B10815057 廖聖郝 Project 5

1. Write a report to elaborate the compiled RISC-V assembly code in correspondence of your C/C++ source code

   Program 1:

```
#include<iostream>
int main(){
    std::cout << "hello_world";
    return 0;
}
```

   從組合語言中可以發現 hello_world 與 main function 的蹤跡:

```
.LC0:
    .string "hello_world"
    .text
    .align  1
    .globl  main
    .type   main, @function
```

   Program 2:

```
int global_init = 87;
int global_non_init;
void function();

int main(){
    int local_init = 66;
    int local_non_init;

    int* dynamic = new int[10];
    function();
    return 0;
}

void function(){
    static int static_variable_init = 1;
    static int static_variable_non_init;
}
```

```
    .type   global_init, @object
    .size   global_init, 4
global_init:
    .word   87
    .globl  global_non_init
    .section    .sbss,"aw",@nobits
    .align  2
    .type   global_non_init, @object
    .size   global_non_init, 4
global_non_init:
    .zero   4
    .text
    .align  1
    .globl  main
    .type   main, @function
```

int 的 size 為 4，所以第二行的.size 欄位為 4

有初始化的變數，初始值會放在.word 欄位

無初始化的全域變數，就會有.zero 欄位，自動初始化為 0

Program 3

```
#include <iostream>
int fib(int n){

    if(n==0)
        return 0;

    if(n==1)
        return 1;

    return (fib(n-1)+fib(n-2));

}

int main(){
    int n;
    std::cout << "Please input an integer to show the last value of Fibonacci Sequence :\n";
    std::cin >> n;
    std::cout << "The Fibonacci Sequence is " << fib(n) << std::endl;
    return 0;
}
```

```
.LFE1540:
    .size   _Z3fibi, .-_Z3fibi
    .section    .rodata
    .align  2
.LC0:
    .string "Please input an integer to show the last value of Fibonacci Sequence :\n"
    .align  2
.LC1:
    .string "The Fibonacci Sequence is "
    .text
    .align  1
    .globl  main
    .type   main, @function
```

可以看到提示的 string 都放在.string 欄位

```
.LFE484:
    .size   _ZL20__gthread_key_deletei, .-_ZL20__gthread_key_deletei
    .local  _ZStL8__ioinit
    .comm   _ZStL8__ioinit,1,4
    .align  1
    .globl  _Z3fibi
    .type   _Z3fibi, @function
```

定義 fib function 的地方

```
.L6:
    lw  a5,-20(s0)
    addi    a5,a5,-1
    mv  a0,a5
    call    _Z3fibi
    mv  s1,a0
    lw  a5,-20(s0)
    addi    a5,a5,-2
    mv  a0,a5
    call    _Z3fibi
    mv  a5,a0
    add a5,s1,a5
```

Fib function 的內部實作

2. Compare both the elf-gcc and linux-gnu-gcc compilation results with and without using the -static compilation option by using objdump

   依據觀察結果得到以下 dumpfile 指令數比較:

   linux(static)(約幾十萬) > elf(not staitc) = elf(static)(約幾萬) > linux(not static)(約幾百行)

   elf 不管有無 static 編譯，結果似乎都沒差多少

   而 linux 有 static 的 dump file 指令數會遠大於沒有 static 的數量。

   有 include iostream 的程式碼 dump file 指令數(約 14 萬，linux(not static)例外)，會遠大於沒有 include iostream 的程式碼指令數(約 2 萬)

3. In your program, declare a variety of C/C++ variable types with and without non-zero initialization and identify the actual physical locations in either final binary program or run-time memory. Summarize your observations.

   觀察後的結論:

   全域變數:若無初始值，則會自動初始為 0

   Static 變數:若無初始值，則會自動初始為 0

   以上 2 種變數因為在編譯時期就已決定記憶體位址(只會有一份，並且生命週期與程式本身一樣長)，所以初始化值不會對執行時期造成影響(無運行成本)。

區域變數:若無初始值，則可能會是任何值(根據該記憶體位址之前寫入之值)

動態配置記憶體(存於 heap) :若無初始值，則可能會是任何值(根據該記憶體位址之前寫入之值)

以上 2 種變數會因為 function call 或是記憶體配置無法預期，所以在每次的調用都會有不同的記憶體位址，所以初始化該變數的時機只能在執行時期，因而增加運行成本。