SWINBURNE UNIVERSITY OF TECHNOLOGY

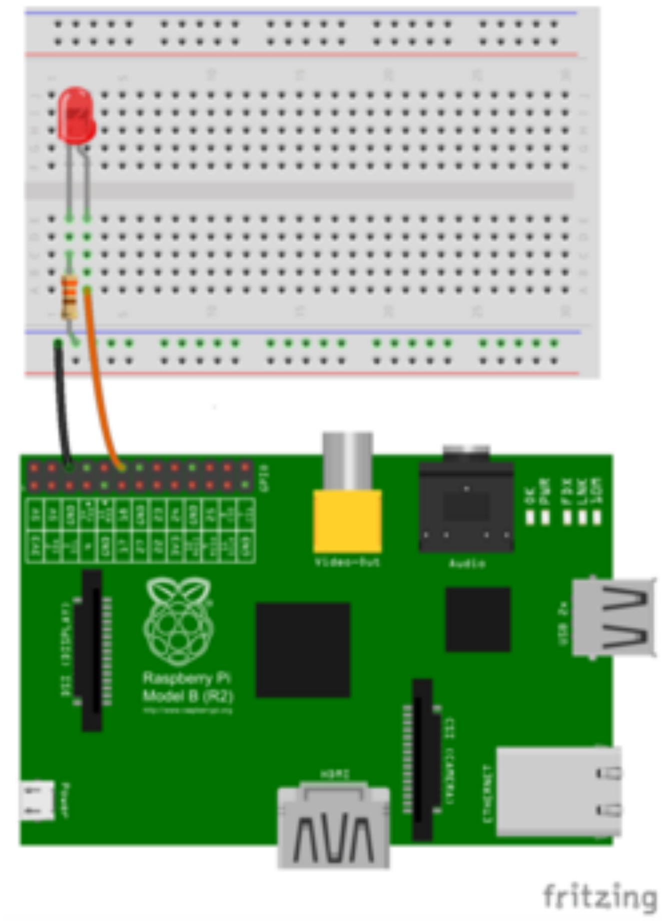
COS10004 Computer Systems

Lecture 7.5 ASM Programming: Turning on an LED (Part 2) – the GPIO chip

CRICOS provider 00111D

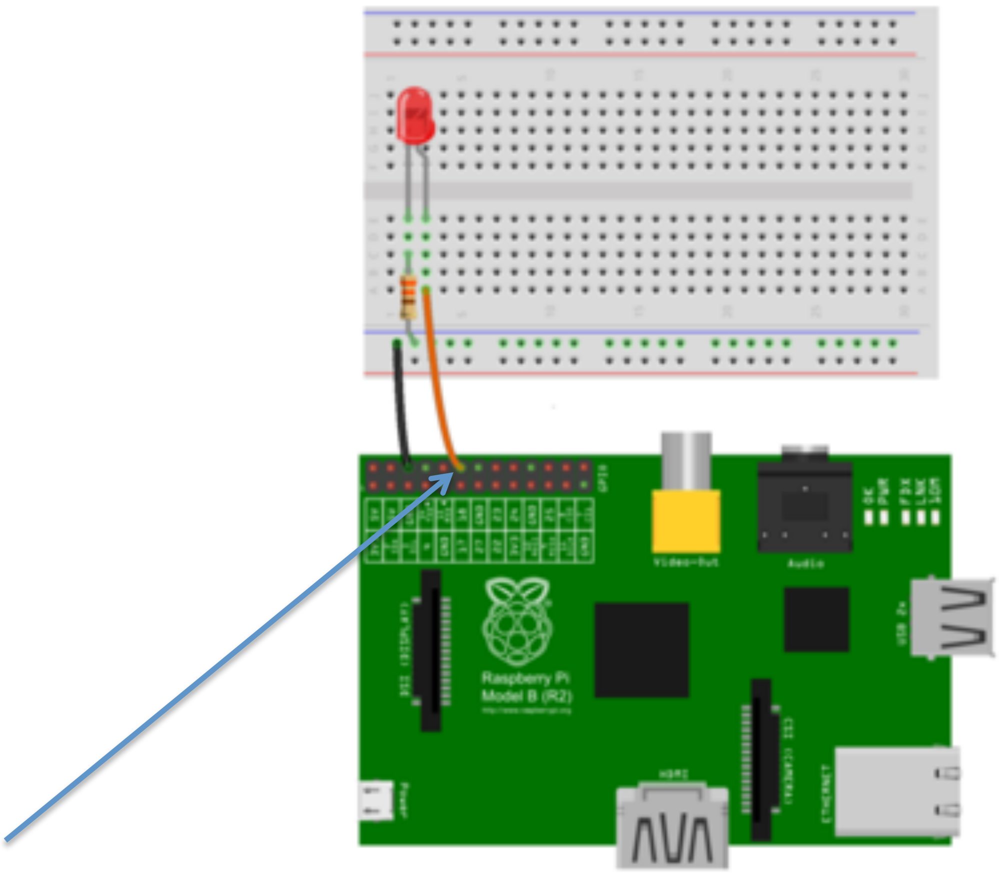
Dr Chris McCarthy

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

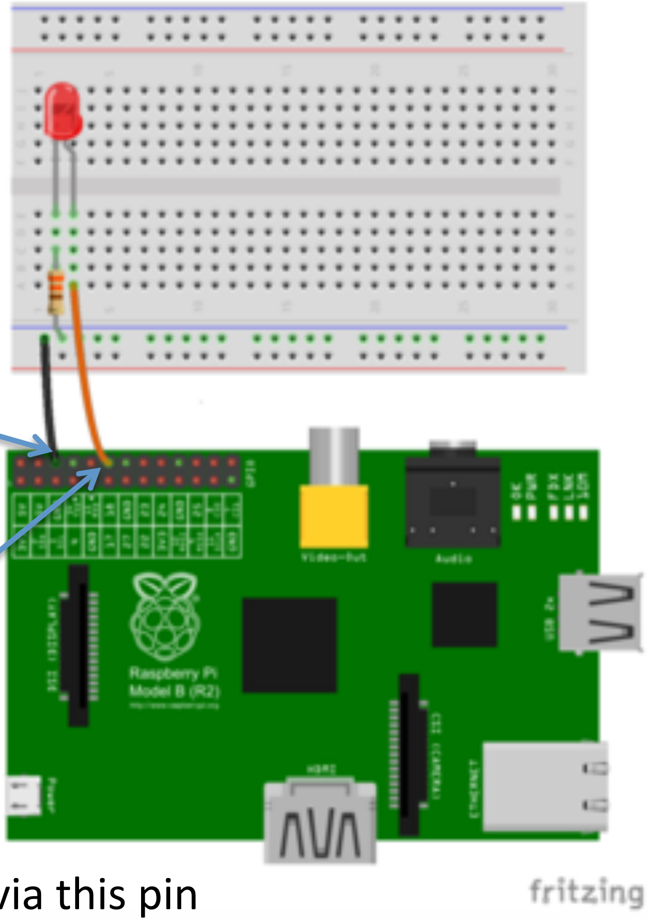


