How to separate out functions in Assembly code.

COS10004 CS Lab 9

Implementing functions

- In this lab you are going to modify existing code by introducing functions
- You are also going to create a separate file for each new function
- The following steps walk you through:
  - Mostly you can just follow the steps, but it is worth first thinking about what you would do yourself first

1. Open Kernel7.asm and separate out GPIO function for initialising LED

```
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer

BASE = $FE000000 ;$3F000000 for RP2 and 3
;GPIO_SETUP
GPIO_OFFSET = $200000
mov r0,BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]     ;set GPIO18 to output

loop$:
  mov r1,#1
  lsl r1,#15
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything else so no need to push/pop
  bl TIMER
  mov r1,#1
  lsl r1,#15
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
  bl TIMER
sub r7,#1
cmp r7,#0
bne  loop$  ;end of outer loop. Runs r7 times
wait:
b wait
include "TIMER.asm"
include "factorialj.asm"
```

This is the code that sets up the GPIO For writing to GPIO 18. Separate this out into its own function called SETUP_LED. It should take the base address as a parameter.

The next slide shows you how, but have a go yourself first !

1. Open Kernel7.asm and separate out GPIO function for initialising LED

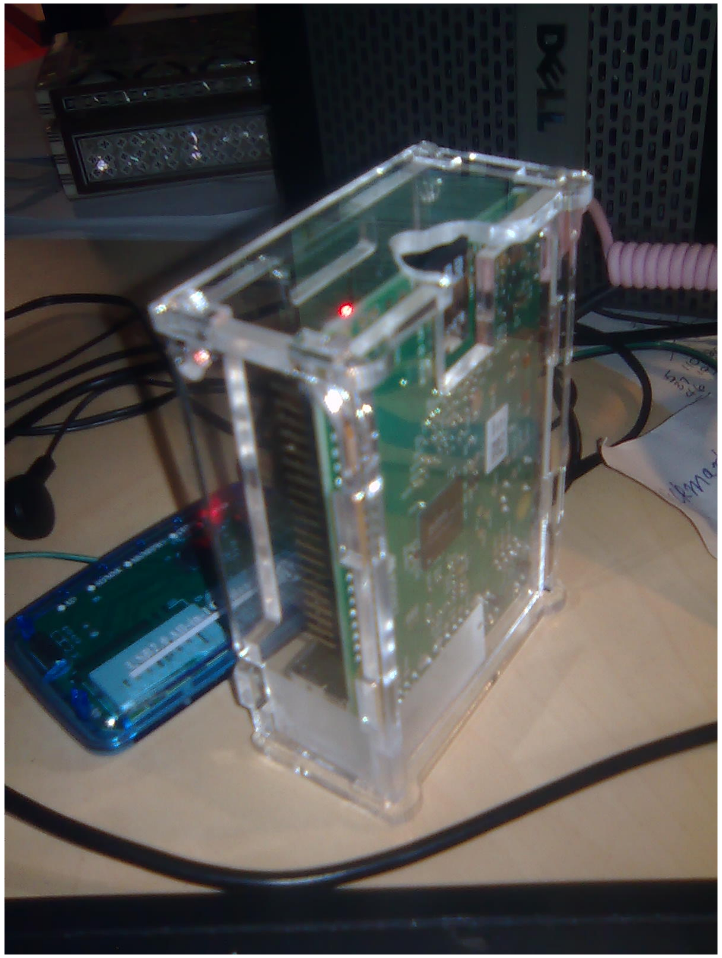
```
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer

BASE = $FE000000 ; $3F000000 for RP2 and 3

;GPIO_SETUP
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED

SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]     ;set GPIO18 to output
bx lr

loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything else so no need to push/pop
  bl TIMER
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
  bl TIMER
sub r7,#1
cmp r7,#0
bne  loop$  ;end of outer loop. Runs r7 times
wait:
b wait
include "TIMER.asm"
```

