

<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>

- <N26112437>_<劉兆軒> AIAS 2023 Lab 3 HW Submission
 - Gitlab code link
 - HW3-1 - Fibonacci Series
 - Assembly Code
 - Simulation Result
 - HW3-2 - Fibonacci Series with C/Assembly Hybrid
 - Assembly Code & C Code
 - Simulation Result
 - HW3-3 - 2x2 Sudoku
 - Assembly Code & C Code
 - Simulation Result
 - Bonus
 - Assembly Code & C Code
 - Simulation Result

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。

```
1  #c code
2  #int fib(int n){
3  #      if(n==0) return 0
4  #  else if(n==1) return 1
5  #      else return fib(n-1)*fib(n-2)
6  #}
7  .text
8  main:
9      li      a0, 16          # 暫存器 a0 儲存 n 值，這邊設定 n=16
10     li      a1, 0
11     jal     fib             # 開始進行 fib 運算
12     j       exit
13 fib:
14     addi    sp, sp, -8      # 進行 stack pointer 的移動
15     sw      ra, 0(sp)      # 儲存 return address
16     sw      a0, 4(sp)      # 儲存 temp data
17     li      t0, 1
18     li      t1, 2
19     blt     a0, t0, ret_zero # if(n==0)return 0
20     blt     a0, t1, ret_one  # if(n==1)return 1
21
22     addi    a0, a0, -1      # f(n-1)
23     jal     fib
24
25     lw      a0, 4(sp)      # 逐一取出
26     addi    a0, a0, -2      # f(n-2)
27     jal     fib
28     j       done
29
30 ret_one:
31     li      a2, 1
32     lw      ra, 0(sp)      #取出ra地址
33     addi    sp, sp, 8
34     add     a1, a2, a1
35     jr      ra
36
37 ret_zero:
38     li      a2, 0
39     lw      ra, 0(sp)      #取出後，將block清掉
40     addi    sp, sp, 8
41     add     a1, a2, a1
42     jr      ra
43 done:
44     lw      ra, 0(sp)      #取出後，將block清掉
45     addi    sp, sp, 8
46     jr      ra
47
48 exit:
49     addi    a0, x0, 1
50     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

```

1  fibonacci_asm:
2  prologue:
3      addi    sp, sp, -4          # 進行 stack pointer 的移動
4      sw      ra, 0(sp)          # 儲存 return address
5      li      a1, 0             # 初始化 a1
6
7  funct_start:
8      jal     formula            # 開始進行 fib 運算
9
10 epilogue:
11     mv a0, a1
12     lw      ra, 0(sp)          # 儲存 return address
13     addi    sp, sp, 4          # 進行 stack pointer 的移動
14     jr      ra
15
16 ret_one:
17     li      t2, 1
18     lw      ra, 0(sp)          # 取出 ra 地址
19     addi    sp, sp, 8
20     add     a1, t2, a1
21     jr      ra
22
23 ret_zero:
24     li      t3, 0
25     lw      ra, 0(sp)          # 取出後，將 block 清掉
26     addi    sp, sp, 8
27     add     a1, t3, a1
28     jr      ra
29
30
31 formula:
32     addi    sp, sp, -8          # 進行 stack pointer 的移動
33     sw      ra, 0(sp)          # 儲存 return address
34     sw      a0, 4(sp)          # 儲存 temp data
35
36
37     li      t0, 1
38     li      t1, 2
39     blt     a0, t0, ret_zero    # if(n==0) return 0
40     blt     a0, t1, ret_one     # if(n==1) return 1
41
42     addi    a0, a0, -1          # f(n-1)
43     jal     formula
44
45     lw      a0, 4(sp)          # 逐一取出
46     addi    a0, a0, -2          # f(n-2)
47     jal     formula
48
49     lw      ra, 0(sp)          # 取出後，將 block 清掉
50     addi    sp, sp, 8
51     jr      ra
52     .size fibonacci_asm, .-fibonacci_asm

```

- fibonacci.c

```
1  int fibonacci_c(int n) {
2      if(n == 0) {
3          return 0;
4      }
5
6      else if(n == 1) {
7          return 1;
8      }
9
10     else {
11         return fibonacci_c(n-1)+fibonacci_c(n-2);
12     }
13 }
```

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


