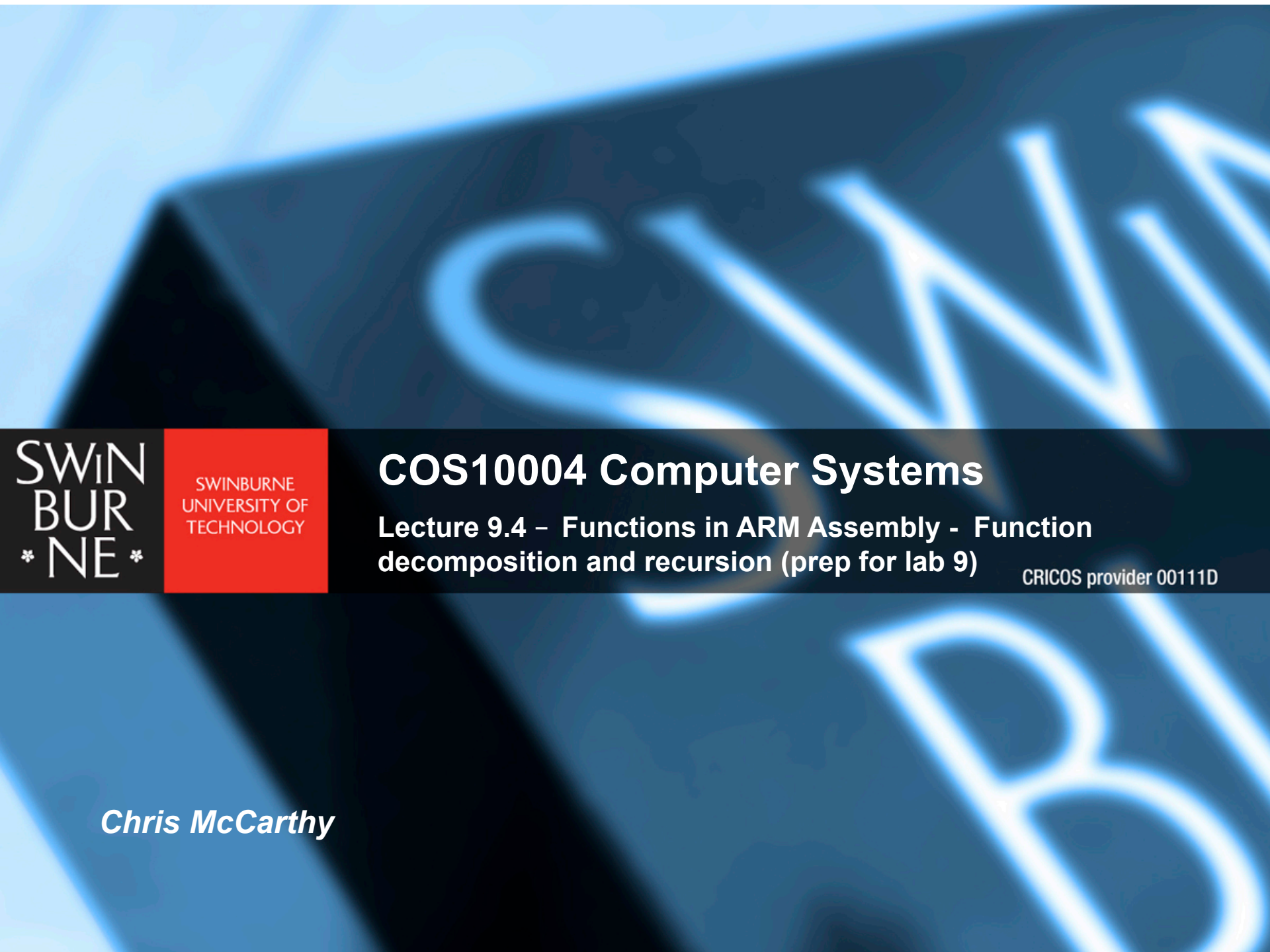

COS10004 Computer Systems

Lecture 9.4 – Functions in ARM Assembly - Function decomposition and recursion (prep for lab 9)

CRICOS provider 00111D

Chris McCarthy

WEEK 9 LAB

- You are going to be “re-factoring” code by inserting asm functions
- The code will use the LED (wired to GPIO18) to flash the answer after computing the factorial of a number
- You are also going to break each function up into separate files:
- In this lecture:
  - How to include asm files in your program
  - Implementing recursive functions (e.g, like factorial)

Prep for Lab 9 - Dividing the work...

