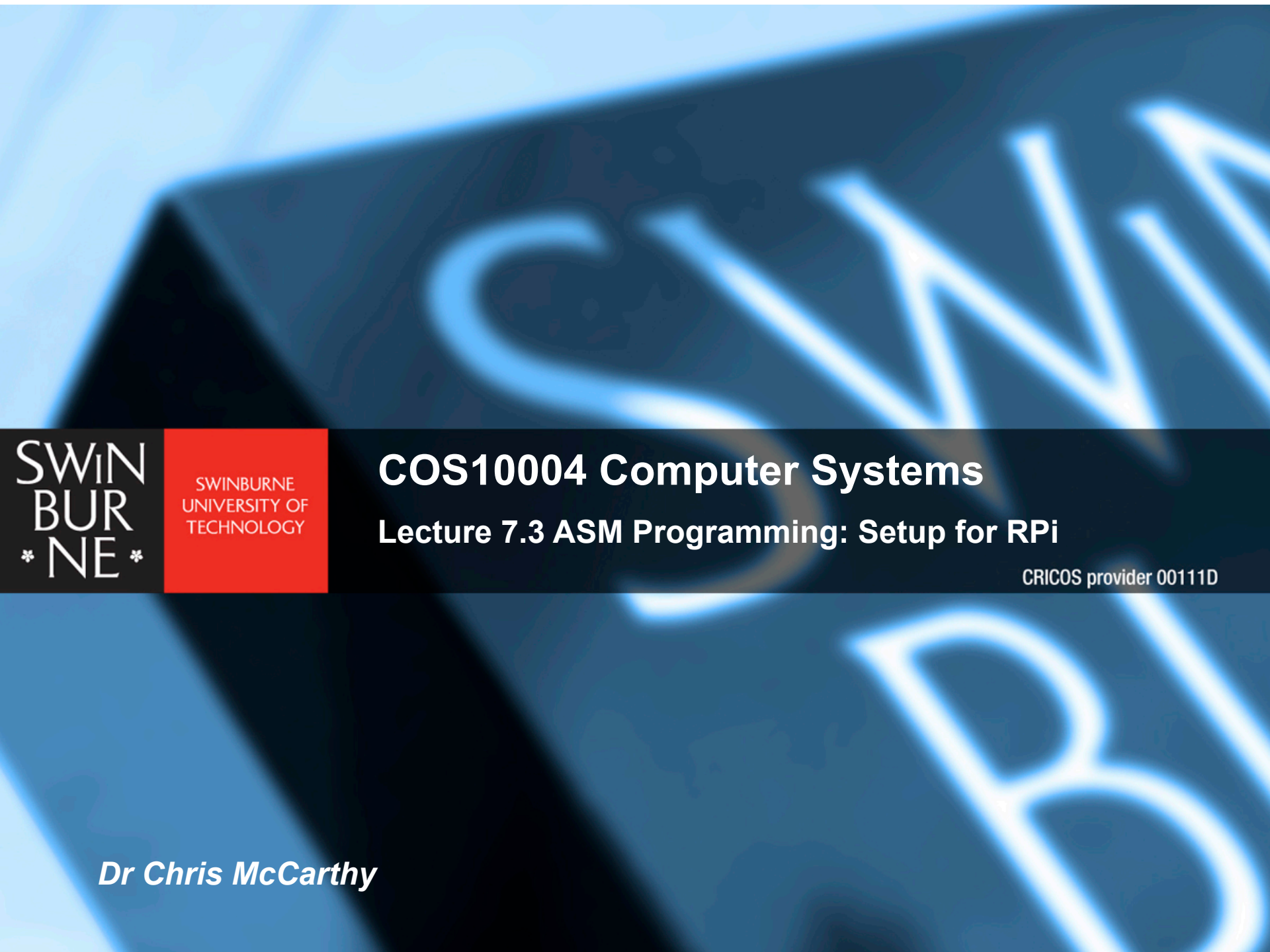

COS10004 Computer Systems

Lecture 7.3 ASM Programming: Setup for RPi

CRICOS provider 00111D

Dr Chris McCarthy

ARM ASSEMBLY PROGRAMMING

- Arm: Advanced RISC Machine
- The most widely used instruction set in the world:
  - 130 billion ARM processors produced
  - mobile devices, routers, IoT devices (eg RPi, Arduino)
- Consists of about 100 instructions in total:
  - We will explore some but certainly not all

