1. 修正 asm.3.c 錯誤

(1) 修改前

```c
asm.3.c
#include <stdio.h>
int main(int argc, char** argv) {
    int a=10, b=20, c=90, d=100;
    printf("a=%d, b=%d, c=%d, d=%d\n", a, b, c, d);
    __asm__ volatile (
      "mov %1, %%rax\n"     //  rax = a
      "mov %2, %%rbx\n"     //  rbx = b
      "add %%rbx, %%rax\n"    // rax += rbx ## no save
      "mov %%rax, %0\n"     // c = rax switch
      : "=m" (c)     //output
      : "g" (a), "g" (b)     //input
      : "rbx", "rax"     //搞爛掉的暫存器
    );

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

```
~/OS_HW/HW2  ./asm.3
a=10, b=20, c=90, d=100
c = a + b
a=120, b=20, c=30, d=100
```

(2)修改後

(i) 修正 BUG：int 改成 long int

(ii) 原因： rax 暫存器為 64bit 專用，用 int 可能會有錯誤，故把 int 改成 long int

(iii) 其他解法：換成用 eax 暫存器來存 int

```c
asm.3_fixbug.c
    #include <stdio.h>
    int main(int argc, char** argv) {
        long int a=10, b=20, c=90, d=100;
        printf("a=%ld, b=%ld, c=%ld, d=%ld\n", a, b, c, d);
        __asm__ volatile (
                          "mov %1, %%rax\n"    //  rax = a
                          "mov %2, %%rbx\n"    //  rbx = b
                          "add %%rbx, %%rax\n"    // rax += rbx
                          "mov %%rax, %0\n"    // c = rax
                          : "=m" (c)  //output, %0
                          : "g" (a), "g" (b) //input, %1, %2
                          : "rbx", "rax" //搞爛掉的暫存器
                          );
        printf("c = a + b\n");
        printf("a=%ld, b=%ld, c=%ld, d=%ld\n", a, b, c, d);
    }
```

```
~/OS_HW/HW2  ./asm.3_fixbug
a=10, b=20, c=90, d=100
c = a + b
a=10, b=20, c=30, d=100
```

2. 修正 asm.5.c 錯誤

(1)修改前

```
+ asm.5.c
#include <stdio.h>
int main(int argc, char** argv) {
    unsigned long msr;
    asm volatile ( "rdtsc\n\t"  // Returns the time in EDX:EAX.
                   "shl $32, %%rdx\n\t"    // Shift the upper bits left.
                   "or %%rdx, %0"          // 'Or' in the lower bits.
                  : "=m" (msr) // msr會放在記憶體
                  :
                  : "rdx");
    printf("msr: %lx\n", msr);
}
```

```
~/OS_HW/HW2  ./asm.5
msr: 6f900000000
```

(2)修改後

(i) 修正 BUG：先 or 存到 rax 暫存器再改用 mov 放進去記憶體

(ii) 原因： 用 or 直接寫入記憶體會導致儲存不完全，所以後面的位元都是 0，故改用 mov 存到記憶體

```
asm.5_fixbug.c
#include <stdio.h>
int main(int argc, char** argv) {
    unsigned long msr;
    asm volatile ( "rdtsc\n\t"  // Returns the time in EDX:EAX.
                   "shl $32, %%rdx\n\t"    // Shift edx to the upper bits left.
                   "or %%rdx, %%rax\n\t"          // 'Or' in the lower bits.
                                  //rdx upper bits is edx or lower bit eax and store to rax
                   "mov %%rax, %0\n" //move will store memory fully
                  : "=m" (msr) // msr會放在記憶體 //=a show that rax low bit from eax
                  :
                  : "rdx");
    printf("msr: %lx\n", msr);
}
```

```
~/OS_HW/HW2  ./asm.5_fixbug
msr: 4db7cd024a1
```

//rdtsc 是用來取得 Intel Pentium 以上 CPU 的 TSC 值（time stamp counters），所經過的 clock 數，其數值為 64 bit，傳回於 EDX:EAX 中

//附錄：Youtube 筆記
C 語言視高階的組合語言
Linux 幾乎都用 C，除了跟 CPU 相關的才會用組合語言寫

asm [volatile] //不要對下面的組合言做優化
(AssemblerTemplate //這部分就是組合語言
: OutputOperands // optional ，組語會輸出的變數
[: InputOperands // optional，組語會讀取的變數
[: Clobbers] // optional]，組合語言搞爛掉的暫存器的值)
//gcc 會自動先存然後再用完後還原
//用指標存取記憶體比較麻煩

Asm.1.c（範例）

