

Mathematics 3670: Laboratory 12

Background

In last week's lab, we saw how it is possible to write subroutines in assembly language. We used registers to pass parameters and introduced local save areas to maintain register values.

These techniques are fine for many situations, but there are limitations. For example, what if we have a routine with more than a small handful of parameters? There aren't enough registers to deal with this. Another problem occurs if we want to use a recursive subroutine, since one save area will not be enough.

For this lab, we will explore a different approach—which uses a **stack** and a collection of **stack frames**—which does not have these shortcomings.

Pre-lab Exercises

1. Review the code given in `lab12a.asm` to get a sense of how it is constructed. Of special interest is the stack frame which is used in each subroutine. Note also how each subroutine now has three distinct pieces: the prolog, body, and epilog. Although this program has substantially more code than previous examples, there is a great deal of regularity to it.
2. Trace the execution of the main program, keeping track of how the stack changes over time. As a starting point, consider the actions of the main program as it calls `DRAW_PIXEL`, which will set up one stack frame. Can you predict the exact content of this stack frame at the point the routine reaches its body (i.e., line 179)? How will it change when control returns to the main program (i.e., line 29)?
3. In a similar way, see if you can follow the execution of the main program as it calls `DRAW_BLOCK`. This is more intricate, since `DRAW_BLOCK` will in turn call `DRAW_PIXEL`. Note that two stack frames will now be used, one for each routine.
4. Read the entire lab to see what you are being asked to do for this week's lab.

Lab Exercises

1. Obtain copies of this week's lab files and place them in your **lc3** folder.
2. Start the simulator and prepare to run the `lab12a` program. Single step through the program and carefully watch the registers to see the program in “slow motion.” Of particular interest is the stack frames. After each subroutine has finished building its stack frame (i.e., at the beginning of the subroutine body), inspect memory to see the frame contents.
3. Modify `DRAW_BLK` as follows. As written, it always produces a 2×2 block of green pixels. Instead, we would like an $s \times s$ block of pixels of a specified color. To do this, use four parameters: the block row, block column, block size, and block color.
4. Prove to yourself that your modified subroutine is working. To do this, adjust the main program so it produces various squares of different sizes and colors.
5. To complete the lab, you have several choices. Choose one of the following:
 - Using ideas from Lab 11, display a color-coded version of Pascal's triangle modulo 2. All subroutines in your program should now be stack-based.
 - Design and implement a subroutine, `DRAW_BRD`, which will display an $n \times n$ checkerboard, using squares of size s . Squares should appear in two different colors, c_1 and c_2 . Your board drawing routine thus has four parameters which control how it is to appear. The board should be placed flush to the upper left-hand corner of the display.

Submissions

Before submitting your work, make sure your name appears in each of the programs you wrote. Also, ensure that each of your programs is generously commented.

Create a **lab12** folder and place copies of all **.asm** programs you wrote in this folder. Submit the folder by dragging it onto the EIU submission icon.

Appendix

Contents of lab12a.asm

```

1      ;;
2      ;; Author: Bill Slough
3      ;;
4      ;; Subroutines with stack frames: improved parameter passing
5      ;;
6      ;; Lights one pixel and one 2x2 square block
7      ;;
8
9      .ORIG x3000
10
11     ;; Main program.....
12
13  START  LD      R6, TOP          ; initialize stack pointer
14         AND     R5, R5, #0      ; frame pointer = null
15
16         LD      R0, ROW         ;
17         ADD     R6, R6, #-1     ;
18         STR     R0, R6, #0      ; push ROW
19
20         LD      R0, COLUMN      ;
21         ADD     R6, R6, #-1     ;
22         STR     R0, R6, #0      ; push COLUMN
23
24         LD      R0, RED         ;
25         ADD     R6, R6, #-1     ;
26         STR     R0, R6, #0      ; push RED
27
28         JSR     DRAW_PIXEL      ; DRAW_PIXEL(ROW, COLUMN, RED)
29         ADD     R6, R6, #3      ; remove parameters from stack frame
30
31         LD      R0, BROW        ;
32         ADD     R6, R6, #-1     ;
33         STR     R0, R6, #0      ; push BROW
34
35         LD      R0, BCOLUMN     ;
36         ADD     R6, R6, #-1     ;
37         STR     R0, R6, #0      ; push BCOLUMN
38
39         JSR     DRAW_BLK       ; DRAW_BLOCK(BROW, BCOLUMN)
40         ADD     R6, R6, #2      ; remove parameters from stack frame
41  STOP   HALT
42
43  TOP    .FILL   x6000          ; where does the stack begin?
44
45  ROW    .FILL   2              ; which pixel row?
46  COLUMN .FILL   8              ; which pixel column?
47  RED    .FILL   x7C00         ; desired color for pixel
48