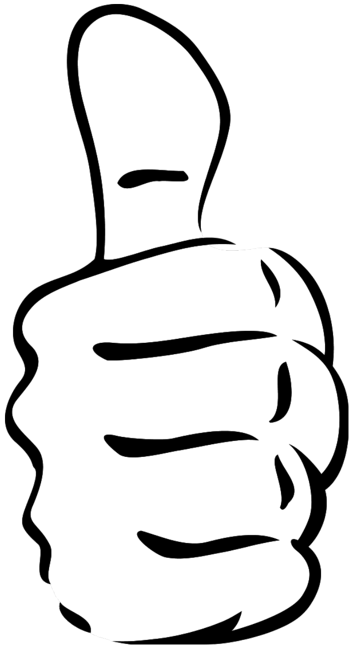
1. Open Kernel7.asm and separate out GPIO function for initialising LED

```
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer

BASE = $3F000000 ;RP2

;GPIO_SETUP
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED

loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything else so no need to push/pop
  bl TIMER
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
  bl TIMER
sub r7,#1
cmp r7,#0
bne  loop$  ;end of outer loop. Runs r7 times
wait:
b wait
include "TIMER.asm"
include "factorialj.asm"

SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]     ;set GPIO18 to output
bx lr
```

To tidy things up, move the SETUP_LED function to the bottom of the file. Program control will jump to this address when it executes bl SETUP_LED (and return of course when it is complete - when bx lr is executed !)

compile and test





## 2.	Separate out GPIO function for Flashing LED

```
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer
BASE = $FE000000 ;RP4

;GPIO_SETUP
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED

loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything else so no need to push/pop
  bl TIMER
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
  bl TIMER
sub r7,#1
cmp r7,#0
bne  loop$  ;end of outer loop. Runs r7 times
wait:
b wait
include "TIMER.asm"
include "factorialj.asm"

SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]     ;set GPIO18 to output
bx lr
```

OK – now its time to create a function
For flashing the LED – create a function
Called FLASH and move this code into it.
The function will need two arguments, the
BASE address (r0) and the number of flashes to flash
(i.e. r7 - the result of the factorial function)

Again – have a go before you skip ahead and see our solution!

2. Separate out GPIO function for Flashing LED

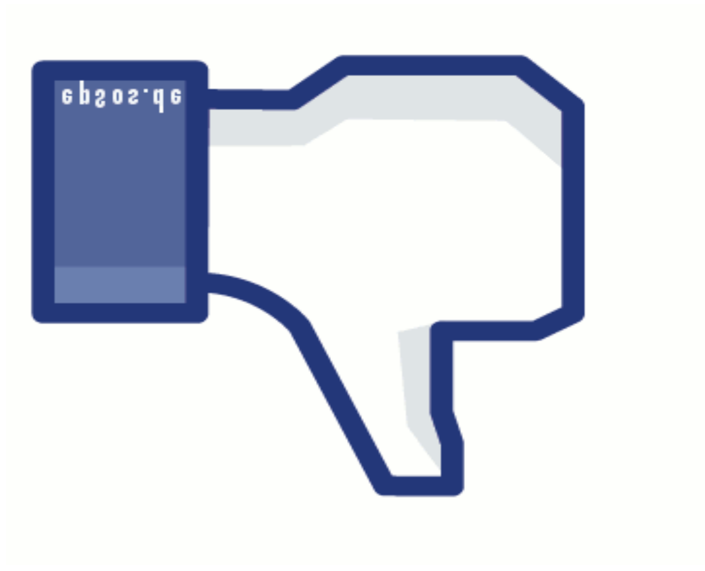
```
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer
BASE = $3F000000 ;RP2 ;GPIO_SETUP
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED

push {r0,r1}
mov r0,BASE
mov r1,r7
bl FLASH
pop {r0,r1}

FLASH:
;param r0=BASE
;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything else so no need to push/pop
  bl TIMER
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
  bl TIMER
sub r7,#1
cmp r7,#0
bne  loop$  ;end of outer loop. Runs r7 times
bx lr

wait:
b wait
include "TIMER.asm"
include "factorialj.asm"
```

