<N26112437>_<劉兆軒> AIAS 2023 Lab 3 HW Submission

1. 請不要用這份template 交作業, 建立一個新的codimd 檔案, 然後copy & paste 這個template 到你創建的檔案做修改。
2. 請修改你的學號與姓名在上面的title 跟這裡, 以避免TA 修改作業時把檔案跟人弄錯了
3. 在Playlab 作業中心繳交作業時, 請用你創建的檔案鏈結繳交, 其他相關的資料與鏈結請依照Template 規定的格式記載於codimd 上。

記得在文件標題上修改你的 <學號> <姓名>

Gitlab code link

Please paste the link to your private Gitlab repository for this homework submission here.

- https://playlab.computing.ncku.edu.tw:4001/kevin1217/lab03 (https://playlab.computing.ncku.edu.tw:4001/kevin1217/lab03)

HW3-1 - Fibonacci Series

Assembly Code

> 請放上你的程式碼並加上註解，讓 TA明白你是如何完成的。

```
#c code
#int fib(int n){
#       if(n==0) return 0
#   else if(n==1) return 1
#       else return fib(n-1)*fib(n-2)
#}
.text
main:
    li     a0, 16             # 暫存器 a0 儲存 n 值，這邊設定 n=16
    li     a1,0
    jal    fib                # 開始進行 fib 運算
    j      exit
fib:
    addi   sp, sp, -8         # 進行 stack pointer 的移動
    sw     ra, 0(sp)          # 儲存 return address
    sw     a0, 4(sp)          # 儲存 temp data
    li     t0, 1
    li     t1, 2
    blt    a0, t0, ret_zero   # if(n==0)return 0
    blt    a0, t1, ret_one    # if(n==1)return 1

    addi   a0, a0, -1         # f(n-1)
    jal    fib

    lw     a0, 4(sp)          # 逐一取出
    addi   a0, a0, -2         # f(n-2)
    jal    fib
    j      done

ret_one:
    li     a2, 1
    lw     ra, 0(sp)          #取出ra地址
    addi   sp, sp, 8
    add    a1, a2, a1
    jr     ra

ret_zero:
    li     a2, 0
    lw     ra, 0(sp)          #取出後，將block清掉
    addi   sp, sp, 8
    add    a1, a2, a1
    jr     ra
done:
    lw     ra, 0(sp)          #取出後，將block清掉
    addi   sp, sp, 8
    jr     ra

exit:
    addi a0,x0 ,1
    ecall
```

Simulation Result

```
987
```

HW3-2 - Fibonacci Series with C/Assembly Hybrid

Assembly Code & C Code

> 請放上你的程式碼並加上註解，讓 TA明白你是如何完成的。

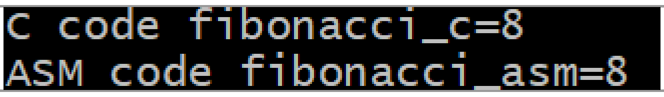
- `fibonacci.S`

```
fibonacci_asm:
prologue:
    addi   sp, sp, -4         # 進行 stack pointer 的移動
    sw     ra, 0(sp)          # 儲存 return address
    li     a1,0ho

funct_start:
jal    formula                # 開始進行 fib 運算

epilogue:
    mv a0,a1
    lw     ra, 0(sp)          # 儲存 return address
    addi   sp, sp, 4          # 進行 stack pointer 的移動
    jr     ra

ret_one:
    li     t2, 1
    lw     ra, 0(sp)          #取出ra地址
    addi   sp, sp, 8
    add    a1, t2, a1
    jr     ra

ret_zero:
    li     t3, 0
    lw     ra, 0(sp)          #取出後，將block清掉
    addi   sp, sp, 8
    add    a1, t3, a1
    jr     ra


formula:
    addi   sp, sp, -8         # 進行 stack pointer 的移動
    sw     ra, 0(sp)          # 儲存 return address
    sw     a0, 4(sp)          # 儲存 temp data


    li     t0, 1
    li     t1, 2
    blt    a0, t0, ret_zero   # if(n==0)return 0
    blt    a0, t1, ret_one    # if(n==1)return 1

    addi   a0, a0, -1         # f(n-1)
    jal    formula

    lw     a0, 4(sp)          # 逐一取出
    addi   a0, a0, -2         # f(n-2)
    jal    formula

    lw     ra, 0(sp)          #取出後，將block清掉
    addi   sp, sp, 8
    jr     ra
    .size fibonacci_asm, .-fibonacci_asm
```

