

## Mathematics 3670: Another Look at Stack Frames

Address	Contents
x5fdc	
x5fdd	
x5fde	
x5fdf	
x5fe0	
x5fe1	
x5fe2	
x5fe3	
x5fe4	
x5fe5	
x5fe6	
x5fe7	
x5fe8	
x5fe9	
x5fea	
x5feb	
x5fec	
x5fed	
x5fee	
x5fef	
x5ff0	
x5ff1	
x5ff2	
x5ff3	
x5ff4	
x5ff5	
x5ff6	
x5ff7	
x5ff8	
x5ff9	
x5ffa	
x5ffb	
x5ffc	
x5ffd	
x5ffe	
x5fff	
x6000	

Figure 1: A portion of the LC-3 memory. Where are the stack frames?

Address	Contents
x5fdc	
x5fdd	
x5fde	
x5fdf	
x5fe0	
x5fe1	
x5fe2	
x5fe3	
x5fe4	
x5fe5	
x5fe6	
x5fe7	
x5fe8	
x5fe9	
x5fea	
x5feb	
x5fec	
x5fed	
x5fee	
x5fef	
x5ff0	
x5ff1	
x5ff2	
x5ff3	
x5ff4	
x5ff5	
x5ff6	
x5ff7	
x5ff8	
x5ff9	
x5ffa	
x5ffb	
x5ffc	
x5ffd	
x5ffe	
x5fff	
x6000	

Figure 2: A portion of the LC-3 memory. Where are the stack frames?

## Appendix

### Contents of factorial.asm

```

1      ;;
2      ;; Author: Bill Slough
3      ;;
4      ;; Illustration of a simple recursive function, N!
5      ;;
6
7      .ORIG x3000
8
9      ;; Main program.....
10
11     START  LD      R6, TOP          ; initialize stack pointer
12           AND     R5, R5, #0      ; frame pointer = null
13
14           LD      R0, M           ;
15           ADD     R6, R6, #-1     ;
16           STR     R0, R6, #0      ; push M
17
18           JSR     FACT           ;
19           LDR     R1, R6, #0      ;
20           ADD     R6, R6, #2      ; R1 = FACT(M)
21
22           ADD     R6, R6, #-1     ;
23           STR     R0, R6, #0      ;
24           JSR     PUTHEX         ; puthex(R0)
25           ADD     R6, R6, #1      ;
26
27           LEA     R0, MSG         ;
28           PUTS   ; write(" factorial is ")
29
30           ADD     R6, R6, #-1     ;
31           STR     R1, R6, #0      ;
32           JSR     PUTHEX         ; puthex(R1)
33           ADD     R6, R6, #1      ;
34
35           LEA     R0, NEWLINE     ;
36           PUTS   ; write("\n")
37     STOP  HALT
38
39     TOP   .FILL   x6000          ; where does the stack begin?
40     M     .FILL   4
41     MSG   .STRINGZ " factorial is "
42     NEWLINE .STRINGZ "\n"
43
44     ;; Subroutine .....
45     FACT
46     ;; Computes the value of N factorial
47     ;;
48     ;; Parameters:
49     ;;     N      -- a nonnegative integer
50     ;;
51     ;; Frame offsets:
52     ;;     -1   saved R1
53     ;;     0   saved R0
54     ;;     +1  previous frame pointer
55     ;;     +2  return address

```

```

56      ;;      +3 N
57
58      ;; PROLOG
59      ADD     R6,R6,#-1      ;
60      STR     R7,R6,#0      ; push return address
61
62      ADD     R6,R6,#-1      ;
63      STR     R5,R6,#0      ; push previous frame pointer
64
65      ADD     R5,R6,#-1      ; establish frame pointer for this frame
66
67      STR     R0,R6,#-1      ; save registers being used in this routine
68      STR     R1,R6,#-2      ;
69      ADD     R6,R6,#-2      ; adjust stack pointer to the top of the frame
70
71      ;; BODY
72  IF     LDR     R0,R5,#3      ; if (n = 0) then
73      BRnp    ELSE          ;
74  THEN  ADD     R0,R0,#1      ; return 1
75      BR     ENDIF         ;
76  ELSE  ADD     R0,R0,#-1     ; else
77      ADD     R6,R6,#-1     ;
78      STR     R0,R6,#0      ; push (n-1)
79      JSR     FACT         ;
80      LDR     R0,R6,#0      ;
81      ADD     R6,R6,#2      ; R0 = FACT(n-1)
82      LDR     R1,R5,#3      ;
83      MUL     R0,R0,R1      ; return n * FACT(n-1)
84  ENDIF ; end if
85      ;; EPILOG
86      LDR     R7,R5,#2      ; get the return address
87      STR     R0,R5,#2      ; deposit the return value
88      LDR     R1,R5,#-1     ; restore R1
89      LDR     R0,R5,#0      ; restore R0
90      LDR     R5,R5,#1      ; restore the previous frame pointer
91      ADD     R6,R6,#3      ; adjust the stack pointer, deallocate frame
92      RET
93
94      ;; Subroutine .....
95  PUTHEX
96      ;; Displays the hexadecimal representation of a given quantity
97      ;;
98      ;; Parameters:
99      ;;      N      -- a 16-bit quantity
100     ;;
101     ;; Frame offsets:
102     ;;      -4  saved R4
103     ;;      -3  saved R3
104     ;;      -2  saved R2
105     ;;      -1  saved R1
106     ;;      0   saved R0
107     ;;      +1  previous frame pointer
108     ;;      +2  return address
109     ;;      +3  N
110
111     ;; PROLOG
112     ADD     R6,R6,#-1      ;
113     STR     R7,R6,#0      ; push return address

