حسین انجیدنی - 400100746

سوال چهارم

در این کد اسمبلی برای پیاده سازی برج هانوی از استک استفاده کردم چرا که پیاده سازی بازگشتی در اسمبلی با توجه به محدودیت ذخیره تعداد بار های return از یک تابع غیرممکن است پس باید از روشی که من اینجا از استک استفاده کردم آن را غیر بازگشتی کنیم.

برا استفاده از استک باید داده ساختار آن را پیاده کنیم که شامل این توابع است:

createStack, isFull, isEmpty, push, pop

اکنون هر تابع را جدا از کد می آورم:

:Create stack

این تابع استک را میسازد و حافظه ای را به آن اختصاص میدهد. ابتدا تعداد المان ها را ذخیره میکند و سپس فضایی را برای هر یک مشخص میکند.

```
createStack:
    push    {fp, lr}
    add     fp, sp, #4
    sub     sp, sp, #16
    str     r0, [fp, #-16]
    mov     r0, #12
    bl      malloc
    mov     r3, r0
    str     r3, [fp, #-8]
    ldr     r3, [fp, #-8]
    ldr     r2, [fp, #-16]
    str     r2, [r3]
    ldr     r3, [fp, #-8]
    mvn     r2, #0
    str     r2, [r3, #4]
    ldr     r3, [fp, #-8]
    ldr     r3, [r3]
    lsl     r3, r3, #2
    mov     r0, r3
    bl      malloc
    mov     r3, r0
    mov     r2, r3
    ldr     r3, [fp, #-8]
    str     r2, [r3, #8]
    ldr     r3, [fp, #-8]
    mov     r0, r3
    sub     sp, fp, #4
    pop     {fp, pc}
```

:isFull

در این تابع برسی میکنیم که آیا تعداد اعضای قرار گرفته درون استک برابر با تعداد اعضای کنونی استک هست یا خیر؟

```
isFull:
    str     fp, [sp, #-4]!
    add     fp, sp, #0
    sub     sp, sp, #12
    str     r0, [fp, #-8]
    ldr     r3, [fp, #-8]
    ldr     r2, [r3, #4]
    ldr     r3, [fp, #-8]
    ldr     r3, [r3]
    sub     r3, r3, #1
    cmp     r2, r3
    moveq   r3, #1
    movne   r3, #0
    and     r3, r3, #255
    mov     r0, r3
    add     sp, fp, #0
    ldr     fp, [sp], #4
    bx      lr
```

:isEmpty

این تابع همانند تابع قبل است فقط صفر بودن را مورد بررسی قرار میدهد.

```
isEmpty:
    str     fp, [sp, #-4]!
    add     fp, sp, #0
    sub     sp, sp, #12
    str     r0, [fp, #-8]
    ldr     r3, [fp, #-8]
    ldr     r3, [r3, #4]
    cmn     r3, #1
    moveq   r3, #1
    movne   r3, #0
    and     r3, r3, #255
    mov     r0, r3
    add     sp, fp, #0
    ldr     fp, [sp], #4
    bx      lr
```

:Push

```
push:
    push    {fp, lr}
    add     fp, sp, #4
    sub     sp, sp, #8
    str     r0, [fp, #-8]
    str     r1, [fp, #-12]
    ldr     r0, [fp, #-8]
    bl      isFull
    mov     r3, r0
    cmp     r3, #0
    bne     .L10
    ldr     r3, [fp, #-8]
    ldr     r2, [r3, #8]
    ldr     r3, [fp, #-8]
    ldr     r3, [r3, #4]
    add     r1, r3, #1
    ldr     r3, [fp, #-8]
    str     r1, [r3, #4]
    ldr     r3, [fp, #-8]
    ldr     r3, [r3, #4]
    lsl     r3, r3, #2
    add     r3, r2, r3
    ldr     r2, [fp, #-12]
    str     r2, [r3]
    b       .L7
.L10:
    nop
.L7:
    sub     sp, fp, #4
    pop     {fp, pc}
```

پس از بررسی آنکه استک پر نباشد، یکی از خانه هایی که به این استک اختصاص داده شده را انتخاب میکنیم و این را به جایش قرار میدهیم.

:Pop

بالایی ترین مقدار استک را در صورت انکه استک خالی نباشد برمیگردانند.

```
pop:
    push    {fp, lr}
    add     fp, sp, #4
    sub     sp, sp, #8
    str     r0, [fp, #-8]
    ldr     r0, [fp, #-8]
    bl      isEmpty
    mov     r3, r0
    cmp     r3, #0
    beq     .L12
    mvn     r3, #0
    b       .L13
.L12:
    ldr     r3, [fp, #-8]
    ldr     r2, [r3, #8]
    ldr     r3, [fp, #-8]
    ldr     r3, [r3, #4]
    sub     r0, r3, #1
    ldr     r1, [fp, #-8]
    str     r0, [r1, #4]
    lsl     r3, r3, #2
    add     r3, r2, r3
    ldr     r3, [r3]
.L13:
    mov     r0, r3
    sub     sp, fp, #4
    pop     {fp, pc}
```

توابع کمکی L12 و L13 به ترتیب کار اصلی در pop را انجام میدهند در تابع اول در صورت خالی نبودن استک، آدرس اولین خانه ان را پیدا میکند و تا سطر آن بالا میرود (با توجه به انکه تعداد اعضا را ذخیره میکنیم میتوان از اندازه تعداد اعضا صربدر سایز هر عضو استفاده کرد) و آن را انتخاب میکند و از سایز استک میکاهد. و تابع دوم نیز پس از تمامی عملیات ها و بررسی شدن وضعیت استک مقدار پاپ شده یا در صورت خالی بودن منهای یک را نمایش میدهد.