- `fibonacci.c`

```c
int fibonacci_c(int n) {
    if(n == 0) {
        return 0;
    }

    else if(n == 1) {
        return 1;
    }

    else {
        return fibonacci_c(n-1)+fibonacci_c(n-2);
    }
}
```

Simulation Result



```
C code fibonacci_c=8
ASM code fibonacci_asm=8
```

HW3-3 - 2x2 Sudoku

Assembly Code & C Code

> 請放上你的程式碼並加上註解，讓 TA明白你是如何完成的。

- `main.c`

```c
//main.c
#include <stdio.h>
#include <stdlib.h>
#include "sudoku_2x2_c.h"
#define SIZE 16

//char test_asm_data[16] = {0, 0, 0, 0,
//                           0, 4, 3, 2,
//                           0, 3, 1, 4,
//                           0, 2, 4, 1 };
//char test_c_data[16] = { 4, 2, 1, 3,
//                           1, 3, 4, 2,
//                           2, 1, 0, 4,
//                           3, 4, 2, 1 };
//char test_asm_data[16] = { 0, 2, 1, 3,
//                           1, 0, 4, 2,
//                           2, 1, 0, 4,
//                           3, 4, 2, 0 };

char test_c_data[16] = { 0, 0, 2, 0,
                         0, 0, 0, 4,
                         2, 3, 0, 0,
                         0, 4, 0, 0 };

//
//char test_asm_data[16] = { 0, 0, 2, 0,
//                             0, 0, 0, 4,
//                             2, 3, 0, 0,
//                             0, 4, 0, 0 };

char test_asm_data[16] = { 4, 1, 2, 3,
                           3, 0, 1, 4,
                           2, 3, 4, 0,
                           0, 4, 0, 2 };
void print_sudoku_result() {
    int i;
    char str[25];
    puts("Output c & assembly function result\n");
    puts("c result :\n");

    for( i=0 ; i<SIZE ; i++) {
        int j= *(test_c_data+i);
        itoa(j, str,10);
        puts(str);
    }

    puts("\n\nassembly result :\n");
    for( i=0 ; i<SIZE ; i++) {
        int j= *(test_asm_data+i);
        itoa(j, str, 10);
        puts(str);
    }

    int flag = 0;
    for( i=0 ; i<SIZE ; i++) {
        if (*(test_c_data+i) != *(test_asm_data+i)) {
            flag = 1;
            break;
        }
    }

    if (flag == 1){
        puts("\n\nyour c & assembly got different result ... QQ ...\n");
    }
    else {
        puts("\n\nyour c & assembly got same result!\n");
    }
}


void sudoku_2x2_asm(char *test_asm_data); // TODO, sudoku_2x2_asm.S

void sudoku_2x2_c(char *test_c_data); // TODO, sudoku_2x2_c.S

int main() {
    sudoku_2x2_c(test_c_data);
    sudoku_2x2_asm(test_asm_data);
    print_sudoku_result();
    return 0;
}
```