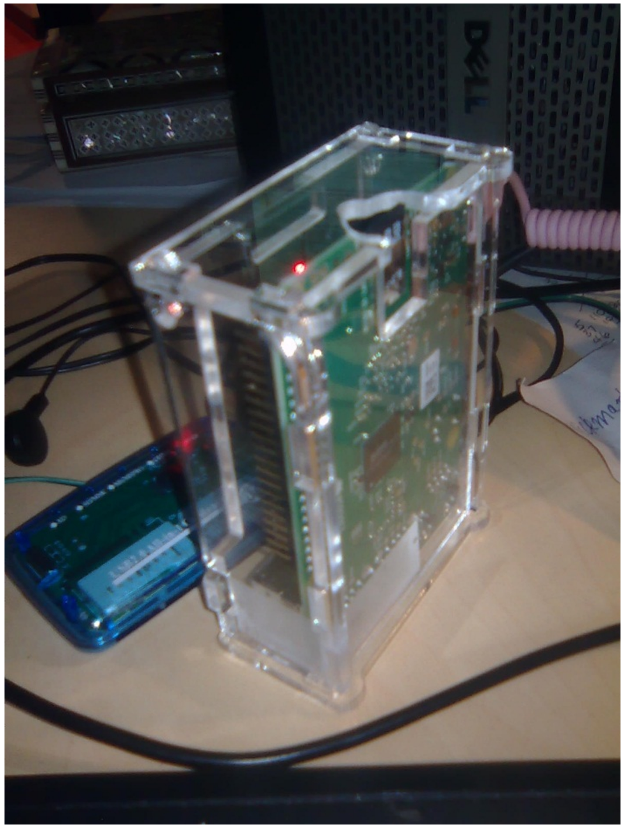
2. Separate out GPIO function for Flashing LED

```
...
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED

push {r0, r1}
mov r0,BASE
mov r1,r7
bl FLASH
pop {r0, r1}

wait:
b wait
include "TIMER.asm"
include "factorialj.asm"

SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]      ;set GPIO18 to output
bx lr

FLASH:
;param r0=BASE
;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything else so no need to push/pop
  bl TIMER
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
  bl TIMER
sub r7,#1
cmp r7,#0
bne  loop$  ;end of outer loop. Runs r7 times
```

As before – tidy things up by moving the FLASH function to the bottom.

compile and test





compile and test

- Problem. Flashes for more than 24 times.
- Why ?
  - this is a challenge to work out, but follow the logic and see if you can, then proceed to fix it.
  - Hint: look at what happens to the lr register when FLASH is executed (which also calls TIMER)… I mentioned this issue in lectures!

OK – the gory details then …

- The link register has been overwritten by the nested function call:

1. call FACTORIAL (from factorialj.asm) overwrites lr – all good
2. returns 4! (24) (copies lr to pc) – all good
3. put 24 in r7
4. call SETUP_LED (from GPIO.asm) overwrites lr – all good
5. returns (copies lr to pc) – all good
6. call FLASH (from GPIO.asm) overwrites lr - so far so good
7. FLASH reads value in r0 (4!)
8. FLASH calls TIMER (from timer2_2param.asm) overwrites lr – not so good
9. and repeats 24 times (copies lr to pc)
10. FLASH returns (copies lr to pc)
    - wrong lr - correct one has been overwritten

11. return and run infinite loop

Solution

Edit the FLASH function to backup the value in "lr" onto the stack before TIMER is called (and to restore from the backup after TIMER completes).

