1 Introduction

After printing many review materials double-sided, we find it disturbing to flip them upside down during the review. Assume that N pages of the materials are double-sided printed, each side containing one of N chapters in the textbook. We wonder if there is a way to flip some of them so that we can cover all the N chapters at once.

In this program, we need to find out one flipping solution, which can let us cover all the N chapters at once.

For example if the Input is following :

5
3 2
2 5
4 1
1 3
2 1

The output should be : 3 5 4 1 2 (ignore the echo of input)

That means if we just flip the second page (swap 2 5 , the third line of input), and then we will get the first colum as the answer, which let us cover all the 5 chapters at once.

2 Algorithm Specification

The first thing we should do is read the input and do process of them. We use **trap x20** to read. Because the *N* may be larger than 10, we should use the special character like **Space** and **Enter** as a signal. After finishing the read of a number ,we transfer it into digit rather than ascii code and store it into memory.

```
num ← 0 \\ initialize
trap x20
trap x21
while R0 != \n
    num ← 10 * num + R0 - '0'
for i = 0 to num-1
    num ← 0 \\ initialize
    trap x20
    trap x21
    while R0 != ' '
        num ← 10 * num + R0 - '0'
    page[2*i] ← num
    num ← 0 \\ initialize
    trap x20
    trap x21
    while R0 != ' '
        num ← 10 * num + R0 - '0'
    page[2*i + 1] ← num

```

Then, we use the recursion to determine the flip condition. To simplify the coding procedure, we use an array to store whether one page need flip. Once we determine the ith page need to flip, then we recursively call function itself to get the flip condition of (i+1)th page to nth page. This procedure will form a complete binary tree. When the program comes to the leave of the binary tree, the program will call the check function to check whether in this flip condition, we can cover all the N chapters at once.

```
    function flip(int n)
        if n == num
            call check()
            return
        flip[n] ← 0
        call flip(n-1)
        flip[n] ← 1
        call flip(n-1)
        flip[n] ← 0 \\recover to be init
```

When the program comes to the leave of binary tree, the check function will be called. In the check function, we check whether the chosen chapter of each page can cover from 1 to num. Firstly, according to the flip condition, we take out the front chapter or back chapter of each page and store the result into memory. Then we form a label area, according to the result we get before to set the label area. That means if the result contain chapter 4, then label[3] will be set to 1. Finally, we go through the label area. If it's all 1 in label area, then we just output the result we obtained before and halt the program.

```
    function check()
        for i = 0 to num - 1
            if flip[i] == 1
                result[i] ← page[2*i + 1]
            else
                result[i] ← page[2*i]
        // clear the label area
        for i = 0 to num - 1
            label[i] ← 0
        for i = 0 to num - 1
            label[result[i]-1] ← 1
        for i = 0 to num - 1
            if label[i] == 0
                return
        //output
        for i = 0 to num - 1
            R0 ← result[i]
            trap x21
        halt
```

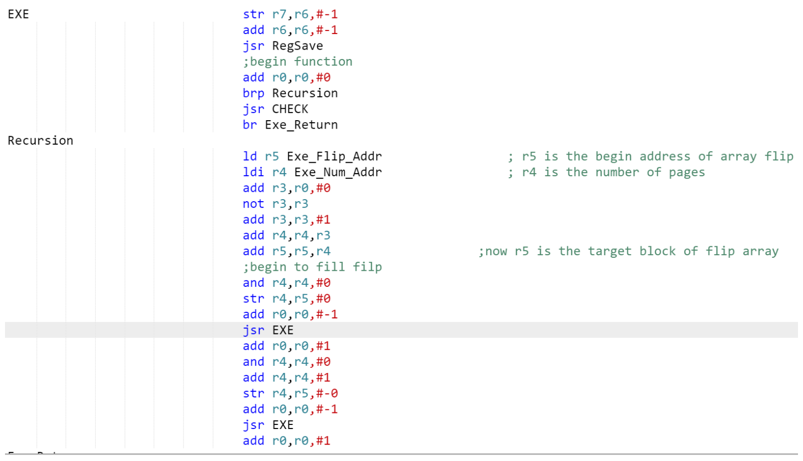
3 Q and A

- Q: what is your recursive algorithm?

  A: in the program, we do the recursive algorithm accroding to the flip condition. Once we set the ith page to be flipped or not, we then recursively set the (i+1)th to nth pages. In this way, the program will form a binary tree. When the program comes to the leave of binary tree, we call the check function to check the condition.

