Mathematics 3670: Laboratory 11

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

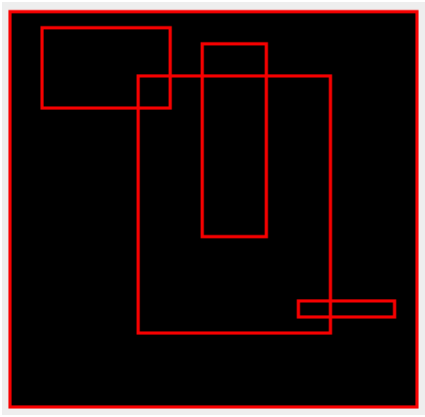
1. Review the code given in `lab11a.asm` to get a sense of how it is constructed. Of special interest is the stack frame which is used in each subroutine. Note also how each subroutine now has three distinct pieces: the prolog, body, and epilog. Although this program has substantially more code than 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. Be sure to read the entire lab to see what you are being asked to do for this week's lab.

5. When `lab11c.asm` is executed, the following output will be produced:



Carefully study the code in this program, paying special attention to the following.

- Observe the role of the `P` and `Q` arrays in the main program.
- The subroutine `DRAW_RECT` draws a rectangle in a specified color. A rectangle is specified by providing the coordinates of its upper-left and lower-right corner points.
- Verify that the prolog and epilog sections of `DRAW_RECT` fulfill their obligations regarding the stack frame.

- Verify that the body of `DRAW_RECT` lights the correct pixels for a given rectangle.

6. For the lab, you will add several subroutines to this program. One of these, `GET_PIXEL`, will return the color of any one pixel of the video display. This routine takes two parameters — the row and column of the pixel in question. The return value is a 15-bit value which gives the color of the pixel.

   Here's how we want it to work. The caller pushes the coordinates of a pixel onto the stack, then calls `GET_PIXEL`. Upon return, the caller sees the return value on top of the stack; below that are the coordinates of the pixel. To maintain correct stack behavior, the caller will push two values and pop three.

   Design the code for this routine.

7. A number of paint programs have a bucket tool that allows "paint" to be "poured" into a region. One way to accomplish this is to use an algorithm known as "flood fill," as shown in Figure 1.

   Design a subroutine, `F_FILL`, which will perform the flood fill algorithm. This routine takes four parameters: `row` and `column` (which together specify the location of a pixel) and two colors: `color1` and `color2`. In response, this routine finds the largest block of pixels which have the first color. Each pixel in this block can reach the given pixel by taking a number of horizontal or vertical steps. The flood fill algorithm changes the color of each of the pixels in this block to the second color.

   Design the code for this routine.

```
// (row,column) specifies the location of a pixel.
// FloodFill identifies the largest block of pixels
// containing this pixel which have the first color;
// all the pixels in this block are changed to the
// second color.
FloodFill(row, column, color1, color2)
If GetPixel(row, column) != color1
  return

DrawPixel(row  , column  , color2)         // Change this pixel's color
FloodFill(row  , column-1, color1, color2) // Flood west
FloodFill(row  , column+1, color1, color2) // Flood east
FloodFill(row-1, column  , color1, color2) // Flood north
FloodFill(row+1, column  , color1, color2) // Flood south
```

Figure 1: Flood fill algorithm.

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 `lab11a` 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. Do **one** of the following:

   - Using ideas from Lab 10, 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.

6. Prepare and run the `lab11b` program. The output is not very exciting, since it only draws two pixels. However, this program can serve as a simple test of your `GET_PIXEL` routine.

   Modify `lab11b.asm` by adding the code for `GET_PIXEL`. Adjust the main program so that it calls `GET_PIXEL` three times, for the following pixels: $(3, 8), (3, 9)$, and $(3, 10)$.

   When you run the program, put breakpoints immediately after each `JSR` instruction; verify the correct color is returned in each case.