We "write" bits to the LED via this pin

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

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
```

What about these numbers ?

Where did they come from
And what do they mean ?

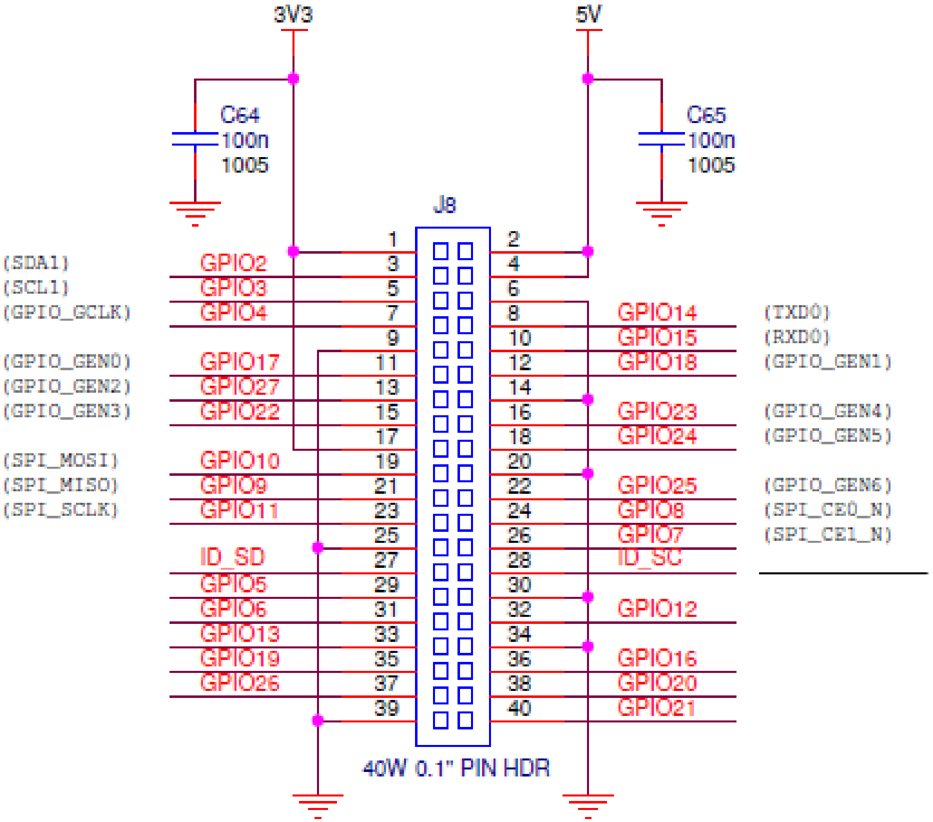
These numbers all refer to settings and programming of the GPIO registers.