4 essential parts of code

Fig 1 is the implement of recursion operation

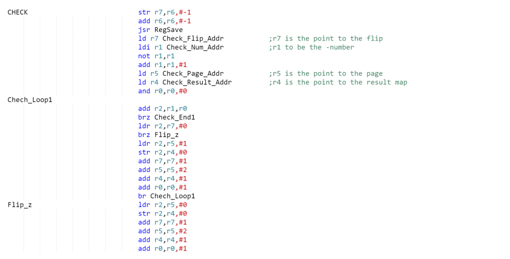


```
EXE                             str r7,r6,#-1
                                add r6,r6,#-1
                                jsr RegSave
                                ;begin function
                                add r0,r0,#0
                                brp Recursion
                                jsr CHECK
                                br Exe_Return
Recursion
                                ld r5 Exe_Flip_Addr                 ; r5 is the begin address of array flip
                                ldi r4 Exe_Num_Addr                 ; r4 is the number of pages
                                add r3,r0,#0
                                not r3,r3
                                add r3,r3,#1
                                add r4,r4,r3
                                add r5,r5,r4                    ;now r5 is the target block of flip array
                                ;begin to fill filp
                                and r4,r4,#0
                                str r4,r5,#0
                                add r0,r0,#-1
                                jsr EXE
                                add r0,r0,#1
                                and r4,r4,#0
                                add r4,r4,#1
                                str r4,r5,#-0
                                add r0,r0,#-1
                                jsr EXE
                                add r0,r0,#1
```

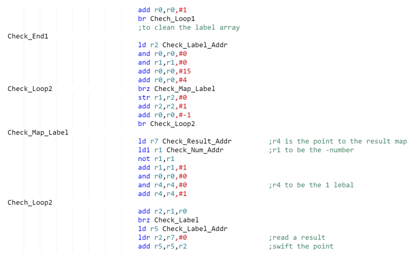
Figure 1: Fig 1

Fig 2 is the implement of check function

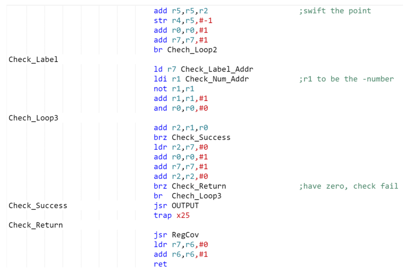
Figure 2: Fig 2



```
CHECK                           str r7,r6,#-1
                                add r6,r6,#-1
                                jsr RegSave
                                ld r7 Check_Flip_Addr           ;r7 is the point to the flip
                                ldi r1 Check_Num_Addr           ;r1 to be the -number
                                not r1,r1
                                add r1,r1,#1
                                ld r5 Check_Page_Addr           ;r5 is the point to the page
                                ld r4 Check_Result_Addr         ;r4 is the point to the result map
                                and r0,r0,#0
Chech_Loop1
                                add r2,r1,r0
                                brz Check_End1
                                ldr r2,r7,#0
                                brz Flip_z
                                ldr r2,r5,#1
                                str r2,r4,#0
                                add r7,r7,#1
                                add r5,r5,#2
                                add r4,r4,#1
                                add r0,r0,#1
                                br Chech_Loop1
Flip_z                          ldr r2,r5,#0
                                str r2,r4,#0
                                add r7,r7,#1
                                add r5,r5,#2
                                add r4,r4,#1
                                add r0,r0,#1
```



```
                                add r0,r0,#1
                                br Chech_Loop1
                                ;to clean the label array
Check_End1
                                ld r2 Check_Label_Addr
                                and r0,r0,#0
                                and r1,r1,#0
                                add r0,r0,#15
                                add r0,r0,#4
Check_Loop2                     brz Check_Map_Label
                                str r1,r2,#0
                                add r2,r2,#1
                                add r0,r0,#-1
                                br Check_Loop2
Check_Map_Label
                                ld r7 Check_Result_Addr         ;r4 is the point to the result map
                                ldi r1 Check_Num_Addr           ;r1 to be the -number
                                not r1,r1
                                add r1,r1,#1
                                and r0,r0,#0
                                and r4,r4,#0                    ;r4 to be the 1 lebal
                                add r4,r4,#1
Chech_Loop2
                                add r2,r1,r0
                                brz Check_Label
                                ld r5 Check_Label_Addr
                                ldr r2,r7,#0                    ;read a result
                                add r5,r5,r2                    ;swift the point
```



```
                                add r5,r5,r2                    ;swift the point
                                str r4,r5,#-1
                                add r0,r0,#1
                                add r7,r7,#1
                                br Chech_Loop2
Check_Label
                                ld r7 Check_Label_Addr
                                ldi r1 Check_Num_Addr           ;r1 to be the -number
                                not r1,r1
                                add r1,r1,#1
                                and r0,r0,#0
Chech_Loop3
                                add r2,r1,r0
                                brz Check_Success
                                ldr r2,r7,#0
                                add r0,r0,#1
                                add r7,r7,#1
                                add r2,r2,#0
                                brz Check_Return                ;have zero, check fail
                                br  Chech_Loop3
Check_Success                   jsr OUTPUT
                                trap x25
Check_Return
                                jsr RegCov
                                ldr r7,r6,#0
                                add r6,r6,#1
                                ret
```