- Have a go yourself before skipping ahead

2. Separate out GPIO function for Flashing LED

```
...
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED
mov r0,BASE
mov r1,r7
bl FLASH
wait:
b wait
include "TIMER.asm"
include "factorialj.asm"
SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]     ;set GPIO18 to output
bx lr
FLASH:
;param r0=BASE
;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything else so no need to push/pop
     push {lr}
     bl TIMER
     pop {lr}
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
     push {lr}
     bl TIMER
     pop {lr}
sub r7,#1
cmp r7,#0
bne  loop$  ;end of outer loop. Runs r7 times
bx lr
```

these fixed it.

3. Move GPIO functions into their own file

```
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer
BASE = $3F000000 ;RP2 ;GPIO_SETUP
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED
mov r0,BASE
mov r1,r7
bl FLASH
wait:
b wait

include "TIMER.asm"
include "factorialj.asm"
SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
```

To tidy things up, move the code for Setting up and flashing LED into a single file called gpio.asm (and include it in kernel7.asm)

```
str r1,[r0,#4]     ;set GPIO18 to output
bx lr

FLASH:
;param r0=BASE
;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything
else so no need to push/pop
     push {lr}
     bl TIMER
     pop {lr}
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
     push {lr}
     bl TIMER
     pop {lr}
sub r7,r7,#1
cmp r7,#0
ble  loop$  ;end of outer loop. Runs r7 times
bx lr
```

```
;GPIO.asm
SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]     ;set GPIO18 to output
bx lr

FLASH:
;param r0=BASE
;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything
else so no need to push/pop
   push {lr}
     bl TIMER
     pop {lr}
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
   push {lr}
     bl TIMER
     pop {lr}
sub r7,r7,#1
cmp r7,#0
bgt  loop$  ;end of outer loop. Runs r7 times
bx lr
```

This is what your gpio.asm file should now look like

```
;kernel7.asm
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer
BASE = $3F000000 ;RP2 ;GPIO_SETUP
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED
mov r0,BASE
mov r1,r7
bl FLASH
wait:
b wait

include "TIMER.asm"
include "factorialj.asm"
include "GPIO.asm"
```

This is what your kernel7.asm file should now look like.

Remember to include GPIO.asm !

```
;kernel7.asm
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer
BASE = $3F000000 ;RP2 ;GPIO_SETUP
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED
GPIO_OFFSET = $200000
mov r0,BASE
bl SETUP_LED
mov r0,BASE
mov r1,r7
bl FLASH
wait:
b wait

include "TIMER.asm"
include "factorialj.asm"
include "GPIO.asm"
```

We don't actually use GPIO_OFFSET in kernel7, so we'll move it to GPIO.asm where it is relevant.

3. Move GPIO functions into their own file

```
;kernel7.asm
;Calculate
mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer
BASE = $3F000000 ;RP2 ;GPIO_SETUP
[redacted]
mov r0,BASE
bl SETUP_LED
[redacted]
mov r0,BASE
bl SETUP_LED
mov r0,BASE
mov r1,r7
bl FLASH
wait:
b wait

include "TIMER.asm"
include "factorialj.asm"
include "GPIO.asm"
```

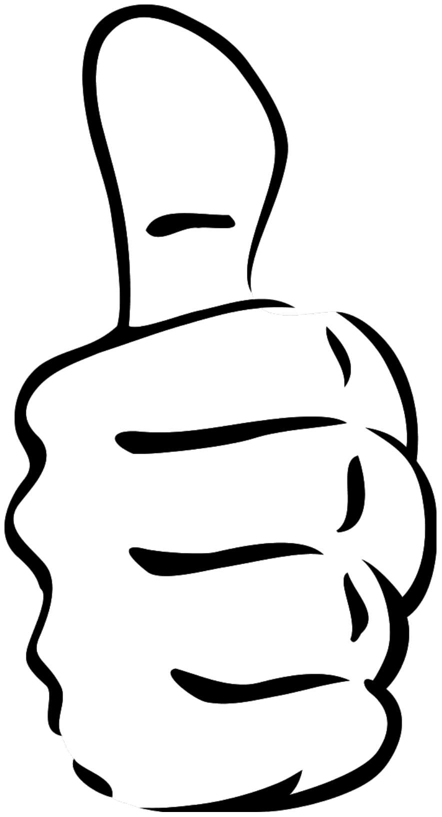
So delete GPIO_OFFSET assignment from here