```
1 //main.c
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "sudoku_2x2_c.h"
5 #define SIZE 16
6
7 //char test_asm_data[16] = {0, 0, 0, 0,
8 //                          0, 4, 3, 2,
9 //                          0, 3, 1, 4,
10 //                          0, 2, 4, 1 };
11 //char test_c_data[16] = { 4, 2, 1, 3,
12 //                        1, 3, 4, 2,
13 //                        2, 1, 0, 4,
14 //                        3, 4, 2, 1 };
15 //char test_asm_data[16] = { 0, 2, 1, 3,
16 //                          1, 0, 4, 2,
17 //                          2, 1, 0, 4,
18 //                          3, 4, 2, 0 };
19
20 char test_c_data[16] = { 0, 0, 2, 0,
21                        0, 0, 0, 4,
22                        2, 3, 0, 0,
23                        0, 4, 0, 0 };
24
25 //
26 //char test_asm_data[16] = { 0, 0, 2, 0,
27 //                          0, 0, 0, 4,
28 //                          2, 3, 0, 0,
29 //                          0, 4, 0, 0 };
30
31 char test_asm_data[16] = { 4, 1, 2, 3,
32                          3, 0, 1, 4,
33                          2, 3, 4, 0,
34                          0, 4, 0, 2 };
35 void print_sudoku_result() {
36     int i;
37     char str[25];
38     puts("Output c & assembly function result\n");
39     puts("c result :\n");
40
41     for( i=0 ; i<SIZE ; i++) {
42         int j= *(test_c_data+i);
43         itoa(j, str,10);
44         puts(str);
45     }
46
47     puts("\n\nassembly result :\n");
48     for( i=0 ; i<SIZE ; i++) {
49         int j= *(test_asm_data+i);
50         itoa(j, str, 10);
51         puts(str);
52     }
53 }
```



```
54     int flag = 0;
55     for( i=0 ; i<SIZE ; i++) {
56         if (*(test_c_data+i) != *(test_asm_data+i)) {
57             flag = 1;
58             break;
59         }
60     }
61
62     if (flag == 1){
63         puts("\n\nyour c & assembly got different result ... QQ ...\n");
64     }
65     else {
66         puts("\n\nyour c & assembly got same result!\n");
67     }
68 }
69
70
71 void sudoku_2x2_asm(char *test_asm_data); // TODO, sudoku_2x2_asm.S
72
73 void sudoku_2x2_c(char *test_c_data); // TODO, sudoku_2x2_c.S
74
75 int main() {
76     sudoku_2x2_c(test_c_data);
77     sudoku_2x2_asm(test_asm_data);
78     print_sudoku_result();
79     return 0;
80 }
```

- sudoku_2x2_asm.S


```

1  # sudoku_2x2_asm.S
2
3      .text                                # code section
4      .global sudoku_2x2_asm              # declare the asm function as a global func
5      .type sudoku_2x2_asm, @function    # define sum_asm as a function
6  sudoku_2x2_asm:
7
8  sudoku_2x2_c:
9      #先存ra和s0到stack
10     addi    sp,sp,-48
11     sw      ra,44(sp)
12     sw      s0,40(sp)
13     #a0=0
14     #a1=輸入矩陣之addr
15     addi    s0,sp,48
16     sw      a0,-36(s0)
17     lw      a1,-36(s0)
18     li      a0,0
19     jal     ra,solve
20     sw      a0,-20(s0)
21     nop
22     lw      ra,44(sp)
23     lw      s0,40(sp)
24     addi    sp,sp,48
25     ret
26
27 solve:
28     #a0:陣列addr
29     #將原始a0的值放到a4內
30     addi    sp,sp,-48
31     sw      ra,44(sp)
32     sw      s0,40(sp)
33     addi    s0,sp,48
34     sw      a0,-36(s0)
35     sw      a1,-40(s0)
36     lw      a4,-36(s0)
37     #if (index >= 16) return 1
38     li      a5,15
39     bge     a5,a4,solve+0x2c
40     li      a5,1
41     j       solve+0xe4
42     lw      a5,-36(s0)
43     lw      a4,-40(s0)
44     add     a5,a4,a5
45     lbu     a5,0(a5)
46     beqz    a5,solve+0x5c
47     lw      a5,-36(s0)
48     addi    a5,a5,1
49     lw      a1,-40(s0)
50     mv      a0,a5
51     #if (set[index] > 0) return solve(index + 1, set);
52     jal     ra,solve
53     mv      a5,a0

```

