COS10004: Computer Systems

Lab 10

Name: SWH00420 Tran Quoc Dung

Student ID: 103803891

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

Upon calling DrawPixel, the arguments x and y are necessary, as well as what occurs in the drawPixel function would be that they show on display whenever the Pi is plugged into the screen.

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 ?

Whenever Raspberry Pi is plugged into the monitor, the code displays a pixel so that the appropriate need may be displayed.

9. Assemble and test your code regularly, and keep copies so you can back track and try different things.

```
;FILE DrawPixel.asm

;author: JHH for Comp Syst 2015

;version 2 - handles different values of BITS_PER_PIXEL

drawpixel:

 ;paramesters:

 ;r0 = screen memory address incl channel number

 ;r1 = x

 ;r2 = y

 ;r3 = colour (16 bit RGB)
```

```
;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
 ;calc y term (y * SCREEN_X * BITS_PER_PIXEL / BITS PER BYTE)
 mov r8,SCREEN_X ;640
 mul r8,r2 ;* y
 mul r8,r9 ;*BITS_PER_PIXEL
 lsr r8,#3 ;/8 bits per byte
 add r0,r8 ;add y term
 ;r3 is what we want to write
cmp r9,#8; if BITS_PER_PIXEL == 8
beq dp_eight
 cmp r9,#16;
 beq dp_sixteen
;assume 32
 str r6,[r0] ;for 32-bit colour
 b dp_endif
dp_sixteen:
 strh r3,[r0] ;copy low bytes (Half) to r0
 b dp_endif
dp_eight:
 strb r6,[r0] ;for 8-bit colour
dp_endif:
bx lr
```

```
; 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 = $FE000000 ; 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
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 r4, #1 ;x ordinate
mov r5, #1 ;y
;set colour - while for 8BPP, Yellow for 16BPP
mov r9,BITS_PER_PIXEL
cmp r9,#8; if BITS_PER_PIXEL == 16
beq sp_eight
;assume 16
 mov r6,$FF00
 orr r6,$00EE ; yellow
 b sp_endif
sp_eight:
 mov r6,#1 ;white for 8-bit colour
sp_endif:
lineloop:
 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}

add r4,#1
```

```
 mov r8,SCREEN_X AND $FF00
 orr r8,SCREEN_X AND $00FF ;640 = 0x0280
 cmp r4,r8
bls lineloop ;branch less than or same
;flash the LED to show we're almost finished
push {r0-r9}
mov r0,BASE
bl FLASH
pop {r0-r9}

push {r0-r3}
 mov r0,r7 ;screen address
 mov r1,#1 ;x
 mov r2,#419 ;y
 mov r3,r6 ;colour

 bl drawpixel
 pop {r0-r3}
 push {r0-r3}
 mov r0,r7 ;screen address
 mov r1,#559 ;x
 mov r2,#419 ;y
 mov r3,r6 ;colour

bl drawpixel
 pop {r0-r3}
 ; draw pixel in middle of the screen
 push {r0-r3}
```

```
 mov r0,r7
 mov r1,#319 ;x
 mov r2,#239 ;y
 mov r3,r6 ;colour

 bl drawpixel
 pop {r0-r3}
Loop:
 b Loop ;wait forever
CoreLoop: ; Infinite Loop For Core 1..3
 b CoreLoop
include "FBinit8.asm"
include "timer2_2Param.asm"
include "flash.asm"
include "drawpixel.asm"
format binary as 'img' ;must be first
mov sp,$1000 ;initialise stack pointer
;momentary switch swaps LEDs
org $8000
;GPIOs:
;pin 17: +3.3v
;pin 19: GPI10 (input)
; outputs:
;pin 12: GPIO18 (LED 1)
;pin 14: GPIO23 (LED 2)
;NC: pin 19 not connected (GPIO 10)
;Pull-up: pin 19 connected to +3.3V (pin 17)
BASE = $FE000000 ; Swap with $3F000000 for Pi 2/ 3B/ 3B+
```

```
GPIO_OFFSET = $200000
mov r0,BASE
orr r0,GPIO_OFFSET ;Base address of GPIO
ldr r1,[r0,#4] ;read function register for GPIO 10 - 19
bic r1,r1, #7 ; ; #27 ;bit clear 27 = 9 * 3 = read access
str r1,[r0,#4];10 input
;set up outputs
ldr r10,[r0,#4] ; LED 1 (GPIO18)
orr r10, $1000000 ;set bit 24
str r10,[r0,#4] ; GPIO18 output
ldr r2,[r0,#8] ; LED 2 (GPIO23)
orr r2, $200 ;set bit 9
str r2,[r0,#8] ; GPIO23 output
;activate LED 1
mov r2,#1
lsl r2,#18 ;bit 18 to write to GPIO18
;disable LED 2
mov r10,#1
lsl r10,#23 ;bit 23 to write to GPIO23
;poll GPIO10 and swap LEDS if high
loop$:
;read first block of GPIOs
ldr r9,[r0,#52] ;read gpios 0-31
tst r9,#1024 ; use tst to check bit 10
bne led2 ;if ==0
;else LED 1
str r2,[r0,#28] ;Turn on LED 1
b cont
```

```
led2:
str r10,[r0,#28] ;Turn on LED 2
cont:
;call timer
push {r0-r3}
mov r0,BASE
mov r1,$70000
orr r1,$0A100
orr r1,$00020 ;TIMER_MICROSECONDS = 500,000
bl Delay
pop {r0-r3}
ldr r9,[r0,#52] ;read gpios 0-31
tst r9,#1024 ; use tst to check bit 10
bne led2_2 ; if ==0
;else
str r2,[r0,#40] ;Turn off LED 1
b cont2
led2_2:
str r10,[r0,#40] ;Turn off LED 2
cont2:
;call timer
push {r0-r3}
mov r0,BASE
mov r1,$70000
orr r1,$0A100
orr r1,$00020 ;TIMER_MICROSECONDS = 500,000
bl Delay
pop {r0-r3}
```

b loop$

include "timer2_2Param.asm

Photo: