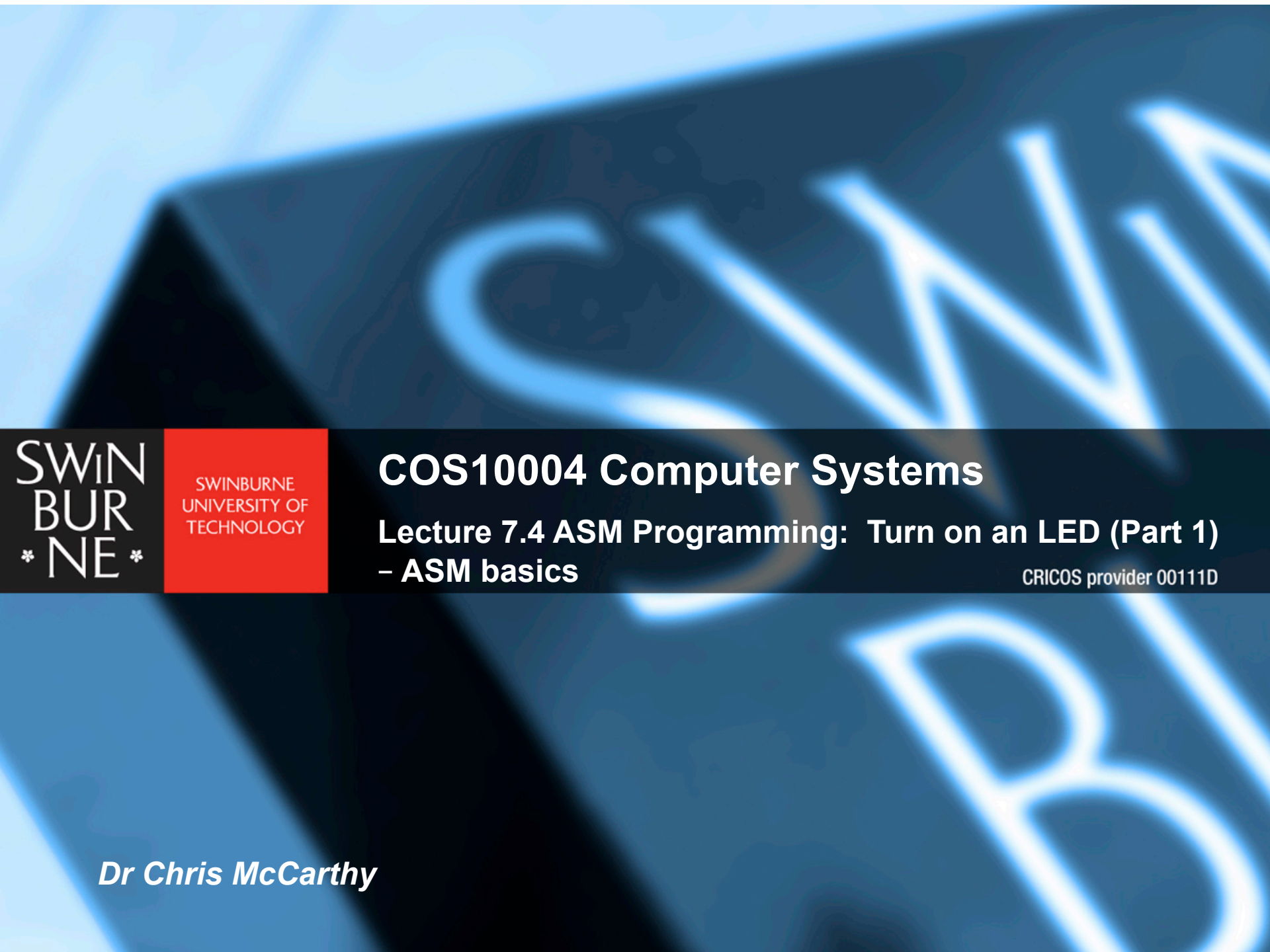


COS10004 Computer Systems

Lecture 7.4 ASM Programming: Turn on an LED (Part 1) – ASM basics

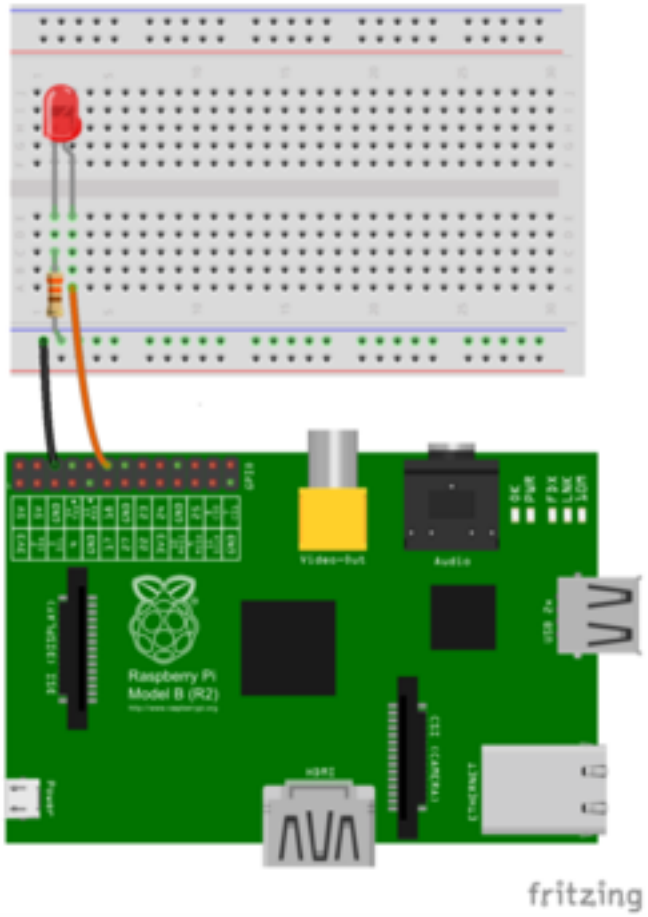
CRICOS provider 00111D