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.

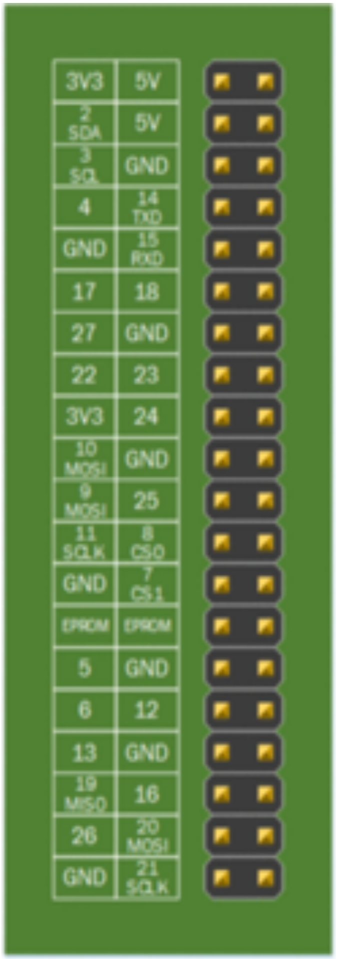
GPIO

- The General Purpose Input/Output chip
- The GPIO chip has 54 registers which can be read, set high or set low.
- They are referred to as GPIO0, GPIO1 …
- Some are connected to physical pins on the R Pi board
- Others are connected to hardware on the board.

GPIO LAYOUT FOR RPI 2B, 3B AND 4



GPIO EXPANSION



3V3	5V
2 SDA	5V
3 SCL	GND
4	14 TXD
GND	15 RXD
17	18
27	GND
22	23
3V3	24
10 MOSI	GND
9 MOSI	25
11 SCLK	8 CS0
GND	7 CS1
EPROM	EPROM
5	GND
6	12
13	GND
19 MISO	16
26	20 MOSI
GND	21 SCLK

SETTING A HARDWARE PIN ON

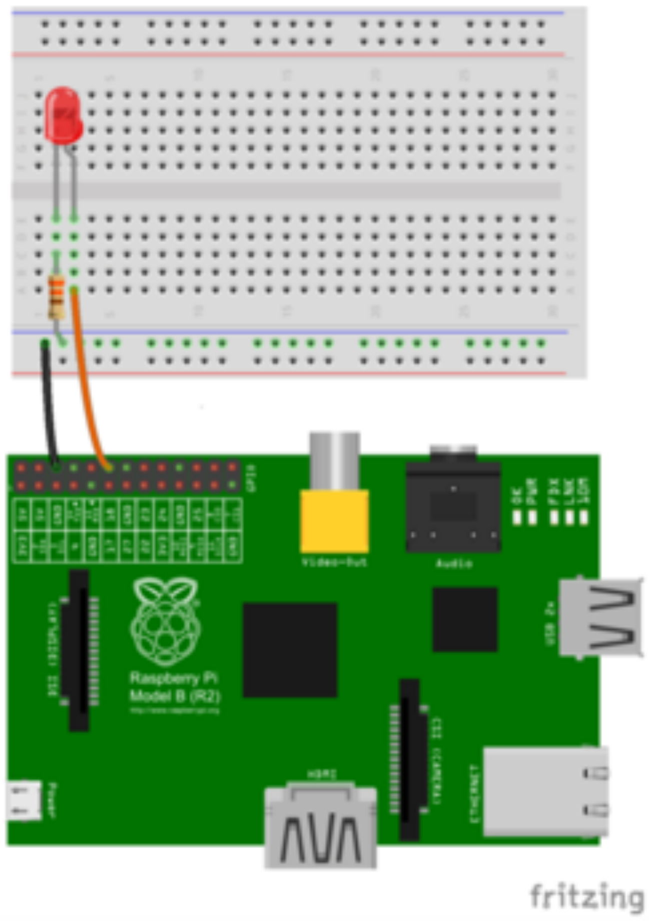
- Establish which GPIO is associated with the Pin.