```

```

114
115     ADD     R6,R6,#-1           ;
116     STR     R5,R6,#0           ; push previous frame pointer
117
118     ADD     R5,R6,#-1           ; establish frame pointer for this frame
119
120
121     STR     R0,R6,#-1           ; save registers being used in this routine
122     STR     R1,R6,#-2           ;
123     STR     R2,R6,#-3           ;
124     STR     R3,R6,#-4           ;
125     STR     R4,R6,#-5           ;
126     ADD     R6,R6,#-5           ; adjust stack pointer to the top of the frame
127
128     ;; BODY
129     LDR     R0,R5,#3             ; get N from stack frame
130     LD      R2,MASK2            ; mask for least significant hex digit
131
132     AND     R1,R1,#0            ;
133     ADD     R1,R1,#3            ; i = 3
134 LOOP  ; repeat
135     AND     R3,R0,R2            ; isolate least significant digit
136
137     LEA    R4,DIGITS            ;
138     ADD     R4,R4,R3            ;
139     LDR     R4,R4,#0            ; hex_digit = hex(least significant digit)
140
141     LEA    R3,RESULT            ;
142     ADD     R3,R3,R1            ;
143     STR     R4,R3,#1            ; result[i + 1] = hex_digit
144
145     ADD     R6,R6,#-1           ;
146     STR     R0,R6,#0           ; push N
147     JSR    SHIFT4              ;
148     LDR     R0,R6,#0           ;
149     ADD     R6,R6,#2           ; N = SHIFT4(N)
150
151     ADD     R1,R1,#-1           ; i = i - 1
152     BRzp   LOOP                ; until (i < 0)
153
154     LEA    R0,RESULT            ;
155     PUTS   ; write(result)
156
157     ;; EPILOG
158     LDR     R4,R5,#-4           ; restore R4
159     LDR     R3,R5,#-3           ; restore R3
160     LDR     R2,R5,#-2           ; restore R2
161     LDR     R1,R5,#-1           ; restore R1
162     LDR     R0,R5,#0           ; restore R0
163     LDR     R7,R5,#2           ; get the return address
164     LDR     R5,R5,#1           ; restore the previous frame pointer
165     ADD     R6,R6,#7           ; adjust the stack pointer, deallocate frame
166     RET
167 DIGITS  .STRINGZ      "0123456789ABCDEF"
168 RESULT  .STRINGZ      "x...."
169 MASK2   .FILL        x000F           ; mask for least significant hex digit
170
171     ;; Subroutine .....
```

```

172  SHIFT4
173      ;; Shift right 4 bits
174      ;;
175      ;; Parameters:
176      ;;     N      -- a 16-bit quantity
177      ;;
178      ;; Return value:
179      ;;     the 16 bit value obtained by shifting N right 4 bits
180      ;;
181      ;; Frame offsets:
182      ;;     -4  saved R4
183      ;;     -3  saved R3
184      ;;     -2  saved R2
185      ;;     -1  saved R1
186      ;;     0   saved R0
187      ;;     +1  previous frame pointer
188      ;;     +2  return address
189      ;;     +3  N
190
191      ;; PROLOG
192      ADD    R6,R6,#-1      ;
193      STR    R7,R6,#0      ; push return address
194
195      ADD    R6,R6,#-1      ;
196      STR    R5,R6,#0      ; push previous frame pointer
197
198      ADD    R5,R6,#-1      ; establish frame pointer for this frame
199
200
201      STR    R0,R6,#-1      ; save registers being used in this routine
202      STR    R1,R6,#-2      ;
203      STR    R2,R6,#-3      ;
204      STR    R3,R6,#-4      ;
205      STR    R4,R6,#-5      ;
206      ADD    R6,R6,#-5      ; adjust stack pointer to the top of the frame
207
208      ;; BODY
209      AND    R0,R0,#0      ; result = 0
210      AND    R3,R3,#0      ;
211      ADD    R3,R3,#1      ; x = 1
212      LD     R1,MASK      ; mask (for bit 4)
213      LD     R2,COUNT     ; i = 12
214  LOOP2      ; repeat
215      LDR    R4,R5,#3      ;
216  IF2      AND    R4,R4,R1      ; if (current bit of N = 1) then
217      BRz   EIF2          ;
218      ADD    R0,R0,R3      ; result = result + x
219  EIF2      ; end if
220      ADD    R3,R3,R3      ; x = left_shift(x)
221      ADD    R1,R1,R1      ; mask = left_shift(mask)
222      ADD    R2,R2,#-1     ; i = i - 1
223      BRp   LOOP2        ; until (i = 0)
224  ELOOP2
225      ;; EPILOG
226      LDR    R7,R5,#2      ; get the return address
227      STR    R0,R5,#2      ; deposit the return value
228      LDR    R4,R5,#-4     ; restore R4
229      LDR    R3,R5,#-3     ; restore R3

```

```
230          LDR    R2,R5,#-2          ; restore R2
231          LDR    R1,R5,#-1          ; restore R1
232          LDR    R0,R5,#0          ; restore R0
233          LDR    R5,R5,#1          ; restore the previous frame pointer
234          ADD    R6,R6,#6          ; adjust the stack pointer, deallocate frame
235          RET
236  MASK      .FILL  x0010          ; mask for bit 4
237  COUNT     .FILL  12            ; loop count
238          .END
```