CH10: A Calculator

Main Algorithm

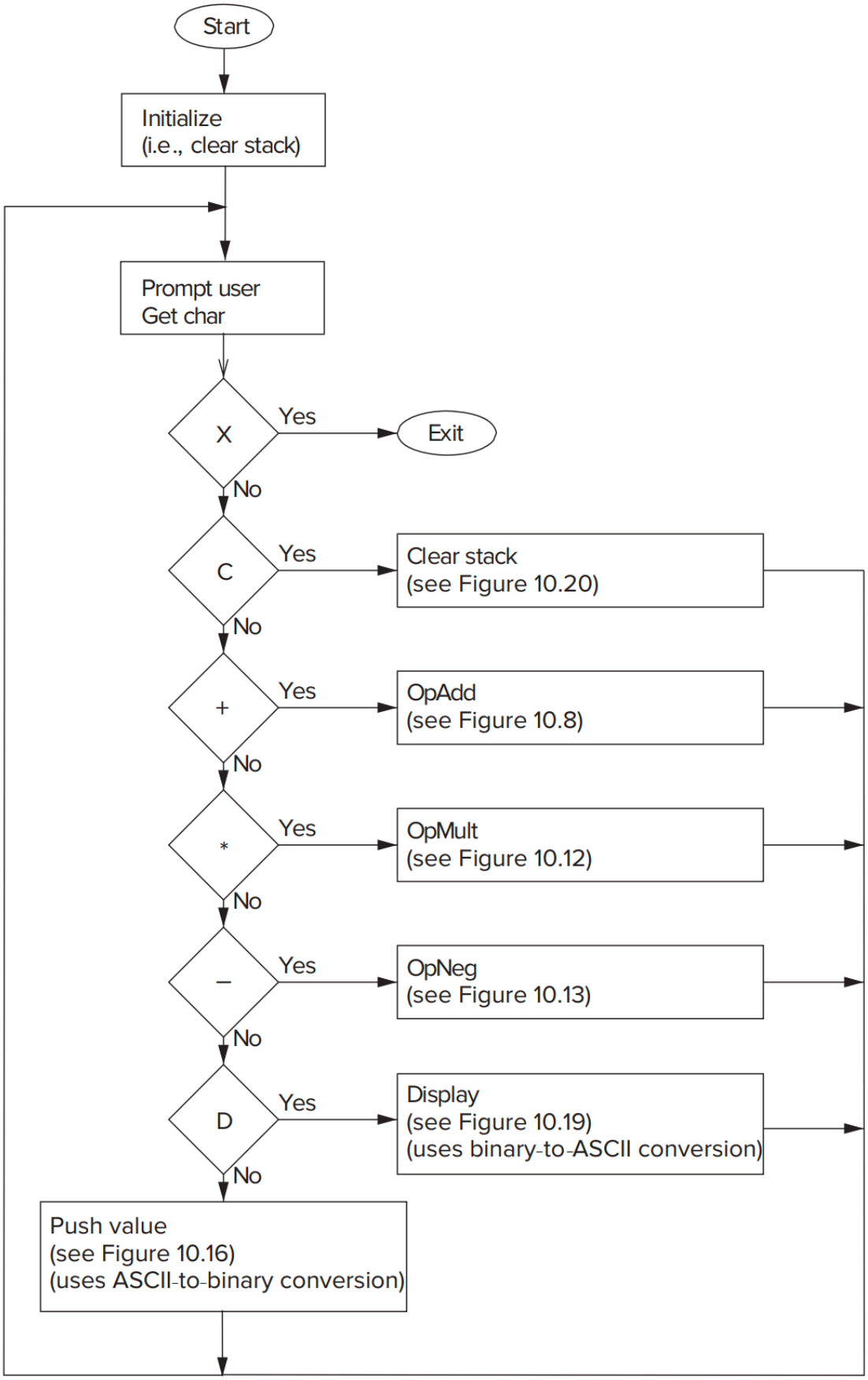

Figure 10.14 The calculator, overview.

```
01    ;
02    ;  The Calculator, Main Algorithm
03    ;
04                LEA       R6,StackBase   ; Initialize the Stack Pointer.
05                ADD       R6,R6,#1       ; R6 = StackBase + 1 --> empty stack
```

```
06
07   NewCommand  LEA       R0,PromptMsg
08               PUTS
09               GETC
0A               OUT
0B   ;
0C   ; Check the command
0D   ;
0E   TestX       LD        R1,NegX       ; Check for X.
0F               ADD       R1,R1,R0
10               BRnp      TestC
11               HALT
12   ;
13   TestC       LD        R1,NegC       ; Check for C.
14               ADD       R1,R1,R0
15               BRnp      TestAdd
16               JSR       OpClear       ; See Figure 10.20
17               BRnzp     NewCommand
18   ;
19   TestAdd     LD        R1,NegPlus    ; Check for +
1A               ADD       R1,R1,R0
1B               BRnp      TestMult
1C               JSR       OpAdd         ; See Figure 10.8
1D               BRnzp     NewCommand
1E   ;
1F   TestMult    LD        R1,NegMult    ; Check for *
20               ADD       R1,R1,R0
21               BRnp      TestMinus