```

54      j      solve+0xe4
55      #index + 1
56      li      a5,1
57      sw      a5,-20(s0)
58      #solve(index + 1
59      j      solve+0xc4
60      lw      a5,-36(s0)
61      lw      a4,-40(s0)
62      add     a5,a4,a5
63      lw      a4,-20(s0)
64      zext.b   a4,a4
65      sb      a4,0(a5)
66      lw      a1,-40(s0)
67      lw      a0,-36(s0)
68      #if (check(index,set) && solve(index + 1,set))
69      jal     ra,check
70      mv      a5,a0
71      beqz    a5,solve+0xb8
72      lw      a5,-36(s0)
73      #index + 1
74      addi    a5,a5,1
75      lw      a1,-40(s0)
76      mv      a0,a5
77      jal     ra,solve
78      mv      a5,a0
79      #n <= 4
80      beqz    a5,solve+0xb8
81      #index + 1
82      li      a5,1
83      #solve(index + 1,set)
84      j      solve+0xe4
85      lw      a5,-20(s0)
86      addi    a5,a5,1
87      sw      a5,-20(s0)
88      lw      a4,-20(s0)
89      li      a5,4
90      bge     a5,a4,solve+0x68
91      lw      a5,-36(s0)
92      lw      a4,-40(s0)
93      add     a5,a4,a5
94      sb      zero,0(a5)
95      #set[index] = 0;
96      li      a5,0
97      mv      a0,a5
98      lw      ra,44(sp)
99      lw      s0,40(sp)
100     addi    sp,sp,48
101     ret
102
103
104     check:
105         #if (col(index, set) && row(index, set) && box(index, set)) return 1;
106         addi    sp,sp,-32

```

```

107     sw      ra,28(sp)
108     sw      s0,24(sp)
109     addi    s0,sp,32
110     sw      a0,-20(s0)
111     sw      a1,-24(s0)
112     #a0為陣列addr a1為index
113     lw      a1,-24(s0)
114     lw      a0,-20(s0)
115     #col(index, set)
116     jal     ra,col
117     mv      a5,a0
118     beqz    a5,check+0x5c
119     lw      a1,-24(s0)
120     lw      a0,-20(s0)
121     #row(index, set)
122     jal     ra,row
123     mv      a5,a0
124     beqz    a5,check+0x5c
125     lw      a1,-24(s0)
126     lw      a0,-20(s0)
127     #box(index, set)
128     jal     ra,box
129     mv      a5,a0
130     beqz    a5,check+0x5c
131     #return 1;
132     li      a5,1
133     j       check+0x5c
134     mv      a0,a5
135     lw      ra,28(sp)
136     lw      s0,24(sp)
137     addi    sp,sp,32
138     ret
139
140 col:
141     addi    sp,sp,-48
142     sw      s0,44(sp)
143     addi    s0,sp,48
144     sw      a0,-36(s0)
145     sw      a1,-40(s0)
146     ##a0為陣列addr a1為index
147     lw      a4,-36(s0)
148     #int col_num = index % 4;
149     srai    a5,a4,0x1f
150     srli    a5,a5,0x1e
151     #for (int i = col_num ; i <= col_num +12; i = i + 4)
152     add     a4,a4,a5
153     andi    a4,a4,3
154     sub     a5,a4,a5
155     sw      a5,-24(s0)
156     lw      a5,-24(s0)
157     sw      a5,-20(s0)
158     j       col+0x80
159     lw      a4,-20(s0)

```

