Lab 10

Thanh Minh

4. It calls the parameters (x,y,colors)from the drawpixel file so that the screen can draw the output. These parameters are calculated and stored in the registers.

5. The Drawpixel function is called when the bl drawpixel is mentioned. The x value assign to r1 and y value will be assign to r2. The colour is first stored in r6 but then it is moved to r3.

Kernel.asm

; 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

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 == 8

beq sp\_eight

;assume 16

mov r6,$FF00

orr r6,$000E ; 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

;assume BITS\_PER\_PIXEL, SCREEN\_X are shared constants

bl drawpixel

pop {r0-r3}

;increment and test

add r4,#1

;mov r8,SCREEN\_X AND $FF00

;orr r8,SCREEN\_X AND $00FF ;640 = 0x0280

cmp r4,100

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}

mov r4,#1

mov r5,#1

lineloop1:

;draw corners of screen

push {r0-r3}

mov r0,r7 ;screen address

mov r1,r4 ;x

mov r2,r5 ;y

mov r3,r6 ;colour

;assume BITS\_PER\_PIXEL, SCREEN\_X are shared constants

bl drawpixel

pop {r0-r3}

add r5,#1

cmp r5,#100

bls lineloop1

lineloop2:

push {r0-r3}

mov r0,r7 ;screen address

mov r1,r4 ;x

mov r2,r5 ;y

mov r3,r6 ;colour

;assume BITS\_PER\_PIXEL, SCREEN\_X are shared constants

bl drawpixel

pop {r0-r3}

add r5,#1

cmp r5,#100

bls lineloop2

mov r4,#1

mov r5,#100

lineloop3:

; draw pixel in middle of the screen

push {r0-r3}

mov r0,r7

mov r1,r4 ;x

mov r2,r5 ;y

mov r3,r6 ;colour

; assume BITS\_PER\_PIXEL, SCREEN\_X are shared constants

bl drawpixel

pop {r0-r3}

add r4,#1

cmp r4,#100

bls lineloop3

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"