lab04-Baguenaudier

PB22081571 薄震宇

2023 年 12 月 14 日

目录

1 实验目的

本次实验的目的在于熟悉递归子程序的使用。利用递归的方法将一个逐步实现较为困难的问题拆分成多次相同的操作，将复杂的问题简单化。在使用递归子程序的同时，加深对于栈这一数据结构的理解。

2 实验原理

正如上面所说，本次实验的关键在于完成递归子程序 **REMOVE** 和 **PUT**。其中 **REMOVE(i)** 用于将前 i 个环从板上移除，即将 state 的前 n 位由 1 变为 0；而 **PUT(i)** 用于将前 i 个环放到板上，即将 state 的前 n 位由 0 变为 1。二者是相反的操作。所以下面先实现 **REMOVE**：

利用下面的关系式

$$R(0) = nothing, R(1) = removethe1^{st}ring$$

$$R(i) = R(i-2) + removethei^{th}ring + P(i-2) + R(i-1), i \geqslant 2$$

可以得到 **REMOVE(n,state)** 子程序的逻辑如下：

- 若 $n = 0$，则直接返回 state
- 若 $n = 1$, 则将 state 的最后一位由 0 变为 1，直接对 state 加 1 即可，然后存储操作后的 state 于特定的存储空间，再返回 state。
- 若以上两种情况均不成立，则先将 state 的前 n-2 位全部变为 1，即调用 **REMOVE(n-2,state)**，然后再将 state 的第 n 位由 0 变为 1，因为 state 是二进制数，所以将 state 加上 2^{n-1} 即可，然后存储操作后的 state 于特定的存储空间；然后再调用 **PUT(n-2,state)** 将 state 的前 n-2 位复原为 0，最后再调用 **REMOVE(n-1,state)** 将 state 的前 n-1 位由 0 变为 1。

上述过程中所提到的将 state 存储于特定的存储空间中，需要使用一个全局变量来保存存储的地址，在汇编语言中，可以专门使用一个寄存器来存储。

在实现了 **REMOVE** 子程序后，只需要对它稍加修改即可：

- 由于 **REMOVE(i)** 是要将 state 的前 i 位由 0 变成 1，而 **PUT(i)** 是要将 state 的前 i 位由 1 变成 0, 所以在 **PUT** 子程序里需要将 **REMOVE** 子程序中的加法改为减法
- **REMOVE** 子程序中是 $REMOVE(i-2)+removethei^{th}ring+P(i-2)+R(i-1), i \geqslant 2$,在 **PUT** 子程序中则是 $PUT(i-2)+putthei^{th}ring+R(i-2)+P(i-1), i \geqslant 2$

在实现了以上两个递归子程序后，在主程序中调用 **REMOVE** 子程序即可

3 实验过程

3.1 C 语言实现

因为 C 语言中已经实现了递归函数，不需要考虑中间的变量的保存，函数的返回等问题，所以先用 C 语言来描述上面的过程，一方面是可以检测思路的正确性，另一方面是在用汇编语言实现时可以照着 C 语言来做的对应的修改，使得逻辑上更为清晰。

C 语言代码如下：

```
#include<stdio.h>
#include<math.h>

int REMOVE(int,int);
int PUT(int,int);

int STATE[10000] = {0};//存储每一步操作后的state
int times;//times表示操作的次数

int main(){
    int n;/*the value of n in x3100*/
    printf("Please input n:");
    scanf("%d",&n);
    STATE[0] = n;
    times = 0;
    REMOVE(n,0);
    for(int i = 0;i <= times;i++)
        printf("%d\n",STATE[i]);
    return 0;
}

int REMOVE(int n,int state){
    /*these two args means*/
    /*(1-n rings will be removed,all rings' state
```

```
        at now)*/
    /*the return value means*/
    /*after these operations, what all rings' state
        are*/
    /*Note that*/
    /*you should store the state of all rings at
       the specific memory*/
    /*in an appropriate location*/
    if(n==0) return state;/*the state remains*/
    if(n==1){
        /*change the 1st ring's state*/
        state++;//改变state
        STATE[++times] = state;//存储当前操作后的
            state
        return state/*all rings' state at now*/;
    }
    /*REMOVE the first n-2 rings*/
    /*REMOVE the n-th ring*/
    /*PUT the first n-2 rings*/
    /*REMOVE the first n-1 rings*/
    state = REMOVE(n-2, state);
    state += pow(10,n-1);//将state的第n位变成1(用二
       进制表示时需要改成加2^(n-1))
    STATE[++times] = state;
    state = PUT(n-2, state);
    state = REMOVE(n-1, state);
    return state/*all rings' state at now*/;
}

int PUT(int n,int state){
    /*these two args means*/
    /*(1-n rings will be put,all rings' state at
       now)*/
```

```
    /*the return value means*/
    /*after these operations, what all rings' state
        are*/
    /*you just need to inverse REMOVE*/
    if(n == 0)  return state;
    if(n == 1)
    {
        state--;//改变state
        STATE[++times] = state;//存储当前操作后的
            state
        return state;
    }
    state = PUT(n-2, state);
    state -= pow(10,n-1);
    STATE[++times] = state;
    state = REMOVE(n-2,state);
    state = PUT(n-1, state);
    return state;
}
```