- Find the associated “function” register and set appropriate bits to indicate your intention to write to that GPIO
- Find the appropriate “write” register and set the output bit to 1.

Setting a hardware PIN ON

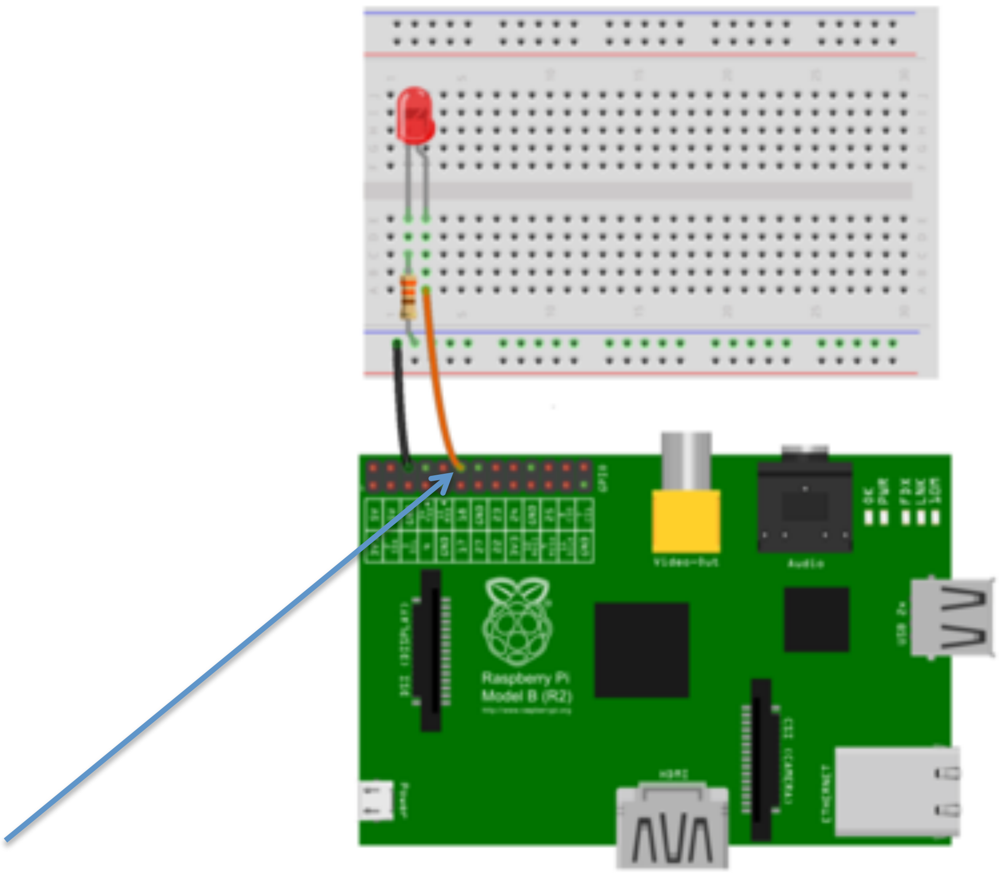
- Establish which GPIO is associated with the Pin.
- Find the associated "function" register and set appropriate bits to indicate your intention to write to that GPIO
- Find the appropriate "write" register and set the output bit to 1.

What could possibly go wrong !