Dr Chris McCarthy

Turning on a Light!

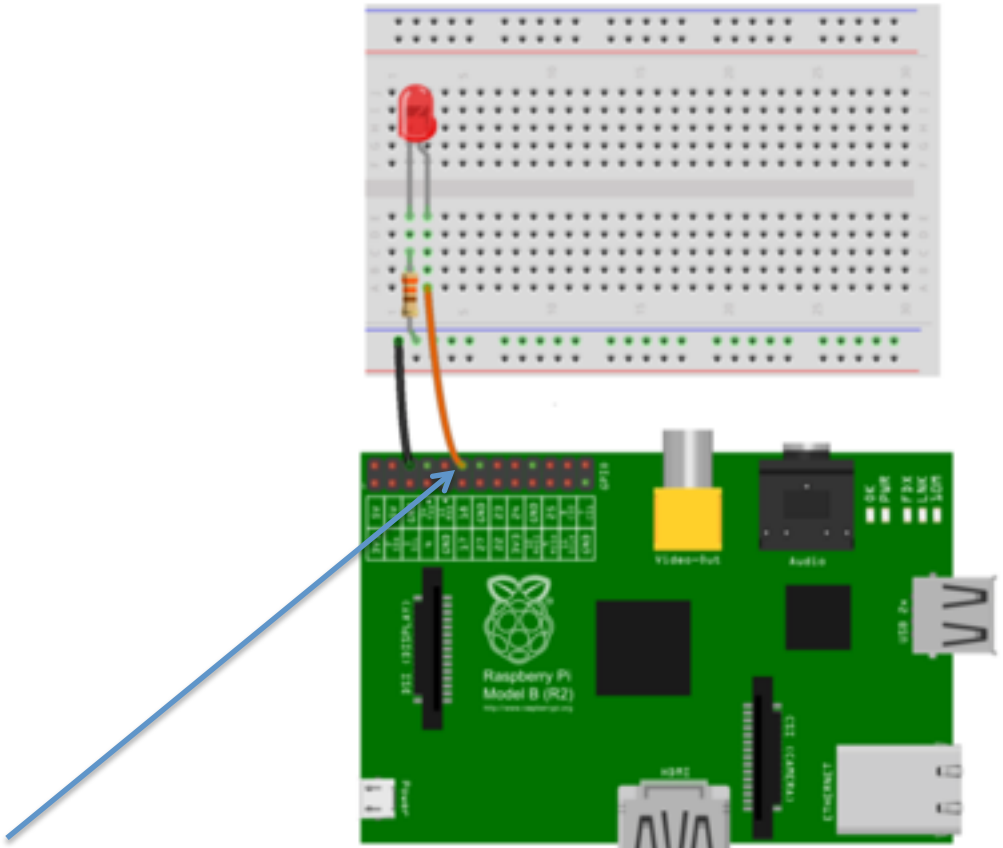
- We are now ready to program, so lets start playing !
- We're going to walk through the asm code for turning on an LED connected to a pin

