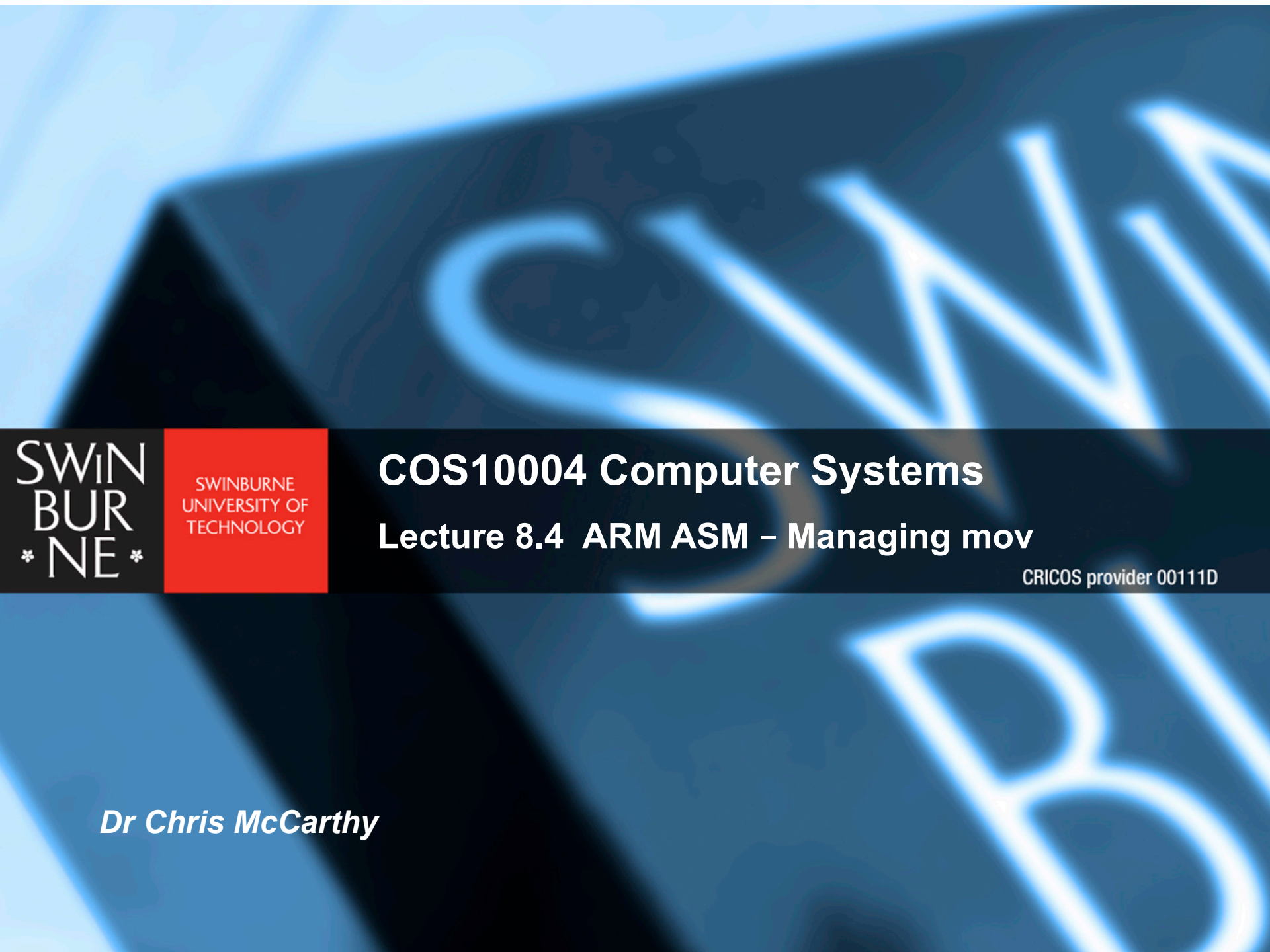


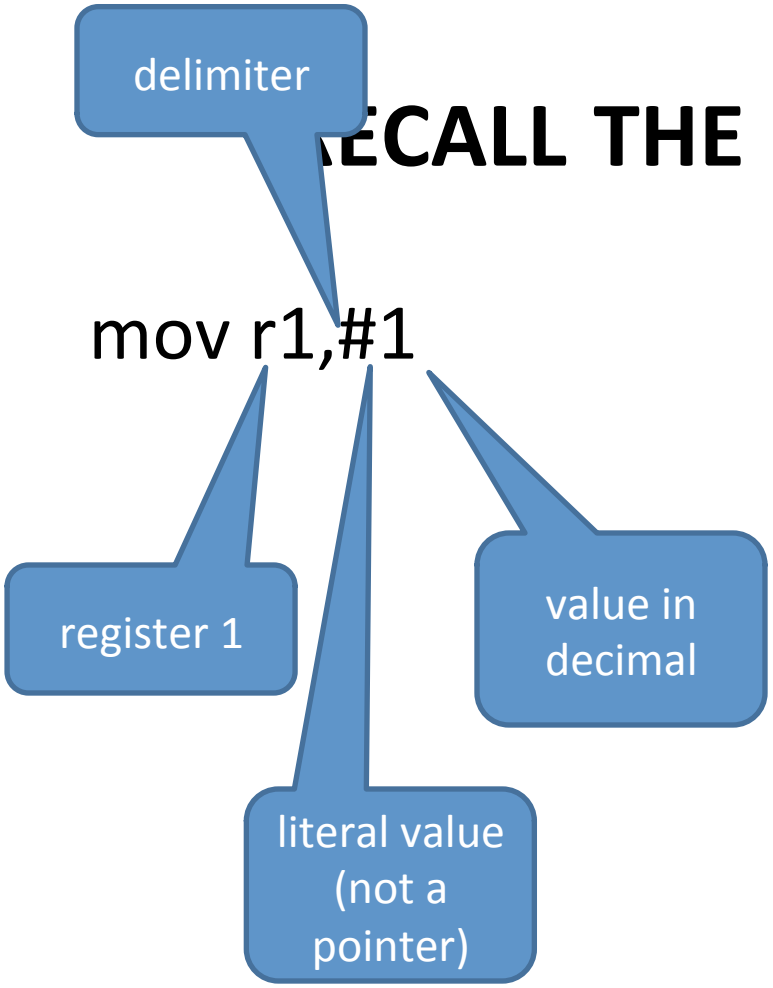
COS10004 Computer Systems

Lecture 8.4 ARM ASM – Managing mov

CRICOS provider 00111D

Dr Chris McCarthy

RECALL THE MOV COMMAND



- load register 1 with the value 1*

*not all numbers can be used

MANAGING NUMBERS WITH MOV

- The *mov* op code is really fast – 1 clock cycle
- It combines the operation (mov) and the operand in the one 32-bit word.
- The ALU/CPU can process this *immediately*
  - no copying from memory (using pointers)
  - No construction of instruction for the ALU.
- BUT:
  - it only accepts some numbers (those with at least 24 bits set to 0.

How does MOV manage numbers ?

- The 32 bit mov instructions has only 8 bits available to store the value to be moved!
- Other bits are for:
  - 20 bit op-code (the number representing the instruction − all instructions have an op-code)
  - 4 bits for a the ROR .
- The binary representation of the number to be moved must be rotated so that the op-code and the 4-bit ROR can be written over the unused 24 bits
  - This is done with a hardware shift register called a “barrel shifter”.
  - A shift register that wraps around!
- We end up with one 32-bit word containing all parts combined.
- This allows it to transfer a number into a register in **one clock cycle**.