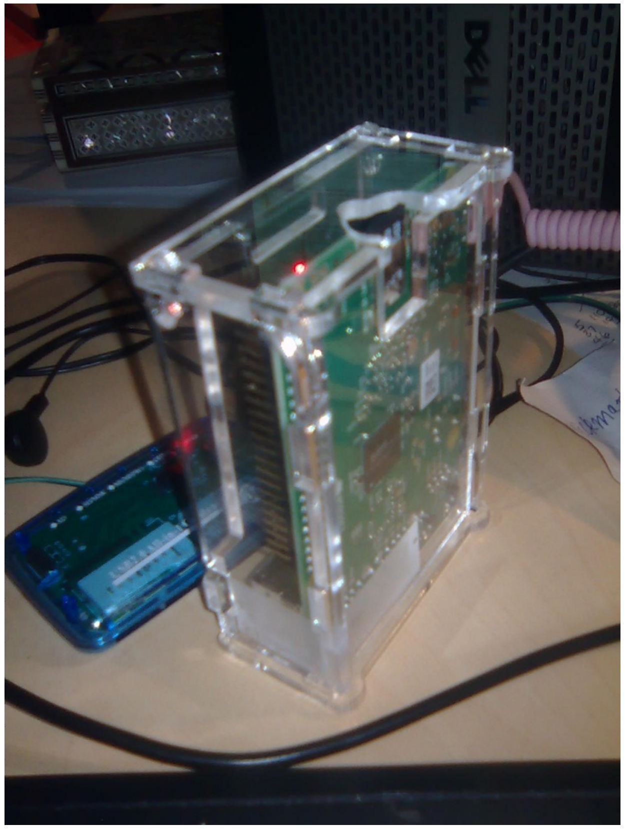
3. Move GPIO functions into their own file

```
;GPIO.asm
GPIO_OFFSET = $200000
SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]      ;set GPIO18 to output


FLASH:

;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything
else so no need to push/pop
   push {lr}
     bl TIMER
     pop {lr}
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
   push {lr}
     bl TIMER
     pop {lr}
sub r7,r7,#1
cmp r7,#0
bgt  loop$  ;end of outer loop. Runs r7 times
bx lr
```

And put it here (GPIO.asm) where it is relevant.

compile and test





Level Up

- Suppose your function needs lots of registers.
- But you're already using them.
- push them onto the stack
- set param registers
- call function
- pop registers
- Replace the dumb timer in Lab 8 with the accurate timer discussed in Week 9’s lecture.

Step 1. Substitute timer function

```
;kernel7.asm

mov r1,#4 ;input
mov sp,$1000  ;make room on the stack
mov r0,r1
bl FACTORIAL
mov r7,r0 ;store answer

BASE = $FE000000 ;RP4
;GPIO_SETUP
mov r0,BASE

bl SETUP_LED

mov r0,BASE
mov r1,r7
bl FLASH

wait:
b wait
include "timer2_2Param.asm"
include "factorialj.asm"
include "GPIO.asm"
```

```
; timer2_2Param.asm

Delay:  ;this function has 2 parameters
TIMER_OFFSET=$3000
mov r3,r0  ;BASE   - depends on Pi model
orr r3,TIMER_OFFSET
mov r4,r1  ;$80000 passed as a parameter
ldrd r6,r7,[r3,#4]
mov r5,r6
loopt1:  ;label still has to be
different from one in _start
  ldrd r6,r7,[r3,#4]
  sub r8,r6,r5
  cmp  r8,r4
  bls loopt1
bx lr  ;return
```