First we wire it up



See https://www.youtube.com/watch?v=Rd9kvVs1lSQ for my tutorial on wiring this circuit

First we wire it up

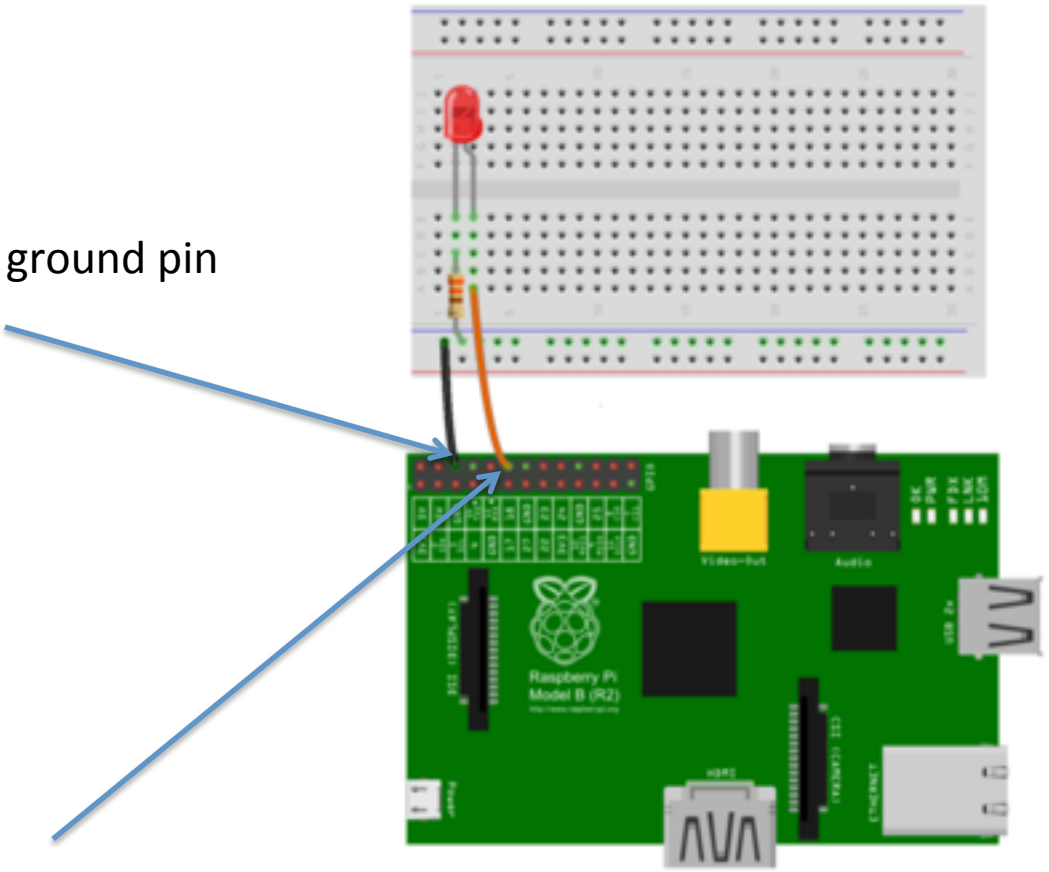


We “write” bits to the LED via this pin

See https://www.youtube.com/watch?v=Rd9kvVs1lSQ for my tutorial on wiring this circuit

First we wire it up

This is the ground pin



We “write” bits to the LED via this pin

See https://www.youtube.com/watch?v=Rd9kvVs1lSQ for my tutorial on wiring this circuit

- Lets look at the code in FASMARM first, then unpack how it works

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000

mov r0,BASE
orr r0,GPIO_OFFSET        ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24                ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]            ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18                 ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]           ;write it into first block of pull-up register

loop$:
b loop$                   ;loop forever
```

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000
```

```
mov r0,BASE
orr r0,GPIO_OFFSET        ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24                ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]            ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18                ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]           ;write it into first block of pull-up register

loop$:
b loop$                   ;loop forever
```

We can set constants.

The name on the left must be a unique name. Make it sensible so you know what it represents.

The value on the right can be expressed as a decimal value or HEX.