OK – BACK TO THIS FLASHY LED THING

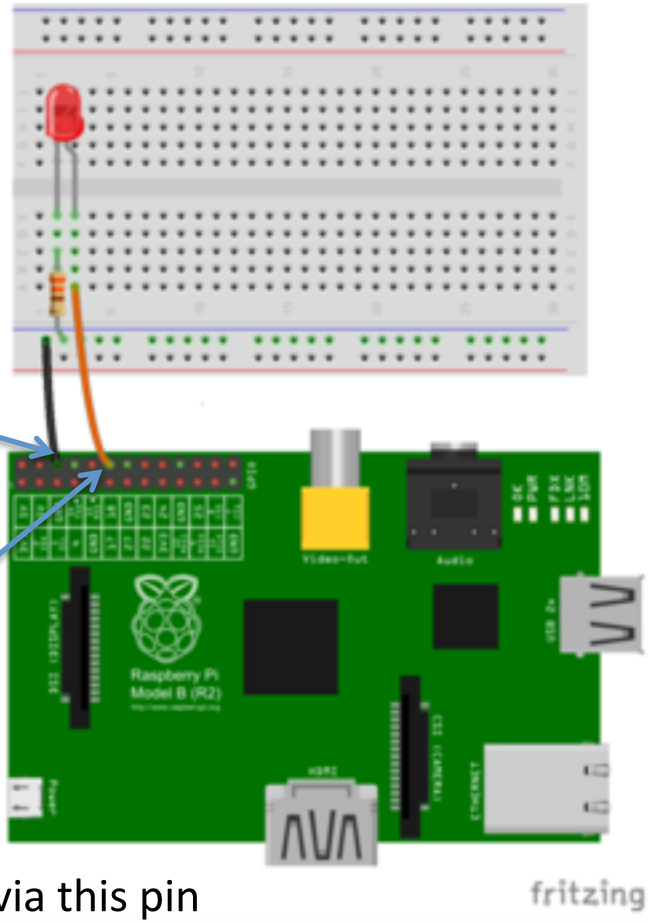


OK – BACK TO THIS FLASHY LED THING



We “write” bits to the LED via this pin

OK – BACK TO THIS FLASHY LED THING

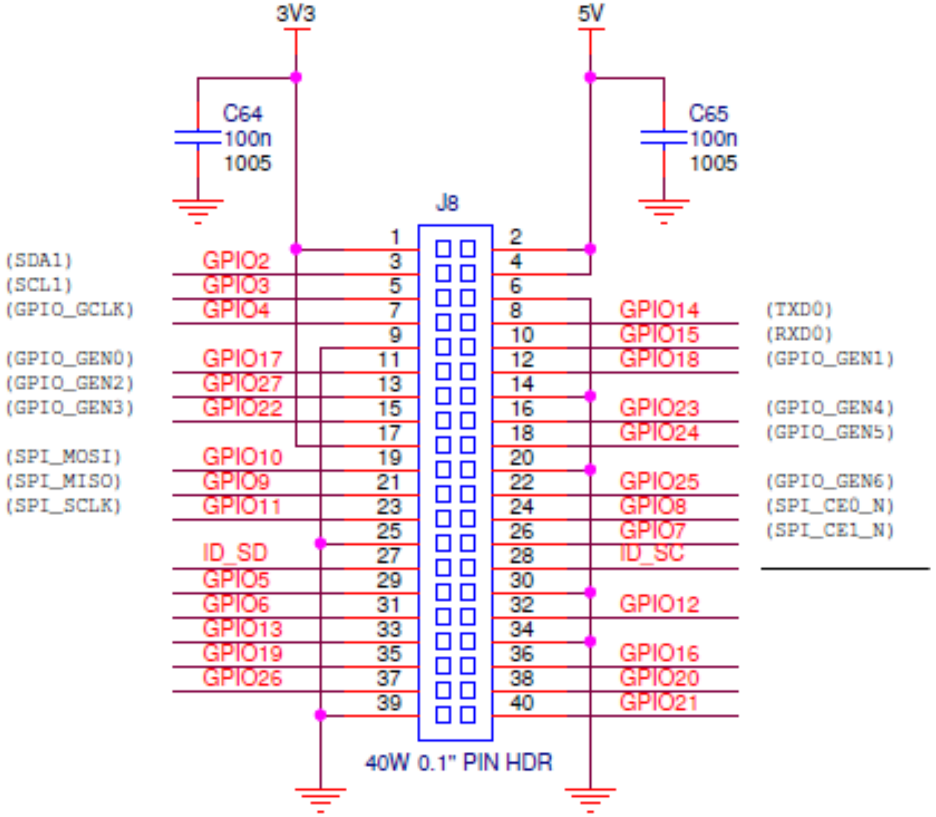


This is the ground pin

We “write” bits to the LED via this pin

So which GPIO register ?

