไม่มีเงื่อนไข

ในระดับภาษา assembly เราไม่ได้เกี่ยวข้องกับตัว PC โดยตรง โดยเราจะใช้ label แทน address ของคำสั่งที่ branch/jump จะกระโดดไปหาถ้าเกิดสภาวะที่เหมาะสม

แม้ว่า MIPS จะมีคำสั่ง branch/jump หลายประเภท ในวิชานี้เราอยากให้จำกัดอยู่เพียงสามคำสั่งดังต่อไปนี้

j L1 # jump ไปที่คำสั่งที่ด้านหน้ามี label L1 ติดอยู่เสมอ
bne rs, rt, L1 # jump ไปที่คำสั่งที่ด้านหน้ามี label L1 ติดอยู่ก็ต่อเมื่อค่าใน register rs != ค่าใน register rt
beq rs, rt, L1 # jump ไปที่คำสั่งที่ด้านหน้ามี label L1 ติดอยู่ก็ต่อเมื่อค่าใน register rs == ค่าใน register rt

ในเมื่อเรามีเพียง beq และ bne ถ้าเราต้องการ jump ตามเงื่อนไขของ relational operator ตัวอื่นๆเช่น < <= > >= เราจะทำอย่างไร คำตอบคือเราจะใช้คำสั่ง beq และ bne ร่วมกับคำสั่ง slt และ slti (ย้อนกลับไปทบทวนคำสั่งสองชนิดนี้จากบทเรียนก่อนหน้านี้)

ถ้าเราต้องการ jump ไปที่ L1 ในกรณีที่ $8 < $9 เราใช้คำสั่ง MIPS 2 คำสั่งต่อไปนี้

slt $10, $8, $9 # if $8 < $9 then $10 = 1 else $10 = 0
bne $10, $0, L1 # if $10 != 0 goto L1

ถ้าเราต้องการ jump ไปที่ L1 ในกรณีที่ $8 <= $9 เราใช้คำสั่ง MIPS 2 คำสั่งต่อไปนี้

slt $10, $9, $8 # if $8 > $9 then $10 = 1 else $10 = 0
beq $10, $0, L1 # if $10 == 0 goto L1

(กล่าวคือเราใช้สภาวะตรงข้าม $8 > $9 ในคำสั่ง slt โดยที่สภาวะนี้ที่จะทำให้ $10 = 1 เราจะไม่ jump ไป L1 แต่สภาวะที่เราจะ jump คือสภาวะที่ทำให้ $10 = 0 นั่นก็คือสภาวะที่ $8 <= $9)

ถ้านิสิตเข้าใจการใช้ slt ร่วมกับ beq และ bne แล้ว ให้ลองคิดดูว่าถ้าเราจะ jump ในกรณีที่ $8 > $9 หรือในกรณีที่ $8 >= $9 ไปที่ L1 เราจะเขียน MIPS assembly ในลักษณะที่คล้ายกับตัวอย่างด้านบนที่ได้ยกมาอย่างไร

ตัวอย่างการแปลงจากภาษาซีเป็น MIPS assembly ที่ใช้ชุดคำสั่ง branch

if แบบง่ายที่สุด:

```
int a,b;
if (a>b) a=0;
b += a;

$16 เก็บค่า a, $17 เก็บค่า b
slt $8, $17, $16 # (ถ้า $8==1 แล้ว a>b แต่ถ้า $8==0 แล้ว a<=b)
beq $8, $0, L1 # branch ไปที่ L1 ถ้า $8 == 0
addu $16, $0, $0 # a=0
L1:
addu $17, $17, $16 # b+=a
```

if-else:

```
if (a>=b) a=0;
else a=1;
b+=a;

slt $8, $16, $17 # เปรียบเทียบ a<b
bne $8, $0, L1 # ถ้า a<b (นั่นคือ $8 != 0) ไปที่ L1
addu $16, $0, $0 # a=0
j L2
L1: addiu $16, $0, 1 # a=1
L2: addu $17, $17, $16 # b+=a
```

for (หรือ while):

```
int i, j;
for (i=0; i<10; i++)
    j += i;
j++;
```

ลองถอดโครงสร้างของ for ออกแล้วเขียนใหม่ให้ใกล้เคียงกับ assembly มากขึ้น

```
i=0;
loop:
if (!(i<10)) goto end;
j+=i;
i++;
goto loop;
end:
 j++;
```

ให้ i เก็บอยู่ที่ $16 และ ให้ j เก็บอยู่ที่ $17 ทำการแปลงเป็น assembly ได้ดังต่อไปนี้

```
addu $16, $0, $0
loop:
slti $18, $16, 10   # ถ้า $16<10 จะได้ $18=1 ถ้า ! ($16<10) จะได้ $18=0
beq $18, $0, end
addu $17, $17, $16  # j+=i
addiu $16, $16, 1   # i++
j loop        # หรือ beq $0, $0, loop
end:
addiu $17, $17, 1
```

ดูตัวอย่างการแปลโค้ดที่ซับซ้อนเพิ่มเติมได้ที่ไฟล์ที่แนบมากับเนื้อหานี้

asm_sample.txt

Character กับ String

ตัวอย่างการเก็บ string ในภาษาซีใน memory ของ MIPS

K	A	S	E	T	\0
addr	addr + 1	addr + 2	addr + 3	addr + 4	addr + 5

ตัวอักษรแต่ละตัวแทนด้วยรูปแบบบิทที่กำหนดโดย ASCII

A = American

S = Standard

C = Code

I = Information

I = Interchange

ดังนั้น string ด้านบนจะถูกเก็บดังต่อไปนี้

0x4B	0x61	0x73	0x65	0x74	0x00

address ต่ำ address สูง

ความเป็น big หรือ little endian ไม่มีผลกับข้อมูลแบบ string ที่เป็น array ของ character ที่แต่ละตัวมีขนาด 1 byte สำหรับข้อมูลแบบนี้ address ต่ำที่สุดจะเก็บค่า character ตัวแรกเสมอ

ให้ s เก็บ string "I love Kasetsart" ต้องการนับจำนวน ‘a’ ที่อยู่ใน string s

```
count=0;
while (*s != 0) {
    if (*s == 'a') count++;
    s++;
}
```

ถอดโครงสร้างของ while ออก เขียนโค๊ดซีโดยใช้ goto ที่ฟังก์ชั่นการทำงานเทียบเท่ากับโค๊ดด้านบน

```
count = 0;
if (*s==0) goto end;
if (*s != 'a') goto L1;
count ++;
L1:
s++;
end:
```

ให้ count เก็บอยู่ใน $16 และ address เริ่มต้นของ s อยู่ใน $17 และแนะนำคำสั่งใหม่ที่อำนวยความสะดวกในการโหลดค่าคงที่เข้าไว้ที่ register: li (ย่อมาจาก load immediate) โดย li $7, 0xac23 หมายถึงการนำค่าคงที่ 0xac23 ที่ทำ sign extension แล้วไปเก็บไว้ที่ $7 ซึ่งจะให้ความเหมายเหมือนกับ addiu $7, $0, 0xac23

โค๊ดภาษาซีด้านบนแปลงเป็น MIPS assembly ได้ดังนี้

```
li  $16, 0
loop:
lbu    $18, 0($17)
beq    $18, $0, end
li  $19, 0x61 # load รหัส ASCII ของ a เข้าที่ $19
bne    $18, $19, L1
addiu  $16, $16, 1 # count++
L1:
addiu  $17, $17, 1 #s++
beq    $0, $0, loop # หรือ j loop
end:
```