Decimal starts with a #

Hex values start with a $

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000

mov r0,BASE
orr r0,GPIO_OFFSET      ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24              ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]          ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18              ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]         ;write it into first block of pull-up register

loop$:
b loop$                 ;loop forever
```

Instructions to execute.

Here you can see the following instructions being used:
Mov, orr, lsl, str, b

We'll look at them in more detail shortly

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000

mov r0,BASE
orr r0,GPIO_OFFSET      ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24              ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]          ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18              ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]         ;write it into first block of pull-up register

loop$:
b loop$                 ;loop forever
```

In-line comments (non executable) start with a semi colon

Use these to make your code Easily readable and interpretable

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000

mov r0,BASE
orr r0,GPIO_OFFSET        ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24                ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]            ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18                ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]           ;write it into first block of pull-up register

loop$:
b loop$                   ;loop forever
```

Labels mark a location (an actual address!) in your asm code.

They are uniquely named and end with a colon.

These allow you to jump from one location to another in your code
(eg. for looping, branching function calls etc)

In this case its being used as part of an infinite loop ..but why ?

ASSEMBLY

mov ;move value into register (does not require RAM)*

lsl ;logical left shift (double a number n times)

str ;store (write) value from register to a *memory* location

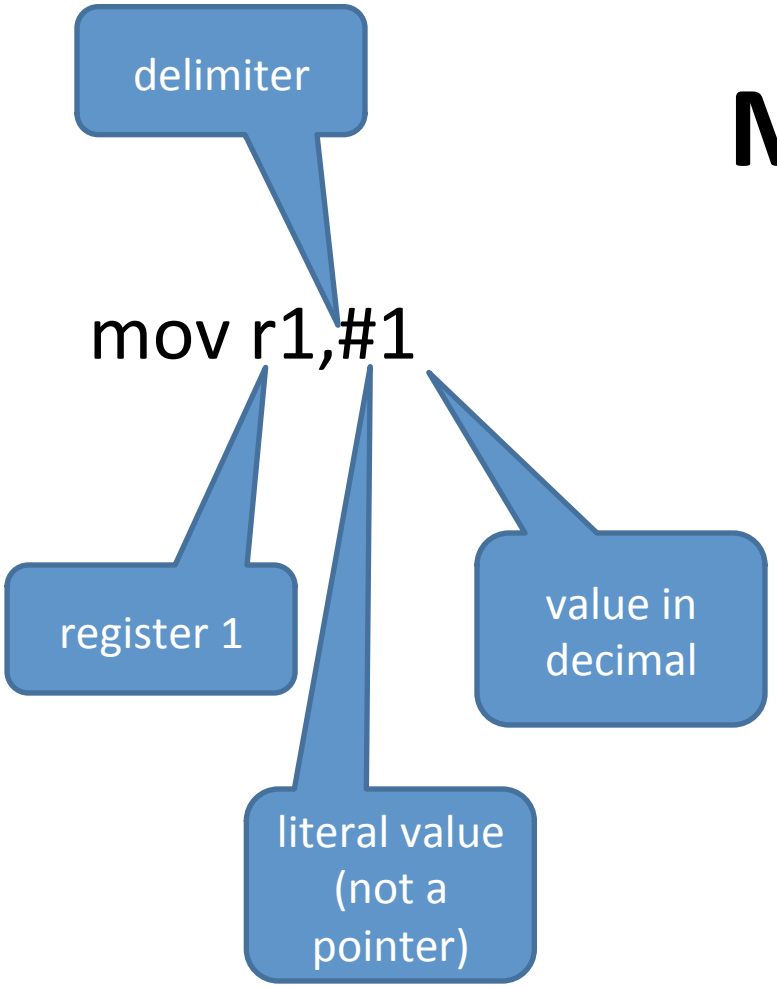
Loop: ;a label**

b ;branch (goto)

***mov** cannot be used to represent some 32-bit numbers. The alternative is **ldr**, which we will cover later. For more explanations look here: http://stackoverflow.com/questions/14046686/why-use-ldr-over-mov-or-vice-versa-in-arm-assembly

**In some ASM the word *loop* is reserved, so people use loop$, l00p, loop2 etc.

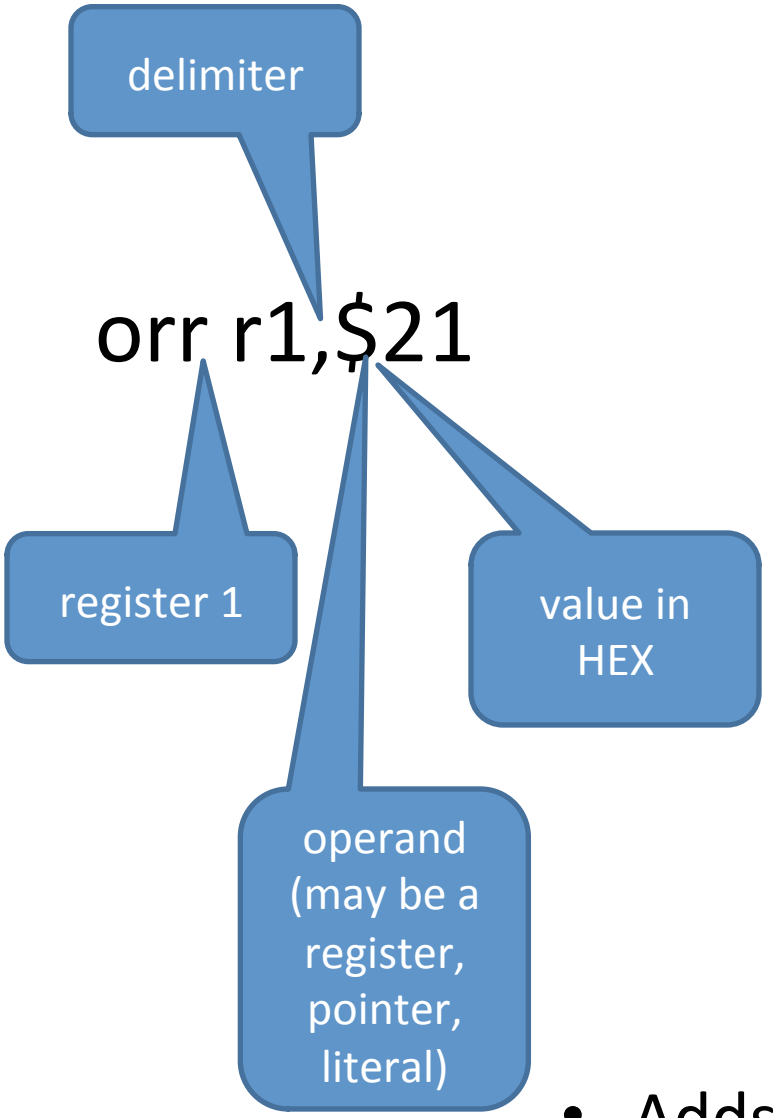
MOVE



- load register 1 with the value 1*

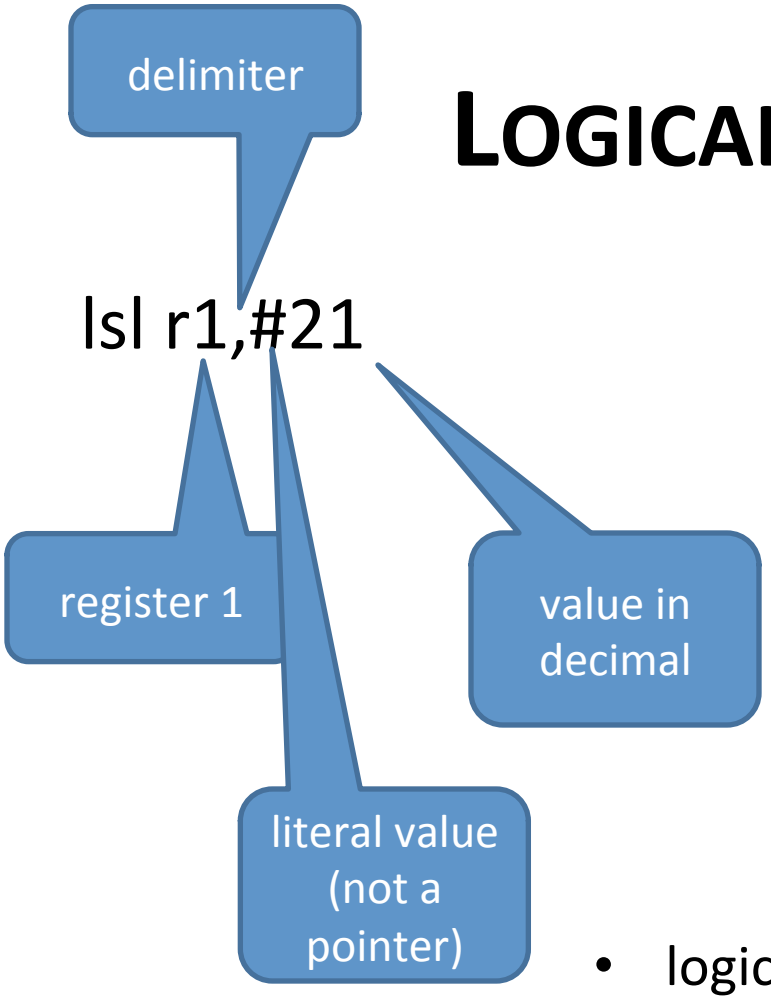
*not all numbers can be used

OR



- r1 = r1 OR operand
- Adds two numbers together without possibility of overflow or carry operations

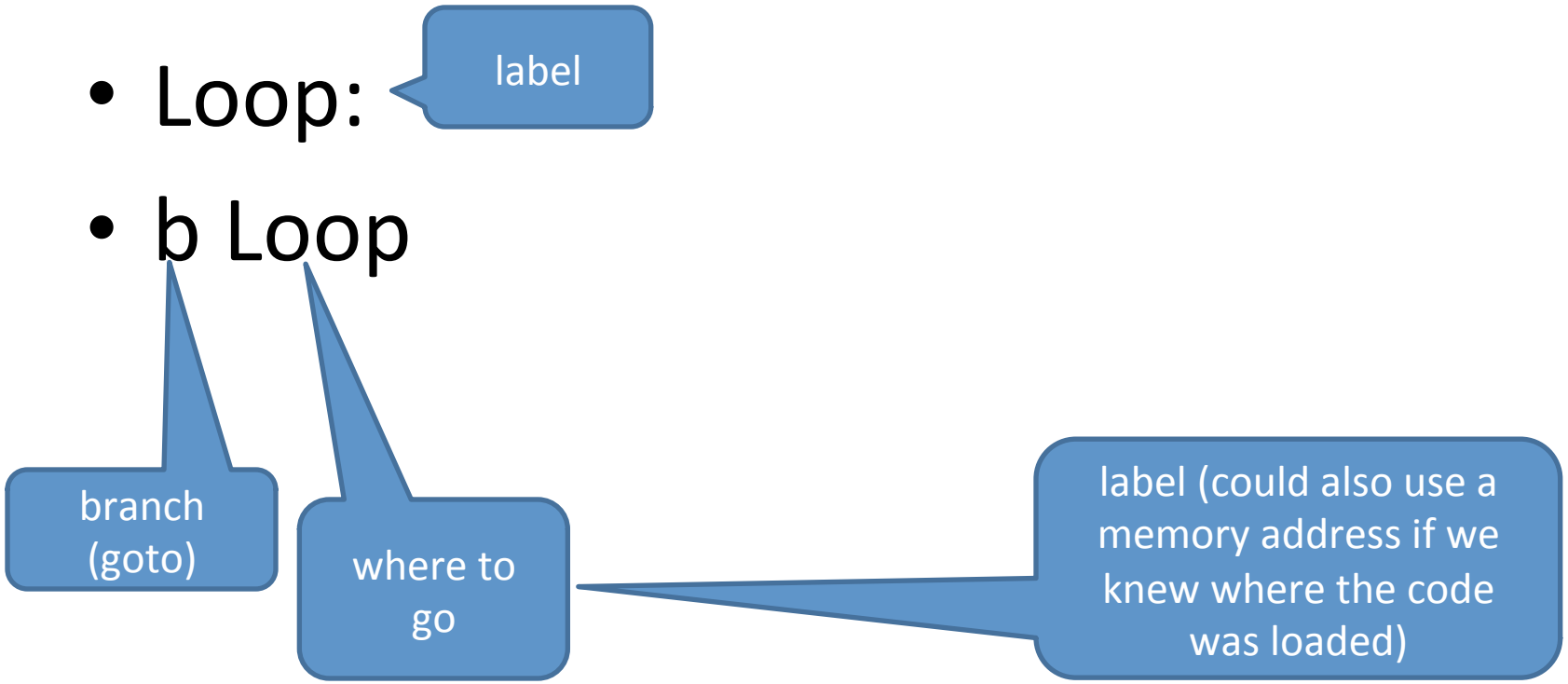
Logical Shift Left



- logical shift left (double) register 1 21 times
  - (multiply by 2^{21})
  - (r1 = 2097152 = 0x200000)
    - (bit 21 is set)
- (00000000 00100000 00000000 00000000)

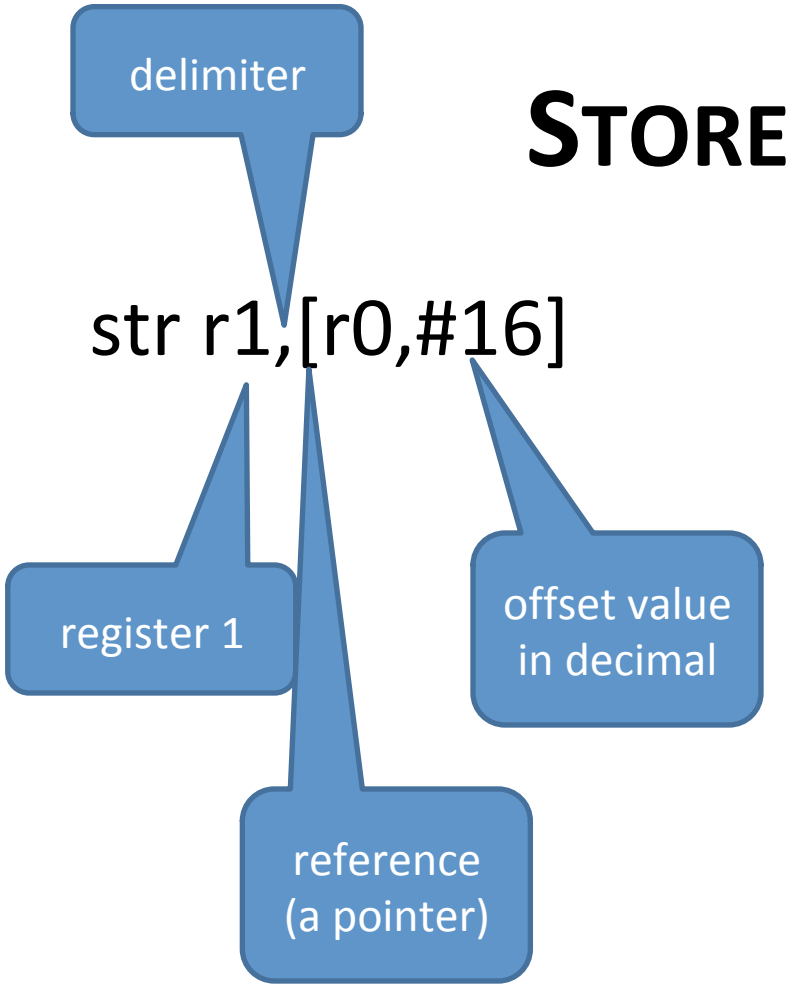
BRANCH

- Loop:
- b Loop



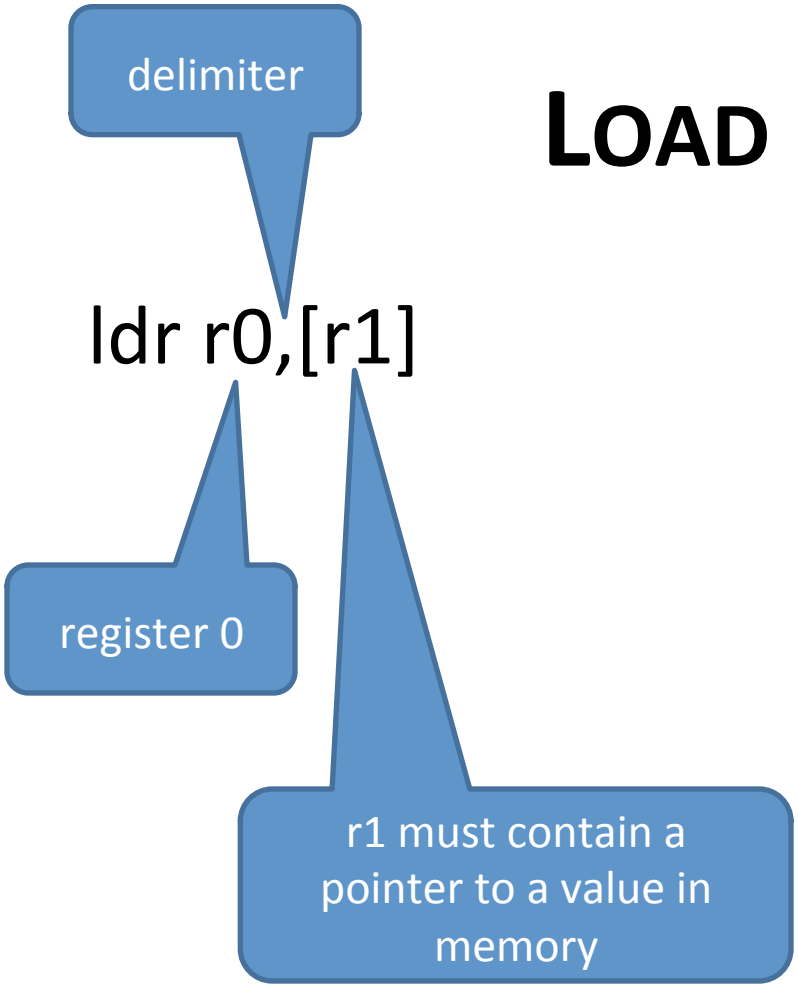
- loop forever
- stops it from crashing

STORE REGISTER



- write value r1 into the memory location: 16 bytes after the value in r0
  - [0x20200010] = 0x200000

LOAD REGISTER



- load register 0 with the value pointed to by r1

General Purpose Registers

- The Armv7 CPU has 13 General Purpose 32-bit registers to work with.
  - Armv8 (e.g.,RPi4) actually has 31 64-bit registers.
- r0, r1, ... r12.
- We use these registers to load in values, perform operations and write back out to memory
- We also use them to pass arguments to functions (we'll get to these later)

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000

mov r0,BASE
orr r0,GPIO_OFFSET        ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24                ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]            ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18                 ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]           ;write it into first block of pull-up register

loop$:
b loop$                   ;loop forever
```

