hw7 key

by LYC

(8') **8.1 What are the defining characteristics of a stack?**

LIFO (Last in First Out)

(表达到类似意思即可)

(25') **8.7 Rewrite the PUSH and POP routines to handle stack elements of arbitrary sizes.**

```
; Subroutines for carrying out the PUSH and POP functions. This
; program works with a stack consisting of memory locations x3FFF
; (BASE) through x3FFB (MAX). R6 is the stack pointer. R3 contains
; the size of the stack element. R4 is a pointer specifying the
; location of the element to PUSH from or the space to POP to.
;
POP         ST R2, Save2 ; are needed by POP.
            ST R1, Save1
            ST R0, Save0
            LD R1, BASE ; BASE contains -x3FFF.
            ADD R1, R1, #-1 ; R1 contains -x4000.
            ADD R2, R6, R1 ; Compare stack pointer to x4000
            BRz fail_exit ; Branch if stack is empty.
            ADD R0, R4, #0
            ADD R1, R3, #0
            ADD R5, R6, R3
            ADD R5, R5, #-1
            ADD R6, R6, R3
pop_loop    LDR R2, R5, #0
            STR R2, R0, #0
            ADD R0, R0, #1
            ADD R5, R5, #-1
            ADD R1, R1, #-1
            BRp pop_loop
            BRnzp success_exit
PUSH        ST R2, Save2 ; Save registers that
            ST R1, Save1 ; are needed by PUSH.
            ST R0, Save0
            LD R1,MAX ; MAX contains -x3FFB
            ADD R2,R6,R1 ; Compare stack pointer to -x3FFB
            BRz fail_exit ; Branch if stack is full.
            ADD R0, R4, #0
            ADD R1, R3, #0
            ADD R5, R6, #-1
            NOT R2, R3
            ADD R2, R2, #1
            ADD R6, R6, R2
push_loop   LDR R2, R0, #0
```

```
             STR R2, R5, #0
             ADD R0, R0, #1
             ADD R5, R5, #-1
             ADD R1, R1, #-1
             BRp push_loop
success_exit LD R0, Save0
             LD R1, Save1 ; Restore original
             LD R2, Save2 ; register values.
             AND R5, R5, #0 ; R5 <-- success.
             RET
fail_exit    LD R0, Save0
             LD R1, Save1 ; Restore original
             LD R2, Save2 ; register values.
             AND R5, R5, #0
             ADD R5, R5, #1 ; R5 <-- failure.
             RET
BASE         .FILL xC001 ; BASE contains -x3FFF.
MAX          .FILL xC005
Save0        .FILL x0000
Save1        .FILL x0000
Save2        .FILL x0000
```

(若没有通过注释或者其它文字说明清楚如何表示任意大小元素，如用哪个寄存器表示输入大小或指针，只有汇编语句，扣5‘；没有保护寄存器扣5’；没有溢出判断扣3‘；栈指针不是R6扣3’；汇编语句错误如LDR写成LD一句扣3‘；其它错误情况酌情扣分)

(如果是通过调用原PUSH和POP来实现本题也算对)

(照抄课本P279原PUSH和POP程序，改都不改，全扣)

(如与本答案完全一致也给分，不过要是抄都没抄完整就多扣点)

(15') **8.8 The following operations are performed on a stack:**

PUSH A, PUSH B, POP, PUSH C, PUSH D, POP, PUSH E, POP, POP, PUSH F

a. What does the stack contain after the PUSH F?

b. At which point does the stack contain the most elements? Without removing the elements left on the stack from the previous operations, we perform:

PUSH G, PUSH H, PUSH I, PUSH J, POP, PUSH K, POP, POP, POP, PUSH L, POP, POP, PUSH M

c. What does the stack contain now?

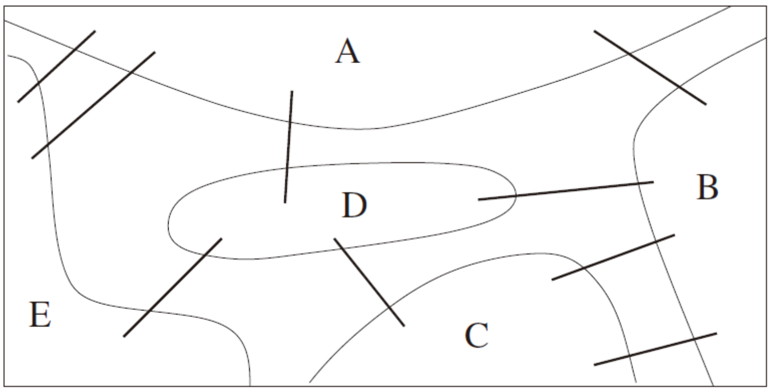
a. (bottom) AF (top)

b. after PUSH J and after PUSH K

c. (bottom) AFM (top)

(每问5’，b小问两个PUSH都要回答到，若只回答其中一个则扣2‘)

(25') **8.12 Many cities, like New York City, Stockholm, Konigsberg, etc., consist of several areas connected by bridges. The following figure shows a map of FiveParts, a city made up of five areas A, B, C, D, E, with the areas connected by nine bridges as shown.**