Step 2. Look for reused registers

```
 ;gpio.asm
GPIO_OFFSET = $200000
SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]      ;set GPIO18 to output
bx lr

FLASH:
;param r0=BASE
;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
  mov r2,$0F0000  ;not using r2 for anything else so
no need to push/pop it
   push {lr}
     bl TIMER
     pop {lr}
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
  mov r2,$0F0000
   push {lr}
     bl TIMER
     pop {lr}
sub r7,r7,#1
```

r0,r1,r2,r7

```
cmp r7,#0
bgt  loop$  ;end of outer loop. Runs r7 times
bx lr
```

```
 ; timer2_2Param.asm
```

```
Delay:  ;this function has 2 parameters
TIMER_OFFSET=$3000
mov r3,r0  ;BASE   - depends on Pi model
orr r3,TIMER_OFFSET
mov r4,r1  ;$80000 passed as a parameter
ldrd r6,r7,[r3,#4]
mov r5,r6
loopt1:  ;label still has to be different from one
in _start
  ldrd r6,r7,[r3,#4]
  sub r8,r6,r5
  cmp  r8,r4
  bls loopt1
bx lr  ;return
```

r0,r1,r3,r4,r5,r6,r7,r8

Step 2. Look for required constants, labels and fix

```
 ;gpio.asm
GPIO_OFFSET = $200000
SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#14]      ;set GPIO18 to output
bx lr

FLASH:
;param r0=BASE
;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]  ;turn LED on
   push {r0,r1,r7,lr} ;r0,r1,r7 in use push and then
set parameters
  mov r0,BASE
  mov r1,$0F0000
  bl TIMER
  pop {r0,r1,r7,lr}
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]  ;turn LED off
 push {r0,r1,r7,lr} ;r0,r1,r7 in use push and then
set parameters
  mov r0,BASE
  mov r1,$0F0000
```

```
  bl TIMER
  pop {r0,r1,r7,lr}
sub r7,r7,#1
cmp r7,#0
bgt  loop$   ;end of outer loop. Runs r7 times
bx lr


 ; timer2_2Param.asm

Delay:  ;this function has 2 parameters
TIMER_OFFSET=$3000
mov r3,r0  ;BASE   - depends on Pi model
orr r3,TIMER_OFFSET
mov r4,r1  ;$80000 passed as a parameter
ldrd r6,r7,[r3,#4]
mov r5,r6
loopt1:  ;label still has to be different from one
in _start
  ldrd r6,r7,[r3,#4]
  sub r8,r6,r5
  cmp  r8,r4
  bls loopt1
bx lr  ;return
```

r0=BASE (1st param), but it's overwritten here

FLASH calls TIMER, but the new timer is calling Delay, so we need to replace the label to branch to

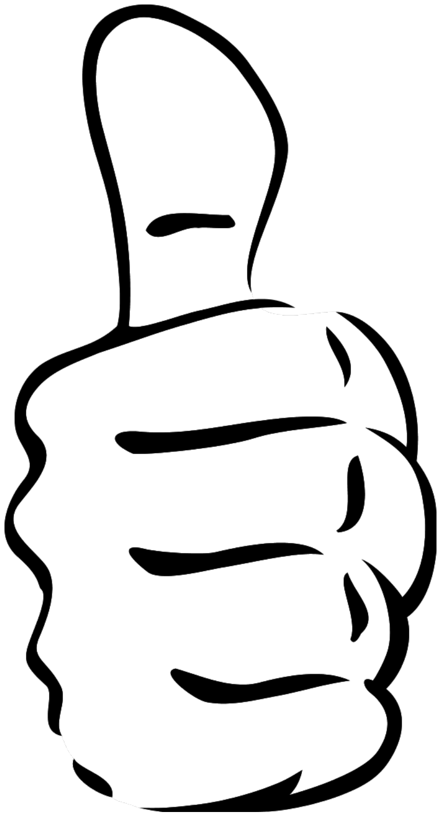
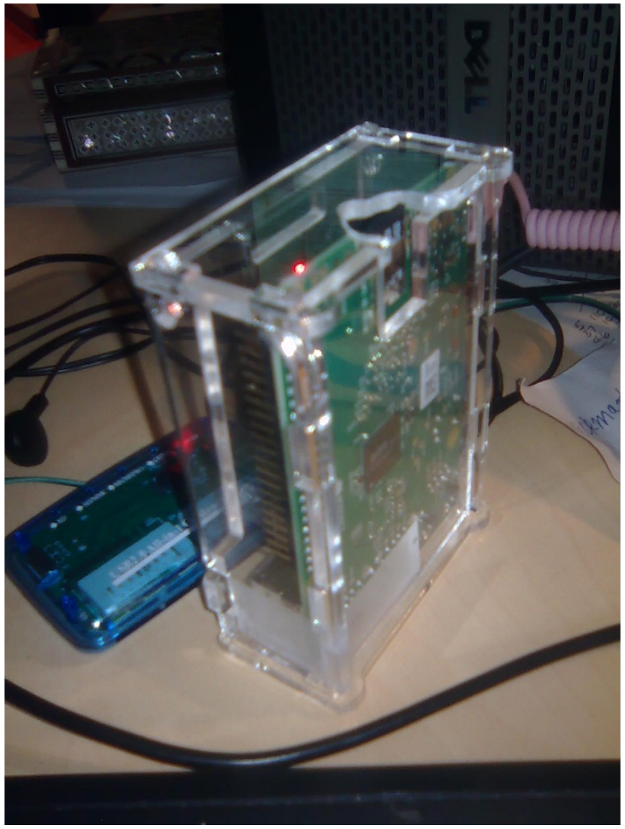
Step 3. Test

r0=BASE (1st param), backed up here

```
 ;gpio.asm
GPIO_OFFSET = $200000
SETUP_LED:
;param r0=BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#24
str r1,[r0,#4]      ;set GPIO18 to output
bx lr

FLASH:
;param r0=BASE
;need BASE for timer, so copy to r2 here
mov r2,r0
;param r1 = number of flashes
orr r0,GPIO_OFFSET
mov r7,r1
loop$:
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#28]   ;turn LED on
   push {r0,r1,r7,lr} ;r0,r1,r7 in use push and then
set parameters
  mov r0,BASE
  mov r1,$0F0000
  bl Delay
  pop {r0,r1,r7,lr}
  mov r1,#1
  lsl r1,#18
  str r1,[r0,#40]   ;turn LED off
 push {r0,r1,r7,lr} ;r0,r1,r7 in use push and then
set parameters
  mov r0,BASE
  mov r1,$0F0000
  bl Delay
  pop {r0,r1,r7,lr}
sub r7,r7,#1
cmp r7,#0
bgt  loop$   ;end of outer loop. Runs r7 times
bx lr


 ; timer2_2Param.asm

Delay:  ;this function has 2 parameters
TIMER_OFFSET=$3000
mov r3,r0   ;BASE   - depends on Pi model
orr r3,TIMER_OFFSET
mov r4,r1   ;$80000 passed as a parameter
ldrd r6,r7,[r3,#4]
mov r5,r6
loopt1:  ;label still
in _start
  ldrd r6,r7,[r3,#4]
  sub r8,r6,r5
  cmp  r8,r4
  bls loopt1
bx lr  ;return
```

FLASH calls TIMER, but the new timer is calling Delay, so we need to replace the label to branch to

compile and test

Better software design

- Should put the includes in the files that use the function
- kernel7.img should include factorialj.asm and GPIO.asm
- GPIO.asm should include timer2_2param.asm
- Should not have to rely on constants or registers being shared between files. (but they are, so keep labels and names unique).