- sudoku_2x2_asm.S

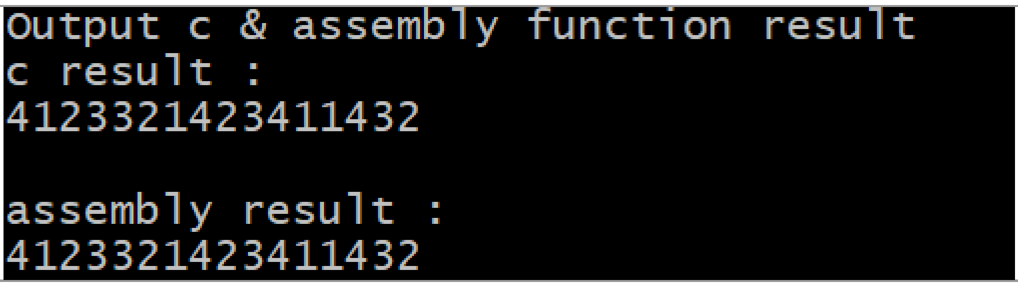
```
# sudoku_2x2_asm.S

    .text                           # code section
    .global sudoku_2x2_asm          # declare the asm function as a global func
    .type sudoku_2x2_asm, @function # define sum_asm as a function
sudoku_2x2_asm:

sudoku_2x2_c:
    #先存ra和s0到stack
    addi    sp,sp,-48
    sw      ra,44(sp)
    sw      s0,40(sp)
    #a0=0
    #a1=輸入矩陣之addr
    addi    s0,sp,48
    sw      a0,-36(s0)
    lw      a1,-36(s0)
    li      a0,0
    jal     ra,solve
    sw      a0,-20(s0)
    nop
    lw      ra,44(sp)
    lw      s0,40(sp)
    addi    sp,sp,48
    ret

solve:
    #a0:陣列addr
    #將原始a0的值放到a4內
    addi    sp,sp,-48
    sw      ra,44(sp)
    sw      s0,40(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    lw      a4,-36(s0)
    #if (index >= 16) return 1
    li      a5,15
    bge     a5,a4,solve+0x2c
    li      a5,1
    j       solve+0xe4
    lw      a5,-36(s0)
    lw      a4,-40(s0)
    add     a5,a4,a5
    lbu     a5,0(a5)
    beqz    a5,solve+0x5c
    lw      a5,-36(s0)
    addi    a5,a5,1
    lw      a1,-40(s0)
    mv      a0,a5
    #if (set[index] > 0) return solve(index + 1, set);
    jal     ra,solve
    mv      a5,a0
    j       solve+0xe4
    #index + 1
    li      a5,1
    sw      a5,-20(s0)
    #solve(index + 1
    j       solve+0xc4
    lw      a5,-36(s0)
    lw      a4,-40(s0)
    add     a5,a4,a5
    lw      a4,-20(s0)
    zext.b  a4,a4
    sb      a4,0(a5)
    lw      a1,-40(s0)
    lw      a0,-36(s0)
    #if (check(index,set) && solve(index + 1,set))
    jal     ra,check
    mv      a5,a0
    beqz    a5,solve+0xb8
    lw      a5,-36(s0)
    #index + 1
    addi    a5,a5,1
    lw      a1,-40(s0)
    mv      a0,a5
    jal     ra,solve
    mv      a5,a0
    #n <= 4
    beqz    a5,solve+0xb8
    #index + 1
    li      a5,1
    #solve(index + 1,set)
    j       solve+0xe4
    lw      a5,-20(s0)
    addi    a5,a5,1
    sw      a5,-20(s0)
    lw      a4,-20(s0)
    li      a5,4
    bge     a5,a4,solve+0x68
    lw      a5,-36(s0)
    lw      a4,-40(s0)
    add     a5,a4,a5
    sb      zero,0(a5)
    #set[index] = 0;
    li      a5,0
    mv      a0,a5
    lw      ra,44(sp)
    lw      s0,40(sp)
    addi    sp,sp,48
    ret


check:
    #if (col(index, set) && row(index, set) && box(index, set)) return 1;
    addi    sp,sp,-32
    sw      ra,28(sp)
    sw      s0,24(sp)
    addi    s0,sp,32
    sw      a0,-20(s0)
    sw      a1,-24(s0)
    #a0為陣列addr a1為index
    lw      a1,-24(s0)
    lw      a0,-20(s0)
    #col(index, set)
    jal     ra,col
    mv      a5,a0
    beqz    a5,check+0x5c
    lw      a1,-24(s0)
    lw      a0,-20(s0)
    #row(index, set)
    jal     ra,row
    mv      a5,a0
    beqz    a5,check+0x5c
    lw      a1,-24(s0)
    lw      a0,-20(s0)
    #box(index, set)
    jal     ra,box
    mv      a5,a0
    beqz    a5,check+0x5c
    #return 1;
    li      a5,1
    j       check+0x5c
    mv      a0,a5
    lw      ra,28(sp)
    lw      s0,24(sp)
    addi    sp,sp,32
    ret

col:
    addi    sp,sp,-48
    sw      s0,44(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    ##a0為陣列addr a1為index
    lw      a4,-36(s0)
    #int col_num = index % 4;
    srai    a5,a4,0x1f
    srli    a5,a5,0x1e
    #for (int i = col_num ; i <= col_num +12; i = i + 4)
    add     a4,a4,a5
    andi    a4,a4,3
    sub     a5,a4,a5
    sw      a5,-24(s0)
    lw      a5,-24(s0)
    sw      a5,-20(s0)
    j       col+0x80
    lw      a4,-20(s0)
    lw      a5,-36(s0)
    #if (i != index)
    beq     a4,a5,col+0x74
    lw      a5,-36(s0)
    lw      a4,-40(s0)
    add     a5,a4,a5
    lbu     a4,0(a5)
    lw      a5,-20(s0)
    lw      a3,-40(s0)
    add     a5,a3,a5
    lbu     a5,0(a5)
    #if (set[index] == set[i])
    bne     a4,a5,col+0x74
    li      a5,0
    j       col+0x94
    lw      a5,-20(s0)