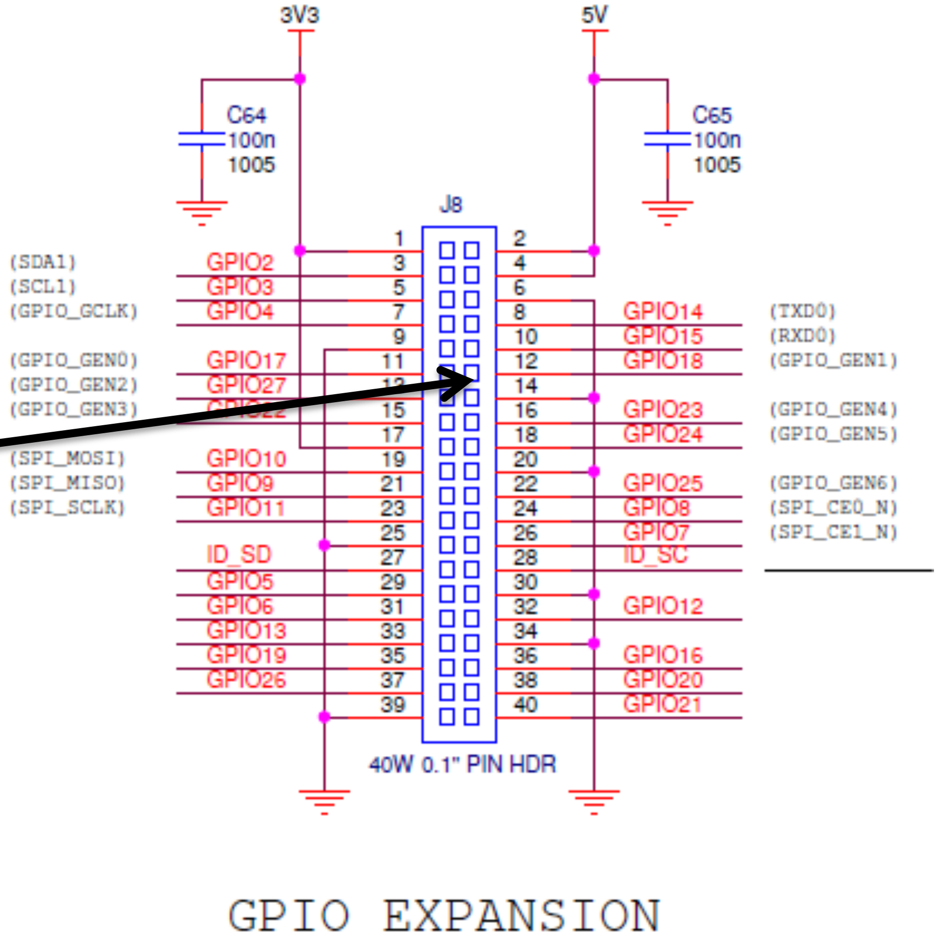
- Now locate the "writing" pin in the diagram GPIO register



GPIO EXPANSION

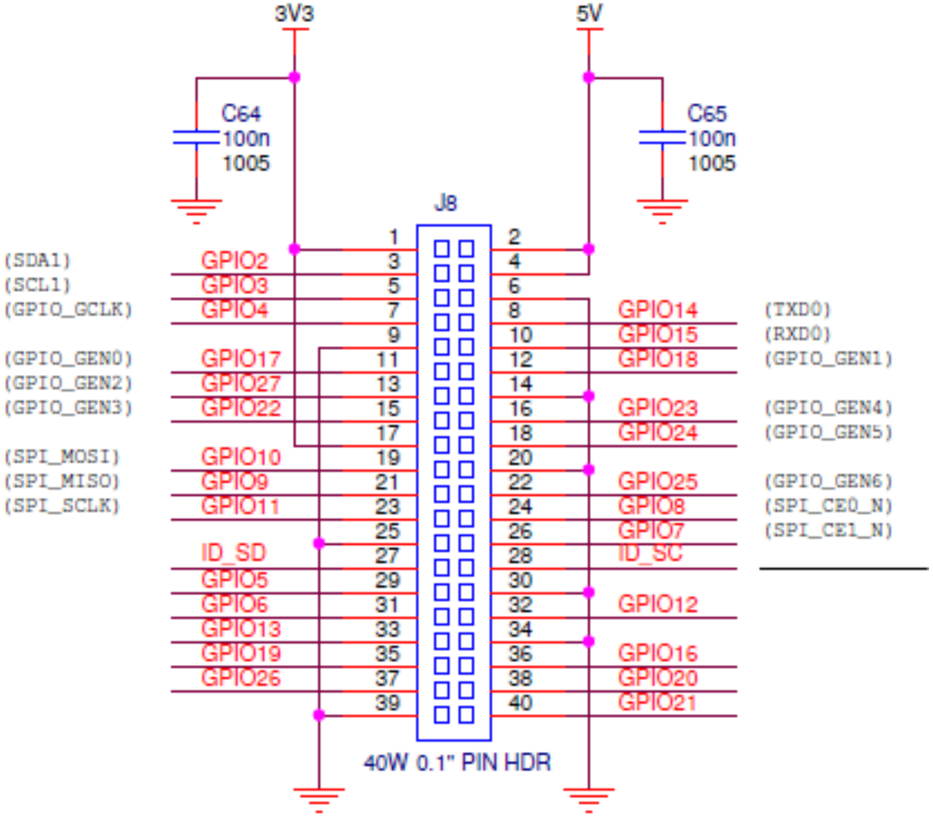
So which GPIO register ?

- Now locate the "writing" pin in the diagram GPIO register
- It is GPIO18



So Which GPIO register ?