```

```

49  BROW    .FILL    3                ; which block row?
50  BCOLUMN .FILL    8                ; which block column?
51
52          ;; Subroutine .....
53  DRAW_BLK
54          ;; Draw a 2x2 green block at a specified location
55          ;;
56          ;; Parameters:
57          ;;     b_row    -- which row of blocks
58          ;;     b_column -- which column of blocks
59          ;;
60          ;; Frame offsets:
61          ;;     -2  saved R2
62          ;;     -1  saved R1
63          ;;     0   saved R0
64          ;;     +1  previous frame pointer
65          ;;     +2  return address
66          ;;     +3  column
67          ;;     +4  row
68
69          ;; PROLOG
70          ADD     R6,R6,#-1          ;
71          STR     R7,R6,#0          ; push return address
72
73          ADD     R6,R6,#-1          ;
74          STR     R5,R6,#0          ; push previous frame pointer
75
76          ADD     R5,R6,#-1          ; establish frame pointer for this frame
77
78          STR     R0,R6,#-1          ; save registers being used in this routine
79          STR     R1,R6,#-2          ;
80          STR     R2,R6,#-3          ;
81          ADD     R6,R6,#-3          ; adjust stack pointer to the top of the frame
82
83          ;; BODY
84          LDR     R0,R5,#4          ; R0 = b_row
85          LDR     R1,R5,#3          ; R1 = b_column
86          LD      R2,GREEN          ; R2 = green
87
88          ADD     R0,R0,R0          ; R0 = 2 * b_row
89          ADD     R1,R1,R1          ; R1 = 2 * b_column
90
91          ADD     R6,R6,#-1          ; upper left pixel in block
92          STR     R0,R6,#0          ; push R0
93          ADD     R6,R6,#-1          ;
94          STR     R1,R6,#0          ; push R1
95          ADD     R6,R6,#-1          ;
96          STR     R2,R6,#0          ; push R2
97          JSR     DRAW_PIXEL        ; draw_pixel(2 * b_row, 2 * b_column, GREEN)
98          ADD     R6,R6,#3          ; remove parameters
99
100         ADD     R1,R1,#1          ; upper right pixel in block
101         ADD     R6,R6,#-1          ;
102         STR     R0,R6,#0          ; push R0
103         ADD     R6,R6,#-1          ;
104         STR     R1,R6,#0          ; push R1
105         ADD     R6,R6,#-1          ;
106         STR     R2,R6,#0          ; push R2