```

160     lw      a5, -36(s0)
161     #if (i != index)
162     beq      a4, a5, col+0x74
163     lw      a5, -36(s0)
164     lw      a4, -40(s0)
165     add      a5, a4, a5
166     lbu      a4, 0(a5)
167     lw      a5, -20(s0)
168     lw      a3, -40(s0)
169     add      a5, a3, a5
170     lbu      a5, 0(a5)
171     #if (set[index] == set[i])
172     bne      a4, a5, col+0x74
173     li      a5, 0
174     j        col+0x94
175     lw      a5, -20(s0)
176     #i = i + 4
177     addi     a5, a5, 4
178     sw      a5, -20(s0)
179     lw      a5, -24(s0)
180     #i <= col_num + 12
181     addi     a5, a5, 11
182     lw      a4, -20(s0)
183     bge      a5, a4, col+0x3c
184     #return 1
185     li      a5, 1
186     mv      a0, a5
187     lw      s0, 44(sp)
188     addi     sp, sp, 48
189     ret
190
191 row:
192     addi     sp, sp, -48
193     sw      s0, 44(sp)
194     addi     s0, sp, 48
195     #a0為陣列addr a1為index
196     sw      a0, -36(s0)
197     sw      a1, -40(s0)
198     lw      a5, -36(s0)
199     #int row_num = (index / 4) * 4;
200     srai     a4, a5, 0x1f
201     andi     a4, a4, 3
202     add      a5, a4, a5
203     srai     a5, a5, 0x2
204     slli     a5, a5, 0x2
205     #for (i = row_num; i < row_num + 4; i++)
206     sw      a5, -24(s0)
207     lw      a5, -24(s0)
208     sw      a5, -20(s0)
209     #i < row_num + 4
210     j        row+0x80
211     lw      a4, -20(s0)
212     lw      a5, -36(s0)

```

```

213     #1+ (1 != index)
214     beq     a4,a5,row+0x74
215     lw      a5,-36(s0)
216     lw      a4,-40(s0)
217     add     a5,a4,a5
218     lbu     a4,0(a5)
219     lw      a5,-20(s0)
220     lw      a3,-40(s0)
221     add     a5,a3,a5
222     lbu     a5,0(a5)
223     #i < row_num + 4
224     bne     a4,a5,row+0x74
225     li      a5,0
226     j       row+0x94
227     lw      a5,-20(s0)
228     #i++
229     addi    a5,a5,1
230     sw      a5,-20(s0)
231     lw      a5,-24(s0)
232     addi    a5,a5,3
233     lw      a4,-20(s0)
234     bge     a5,a4,row+0x3c
235     li      a5,1
236     mv      a0,a5
237     lw      s0,44(sp)
238     addi    sp,sp,48
239     ret
240
241
242 box:
243     #   int box[16] = { 0,1,4,5,    2,3,6,7,
244     #                       8,9,12,13, 10,11,14,15 };
245     #a5為變數
246     addi    sp,sp,-112
247     sw      ra,108(sp)
248     sw      s0,104(sp)
249     addi    s0,sp,112
250     ###a0為陣列addr a1為index
251     sw      a0,-100(s0)
252     sw      a1,-104(s0)
253     sw      zero,-20(s0)
254     lui     a5,0x80001
255     addi    a4,a5,-60
256     addi    a5,s0,-92
257     #for (int i = box_num; i < box_num + 4; i++)
258     mv      a3,a4
259     li      a4,64
260     mv      a2,a4
261     mv      a1,a3
262     mv      a0,a5
263     #break
264     jal     ra,memcpy
265     sw      zero,-24(s0)
266     i       box+0x00

```

```

266      j      box+0x90
267      lw      a5, -24(s0)
268      slli     a5, a5, 0x2
269      #i < 16
270      addi     a5, a5, -16
271      add      a5, a5, s0
272      lw      a5, -76(a5)
273      lw      a4, -100(s0)
274      #if (index == box[i])
275      bne      a4, a5, box+0x84
276      lw      a5, -24(s0)
277      srai     a4, a5, 0x1f
278      andi     a4, a4, 3
279      add      a5, a4, a5
280      srai     a5, a5, 0x2
281      slli     a5, a5, 0x2
282      sw      a5, -20(s0)
283      j      box+0x9c
284      lw      a5, -24(s0)
285      addi     a5, a5, 1
286      sw      a5, -24(s0)
287      lw      a4, -24(s0)
288      li      a5, 15
289      #box_num = (i / 4) * 4;
290      bge      a5, a4, box+0x48
291      lw      a5, -20(s0)
292      sw      a5, -28(s0)
293      j      box+0x110
294      lw      a5, -28(s0)
295      slli     a5, a5, 0x2
296      addi     a5, a5, -16
297      add      a5, a5, s0
298      lw      a5, -76(a5)
299      lw      a4, -100(s0)
300      #i < box_num + 4
301      beq      a4, a5, box+0x104
302      lw      a5, -100(s0)
303      lw      a4, -104(s0)
304      add      a5, a4, a5
305      lbu      a4, 0(a5)
306      lw      a5, -28(s0)
307      # box_num = (i / 4) * 4;
308      slli     a5, a5, 0x2
309      addi     a5, a5, -16
310      add      a5, a5, s0
311      lw      a5, -76(a5)
312      mv      a3, a5
313      lw      a5, -104(s0)
314      add      a5, a5, a3
315      lbu      a5, 0(a5)
316      # if (box[i] != index)
317      bne      a4, a5, box+0x104
318      li      a5, 0
319      i      box+0x124

```