7. Copy the `GET_PIXEL` subroutine into `lab11c.asm`. Run the program and verify the output is as shown on the first page of this handout.

8. Modify `lab11c.asm` further by adding your `F_FILL` subroutine. To test your work, we want to be able to flood fill, starting at an arbitrary pixel on the screen — using black for the first color and red for the second color. The data values stored at `F_ROW` and `F_COL` specify where the flood fill is to begin. As you test your program, you will change these values.

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 **lab11** 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 lab11a.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
```

Contents of lab11b.asm

```
1           ;;
2           ;; Author: Bill Slough
3           ;;
4           ;; Subroutines with stack frames: improved parameter passing
5           ;;
6           ;; Lights a few pixels
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,ROW                  ;
32          ADD     R6,R6,#-1               ;
33          STR     R0,R6,#0                ; push ROW
34
35          LD      R0,COLUMN               ;
36          ADD     R0,R0,#2                ;
37          ADD     R6,R6,#-1               ;
38          STR     R0,R6,#0                ; push COLUMN + 2
39
40          LD      R0,GREEN                ;
41          ADD     R6,R6,#-1               ;
42          STR     R0,R6,#0                ; push GREEN
43
44          JSR     DRAW_PIXEL              ; DRAW_PIXEL(ROW, COLUMN + 2, GREEN)
45          ADD     R6,R6,#3                ; remove parameters from stack frame
46
47  STOP    HALT
48
49  TOP     .FILL   x6000           ; where does the stack begin?
50
51  RED     .FILL   x7C00           ;
52  GREEN   .FILL   x03E0
53
54  ROW     .FILL   3               ; which row?
55  COLUMN  .FILL   8               ; which column?
56
57          ;; Subroutine ..............................................................
```

```
58  DRAW_PIXEL
59          ;; Draw a pixel on the graphics display unit at a specified location
60          ;;
61          ;; Parameters:
62          ;;      row     -- a value between 0 and 123
63          ;;      column  -- a value between 0 and 127
64          ;;      color   -- a 15-bit RGB value
65          ;;
66          ;; Frame offsets:
67          ;;      -4  saved R4
68          ;;      -3  saved R3
69          ;;      -2  saved R2
70          ;;      -1  saved R1
71          ;;       0  saved R0
72          ;;      +1  previous frame pointer
73          ;;      +2  return address
74          ;;      +3  color
75          ;;      +4  column
76          ;;      +5  row
77
78          ;; PROLOG
79          ADD     R6,R6,#-1               ;
80          STR     R7,R6,#0                ; push return address
81
82          ADD     R6,R6,#-1               ;
83          STR     R5,R6,#0                ; push previous frame pointer
84
85          ADD     R5,R6,#-1               ; establish frame pointer for this frame
86
87          STR     R0,R6,#-1               ; save registers being used in this routine
88          STR     R1,R6,#-2               ;
89          STR     R2,R6,#-3               ;
90          STR     R3,R6,#-4               ;
91          STR     R4,R6,#-5               ;
92          ADD     R6,R6,#-5               ; adjust stack pointer to the top of the frame
93
94          ;; BODY
95          LDR     R0,R5,#5                ; R0 = row
96          LDR     R1,R5,#4                ; R1 = column
97          LDR     R2,R5,#3                ; R2 = color
98
99          ADD     R3,R0,#0                ;
100         LD      R4,PIX_PER_ROW  ;
101         MUL     R3,R3,R4                ;
102         ADD     R3,R3,R1                ; R3 = PIX_PER_ROW * row + column
103
104         LD      R4,VIDEO_ADDR
105         ADD     R4,R4,R3                ; R4 = address of desired pixel
106
107         STR     R2,R4,#0                ; Video[row,column] = color
108
109         ;; EPILOG
110         LDR     R4,R5,#-4               ; restore R4
111         LDR     R3,R5,#-3               ; restore R3
112         LDR     R2,R5,#-2               ; restore R2
113         LDR     R1,R5,#-1               ; restore R1
114         LDR     R0,R5,#0                ; restore R0
115         LDR     R7,R5,#2                ; get the return address
```