22               JSR       OpMult        ; See Figure 10.12
23               BRnzp     NewCommand
24   ;
25   TestMinus   LD        R1,NegMinus   ; Check for -
26               ADD       R1,R1,R0
27               BRnp      TestD
28               JSR       OpNeg         ; See Figure 10.13
29               BRnzp     NewCommand
2A   ;
2B   TestD       LD        R1,NegD       ; Check for D
2C               ADD       R1,R1,R0
2D               BRnp      EnterNumber
2E               JSR       OpDisplay     ; See Figure 10.19
2F               BRnzp     NewCommand
30   ;
31   ; Then we must be entering an integer
32   ;
33   EnterNumber JSR       PushValue     ; See Figure 10.16
34               BRnzp     NewCommand
35   ;
36   PromptMsg   .FILL     x000A
37               .STRINGZ "Enter a command:"
38   NegX        .FILL     xFFA8
39   NegC        .FILL     xFFBD
3A   NegPlus     .FILL     xFFD5
3B   NegMinus    .FILL     xFFD3
3C   NegMult     .FILL     xFFD6
3D   NegD        .FILL     xFFBC
3E
3F   ; Globals
40   StackMax    .BLKW     #9
41   StackBase   .BLKW     #1
42   ASCIIBUFF   .BLKW     #4
43               .FILL     x0000 ; ASCIIBUFF sentinel
```

Figure 10.15 The calculator's main algorithm.