- gpio.asm - code for accessing the LED controlled by a GPIO register
- timer.asm - code for using the system timer
- flash.asm - code for flashing the LED *n* times (calls gpio and wait)
- kernel7.asm – startup code for calling flash

INCLUDES

- Put each function (or group of related functions) in a dedicated source file.
- The *include* command will combine them with your main.asm (the one you compile) and assemble as one source file.
- *include* literally performs a text substitution:
  - Done prior to compilation
  - To be safe, I put includes at the bottom of the asm file I am including into

PUTTING IT TOGETHER...

```
;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 = $200000
mov r0,BASE
orr r0,GPIO_OFFSET
mov r1,#1
lsl r1,#21
str r1,[r0,#16]     ;set GPIO47 to output
loop$:
  mov r1,#1
  lsl r1,#15
  str r1,[r0,#32]  ;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,#44]  ;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"
```

RECURSION

- Functions can call other functions, which includes themselves !
  - This is known as recursion
- When a function calls itself, a new *copy* of the function is needed
  - While the older copy that called it waits for it to return (just like any function)
- Each copy operates with its own local variables
  - This means backing up relevant registers using the stack!

RECURSION

- Using the stack there is almost no limit to how many processes we can launch.
- We can get a function to call itself over and over again because the stack will hold all the temporary values (inputs, outputs) and play them back in the right order.
- We can program an algorithm to keep going (refining the answer) until it reaches a determined “base case”
  - e.g. reaching the end of a list when computing the sum of a list of numbers

THINGS WE NEED FOR RECURSION

1. A function must call itself.
2. A function must pass a parameter to itself.
3. The parameter must change in a systematic way.
4. A function must have an exit condition (so that it will stop calling itself.
   - In ASM, we need to push the LR onto the stack for each function call so that all of the calls return a value.
   - Each return will use a different value of LR.

Following code adapted from: http://www.slideshare.net/StephanCadene/arm-procedure-calling-conventions-and-recursion

FACTORIAL

factorialj.asm

- Factorial(n) – n*n-1*n-2*n-3*…*1
- e.g. 4! = 4*3*2*1

```
FACTORIAL:
sub r1,r1,#1    ;3. r1 approaches 1
cmp r1,#1       ;4. exit if 1
beq EXIT
mul r0,r0,r1    ;total=total*param
push {r1,lr}    ;2. push onto the stack,
                ;preserving the PC.
bl FACTORIAL    ;1. call FACTORIAL
EXIT:
pop {r1,lr}   ;pop off the stack
bx lr         ;RETURN
```

CALLING FACTORIALJ.ASM

```
format binary as 'img'   ;must be first
;kernel7.asm
;r0 = input param
;r1 = working answer initialised to 4
mov r1,#4        ;input
mov sp,$1000     ;make room on the stack
mov r0,r1

bl FACTORIAL
mov r7,r0        ;retrieve answer
include "factorialj.asm"
```

GETTING THE ANSWER OUT

- We can flash the LED ANSWER times.

```
loop$:          ;r7 contains the ANSWER
  mov r1,#1
  lsl r1,#15
  str r1,[r0,#32]   ;turn LED on

  mov r2,$0F0000
  bl TIMER   ;just a dumb timer here

  mov r1,#1
  lsl r1,#15
  str r1,[r0,#44]   ;turn LED off

  mov r2,$0F0000
  bl TIMER

  sub r7,#1
  cmp r7,#0
bne  loop$
```

THE DUMB TIMER FUNCTION

```
;TIMER.asm - dumb timer
;r2=number of loops
TIMER:
  wait1$:
    sub r2,#1
    cmp r2,#0
    bne wait1$
bx lr
```

THE LAB (AND SUMMARY)

- You're going to implement functions and change over the timer in a factorial calculation program.
- The lab walks you through this process
- Pay attention to the need of the stack to backup all relevant registers when doing recursion
  - Think about lr in this context !