RPI ARM PROCESSORS

Model	Processor	Instruction Set
Raspberry Pi 4 B 	Broadcom BCM2711, 1.5 GHz Quad core Cortex-A72	ARMv8 64 bit
Raspberry Pi 3B 	Broadcom BCM2837 , 1.2 GHz Quad Core Cortex-A53	ARMv8 64 bit
Raspberry Pi 2B 	Broadcom BCM2837, 900Mhz Quad Core Cortex-A7	ARMv7 32 bit

ARM instruction sets for RPi

- ARMv8 supports 64 bit operations, but is also 32 bit compatible (RPi 3 & 4)
- We will be working with 32 bit instruction set
- What does that mean ?
  - 32 bit-wide registers
  - 32 bit-wide addresses
- We specify which ARM instruction set using filename conventions

BARE METAL ASM PROGRAMMING

- We will be writing bare metal assembly (asm) programming
- What does this mean ?
  - No OS to help (or hinder) us
  - Direct access to hardware registers and memory addresses (no virtual addressing)
  - Essentially just us and the built in BIOS of the RPi that hands us control

OUR COMPILATION PROCESS

- We write our code using FASMARM (on PC)
  1. Source code (ASM) -> compile -> <filename>.bin
  2. rename <filename>.bin -> kernel7.img
  3. kernel7.img
- We then execute our code on the Pi:
  1. Copy kernel7.img to microSD card,
  2. put card in Pi,
  3. Power-up Pi

ARM: Typical ASM structure

- ASM source code has various sections:

```
; comments
.section .init   ; for defini                    tart
.section .data  ;for hard
.section .text   ; for cod
.glob _start    ;name of
"functions"
_start:   ;label where co

.  ;Means assembler instruction
```

FASM inserts/manages these for us

ARM: FASM ASM SYNTAX

- ASM source code has one section (but you can include other files):

```
;comments
VARIABLELABEL = value  ; hard-coded variables (constants) and arrays
include 'othersourcefile.asm'    ; for other files

format binary as 'img'  ;specify ext of kernel (compiled) file

label:      ;start of function, dest. of goto, start of array or struct

;HEX numbers are represented with a leading $ (not 0x)
;Decimal numbers represented with a leading #
```

There are no…

- { }
- loop structures
- if structures, switch
- function argument lists
- pre-processor directives
- useful error messages

BUT YOU CAN...

- define functions
- define variables
- undefine variables (at run time)
- include files
- work with arrays
- call OS functions (if you have an OS)
- call C functions (if you link to a C library)
- use goto

RPI BOOT PROCESS

1. GPU:
   - (runs 1st stage bootloader (on SoC ROM).
2. 1st bootloader reads SD card, finds *bootcode.bin* (proprietary) and loads it into L2 cache.
3. 2nd bootloader (bootcode.bin) enables RAM, reads GPU firmware (start.elf)
   - (file not needed for RPi 4)
4. start.elf reads *config.txt* for special settings, loads and runs kernel:

`Runs kernel7.img` if it finds it

POST error codes (LED signals) here:
http://elinux.org/R-Pi_Troubleshooting#Power_.2F_Start-up

SD Card files

- You will need a correctly formatted micro SD card to successfully program your Pi
- Micro SD card needs correct files copied to root folder:
  - files are specific to your RPi model
- Files to download and formatting instructions are here:
  - https://github.com/FelipMarti/COS10004-RPi

SUMMARY

- Summarise how we will be programming our Pi using bare metal ARM ASM
- All files to download to your SD card and instructions for formatting can be found here:
  - https://github.com/FelipMarti/COS10004-RPi

- Next lecture… we program !