OpAdd

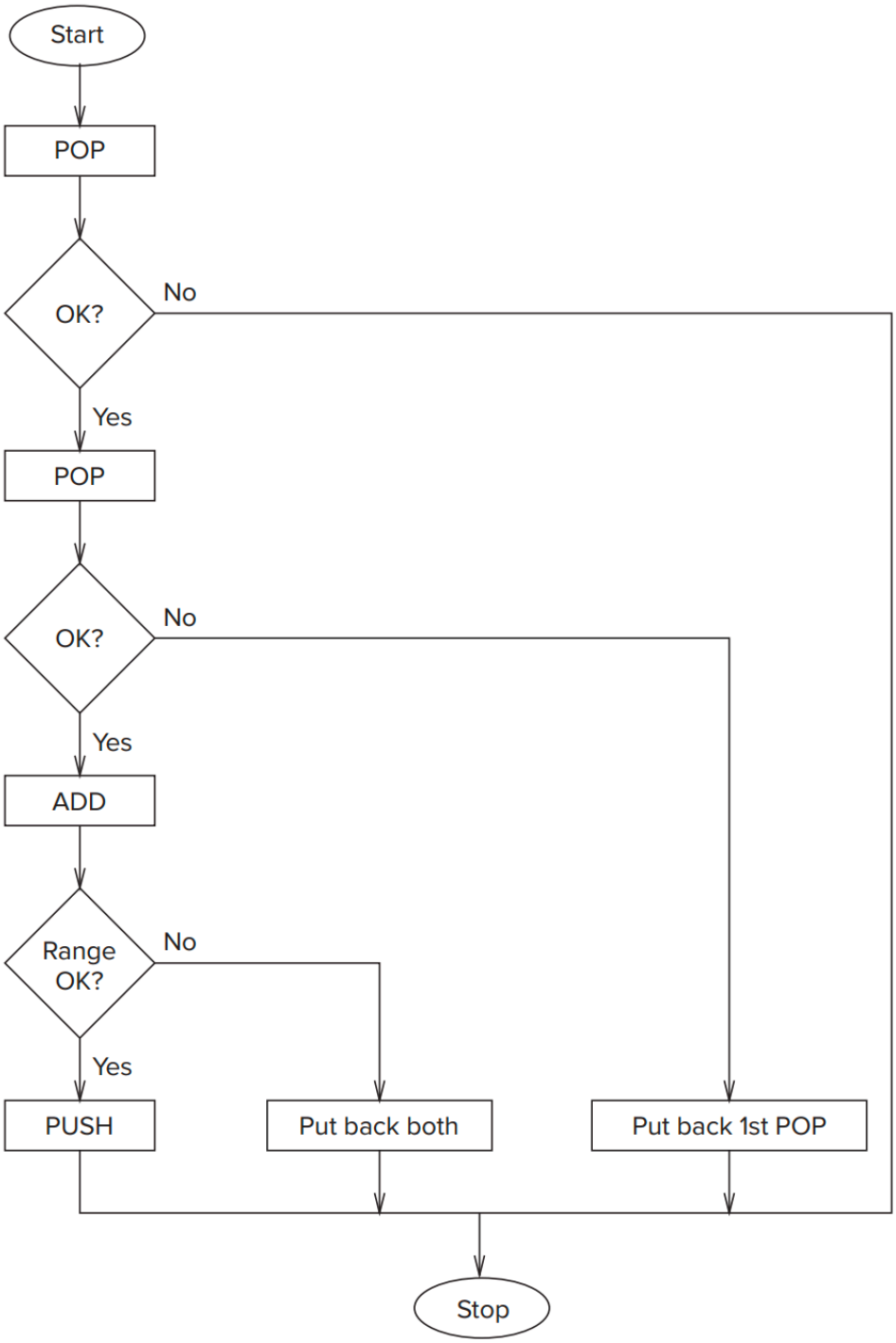

Figure 10.7 Flowchart for OpAdd algorithm.

> Push back can be `ADD R6,R6,#-1`

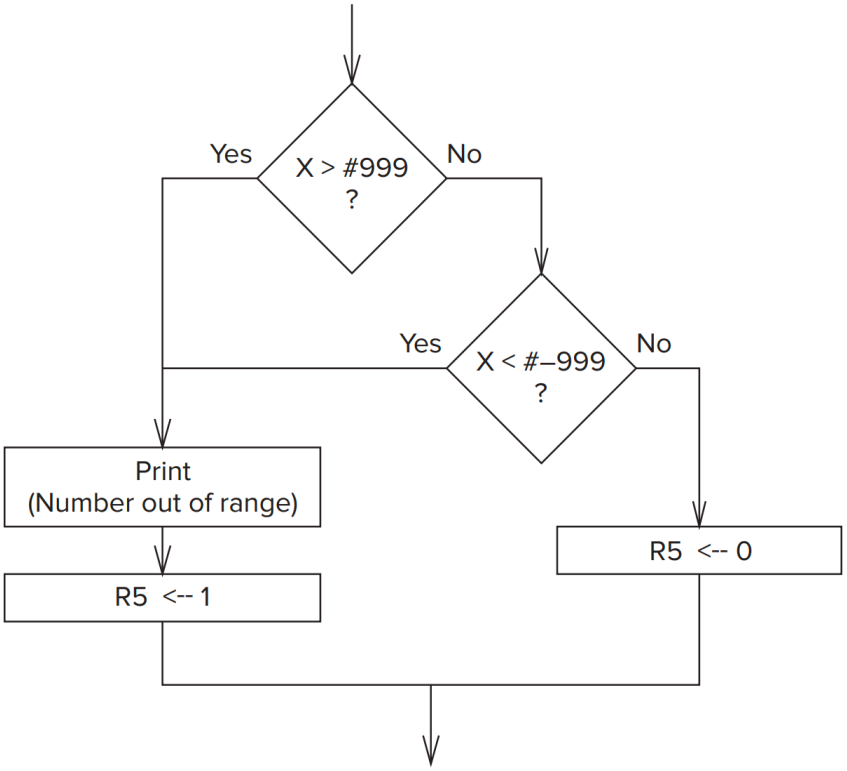

Figure 10.9 The RangeCheck algorithm flowchart.