运行上述 C 语言程序得结果如下：

图 1: n = 3

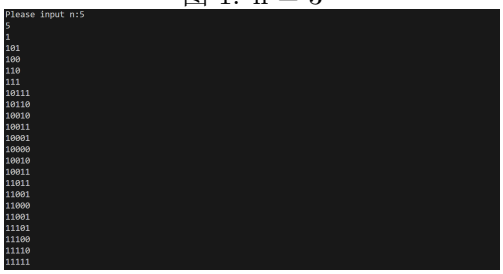

图 2: n = 5

上述结果正确，说明思路正确，然后可以转换为汇编语言。

3.2 流程图

先画出流程图如下：

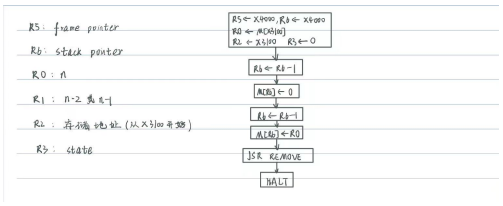

图 3: main

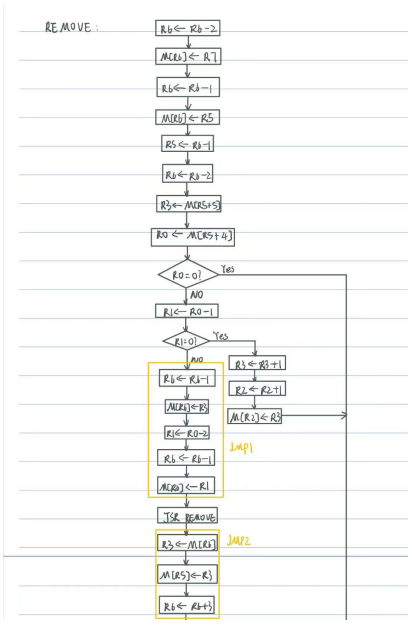

图 4: remove

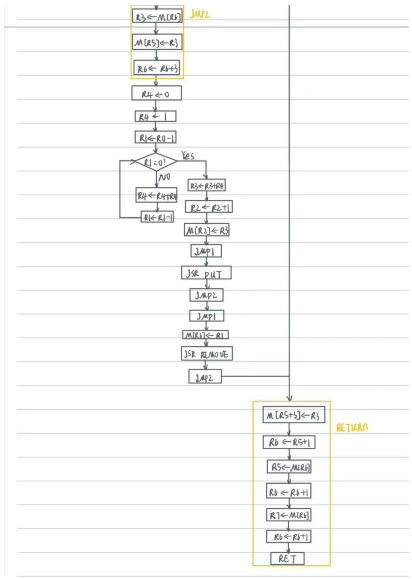

图 5: remove(续)