```
319      j      box+0xa124
320      lw      a5, -28(s0)
321      addi     a5, a5, 1
322      sw      a5, -28(s0)
323      lw      a5, -20(s0)
324      addi     a5, a5, 3
325      lw      a4, -28(s0)
326      #if (set[index] == set[box[i]])
327      bge      a5, a4, box+0xa8
328      li      a5, 1
329      mv      a0, a5
330      lw      ra, 108(sp)
331      lw      s0, 104(sp)
332      addi     sp, sp, 112
333      #return 1;
334      ret
335
336      .size sudoku_2x2_asm, .-sudoku_2x2_asm
```

- sudoku_2x2_c.c


```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include "sudoku_2x2_c.h"
5  void sudoku_2x2_c(char* test_c_data) {
6      int a = solve(0, test_c_data);
7  }
8
9
10 int solve(int index, char* set) {
11     if (index >= 16) {
12         return 1; // 如果檢查完所有的格子，回傳 True
13     }
14     if (set[index] > 0) { // set是一個儲存所有資料的array
15         return solve(index + 1, set); // 如果格子中已經有值了則會往下一格判斷
16     }
17
18     else {
19         // 判斷目前這格在 1~4是否有符合條件
20         // 如果有的話就往下一格作判斷 (遞迴)
21         // 直到每一格都符合條件為止
22         for (int n = 1; n <= 4; n++) {
23             set[index] = n;
24
25             // DFS check function用來檢查當前這格放入這個數值是否正確
26             // solve(index+1) function則是繼續判斷下一格的值
27             if (check(index, set) && solve(index + 1, set))
28                 return 1;
29         }
30     }
31     set[index] = 0; // returns the value to 0 to mark it as empty
32     return 0; // no solution
33 }
34
35 //確認填入的數字是否在橫排、直列、box中是否有重複的數字
36 int check(int index, char *set)
37 {
38     if (col(index, set) && row(index, set) && box(index, set))
39     {
40         return 1;
41     }
42 }
43
44 //判斷直行是否有重複數字
45 int col(int index, char* set)
46 {
47     int col_num = index % 4;
48     for (int i = col_num; i < col_num + 12; i = i + 4)
49     {
50         if (i != index)
51         {
52             if (set[index] == set[i])
53
```

```
54         {
55             return 0;
56         }
57     }
58 }
59 return 1;
60 }
61
62 //判斷橫列是否有重複數字
63 int row(int index, char* set)
64 {
65     int row_num = (index / 4) * 4;
66     int i;
67     for (i = row_num; i < row_num + 4; i++)
68     {
69         if (i != index)
70         {
71             if (set[index] == set[i])
72             {
73                 return 0;
74             }
75         }
76     }
77
78     return 1;
79 }
80
81 //判斷box內是否有重複數字
82 int box(int index, char* set)
83 {
84     int box_num = 0;
85
86     //將4*4的數獨切成4個box
87     //0 1 2 3
88     //4 5 6 7
89     //
90     //8 9 10 11
91     //11 12 13 14
92     int box[16] = { 0,1,4,5, 2,3,6,7,
93                    8,9,12,13, 10,11,14,15 };
94     for (int i = 0; i < 16; i++)
95     {
96         if (index == box[i])
97         {
98             box_num = (i / 4) * 4; //找區間
99             break;
100        }
101    }
102    for (int i = box_num; i < box_num + 4; i++)
103    {
104
105        if (box[i] != index)
106        {
```

```
107         if (set[index] == set[box[i]])
108         {
109             return 0;
110         }
111     }
112 }
113
114 return 1;
115 }
```

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)