You will notice a lot of numbers being referred to in this code.

Lets look at some of these and why they are there

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000

mov r0,BASE
orr r0,GPIO_OFFSET        ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24                ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]            ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18                 ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]           ;write it into first block of pull-up register

loop$:
b loop$                   ;loop forever
```

This is the “peripheral” base address. This specific value represents the base address of all registers on the RPi 4 that interface with peripheral components (eg the GPIO pins)

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET= $200000

mov r0,BASE
orr r0,GPIO_OFFSET        ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24                ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]            ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18                 ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]           ;write it into first block of pull-up register

loop$:
b loop$                   ;loop forever
```

This is the offset from the base address that marks the start of the GPIO registers (which we need to read and write to/from the GPIO pins).

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET= $200000

mov r0,BASE
orr r0,GPIO_OFFSET        ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24                ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]            ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18                 ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]           ;write it into first block of pull-up register

loop$:
b loop$                   ;loop forever
```

These two operations add the GPIO OFFSET to the BASE address.

Mov moves the value referred to by BASE into the register r0

orr has the effect of adding GPIO_OFFSET to the current contents of r0 (BASE).

The result is stored in r0, which now refers to the base memory address of the GPIO registers

WHICH BASE ADDRESS FOR WHICH RPI

RPi Model	2B	2B v1.2	3B	3B+	4B
SoC	BCM2836	BCM2837	BCM2837	BCM2837B0	BCM2711
Memory	512MB	1GB	1GB	1GB	1/2/4GB
Peripheral Base address	0x20000000	0x3F000000	0x3F000000	0x3F000000	0xFE000000
GPIO Offset	0x200000	0x200000	0x200000	0x200000	0x200000

The peripheral base address depends on which model Pi you are using.

Notice however that the GPIO offset value is the same for all

See https://github.com/FelipMarti/COS10004-RPi for more information on Pi model specifics

TURNING ON AN LED

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000

mov r0,BASE
orr r0,GPIO_OFFSET        ;r0 now equals 0xFE200000

mov r1,#1
lsl r1,#24                ;write 1 into r1, lsl 24 times to move the 1 to bit 24
str r1,[r0,#4]            ;write it into 5th (16/4+1)block of function register

mov r1,#1
lsl r1,#18                 ;write 1 into r1, lsl 18 times to move the 1 to bit 18
str r1,[r0,#28]           ;write it into first block of pull-up register

loop$:
b loop$                   ;loop forever
```

What about these numbers ?

Where did they come from
And what do they mean ?

These numbers all refer to settings and programming of the GPIO registers.

To understand this part of the Code we need to understand what the GPIO chip is, and how we interface with it to read to and write from the GPIO header pins.

SUMMARY

- We have seen the structure of a simple Arm asm program
- We have defined some operations:
  - Mov, orr, lsl, str, ldr, b
- ARMv7 gives us 13 general purpose registers (r0-r12) to store values with
- The peripheral base address refers to the start of the peripheral registers:
  - Pi model specific so you need to know which one is for you!
- Next – the GPIO chip and how to program it!