The following program prompts the user to enter two areas and then stores the number of bridges from the first area to the second in location x4500. Your job: On the next page, design the data structure for the city of FiveParts that the following program will use to count the number of bridges between two areas.

```
            .ORIG x3000
            LEA R0, FROM
            TRAP x22
            TRAP x20      ; Inputs a char without banner
            NOT R1, R0
            ADD R1, R1, #1
            LEA R0, TO
            TRAP x22
            TRAP x20
            NOT R0, R0
            ADD R0, R0, #1
            AND R5, R5, #0
            LDI R2, HEAD
SEARCH      BRz DONE
            LDR R3, R2, #0
            ADD R7, R1, R3
            BRz FOUND_FROM
            LDR R2, R2, #1
            BRnzp SEARCH
FOUND_FROM  ADD R2, R2, #2
NEXT_BRIDGE LDR R3, R2, #0
            BRz DONE
            LDR R4, R3, #0
            ADD R7, R0, R4
            BRnp SKIP
            ADD R5, R5, #1  ; Increment Counter
SKIP        ADD R2, R2, #1
            BRnzp NEXT_BRIDGE
DONE        STI R5, ANSWER
            HALT
HEAD        .FILL x3050
ANSWER      .FILL x4500
FROM        .STRINGZ "FROM: "
TO          .STRINGZ "TO: "
            .END
```

Your job is to provide the contents of the memory locations that are needed to specify the data structure for the city of FiveParts, which is needed by the program on the previous page. We have given you the HEAD pointer for the data structure and in addition, five memory locations and the contents of those five locations. We have also supplied more than enough sequential memory locations after each of the five to enable you to finish the job. Use as many of these memory locations as you need.

```
; NODE STRUCTURE
; this city name
; next node address
; bridged city X's node address, bridged city Y's node address, ...(end with
NULL)
        .ORIG x3050
        .FILL A
A       .FILL x0041 ; 'A'
        .FILL B ; next node
        .FILL B
        .FILL D
        .FILL E
        .FILL E
        .FILL x0000
B       .FILL x0042 ; 'B'
        .FILL C ; next node
        .FILL A
        .FILL C
        .FILL C
        .FILL D
        .FILL x0000
C       .FILL x0043 ; 'C'
        .FILL D ; next node
        .FILL B
        .FILL B
        .FILL D
        .FILL x0000
D       .FILL x0044 ; 'D'
        .FILL E ; next node
        .FILL A
        .FILL B
        .FILL C
        .FILL E
        .FILL x0000
E       .FILL x0045 ; 'E'
        .FILL x0000 ; null
        .FILL A
        .FILL A
        .FILL D
        .FILL x0000
        .END
```

A	x4000	x0041
	x4001	x4100
	x4002	x4100
	x4003	xA243
	x4004	xBBBB
	x4005	xBBBB
	x4006	x0000

x3050	x4000

D	xA243	x0042
	xA244	xBBBB
	xA245	x4000
	xA246	x4100
	xA247	x3100
	xA248	xBBBB
	xA249	x0000

B	x4100	x0043
	x4101	x3100
	x4102	x4000
	x4103	x3100
	x4104	x3100
	x4105	xA243
	x4106	x0000

E	xBBBB	x0044
	xBBBC	x0000
	xBBBD	x4000
	xBBBE	x4000
	xBBBF	xA243
	xBBC0	x0000
	xBBC1	

C	x3100	x0045
	x3101	xA243
	x3102	x4100
	x3103	x4100
	x3104	xA243
	x3105	x0000
	x3106	

(以其它形式表示结构也可。注意HEAD处是第一个节点的地址，若x3050直接是第一个节点扣3′；每个节点第一个位置是ASCII码，若不是扣5′；漏写next扣5′；E节点的next是NULL，若不是扣3′；每个节点最后是否以NULL结尾，若不是扣5′；每个节点所存放的与之相邻的节点是指针，若为ASCII码扣5′；两个节点间桥梁数目不正确，每处扣2′；其它错误情况酌情扣分)

(27') **8.14 As you know, the LC-3 ADD instruction adds 16-bit 2’s complement integers. If we wanted to add 32-bit 2’s complement integers, we could do that with the program shown next. Note that the program requires calling subroutine X, which stores into R0 the carry that results from adding R1 and R2. Fill in the missing pieces of both the program and the subroutine X, as identified by the empty boxes. Each empty box corresponds to one instruction or the operands of one instruction.**

Note that a 32-bit operand requires two 16-bit memory locations. A 32-bit operand Y has Y[15:0] stored in address A, and Y[31:16] stored in address A+1.

```
.ORIG x3000
        LEA R3, NUM1
        LEA R4, NUM2
        LEA R5, RESULT
        LDR R1, R3, #0
        LDR R2, R4, #0
        ADD R0, R1, R2
        STR R0, R5, #0
        ---------------(a)
        LDR -----------(b)
        LDR -----------(c)
        ADD R1, -------(d)
        ---------------(e)
        STR R0, -------(f)
        TRAP x25        ;<===added

X       ST R4, SAVER4
        AND R0, R0, #0
        AND R4, R1, R2  ;<===added
        BRn -----------(g)
        ADD R1, R1, #0
        BRn -----------(h)
        ADD -----------(i)
        BRn ADDING
        BRnzp EXIT
ADDING  ADD R4, R1, R2
        BRn EXIT
LABEL   ADD R0, R0, #1
EXIT    LD R4, SAVER4
        RET

NUM1    .BLKW 2
NUM2    .BLKW 2
```

```
RESULT  .BLKW 2
SAVER4  .BLKW 1

.END
```

(a) JSR X

(b) R1, R3, #1

(c) R2, R4, #1 ((b),(c)可交换顺序)

(d) R0, R1 (or R1, R0) (或 R1, R2 or R2, R1)

(e) ADD R0, R1, R2 (or ADD R0, R2, R1) (或 ADD R0, R0, R1 or ADD R0, R1, R0)

(f) R5, #1

(g) LABLE

(h) ADDING

(i) R2, R2, #0

(每空3‘，注意(a)不是BR)