    #i = i + 4
    addi    a5,a5,4
    sw      a5,-20(s0)
    lw      a5,-24(s0)
    #i <= col_num +12
    addi    a5,a5,11
    lw      a4,-20(s0)
    bge     a5,a4,col+0x3c
    #return 1
    li      a5,1
    mv      a0,a5
    lw      s0,44(sp)
    addi    sp,sp,48
    ret

row:
    addi    sp,sp,-48
    sw      s0,44(sp)
    addi    s0,sp,48
    #a0為陣列addr a1為index
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    lw      a5,-36(s0)
    #int row_num = (index / 4) * 4;
    srai    a4,a5,0x1f
    andi    a4,a4,3
    add     a5,a4,a5
    srai    a5,a5,0x2
    slli    a5,a5,0x2
    #for (i = row_num; i < row_num + 4; i++)
    sw      a5,-24(s0)
    lw      a5,-24(s0)
    sw      a5,-20(s0)
    #i < row_num + 4
    j       row+0x80
    lw      a4,-20(s0)
    lw      a5,-36(s0)
    #if (i != index)
    beq     a4,a5,row+0x74
    lw      a5,-36(s0)
    lw      a4,-40(s0)
    add     a5,a4,a5
    lbu     a4,0(a5)
    lw      a5,-20(s0)
    lw      a3,-40(s0)
    add     a5,a3,a5
    lbu     a5,0(a5)
    #i < row_num + 4
    bne     a4,a5,row+0x74
    li      a5,0
    j       row+0x94
    lw      a5,-20(s0)
    #i++
    addi    a5,a5,1
    sw      a5,-20(s0)
    lw      a5,-24(s0)
    addi    a5,a5,3
    lw      a4,-20(s0)
    bge     a5,a4,row+0x3c
    li      a5,1
    mv      a0,a5
    lw      s0,44(sp)
    addi    sp,sp,48
    ret


box:
    #    int box[16] = { 0,1,4,5,    2,3,6,7,
    #                  8,9,12,13,  10,11,14,15 };
    #a5為變數
    addi    sp,sp,-112
    sw      ra,108(sp)
    sw      s0,104(sp)
    addi    s0,sp,112
    ###a0為陣列addr a1為index
    sw      a0,-100(s0)
    sw      a1,-104(s0)
    sw      zero,-20(s0)
    lui     a5,0x80001
    addi    a4,a5,-60
    addi    a5,s0,-92
    #for (int i = box_num; i < box_num + 4; i++)
    mv      a3,a4
    li      a4,64
    mv      a2,a4
    mv      a1,a3
    mv      a0,a5
    #break
    jal     ra,memcpy
    sw      zero,-24(s0)
    j       box+0x90
```

```
        j       box+0x90
        lw      a5,-24(s0)
        slli    a5,a5,0x2
        #i < 16
        addi    a5,a5,-16
        add     a5,a5,s0
        lw      a5,-76(a5)
        lw      a4,-100(s0)
        #if (index == box[i])
        bne     a4,a5,box+0x84
        lw      a5,-24(s0)
        srai    a4,a5,0x1f
        andi    a4,a4,3
        add     a5,a4,a5
        srai    a5,a5,0x2
        slli    a5,a5,0x2
        sw      a5,-20(s0)
        j       box+0x9c
        lw      a5,-24(s0)
        addi    a5,a5,1
        sw      a5,-24(s0)
        lw      a4,-24(s0)
        li      a5,15
        #box_num = (i / 4) * 4;
        bge     a5,a4,box+0x48
        lw      a5,-20(s0)
        sw      a5,-28(s0)
        j       box+0x110
        lw      a5,-28(s0)
        slli    a5,a5,0x2
        addi    a5,a5,-16
        add     a5,a5,s0
        lw      a5,-76(a5)
        lw      a4,-100(s0)
        #i < box_num + 4
        beq     a4,a5,box+0x104
        lw      a5,-100(s0)
        lw      a4,-104(s0)
        add     a5,a4,a5
        lbu     a4,0(a5)
        lw      a5,-28(s0)
        # box_num = (i / 4) * 4;
        slli    a5,a5,0x2
        addi    a5,a5,-16
        add     a5,a5,s0
        lw      a5,-76(a5)
        mv      a3,a5
        lw      a5,-104(s0)
        add     a5,a5,a3
        lbu     a5,0(a5)
        # if (box[i] != index)
        bne     a4,a5,box+0x104
        li      a5,0
        j       box+0x124
        j       box+0x124
        lw      a5,-28(s0)
        addi    a5,a5,1
        sw      a5,-28(s0)
        lw      a5,-20(s0)
        addi    a5,a5,3
        lw      a4,-28(s0)
        #if (set[index] == set[box[i]])
        bge     a5,a4,box+0xa8
        li      a5,1
        mv      a0,a5
        lw      ra,108(sp)
        lw      s0,104(sp)
        addi    sp,sp,112
        #return 1;
        ret

        .size sudoku_2x2_asm, .-sudoku_2x2_asm
```

