Trac Duc Anh Luong - 103488117

Lab 10

4. Identify the parameters required when calling drawPixel, and identify what happens to them in the drawPixel function.

- The parameters are x, y, and colour.
- In the drawpixel.asm file, the values of x, y, and colour were calculated and stored into the registers. The code snippet below shows the calculation of the x value

```
;calculate x term  (x * BITS_PER_PIXEL  / BITS PER BYTE)
mov r8,r1   ;x
mov r9, BITS_PER_PIXEL   ;*BITS_PER_PIXEL (16)
mul r8,r9
lsr r8,#3   ;/8 (bits per byte)
add r0,r8   ;add x term
```

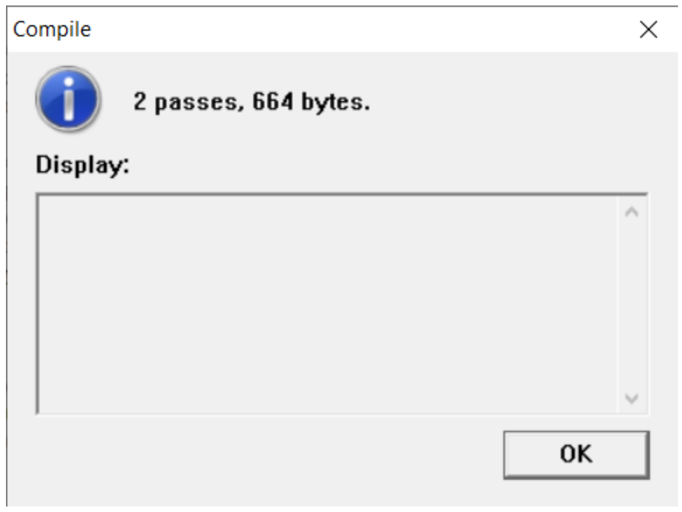
5. Now identify the code which controls the calling of drawPixel, particularly the counter incrementing code and the test to see the loop should continue. What is this code doing?
With bl drawpixel, the drawpixel function then is called into the compiler and runs the code. The value for the x will be assigned to register 1 and the y value will be in r2. Whereas the colour was first initialised into r6 and then it will be moved to r3 in the drawPixel function.

Add your kernel7.asm code of your shape drawer to the submission document

```
; Raspberry Pi B+,2 'Bare Metal' 16BPP Draw Pixel at any XY:
; 1. Setup Frame Buffer
;    assemble struct with screen requirements
;    receive pointer to screen or NULL
; 2. Start loop
;    Send pixel colour to location on screen
;    increment counter and loop if < 640

;note: r6 (colour) is 32-bit/4 byte register.
;at 16 bits/pixel, writing 32bits to adjacent pixels overwrites every
second pixel.
; soln: write lower 2 bytes only (STRH) or lower byte(STRB).
;r0 = pointer + x * BITS_PER_PIXEL/8 + y * SCREEN_X *
BITS_PER_PIXEL/8
format binary as 'img'
;constants


;memory addresses of BASE
BASE = $3F000000 ; use $3F000000 for 3B/3B+ and 2B
```

```
;BASE = $20000000 ;

org $8000
mov sp,$1000

; Return CPU ID (0..3) Of The CPU Executed On
;mrc p15,0,r0,c0,c0,5 ; R0 = Multiprocessor Affinity Register (MPIDR)
;ands r0,3 ; R0 = CPU ID (Bits 0..1)
;bne CoreLoop ; IF (CPU ID != 0) Branch To Infinite Loop (Core ID
1..3)

mov r0,BASE; r0 = BASE
bl FB_Init
;r0 now contains address of screen
;SCREEN_X and BITS_PER_PIXEL are global constants populated by
FB_Init

and r0,$3FFFFFFF ; Convert Mail Box Frame Buffer Pointer From BUS
Address To Physical Address ($CXXXXXXX -> $3XXXXXXX)
str r0,[FB_POINTER] ; Store Frame Buffer Pointer Physical Address

mov r7,r0 ;back-up a copy of the screen address + channel number


; Draw Pixel at (X,Y)
;r0 = address of screen we write to (r7 = backup of screen start
address)

mov r6,#1   ;white for 8-bit colour

mov r4, #10
mov r5, #10

; Draw Box has top left = (10, 10) and bottom right = (18, 26)
horizontal:
   push {r0-r3}
   mov r0,r7    ;screen address
   mov r1,r4 ;x
   mov r2,r5 ;y
   mov r3,r6 ;colour
       bl drawpixel
   pop {r0-r3}

  ;increment and test
  add r4,#1
  mov r8, #18
  cmp r4,r8
bls horizontal      ;branch less than or same

vertical:
   push {r0-r3}
   mov r0,r7    ;screen address
   mov r1,r4 ;x
   mov r2,r5 ;y
   mov r3,r6 ;colour
       bl drawpixel
   pop {r0-r3}

  ;increment and test
  add r5,#1
  mov r9, #27
  cmp r5,r9
bls vertical      ;branch less than or same

mov r4, #10
mov r5, #10

vertical2:
   push {r0-r3}
   mov r0,r7    ;screen address
   mov r1,r4 ;x
   mov r2,r5 ;y
   mov r3,r6 ;colour
       bl drawpixel
   pop {r0-r3}

  ;increment and test
  add r5,#1
  mov r8, #18
  mov r9, #26
  cmp r5,r9
bls vertical2      ;branch less than or same

horizontal2:
   push {r0-r3}
   mov r0,r7    ;screen address
   mov r1,r4 ;x
   mov r2,r5 ;y
   mov r3,r6 ;colour
       bl drawpixel
   pop {r0-r3}

  ;increment and test
  add r4,#1
  mov r8, #18
  cmp r4,r8
bls horizontal2      ;branch less than or same

mov r4, #40
mov r5, #40

vertical3:
   push {r0-r3}
   mov r0,r7    ;screen address
   mov r1,r4 ;x
   mov r2,r5 ;y
   mov r3,r6 ;colour
       bl drawpixel
   pop {r0-r3}

  ;increment and test
  add r5,#1
  mov r9, #80
  cmp r5,r9
bls vertical3      ;branch less than or same

horizontal3:
   push {r0-r3}
   mov r0,r7    ;screen address
   mov r1,r4 ;x
   mov r2,r5 ;y
   mov r3,r6 ;colour
       bl drawpixel
   pop {r0-r3}

  ;increment and test
  add r4,#1
  mov r8, #81
  cmp r4,r8
bls horizontal3      ;branch less than or same

mov r4, #40
mov r5, #40

diagonal:
   push {r0-r3}
   mov r0,r7    ;screen address
   mov r1,r4 ;x
   mov r2,r5 ;y
   mov r3,r6 ;colour
       bl drawpixel
   pop {r0-r3}

  ;increment and test
  add r4,#1
  add r5,#1
  mov r8, #80
  cmp r4,r8
bls diagonal      ;branch less than or same

Loop:
  b Loop  ;wait forever

CoreLoop: ; Infinite Loop For Core 1..3
  b CoreLoop

include "FBinit8.asm"
include "drawpixel.asm"
```

Compile



2 passes, 664 bytes.

Output