8.2 8.8 7.30 8.10

8.2

What is an advantage to using the model in Figure 8.9 to implement a stack vs. the model in Figure 8.8?

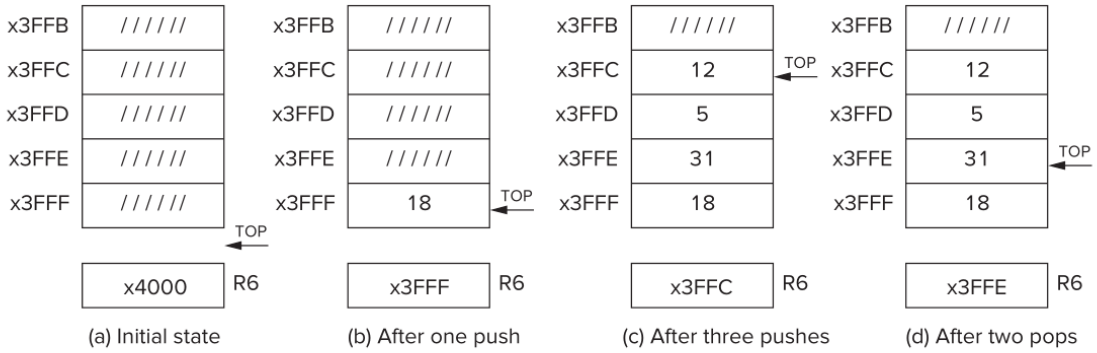

Figure 8.9 A stack, implemented in memory—data entries do not move.

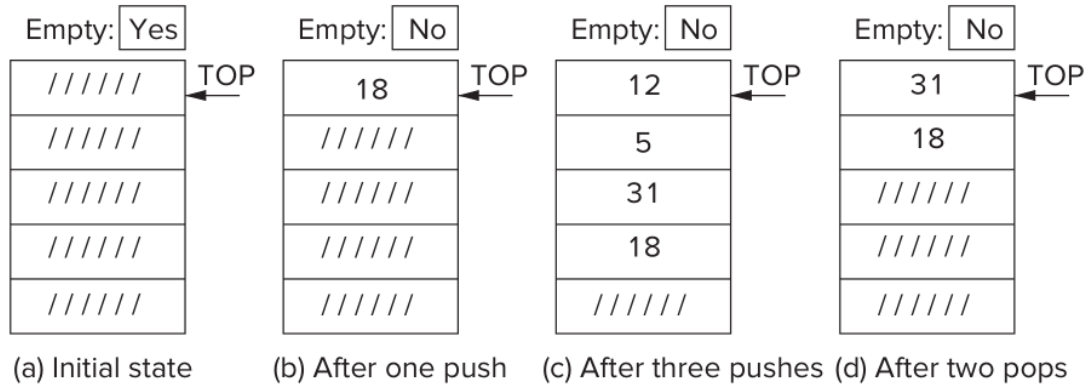

Figure 8.8 A stack, implemented in hardware—data entries move.

A distinguishing feature of the stack of Figure 8.8 is that, like the quarters in the coin holder, as each value is added or removed, all the other values already on the stack move.

unlike the coin holder and computer hardware stack implementations discussed in the previous section (Figure 8.8) ,when values are pushed and popped to and from a stack implemented in sequential memory locations,the data already
stored on the stack ***does not physically move.****(Figure 8.9)* ---From BOOK

Besides，From BOOK

as we will see momentarily, those values 5 and 12 cannot be accessed from memory, as long as every access to memory is controlled by the stack mechanism.

We know that Figure 8.9 don't need extra condition bit and don't need data move cost when push and pop operation

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	**A**
B	AB
POP	A
C	AC
D	ACD
POP	AC
E	ACE
POP	AC
POP	A
F	AF

a:

F(top)
A

b. most elements are

AFGHIJ after Push J

AFGHIK after Push K

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

G	**AFG**
H	AFGH
I	AFGHI
J	AFGHIJ

G	AFG
pop	AFGHI
K	AFGHIK
pop	AFGHI
pop	AFGH
pop	AFG
L	AFGL
pop	AFG
pop	AF
M	AFM

So Finally Stack is

M(top)
F
A

7.30

if (p == StackEND) p = StackBASE

p--,*p=R0 // push R0 to stack

if(size -5 != 0)size++

```
.orig x3000
ADD R0,R0,#1 ;TEST PART
JSR PUSH
ADD R0,R0,#1
JSR PUSH
ADD R0,R0,#1
JSR PUSH
ADD R0,R0,#1
JSR PUSH
ADD R0,R0,#1
JSR PUSH
ADD R0,R0,#1
JSR PUSH
ADD R0,R0,#1

HALT
PUSH
ST R1, SAVER
```

```
LD R1, NEGFULL  ;R1 =  -3FFB
ADD R1, R6, R1  ;R1 = p -3FFB
BRp SKIP        ;(a)    p - 3FFB ==0 means FULL
LD R6, BASE     ;p = x4000 When FULL ,change p = BASE
SKIP
ADD R6, R6, #-1 ;p--
LD R1, MINUS5   ;R1 = -5
ADD R1, R5, R1  ;R1 = R5 - 5
BRz END         ;R5 == 5 END means FULL
ADD R5, R5, #1  ;(b)    else R5++ means size++
END
STR R0,R6,#0    ;(c)
LD R1,SAVER     ;(d)
RET
NEGFULL .FILL xC005 ; x-3FFB
MINUS5 .FILL xFFFB ; #-5
BASE .FILL x4000
SAVER .BLKW #1
.end
```

BRp SKIP (a)

ADD R5, R5, #1 (b)

STR R0,R6,#0 (c)

LD R1,SAVER (d)