กลุ่ม _____

การทดลองที่ 8 Inline assembly

1. Inline assembly

ในการเขียนโปรแกรมภาษาแอสเซมบลี นอกเหนือจากการเขียนด้วยภาษาแอสเซมบลีอย่างเดียว เรายังสามารถใส่ภาษาแอสเซมบลีเข้าไปใน code ภาษาซีได้อีกด้วย ซึ่งจะมีประโยชน์กรณีที่เราต้องการการเขียนโปรแกรมที่รวดเร็วกว่า และเขียนได้ง่ายกว่าด้วยภาษาซี แต่ในส่วนที่ต้องการประสิทธิภาพสูง สามารถเขียนด้วยภาษาแอสเซมบลีได้

การเขียนภาษาแอสเซมบลีแทรกเข้าไปใน code ภาษาซี มีรูปแบบดังนี้

```
asm ("<list of assembly instruction>"
       : <list of write-only parameter formats and names>
       : <list of read-only parameter formats and names>
       : <list of register that are changed by the code>
       );
```

- ตัวอย่างที่ 1 ไม่มี parameter ดังนั้นจึงไม่ต้องมีส่วน ":" ปิดท้ายเลยมีตัวอย่างดังนี้

```
int return1(void) {asm("mov r0, #1");}
```

ฟังก์ชันข้างต้นจะนำค่า 1 ใส่ใน r0 และ return กลับมา เป็นเพียงการส่งค่า return กลับ

- ตัวอย่างที่ 2 อ้างถึง global variable โดยมี parameter โดยตัวอย่างต่อไปนี้

```
#include <stdio.h>

int i,j;

void init(void)
{
     asm("mov r0, #0; add %0, r0, #100; add %1, r0, #200": "+r"(i),"+r"(j));
}

int main(int argc, char *argv[])
{
   init();
   printf("\n i = %d and j = %d \n", i, j);
}
```

ตัวอย่างข้างต้นจะประกอบด้วย 3 คำสั่ง คือ r0=0 : r0+100 นำไปเก็บที่ %0 และ r0+200 นำไปเก็บที่ %1 โดย หลังเครื่องหมาย : จะกำหนด write only parameter ซึ่งในโปรแกรมนี้มี 2 ค่า คือ i กับ j เนื่องจากเป็นการ

นำค่าออกจากส่วน assembly อย่างเดียว ในการอ้างถึงตัวแปรจะใช้ %0, %1 ไล่ไปเรื่อยๆ ตามจำนวนตัวแปรที่อ้างถึง ดังนั้นในกรณีนี้ %0 จะหมายถึง i และ %1 จะหมายถึง j

สำหรับ format กำกับตัวแปร “+r” จะหมายถึง read/write ถ้า “=r” จะหมายถึง write only

<u>หมายเหตุ</u> การเขียนภาษา assembly ในบรรทัดเดียวกันอาจจะดูยาก สามารถแยกบรรทัดโดยใช้ /

- ตัวอย่างที่ 3 อ้างถึง local variable โดยมีตัวอย่างดังนี้

```
#include <stdio.h>

int triple (int n)
{
   asm("add %[value], %[value], %[value], LSL #1" : [value] "+r" (n));
}

int main(void)
{
   int n;
   printf("\nTriple Program!\n");
   printf("\nEnter an integer : ");
   scanf("%d", &n);
   printf("\nThree times %d is %d\n",n, triple(n));
}
```

ในตัวอย่างนี้จะอ้างถึงตัวแปร โดยใช้เครื่องหมาย % แล้วตามด้วยชื่อ ซึ่งอาจจะสะดวกกว่าการใช้ตัวเลข ในการทำงานจริง C Compiler จะนำ n (value) ไปใส่ใน register จากนั้นจึงค่อยทำงานตามคำสั่ง ดังนั้นคำสั่งจริงๆ ก็จะเป็น add r0, r0, r0 LSL#1 (สมมติ register ที่ใช้เป็น r0)

- ตัวอย่างที่ 4 มีทั้งส่วน write only และ read only จากตัวอย่าง

```
#include <stdio.h>

int add(int i, int j)
{
  int res = 0;
  asm ("add %[result], %[input_i], %[input_j]"
    : [result] "=r" (res)
    : [input_i] "r" (i), [input_j] "r" (j)  );
  return res;
}

int main(void)
{
  int a = 1;
  int b = 2;
  int c = 0;

  c = add(a, b);

  printf("Result of %d + %d = %d\n", a, b, c);
}
```

จะแบ่งคำสั่งออกเป็น 3 ส่วน ดังนี้

ส่วนที่ 1 เป็นส่วนคำสั่ง

```
"add %[result], %[input_i], %[input_j]"
```

ส่วนที่ 2 เป็น list ของ output operand ถ้ามีหลายตัวให้คั่นโดยใช้ comma โดยแต่ละชุดจะประกอบด้วย ชื่อใน [ ] ตามด้วย format (หรืออาจเรียกว่า constraint string) ตามด้วย ตัวแปรภาษา c ที่ใช้เป็น output

```
[result] "=r" (res)
```

ส่วนที่ 3 เป็น list ของ input operand ในที่นี้คือ ตัวแปร i และ j

```
[input_i] "r" (i), [input_j] "r" (j)
```

Assignment #2

1. ให้แต่ละกลุ่มเขียนโปรแกรมเปรียบเทียบความเร็วในการทำงานระหว่างภาษา C กับ Assembly โดยใช้ฟังก์ชัน time จับเวลา โดยสร้างฟังก์ชันอะไรก็ได้ เช่น shift เพื่อคูณ 4 หรือ บวกเลข หรืออื่นๆ

```
#include <stdio.h>
#include <time.h>

int main ()
{
    time_t seconds;

    seconds = time(NULL);
    printf("Seconds since January 1, 1970 = %ld\n", seconds);

    return(0);
}
```

   เวลาที่ได้จะเป็นจำนวนวินาทีตั้งแต่ `January 1, 1970` ดังนั้นต้องอ่านค่า 2 ครั้งมาลบกัน ครั้งแรกก่อนทำ และครั้งที่ 2 หลังทำ และเพื่อให้เกิดจำนวนเวลามากๆ ให้วนลูปทำหลายๆ ครั้งเช่น 1 ล้านรอบ การแสดงผลให้แสดงผล 2 ค่า คือ เวลาที่ใช้เมื่อใช้ภาษา C และเวลาที่ใช้เมื่อใช้ภาษาแอสเซมบลี

2. ให้ส่งรายงานด้วยประกอบด้วย Source Code และ ผลการทำงาน
3. ส่งภายใน จ. 1 พ.ค. 66