```
116           LDR     R5,R5,#1                ; restore the previous frame pointer
117           ADD     R6,R6,#7                ; adjust the stack pointer, deallocate frame
118           RET
119
120           ;; Constants used by DRAW_PIXEL
121   VIDEO_ADDR      .FILL   xC000   ; base address of the video display
122   PIX_PER_ROW     .FILL   128     ; number of pixels per row
123
124           .END
```

Contents of lab11c.asm

```
1           ;;
2           ;; Author: Bill Slough
3           ;;
4           ;; Invoking subroutines with stack frames
5           ;;
6           ;; Draws rectangles of varying sizes
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          LEA     R0,P                    ; i = 0
17          LEA     R1,Q
18  LOOP0   LEA     R2,P_END                ; while (corner points remain)
19          NOT     R2,R2                   ;
20          ADD     R2,R2,#1                ;
21          ADD     R2,R0,R2                ;
22          BRz     ELOOP0
23
24          LDR     R3,R0,#0                ;
25          ADD     R6,R6,#-1               ;
26          STR     R3,R6,#0                ;   push P[i].row
27
28          LDR     R3,R0,#1                ;
29          ADD     R6,R6,#-1               ;
30          STR     R3,R6,#0                ;   push P[i].column
31
32          LDR     R3,R1,#0                ;
33          ADD     R6,R6,#-1               ;
34          STR     R3,R6,#0                ;   push Q[i].row
35
36          LDR     R3,R1,#1                ;
37          ADD     R6,R6,#-1               ;
38          STR     R3,R6,#0                ;   push Q[i].column
39
40          LD      R3,RED                  ;
41          ADD     R6,R6,#-1               ;
42          STR     R3,R6,#0                ;   push RED
43
44          JSR     DRAW_RECT               ;   draw_rect(P[i].row, P[i].column,
45          ADD     R6,R6,#5                ;             Q[i].row, Q[i].column, RED)
46
47          ADD     R0,R0,#2                ;
48          ADD     R1,R1,#2                ;   i = i + 1
49          BR      LOOP0                   ; end while
50  ELOOP0
51  STOP    HALT
52
53  TOP     .FILL   xC000                   ; where does the stack begin?
54  RED     .FILL   x7C00                   ; desired color for rectangles
55
56  F_ROW   .FILL   6                       ; where to start the flood fill...
57  F_COL   .FILL   11                      ; currently (6,11)
```

```
58
59          ;; P stores the coordinates of the upper-left corner points
60  P       .FILL   5               ; (5, 10)
61          .FILL   10              ;
62          .FILL   20              ; (20, 40)
63          .FILL   40              ;
64          .FILL   90              ; (90, 90)
65          .FILL   90              ;
66          .FILL   10              ; (10, 60)
67          .FILL   60              ;
68          .FILL   0               ; (0, 0)
69          .FILL   0
70  P_END   .FILL   -1
71
72          ;; Q stores the coordinates of the lower-right corner points
73  Q       .FILL   30              ; (30, 50)
74          .FILL   50              ;
75          .FILL   100             ; (100, 100)
76          .FILL   100             ;
77          .FILL   95              ; (95, 120)
78          .FILL   120             ;
79          .FILL   70              ; (70, 80)
80          .FILL   80              ;
81          .FILL   123             ; (123, 127)
82          .FILL   127
83  Q_END   .FILL   -1
84
85          ;; Subroutine ..............................................................
86  DRAW_RECT
87          ;; Draw a rectangle, given the coordinates of opposing corner points
88          ;;
89          ;; Parameters:
90          ;;      ULrow   (upper left corner)
91          ;;      ULcol
92          ;;      LRrow   (lower right corner)
93          ;;      LRcol
94          ;;      color   (desired color)
95          ;;
96          ;; Frame offsets:
97          ;;      -4  saved R4
98          ;;      -3  saved R3
99          ;;      -2  saved R2
100         ;;      -1  saved R1
101         ;;       0  saved R0
102         ;;      +1  previous frame pointer
103         ;;      +2  return address
104         ;;      +3  color
105         ;;      +4  LRcolumn
106         ;;      +5  LRrow
107         ;;      +6  ULcolumn
108         ;;      +7  ULrow
109
110         ;; PROLOG
111         ADD     R6,R6,#-1               ;
112         STR     R7,R6,#0                ; push return address
113
114         ADD     R6,R6,#-1               ;
115         STR     R5,R6,#0                ; push previous frame pointer
```