- sudoku_2x2_c.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "sudoku_2x2_c.h"
void sudoku_2x2_c(char* test_c_data) {
     int a = solve(0, test_c_data);
}


int solve(int index, char* set) {
    if (index >= 16) {
        return 1;                          // 如果檢查完所有的格子，回傳 True
    }
    if (set[index] > 0) {                  // set是一個儲存所有資料的array
        return solve(index + 1, set);      // 如果格子中已經有值了則會往下一格判斷

    }

    else {
        // 判斷目前這格在 1~4是否有符合條件
        // 如果有的話就往下一格作判斷（遞迴）
        // 直到每一格都符合條件為止
        for (int n = 1; n <= 4; n++) {
            set[index] = n;

            // DFS check function用來檢查當前這格放入這個數值是否正確
            // solve(index+1) function則是繼續判斷下一格的值
            if (check(index,set) && solve(index + 1,set))
                return 1;
        }
    }
    set[index] = 0;         // returns the value to 0 to mark it as empty
    return 0;               // no solution
}

//確認填入的數字是否在橫排、直列、box中是否有重複的數字
int check(int index,char *set)
{
    if (col(index, set) && row(index, set) && box(index, set))
    {
        return 1;
    }
}

//判斷直行是否有重複數字
int col(int index, char* set)
{
    int col_num = index % 4;
    for (int i = col_num ; i < col_num +12; i = i + 4)
    {
        if (i != index)
        {
            if (set[index] == set[i])
            {
                return 0;
            }
        }
    }
    return 1;
}

//判斷橫列是否有重複數字
int row(int index, char* set)
{
    int row_num = (index / 4) * 4;
    int i;
    for (i = row_num; i < row_num + 4; i++)
    {
        if (i != index)
        {
            if (set[index] == set[i])
            {
                return 0;
            }
        }
    }

    return 1;
}

//判斷box內是否有重複數字
int box(int index, char* set)
{
    int box_num = 0;

    //將4*4的數獨切成4個box
    //0  1    2  3
    //4  5    6  7
    //
    //8  9    10 11
    //11 12   13 14
    int box[16] = { 0,1,4,5,    2,3,6,7,
                    8,9,12,13,  10,11,14,15 };
    for (int i = 0; i < 16; i++)
    {
        if (index == box[i])
        {
            box_num = (i / 4) * 4;//找區間
            break;
        }
    }
    for (int i = box_num; i < box_num + 4; i++)
    {

        if (box[i] != index)
        {
            if (set[index] == set[box[i]])
            {
                return 0;
            }
        }
    }

    return 1;
}
```

Simulation Result



```
Output c & assembly function result
c result :
4123321423411432

assembly result :
4123321423411432
```

Bonus

Assembly Code & C Code

請放上你選擇的 leetcode程式碼並加上註解，讓 TA明白你是如何完成的。

Simulation Result

請放上你在 Venus上的模擬結果，驗證程式碼的正確性。(螢幕截圖即可)