- So the LED is controlled via GPIO18.
- The pin has to be pulled up to turn on LED.
- But some other things need to happen first



GPIO EXPANSION

BASE ADDRESS AND GPIO OFFSET

- In our ASM program we will need to access the correct GPIO registers and set specific bits.
- We first need to know where in memory the location of these registers are.
- Specifically we need to know:
  - The BASE address: for all memory we need to access
  - The GPIO offset: the number of bytes from the base address from which GPIO registers start.

Base address and GPIO offset

- For RPi 2B/3B/3B+:

BASE = $3F000000 ; $ means HEX

GPIO_OFFSET=$200000

- For RPi 4:

BASE = $FE000000

GPIO_OFFSET=$200000

PSEUDOCODE

- store location of GPIO (BASE + GPIOADDR) in r0
- Enable “writing” for GPIO18 (our LED)
  - we need to set certain bits in the “function” register to program GPIO18 for writing
- Set output of GPIO18
  - Light on: set bit in the appropriate “write 1” register
  - Light off: set bit in the appropriate “write 0” register
- loop forever

FYI: Turn PWR LED OFF (B+, 2 only)

- store location of GPIO (BASE + GPIOADDR) in r0
- enable output function on GPIO35 (red LED)
  - r1 set to 2^15
  - store r1 into [r0 + 12] #dereferences value in r0+12
- set output of GPIO35 (32 + 3) (turn PWR light off!)
  - r1 set to 2^3 or
  - store r1 into [r0 +44] ;pull low(light off) 40+4
  - store r1 into [r0 +32] ;pull high(light on) 32 = 28+4
- loop forever

But Chris !

How do you know what bits to set for a particular GPIO ?

1. SELECT FUNCTION

start of the BASE+ GPIO (RPi 2/3). Add this address to everything in the GPIO

Each pin programmed by a 3-bit number. Those numbers are packed into 30 bits of each word

Hex	Offset (dec)	32-bit regsiters	**Function Select Register**			
0x3F200000	0	GPIO 0 - 9	**store GPIO start address:**	bits 0-2 = GPIO 0	bits 3-5 = GPIO 1	bits 6-8 = GPIO 2
	1		ldr r0,=0x3F200000	bits 9-11 = GPIO 3	bits 12-14 = GPIO 4	
	2			bits 15-17 = GPIO 5	bits 18-20 = GPIO 6	bits 21-23 = GPIO 7
	3			bits 24-26 = GPIO 8	bits 27-29 = GPIO 9	
0x3F200004	**4**	GPIO 10 - 19	**Enable Write: to GPIO18 (Lab 7)**	bits 0-2 = GPIO 10	bits 3-5 = GPIO 11	bits 6-8 = GPIO 12
	5		mov r1,#1	bits 9-11 = GPIO 13	bits 12-14 = GPIO 14	
	6		lsl r1,#24	bits 15-17 = GPIO 15	bits 18-20 = GPIO 16	bits 21-23 = GPIO 17
	7		str r1,[r0,#4]	bits 24-26 = GPIO **18**	bits 27-29 = GPIO 19	
0x3F200008	**8**	GPIO 20 - 29	**Enable Read: from GPIO24(pin)**	bits 0-2 = GPIO 20	bits 3-5 = GPIO 21	bits 6-8 = GPIO 22
	9		mov r1,#0	bits 9-11 = GPIO 23	bits 12-14 = GPIO 24	
	10		lsl r1,#12	bits 15-17 = GPIO 25	bits 18-20 = GPIO 26	bits 21-23 = GPIO 27
	11		str r1,[r0,#8]	bits 24-26 = GPIO 28	bits 27-29 = GPIO 29	
0x3F20000C	**12**	GPIO 30 - 39		bits 0-2 = GPIO 30	bits 3-5 = GPIO 31	bits 6-8 = GPIO 32
	13			bits 9-11 = GPIO 33	bits 12-14 = GPIO 34	
	14			bits 15-17 = GPIO 35	bits 18-20 = GPIO 36	bits 21-23 = GPIO 37
	15			bits 24-26 = GPIO 38	bits 27-29 = GPIO 39	
0x3F200010	**16**	GPIO 40 - 49		bits 0-2 = GPIO 40	bits 3-5 = GPIO 41	bits 6-8 = GPIO 42
	17			bits 9-11 = GPIO 43	bits 12-14 = GPIO 44	
	18			bits 15-17 = GPIO 45	bits 18-20 = GPIO 46	bits 21-23 = GPIO 47
	19			bits 24-26 = GPIO 8	bits 27-29 = GPIO 49	
0x3F200014	**20**	GPIO 50 - 54	0-7 bits	bits 0-2 = GPIO 50	bits 3-5 = GPIO 51	bits 6-8 = GPIO 52
	21		8-15 bits	bits 9-11 = GPIO 53	bits 12-14 = GPIO 54	
	22		16-22 bits			
	23		23-29 bits			
0x3F200018	24					

Add 4 bytes (1 word) each time we go above 30 bits (10 GPIO pins)

3 registers control writing 0, writing 1 or reading each pin.