Conversion

- **ascii => 2's complement integer**
- **2's complement integer => ascii**

Arithmetic Using a Stack

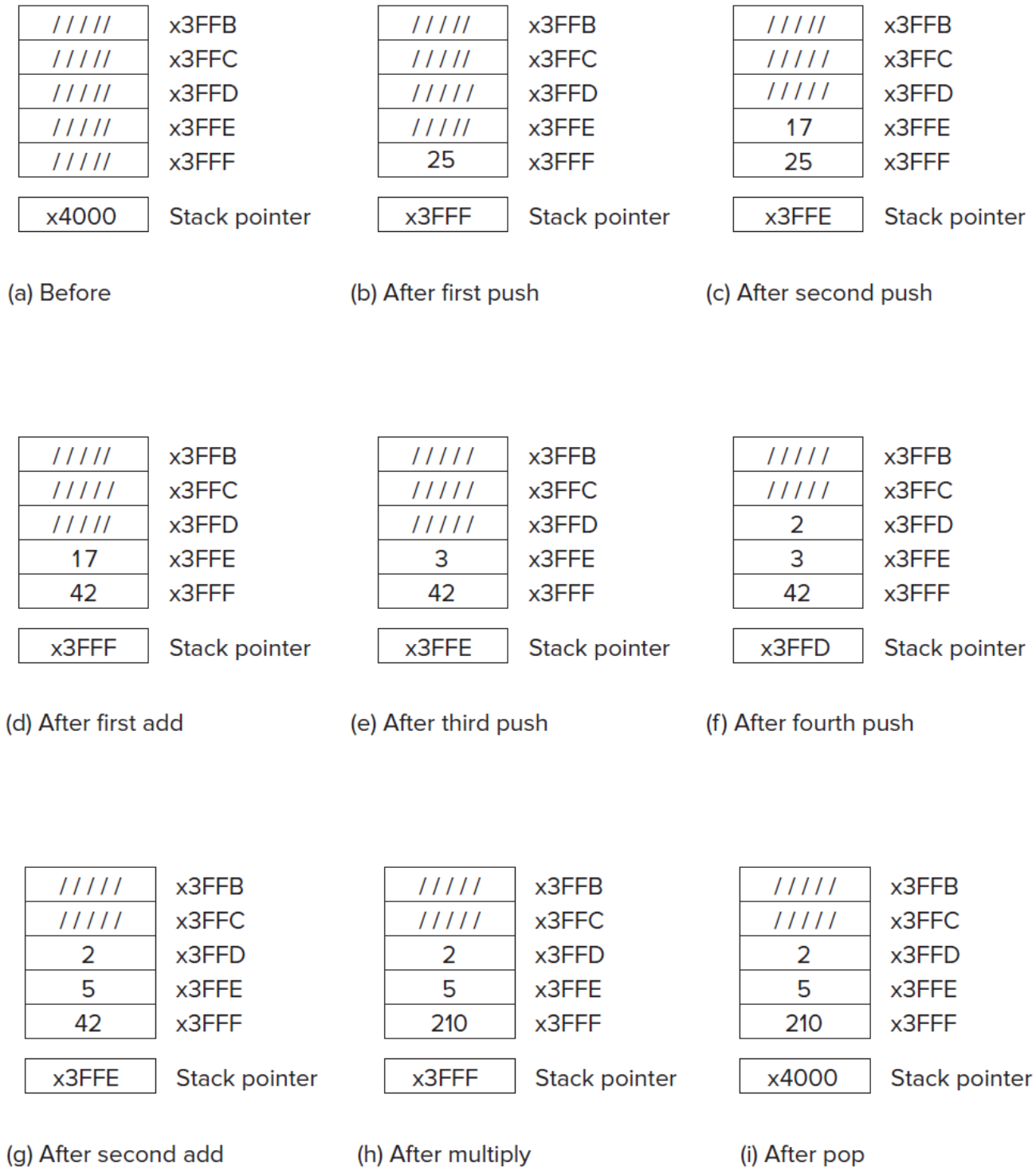

Figure 10.6 Stack usage during the computation of (25 + 17) · (3 + 2).

Multidimensional Arrays

- **1-dimension Array**

  Just like character string & sequentially storage list, we can get A[n] by accessing $A + n$

- **2-dimension Array**

  For 2-dimension Array, we have two storage strategy: Row Major & Column Major. In LC-3, we use Row Major.

  So, for $A[M, N]$ (it means that at most M rows & N columns), we can access $A[i, j]$ by accessing $A + i \times N + j$.

- **3-dimension Array**
  Similar to 2-dimension Array, for $A[M, N, P]$, we can access $A[i, j, k]$ by accessing $A + i \times (N \times P) + j \times P + k$

  > NOTE: in the formulas stated above, we suppose each element just occupies for 1 location, that is, 16 bits. If each element in array needs more than 1 location, the number of locations it need should be taken into consideration.