F.10 Chapter 8 Solutions

8.1 The defining characteristic of a stack is the unique specification of how it is to be accessed. Stack is a LIFO (Last in First Out) structure. This means that the last thing that is put in the stack will be the first one to get out from the stack.

8.3 (a) PUSH R1

(b) POP R0

(c) PUSH R3

(d) POP R7

8.5 One way to check for overflow and underflow conditions is to keep track of a pointer that tracks the bottom of the stack. This pointer can be compared with the address of the first and last addresses of the space allocated for the stack.

```
;
; 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 bottom of the stack.
;

POP             ST  R1, Save1 ; are needed by
                POP. ST    R2, Save2
                ST  R3, Save3
                LD  R1, NBASE   ; BASE contains
                -x3FFF. ADD R1, R1, #-1 ; R1 contains
                -x4000.
                ADD R2, R6, R1  ; Compare bottom of stack to

                x4000 BRz fail_exit   ; Branch if stack is

                empty.
```

```
                LD  R1, BAS     ;Iterate from the top
                    E       of
                            ;the stack
                LDI R0, BAS     ;Load the value from
                    E       the
                NOT R3, R6      ;top of stack
                ADD R3, R3, #1  ;Generate the
                            ;negative of the
                            ;bottom-of-stack
                            pointer
                ADD R6, R6, #1  ;Increment the
                            ;bottom-of-stack
                            ;pointer

pop_loop        ADD   R2, R1, R3  ;Compare iterating
                            ;pointer to
                            ;bottom-of-stack
                pointer BRz success_exit;Branch if no
                more
                            ;entries to shift
                LDR   R2, R1, #-1  ;Load the entry to
                shift STR         R2, R1, #0     ;Shift
                the entry
                ADD   R1, R1, #-1  ;Increment the
                            ;iterating pointer
                BRnzp pop_loop

PUSH            ST  R1,  Save1  ;  Save  registers
                that ST R2, Save2 ; are needed by
                PUSH. ST R3, Save3
                LD R1, MAX ; MAX contains -x3FFB
                ADD R2, R6, R1 ; Compare stack pointer to -x3FFB
                BRz fail_exit ; Branch if stack is full.

                ADD R1, R6, #0 ;Iterate from the bottom
                            ;of stack
                LD  R3, NBASE   ;NBASE contains
                            ;-x3FFF
                ADD R3, R3, #-1   ; R3 = -x4000

push_loop       ADD   R2, R1, R3  ;Compare iterating
                            ;pointer to
                            ;bottom-of-stack
                pointer BRz       push_entry     ;Branch
                if no more
                            ;entries to shift
                LDR   R2, R1, #0  ;Load the entry to
                shift STR         R2, R1, #-1
                ;Shift the entry
                ADD   R1, R1, #1 ;Decrement the
                            ;iterating pointer
                BRnzp push_loop
```

Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.

```
push_entry ADD R6, R6, #-1 ;Increment the
                                ;bottom-of-stack
                pointer STI R0, BASE ;Push a value onto
                stack BRnzp success_exit

success_exit LD R1, Save1 ;Restore original
                LD   R2, Save2   ;register
                values LD        R3, Save3
                AND  R5, R5, #0  ;R5 <---
                success RET

fail_exit       LD   R1, Save1   ;Restore original
                LD   R2, Save2   ;register
                values LD        R3, Save3
                AND  R5, R5, #0
                ADD  R5, R5, #1  ;R5 <---
                failure RET

BASE            .FILL x3FFF
NBASE           .FILL xC001 ; NBASE contains -x3FFF.
MAX             .FILL xC005

Save1           .FILL x0000
Save2           .FILL x0000
Save3           .FILL x0000
```

8.7
```
; 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
                BR      -x3FFB fail_exit ; Branch if stack is
                z       full.
                AD      R0, R4, #0
                D       R1, R3, #0
                AD      R5, R6, #-1
                D
                AD
                D

                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
```

Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.

```
Save1           .FILL   x0000
Save2           .FILL   x0000
```

8.9 (a) BDECJKIHLG

(b)
```
Push Z
Push Y
Pop Y
Push X
Pop X
Push W
Push V
Pop V
Push U
Pop U
Pop W
Pop Z
Push T
Push S
Pop S
Push R
Pop R
Pop T
```

(c) 14 different output streams.

8.11 16 memory locations are needed for the assembled program.

Address of C = **x400F**

After execution, C contains the **average** of the four consecutive values starting at memory location specified in B.

8.13
```
FACT    ST R1, SAVE_R1
              ADD R1, R0, #0
              BRnp SKIP
              ADD R1, R1, #1
              BRnzp DONE
SKIPADD R0, R0, #-1
              BRz DONE
AGAIN   MUL R1, R1, R0
              ADD R0, R0, #-1
              BRnp AGAIN
DONE    ADD R0, R1, #0
              LD R1, SAVER1
              RET
SAVE_R1 .BLKW 1
```

8.15 NOTE: There is an error in the statement of this problem. See Errata Sheet for question. Additionally, this problem would belong in Chapter 9 rather than Chapter 8.

MAR	MDR
x400E	x5020
x400F	xF0F0
x1FFF	**x8002**
x1FFE	x4010
x00F0	**x2000**
x2000	x71BF
x1FFD	**x0000**
x2001	x8000
x1FFE	**x4010**
x1FFF	**x8002**
x4010	xF025

Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.