```
116
117           ADD     R5,R6,#-1               ; establish frame pointer for this frame
118
119           STR     R0,R6,#-1               ; save registers being used in this routine
120           STR     R1,R6,#-2               ;
121           STR     R2,R6,#-3               ;
122           STR     R3,R6,#-4
123           STR     R4,R6,#-5
124           ADD     R6,R6,#-5               ; adjust stack pointer to the top of the frame
125
126           ;; BODY
127           ;; Draw horizontal line segments
128           LDR     R3,R5,#7                ; R3 = ULrow
129           LDR     R4,R5,#5                ; R4 = LRrow
130
131           LDR     R0,R5,#6                ; c = ULcolumn
132   LOOP1   LDR     R1,R5,#4                ; while (c <= LRcolumn)
133           NOT     R2,R0                   ;
134           ADD     R2,R2,#1                ;
135           ADD     R1,R1,R2                ;
136           BRn     ELOOP1                  ;
137   BODY1
138           ADD     R6,R6,#-1               ;
139           STR     R3,R6,#0                ;    push ULrow
140
141           ADD     R6,R6,#-1               ;
142           STR     R0,R6,#0                ;    push c
143
144           LDR     R1,R5,#3                ;
145           ADD     R6,R6,#-1               ;
146           STR     R1,R6,#0                ;    push color
147
148           JSR     DRAW_PIXEL              ;    draw_pixel(ULrow, c, color)
149           ADD     R6,R6,#3                ;    remove parameters
150
151           ADD     R6,R6,#-1               ;
152           STR     R4,R6,#0                ;    push LRrow
153
154           ADD     R6,R6,#-1               ;
155           STR     R0,R6,#0                ;    push c
156
157           ADD     R6,R6,#-1               ;
158           STR     R1,R6,#0                ;    push color
159
160           JSR     DRAW_PIXEL              ;    draw_pixel(LRrow, c, color)
161           ADD     R6,R6,#3                ;
162
163           ADD     R0,R0,#1                ;    c = c + 1
164           BR      LOOP1                   ; end while
165   ELOOP1
166           ;; Draw vertical line segments
167           LDR     R3,R5,#6                ; R3 = ULcolumn
168           LDR     R4,R5,#4                ; R4 = LRcolumn
169
170           LDR     R0,R5,#7                ; r = ULrow
171   LOOP2   LDR     R1,R5,#5                ; while (r <= LRrow)
172           NOT     R2,R0                   ;
173           ADD     R2,R2,#1                ;
```