اکنون که پیاده سازی استک به پایان رسید با استفاده از آن به حل قسمت اصلی سوال میپردازیم.

: Move Disk

در این تابع ابتدا دو استک فرض شده و با اتسفاده از آن ها طبق منطق بازگشتی با استفاده از سه استک fp , lr, pc چابه جایی ها انجام میشود

```
moveDisk:
    push    {fp, lr}
    add     fp, sp, #4
    sub     sp, sp, #16
    str     r0, [fp, #-8]
    str     r1, [fp, #-12]
    str     r2, [fp, #-16]
    str     r3, [fp, #-20]
    ldr     r0, [fp, #-8]
    bl      pop
    mov     r3, r0
    mov     r1, r3
    ldr     r0, [fp, #-12]
    bl      push
    ldr     r3, [fp, #-8]
    ldr     r3, [r3]
    add     r2, r3, #65
    ldr     r3, [fp, #-8]
    ldr     r3, [r3, #4]
    sub     r3, r2, r3
    sub     r1, r3, #2
    ldr     r3, [fp, #-12]
    ldr     r3, [r3]
    add     r2, r3, #65
    ldr     r3, [fp, #-12]
    ldr     r3, [r3, #4]
    sub     r3, r2, r3
    sub     r3, r3, #1
    mov     r2, r1
    ldr     r1, [fp, #-16]
    ldr     r0, .L15
    bl      printf
    ldr     r3, [fp, #-20]
    ldr     r3, [r3]
    add     r2, r3, #1
    ldr     r3, [fp, #-20]
    str     r2, [r3]
    nop
    sub     sp, fp, #4
    pop     {fp, pc}
.L16:
    .align  2
.L15:
    .word   .LC0
```

همچنین در این تابع با توجه به استفاده از تابع print که انتقال ها را نیز نمایش میدهد.

:toweOfHanoi

```
towerOfHanoi:
    push    {fp, lr}
    add     fp, sp, #4
    sub     sp, sp, #24
    str     r0, [fp, #-8]
    str     r1, [fp, #-12]
    str     r2, [fp, #-16]
    str     r3, [fp, #-20]
    ldr     r3, [fp, #-8]
    cmp     r3, #1
    bne     .L18
    ldr     r3, [fp, #4]
    mov     r2, #1
    ldr     r1, [fp, #-16]
    ldr     r0, [fp, #-12]
    bl      moveDisk
    b       .L17
```

```
.L18:
    ldr     r3, [fp, #-8]
    sub     r0, r3, #1
    ldr     r3, [fp, #4]
    str     r3, [sp]
    ldr     r3, [fp, #-16]
    ldr     r2, [fp, #-20]
    ldr     r1, [fp, #-12]
    bl      towerOfHanoi
    ldr     r3, [fp, #4]
    ldr     r2, [fp, #-8]
    ldr     r1, [fp, #-16]
    ldr     r0, [fp, #-12]
    bl      moveDisk
    ldr     r3, [fp, #-8]
    sub     r0, r3, #1
    ldr     r3, [fp, #4]
    str     r3, [sp]
    ldr     r3, [fp, #-12]
    ldr     r2, [fp, #-16]
    ldr     r1, [fp, #-20]
    bl      towerOfHanoi
.L17:
    sub     sp, fp, #4
    pop     {fp, pc}
```

در این تابع دقیقا ترتیب بازگشتی ای که باید در حالتی که برنامه با یک زبان سطح بالتر مانند C نوشته میشد (بدین شکل تصور کردم که هر بازگشت به درخت بازگشت ما سه نود اضافه میکند و تا اخر نود پایین رفتم سپس از انجا کد ها را طوری نوشتم که به ترتیب معکوس به بالا برگردد)

:Main

در این تابع صرفا استک ها را ایجاد کردیم و مقدار های ورودی را گرفتیم، سپس در میله (استک) مقادیر مربوطه را گذاتیم و تابع towerOfHanoi را اجرا کردیم.

اکنون خروجی کد را در درج میکنم:

```
hosein@hoseinonthego:~/Emb/HW03/Q2$ ./a.out
Enter the number of disks: 5
Move disk 1 from rod A to rod E
Move disk 2 from rod B to rod E
Move disk 1 from rod E to rod D
Move disk 3 from rod C to rod E
Move disk 1 from rod D to rod C
Move disk 2 from rod E to rod D
Move disk 1 from rod C to rod C
Move disk 4 from rod D to rod E
Move disk 1 from rod C to rod D
Move disk 2 from rod D to rod D
Move disk 1 from rod D to rod C
Move disk 3 from rod E to rod D
Move disk 1 from rod C to rod E
Move disk 2 from rod D to rod C
Move disk 1 from rod E to rod B
Move disk 5 from rod E to rod E
Move disk 1 from rod B to rod E
Move disk 2 from rod C to rod D
Move disk 1 from rod E to rod C
Move disk 3 from rod D to rod E
Move disk 1 from rod C to rod D
Move disk 2 from rod D to rod D
Move disk 1 from rod D to rod C
Move disk 4 from rod E to rod D
Move disk 1 from rod C to rod C
Move disk 2 from rod D to rod E
Move disk 1 from rod C to rod D
Move disk 3 from rod E to rod C
Move disk 1 from rod D to rod E
Move disk 2 from rod E to rod B
Move disk 1 from rod E to rod A
Total number of moves: 31
hosein@hoseinonthego:~/Emb/HW03/Q2$ S
```