由于 **PUT** 的操作与 **REMOVE** 的操作几乎一样，只有极少部分需要更改，所以就没有再画其流程图。

3.3 汇编语言实现

根据流程图，可以写出汇编代码如下：

```
    ;
;本题的关键在于使用栈来构造递归子程序REMOVE和PUT
;
            .ORIG   x3000
            LD      R5, BOTTOM          ;R5为frame
                pointer，初始化为x4000
            LD      R6, BOTTOM          ;R6为stack
                pointer，初始化为x4000
            LD      R2, START           ;R2为存储
                state的地址，初始化为x3100
            LDR     R0, R2, #0          ;R0存储n的
                值
            AND     R3, R3, #0          ;R3用于存储
                state

            ADD     R6, R6, #-1         ;
            STR     R3, R6, #0          ;state = 0
                入栈
            ADD     R6, R6, #-1         ;
            STR     R0, R6, #0          ;n入栈
            JSR     REMOVE              ;调用REMOVE
                子程序
            BRnzp   OVER                ;调用完毕后
                程序结束，不需要处理返回值


REMOVE      ADD     R6, R6, #-2         ;为返回值预
    留空间
            STR     R7, R6, #0          ;保存返回地
                址
```

```
                ADD     R6, R6, #-1             ;
                STR     R5, R6, #0              ;保存caller
                    's frame pointer
                ADD     R5, R6, #-1             ;
                ADD     R6, R6, #-2             ;为局部变量
                    state和n分配空间

                LDR     R3, R5, #5              ;R3存储
                    state
                LDR     R0, R5, #4              ;R0存储n
                STR     R3, R5, #0              ;存储局部变
                    量state
                STR     R0, R6, #0              ;存储局部变
                    量n
                BRz     RETURN1
                ADD     R1, R0, #-1
                BRz     STORE1                  ;n=1时单独
                    处理
                JSR     JMP1                    ;做子程序调
                    用前的准备
                JSR     REMOVE                  ;递归调用,
                    REMOVE(n-2,state)
                JSR     JMP2                    ;做子程序调
                    用后的处理
                AND     R4, R4, #0
                ADD     R4, R4, #1              ;R4 <- 1
                ADD     R1, R0, #-1
JUDGE1          BRnp    LOOP1                   ;令R4的第n
    位为1, 后面全为0
                ADD     R3, R3, R4              ;令R3的第n
                    位为1
                ADD     R2, R2, #1              ;存储地址加
                    1
```

```
                STR     R3, R2, #0              ;存储state
                JSR     JMP1
                JSR     PUT                     ;PUT(n-2,
                    state)
                JSR     JMP2
                JSR     JMP1
                ADD     R1, R1, #1              ;JMP1中令R1
                    <- R0-2, 这里需要令R1 <- R0-1, 故还
                    需加1
                STR     R1, R6, #0
                JSR     REMOVE                  ;REMOVE(n
                    -1,state)
                JSR     JMP2                    ;JMP1中令R1
                    <- R0-2, 这里需要令R1 <- R0-1, 故还
                    需加1
                BRnzp   RETURN1



PUT             ADD     R6, R6, #-2             ;为返回值预
    留空间
                STR     R7, R6, #0              ;保存返回地
                    址
                ADD     R6, R6, #-1             ;
                STR     R5, R6, #0              ;保存caller
                    's frame pointer
                ADD     R5, R6, #-1             ;
                ADD     R6, R6, #-2             ;为局部变量
                    state和n分配空间

                LDR     R3, R5, #5              ;R3存储
                    state
                LDR     R0, R5, #4              ;R0存储n
                STR     R3, R5, #0              ;存储局部变
```

```
                量state
            STR     R0, R6, #0          ;存储局部变
                量n
            BRz     RETURN1
            ADD     R1, R0, #-1         ;n=1时单独
                处理，将最后一位由1变成0
            BRz     STORE2
            JSR     JMP1                ;做子程序调
                用前的准备
            JSR     PUT                 ;PUT(n-2,
                state)
            JSR     JMP2
            AND     R4, R4, #0
            ADD     R4, R4, #1
            ADD     R1, R0, #-1
JUDGE2      BRnp    LOOP2               ;R4的第n位
    为1，后面全为0
            NOT     R4, R4
            ADD     R4, R4, #1          ;将R4取反加
                1
            ADD     R3, R3, R4          ;将R3的第n
                位由1变成0
            ADD     R2, R2, #1
            STR     R3, R2, #0
            JSR     JMP1
            JSR     REMOVE              ;REMOVE(n
                -2,state)
            JSR     JMP2
            JSR     JMP1
            ADD     R1, R1, #1          ;JMP1中令R1
                <- R0-2, 这里需要令R1 <- R0-1, 故还
                需加1
            STR     R1, R6, #0
```

```
                JSR     PUT                     ;PUT(n-1,
                    state)
                JSR     JMP2
                BRnzp   RETURN1


LOOP1           ADD     R4, R4, R4
                ADD     R1, R1, #-1
                BRnzp   JUDGE1

LOOP2           ADD     R4, R4, R4
                ADD     R1, R1, #-1
                BRnzp   JUDGE2


STORE1          ADD     R3, R3, #1              ;最后一位由
    0变为1
                ADD     R2, R2, #1              ;存储地址自
                    加1
                STR     R3, R2, #0              ;
                BRnzp   RETURN1                 ;

STORE2          ADD     R3, R3, #-1             ;最后一位由
    1变为0
                ADD     R2, R2, #1              ;存储地址自
                    加1
                STR     R3, R2, #0              ;
                BRnzp   RETURN1                 ;


RETURN1         STR     R3, R5, #3
                ADD     R6, R5, #1
                LDR     R5, R6, #0
```

```
                ADD     R6, R6, #1
                LDR     R7, R6, #0
                ADD     R6, R6, #1
                RET


JMP1            ADD     R6, R6, #-1         ;
                STR     R3, R6, #0          ;存储state
                ADD     R1, R0, #-2         ;R1 = n-2
                ADD     R6, R6, #-1         ;
                STR     R1, R6, #0          ;存储n-2
                RET

JMP2            LDR     R3, R6, #0          ;获得返回值
                STR     R3, R5, #0          ;存储返回值
                LDR     R0, R6, #3
                ADD     R6, R6, #3          ;子程序的返
                    回值和局部变量出栈
                RET


START           .FILL   x3100
BOTTOM          .FILL   x4000

OVER            HALT
                .END
```

4 实验结果