```
1. __asm__ volatile(  //不要優化
2.       "movl $100, %%eax\n"      // eax = 100  movl 是後面接的是多長
3.       "movl $200, %%ebx\n"      // ebx = 100
4.       "addl %%ebx, %%eax\n"     // eax += rbx   b+a 存到 a
5.       "movl %%eax, %0\n"        // b = rax
6.       : "=g" (b)                //output, b 的代號是"%0"
7.       : "g" (a), "g" (d)        //input, a 代號是"%1", da 代號是"%2"
8.       : "ebx", "eax"            //搞髒掉的暫存器，gcc 會幫我們還原
9.    );
```

Asm.2.c（範例）

```
1.    __asm__ volatile(
2.                      "mov $100, %%rax\n"  //不是 movl  ax 暫存器 r 代表
                                               // 64 位元 e 是 32 位元
3.                      "mov $200, %%rbx\n"
4.                      "add %%rbx, %%rax\n"
5.                      "mov %%rax, %0\n"
6.                      : "=m" (b)    //output
7.                      : "g" (a), "g" (d) //input
8.                      : "rbx", "rax" //搞爛掉的暫存器
```

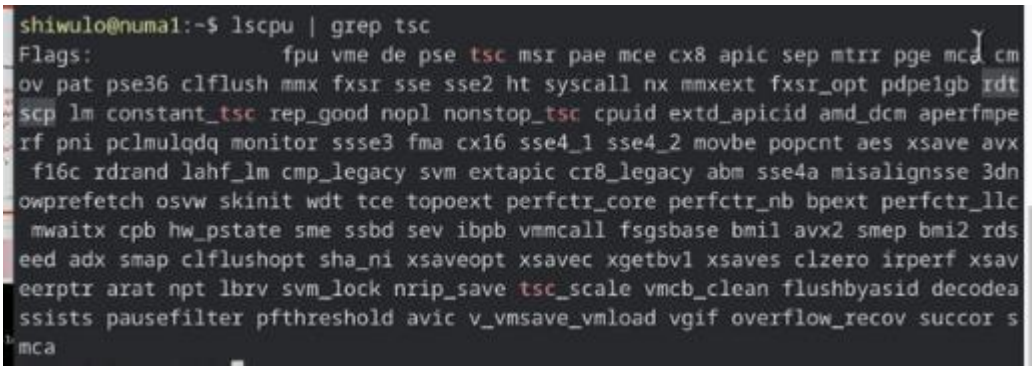
```
m   表示這是一個memory的operand  例如mov eax, data中的data

p   合法的記憶體位址
r   一般通用型register
g   任何的通用型register, memory operand, 立即定址的整數
```

```
constraints在386上有
a         eax
b         ebx
c         ecx
d         edx
S         esi
D         edi
```

X86 運算沒有上指定 operand，會有預設的

//lscpu | grep tsc 列出支援的 TSC 原本是用來量 Cycle 後來被用來量時間



//因為 CPU 會動態調整頻率 使得 tsc 會不準確
//所以有 contant_tsc 獨立於 CPU 的時脈
//nonstop_tsc 提供較精準 constant time
//avx2 向量指令集
//sse sse2 sse3 sse4_1 see4_2 向量指令集 avx = see5, avx2 =see6 (Intel ,AMD)
//avx2 512/8 = 64 一個運算最多處理 64operand
//clzero 寫入 zero 來清除 logical address
//rdrand CPU 支援 RANDOM

Asm.4.c（範例）

```
1. #include <stdio.h>
2. int main(int argc, char** argv) {
3.     unsigned long msr;
4.     asm volatile ( "rdtsc\n\t"  // Returns the time in EDX:EAX.
5.                    "shl $32, %%rdx\n\t"    // Shift the upper
   bits left.
6.                    "or %%rdx, %0"          // 'Or' in the lower
   bits.
7.                    : "=a" (msr)            //msr 會放在 rax 暫存
   器  output
8.                    :   //input
9.                    : "rdx");  //搞爛的暫存器
10.    printf("msr: %lx\n", msr);
11.}
```

Asm.5.c（作業）

```
1. #include <stdio.h>
2. int main(int argc, char** argv) {
3.     unsigned long msr;
4.     asm volatile ( "rdtsc\n\t"  // Returns the time in EDX:EAX.
5.                    "shl $32, %%rdx\n\t"    // Shift the upper
   bits left.
6.                    "or %%rdx, %0"          // 'Or' in the lower
   bits.
7.                    : "=m" (msr)            //msr 會放在記憶體 一
   定要寫入記
                                  //憶體修改上面
8.                    :
9.                    : "rdx");
10.    printf("msr: %lx\n", msr);
11.}
```

X86 記憶體裡面有隱藏的暫存器，讓程式碼變短，減少 MEM 使用空間