```
174          ADD     R1,R1,R2                ;
175          BRn     ELOOP2                  ;
176    BODY2
177          ADD     R6,R6,#-1               ;
178          STR     R0,R6,#0                ;    push r
179
180          ADD     R6,R6,#-1               ;
181          STR     R3,R6,#0                ;    push ULcolumn
182
183          LDR     R1,R5,#3                ;
184          ADD     R6,R6,#-1               ;
185          STR     R1,R6,#0                ;    push color
186
187          JSR     DRAW_PIXEL              ;    draw_pixel(r, ULcolumn, color)
188          ADD     R6,R6,#3                ;    remove parameters
189
190          ADD     R6,R6,#-1               ;
191          STR     R0,R6,#0                ;    push r
192
193          ADD     R6,R6,#-1               ;
194          STR     R4,R6,#0                ;    push LRcolumn
195
196          ADD     R6,R6,#-1               ;
197          STR     R1,R6,#0                ;    push color
198
199          JSR     DRAW_PIXEL              ;    draw_pixel(r, LRcolumn, color)
200          ADD     R6,R6,#3                ;
201
202          ADD     R0,R0,#1                ;    r = r + 1
203          BR      LOOP2                   ; end while
204    ELOOP2
205          ;; EPILOG
206          LDR     R4,R5,#-4
207          LDR     R3,R5,#-3
208          LDR     R2,R5,#-2               ; restore R2
209          LDR     R1,R5,#-1               ; restore R1
210          LDR     R0,R5,#0                ; restore R0
211          LDR     R7,R5,#2                ; get the return address
212          LDR     R5,R5,#1                ; restore the previous frame pointer
213          ADD     R6,R6,#7                ; adjust the stack pointer, deallocate frame
214          RET
215
216          ;; Subroutine ..............................................................
217    DRAW_PIXEL
218          ;; Draw a pixel on the graphics display unit at a specified location
219          ;;
220          ;; Parameters:
221          ;;      row     -- a value between 0 and 123
222          ;;      column  -- a value between 0 and 127
223          ;;      color   -- a 15-bit RGB value
224          ;;
225          ;; Frame offsets:
226          ;;      -4  saved R4
227          ;;      -3  saved R3
228          ;;      -2  saved R2
229          ;;      -1  saved R1
230          ;;       0  saved R0
231          ;;      +1  previous frame pointer
```

```
232          ;;      +2  return address
233          ;;      +3  color
234          ;;      +4  column
235          ;;      +5  row
236
237          ;; PROLOG
238          ADD     R6,R6,#-1               ;
239          STR     R7,R6,#0                ; push return address
240
241          ADD     R6,R6,#-1               ;
242          STR     R5,R6,#0                ; push previous frame pointer
243
244          ADD     R5,R6,#-1               ; establish frame pointer for this frame
245
246          STR     R0,R6,#-1               ; save registers being used in this routine
247          STR     R1,R6,#-2               ;
248          STR     R2,R6,#-3               ;
249          STR     R3,R6,#-4               ;
250          STR     R4,R6,#-5               ;
251          ADD     R6,R6,#-5               ; adjust stack pointer to the top of the frame
252
253          ;; BODY
254          LDR     R0,R5,#5                ; R0 = row
255          LDR     R1,R5,#4                ; R1 = column
256          LDR     R2,R5,#3                ; R2 = color
257
258          ADD     R3,R0,#0                ;
259          LD      R4,PIX_PER_ROW  ;
260          MUL     R3,R3,R4                ;
261          ADD     R3,R3,R1                ; R3 = PIX_PER_ROW * row + column
262
263          LD      R4,VIDEO_ADDR
264          ADD     R4,R4,R3                ; R4 = address of desired pixel
265
266          STR     R2,R4,#0                ; Video[row,column] = color
267
268          ;; EPILOG
269          LDR     R4,R5,#-4               ; restore R4
270          LDR     R3,R5,#-3               ; restore R3
271          LDR     R2,R5,#-2               ; restore R2
272          LDR     R1,R5,#-1               ; restore R1
273          LDR     R0,R5,#0                ; restore R0
274          LDR     R7,R5,#2                ; get the return address
275          LDR     R5,R5,#1                ; restore the previous frame pointer
276          ADD     R6,R6,#7                ; adjust the stack pointer, deallocate frame
277          RET
278
279          ;; Constants used by DRAW_PIXEL
280  VIDEO_ADDR      .FILL   xC000   ; base address of the video display
281  PIX_PER_ROW     .FILL   128     ; number of pixels per row
282
283          .END
```