```

```

107      JSR      DRAW_PIXEL      ; draw_pixel(2 * b_row, 2 * b_column + 1, GREEN)
108      ADD      R6,R6,#3        ; remove parameters
109
110      ADD      R0,R0,#1        ; lower right pixel in block
111      ADD      R6,R6,#-1       ;
112      STR      R0,R6,#0        ; push R0
113      ADD      R6,R6,#-1       ;
114      STR      R1,R6,#0        ; push R1
115      ADD      R6,R6,#-1       ;
116      STR      R2,R6,#0        ; push R2
117      JSR      DRAW_PIXEL      ; draw_pixel(2 * b_row + 1, 2 * b_column + 1, GREEN)
118      ADD      R6,R6,#3        ; remove parameters
119
120      ADD      R1,R1,#-1       ; lower left pixel in block
121      ADD      R6,R6,#-1       ;
122      STR      R0,R6,#0        ; push R0
123      ADD      R6,R6,#-1       ;
124      STR      R1,R6,#0        ; push R1
125      ADD      R6,R6,#-1       ;
126      STR      R2,R6,#0        ; push R2
127      JSR      DRAW_PIXEL      ; draw_pixel(2 * b_row + 1, 2 * b_column, GREEN)
128      ADD      R6,R6,#3        ; remove parameters
129
130      ;; EPILOG
131      LDR      R2,R5,#-2       ; restore R2
132      LDR      R1,R5,#-1       ; restore R1
133      LDR      R0,R5,#0        ; restore R0
134      LDR      R7,R5,#2       ; get the return address
135      LDR      R5,R5,#1       ; restore the previous frame pointer
136      ADD      R6,R6,#5       ; adjust the stack pointer, deallocate frame
137      RET
138
139      ;; Constants used by DRAW_BLK
140  GREEN  .FILL  x03E0
141
142      ;; Subroutine .....
143  DRAW_PIXEL
144      ;; Draw a pixel on the graphics display unit at a specified location
145      ;;
146      ;; Parameters:
147      ;;     row      -- a value between 0 and 123
148      ;;     column  -- a value between 0 and 127
149      ;;     color   -- a 15-bit RGB value
150      ;;
151      ;; Frame offsets:
152      ;;     -4  saved R4
153      ;;     -3  saved R3
154      ;;     -2  saved R2
155      ;;     -1  saved R1
156      ;;     0   saved R0
157      ;;     +1  previous frame pointer
158      ;;     +2  return address
159      ;;     +3  color
160      ;;     +4  column
161      ;;     +5  row
162
163      ;; PROLOG
164      ADD      R6,R6,#-1       ;

```

```

165      STR      R7,R6,#0          ; push return address
166
167      ADD      R6,R6,#-1        ;
168      STR      R5,R6,#0          ; push previous frame pointer
169
170      ADD      R5,R6,#-1        ; establish frame pointer for this frame
171
172      STR      R0,R6,#-1        ; save registers being used in this routine
173      STR      R1,R6,#-2        ;
174      STR      R2,R6,#-3        ;
175      STR      R3,R6,#-4        ;
176      STR      R4,R6,#-5        ;
177      ADD      R6,R6,#-5        ; adjust stack pointer to the top of the frame
178
179      ;; BODY
180      LDR      R0,R5,#5          ; R0 = row
181      LDR      R1,R5,#4          ; R1 = column
182      LDR      R2,R5,#3          ; R2 = color
183
184      ADD      R3,R0,#0          ;
185      LD       R4,PIX_PER_ROW    ;
186      MUL      R3,R3,R4          ;
187      ADD      R3,R3,R1          ; R3 = PIX_PER_ROW * row + column
188
189      LD       R4,VIDEO_ADDR     ;
190      ADD      R4,R4,R3          ; R4 = address of desired pixel
191
192      STR      R2,R4,#0          ; Video[row,column] = color
193
194      ;; EPILOG
195      LDR      R4,R5,#-4          ; restore R4
196      LDR      R3,R5,#-3          ; restore R3
197      LDR      R2,R5,#-2          ; restore R2
198      LDR      R1,R5,#-1          ; restore R1
199      LDR      R0,R5,#0          ; restore R0
200      LDR      R7,R5,#2          ; get the return address
201      LDR      R5,R5,#1          ; restore the previous frame pointer
202      ADD      R6,R6,#7          ; adjust the stack pointer, deallocate frame
203      RET
204
205      ;; Constants used by DRAW_PIXEL
206      VIDEO_ADDR      .FILL    xC000    ; base address of the video display
207      PIX_PER_ROW     .FILL    128      ; number of pixels per row
208
209      .END

```