BARREL SHIFTER

- **ARM CPUs have** a “barrel shifter” – a shift register where the overflow can feed back into the underflow (rotate).
- Allows
  - logical shift left, shift right (move the bits)
  - arithmetic shift right (halve the number)
  - rotate right, other rotates
  - Allows a shift in a word containing the instruction and the operand (i.e. a mov).

http://www.davespace.co.uk/arm/introduction-to-arm/barrel-shifter.html
http://www.slideshare.net/guest56d1b781/arm-fundamentals

IMPLICATION OF THIS APPROACH

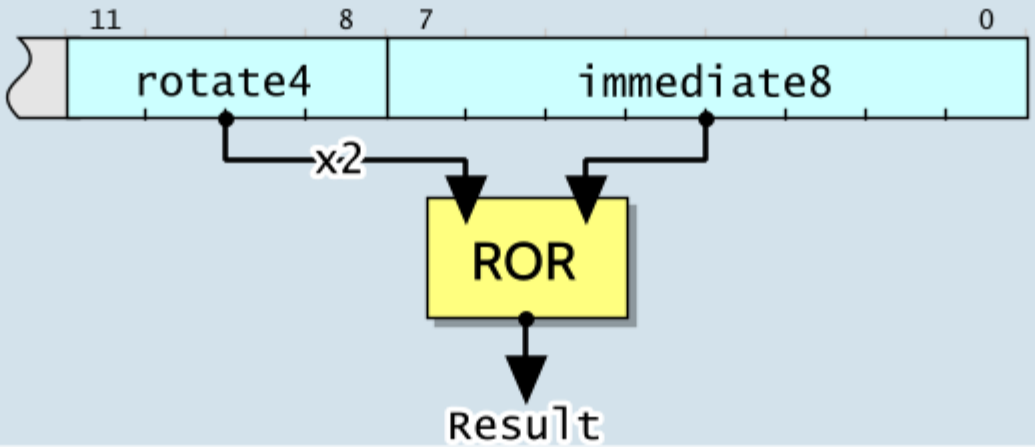
- mov can only handle 8 bit values
- BUT this isn't as limiting as it seems!
- Consider how `mov` maybe combined with `orr ?`

ARM: Introduction to ARM: Immediate Values

by David Thomas on 03 March 2012 — ARM IntroductionToARM Slide

Immediate Values

- You can't fit an arbitrary 32-bit value into a 32-bit instruction word.
- ARM data processing instructions have 12 bits of space for values in their instruction word. This is arranged as a four-bit rotate value and an eight-bit immediate value:



http://www.davespace.co.uk/arm/introduction-to-arm/immediates.html

ORR

- We can construct any 32-bit number by combining parts of the number with the ***orr*** operation.
  - e.g. 500,000 (0x7A120) is forbidden,
- but 0x70000, 0x0A100 and 0x00020 are allowed.

```
mov r3,$70000
orr r3,$0A100
orr r3,$00020
;r3 now equals 0x7A120 or 500,000
```

ORR WITH UNKNOWN NUMBERS

- We can break any number up into 1-byte chunks using a bit mask
- We can AND the value with 0xFF) and then orr (bitwise add) them with the r
- e.g.

some random value

```
mov r0,SOME_VALUE ;won't work … but …

mov r0,SOME_VALUE and $000000FF ;copy across
orr r0,SOME_VALUE and $0000FF00 ;1 byte at a time
orr r0,SOME_VALUE and $00FF0000 ;compiles on
orr r0,SOME_VALUE and $FF000000 ;anything
```

LDR

- Standard ARM ASM allows LDR to also be used to store constants
- Not supported in FASMARM though
  - FASMARM only supports ***ldr*** operations with addresses.

- ***ldr*** is actually a “pseudo instruction” :
  - It tries to perform a mov operation, and converts the immediate value to other things to fudge a working solution.

SUMMARY

- We can flash an LED:
  - pull GPIO pins high and low
  - Insert delays to create flashing effect
- Busy wait timer:
  - V1.0 “dumb timer” CPU dependent
  - V2.0 with timer register (real time delay !)
- `Mov` is a highly optimised number mover
  - Limited to 8 bit values
  - can be combined with orr to assemble values > 8 bits

Next Week

- We build functions in asm !
  - The building blocks of all programming languages!
  - Passing arguments
  - Reusing code
  - Utilising the hardware stack