2. SET VALUE (R/W)

Each pin programmed by a 1-bit number. 32 numbers are packed into 32 bits of each word.

This register writes 1 to the GPIO pin

0x3F20001C	28		Write 1 to GPIO18 (Lab 7)	bits 0-7 = GPIO 0-7
	29	set bit n to turn ON GPIO n	mov r1,#1	bits 8-15 = GPIO 8-15
	30		lsl r1,#18	bits 16-23 = GPIO 16-23
	31		str r1,[r0,#28]	bits 24-31 = GPIO 24-31
0x3F200020	32			bits 0-7 = GPIO 32-39
	33	set bit n to turn ON GPIO 32+n		bits 8-15 = GPIO 40-47
	34			bits 16-22 = GPIO 48-54
	35			
0x3F200024	36			

This register writes 0 to the GPIO pin

0x3F200028	40		Write 1 GPIO18 (Lab 7)	bits 0-7 = GPIO 0-7
	41	set bit n to turn OFF GPIO n	mov r1,#1	bits 8-15 = GPIO 8-15
	42		lsl r1,#18	bits 16-23 = GPIO 16-23
	43		str r1,[r0,#40]	bits 24-31 = GPIO 24-31
0x3F20002C	44			bits 0-7 = GPIO 32-39
	45	set bit n to turn OFF GPIO 32+n		bits 8-15 = GPIO 40-47
	46			bits 16-22 = GPIO 48-54
	47			
0x3F200030	48			

Some GPIO pins need to be sent a 0 to turn the pin on, others need a 1 to turn on.

This register contains state of the GPIO pin (if programmed to read)

0x3F200034	52		bits 0-7 = GPIO 0-7
	53	read bit n to detect GPIO n	bits 8-15 = GPIO 8-15
	54		bits 16-23 = GPIO 16-23
	55		bits 24-31 = GPIO 24-31
0x3F200038	56		bits 0-7 = GPIO 32-39
	57	read bit n to detect GPIO 32+n	bits 8-15 = GPIO 40-47
	58		bits 16-22 = GPIO 48-54
	59		
0x3F20004C	60		

PSEUDOCODE

- store location of GPIO (BASE + GPIO_OFFSET) in r0
- Enable “writing” for GPIO18 (our LED)
  - – Set the 24th bit of r1 (ie r1 = 2^24 or 0x800000)
  - – store r1 into [r0 + 4] ;dereferences value in r0+4
- set output of GPIO18
  - – r1 set to 2^18 or
  - – store r1 into [r0 + 28] (light on)
- loop forever

So now back to the code to see this!

```
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
```

So now back to the code to see this!

```
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
```

Enable “writing” for GPIO18 (our LED)

- Set the 24th bit of r1
- store r1 into [r0 + 4]

So now back to the code to see this!

```
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
```

Write “1” to GPIO18 by:

- Setting the 18th bit to 1 in r1
- store r1 into memoery location [r0 + 28]
- This turns the LED on

So now back to the code to see this!

```
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
```

Loop forever so we do not "fall off the cliff"!

WHY THE LOOP AT THE END ?

- Computer endlessly follows a continuous *instruction cycle* until switched off
- A special register known as the Program Counter (PC) keeps track of *the next instruction* to execute
- PC is a digital counter:
  - It blindly increments automatically after each instruction (but has no idea what its pointing to!)
  - Also set by branch instructions like “b” which jump control to a new instruction address
- The infinite loop stops the PC addressing locations that are not part of the program
  - Keeps it in an infinite holding pattern

DESIGN PATTERN FOR OUR ASM

0. Set BASE address, GPIO_OFFSET address, put in r0

```
BASE = $FE000000   ;  $ means HEX
GPIO_OFFSET=$200000
mov r0,BASE
orr r0,GPIO_OFFSET
```

1. Select GPIO by shifting *1 (using lsl)* to required position (within 32-bit number)

   – lsl r1,#21

     - if more than 32, subtract 32 and add 4 (4 bytes) to offset

2. Select action by storing selection in appropriate register by adding register offset to GPIO base address

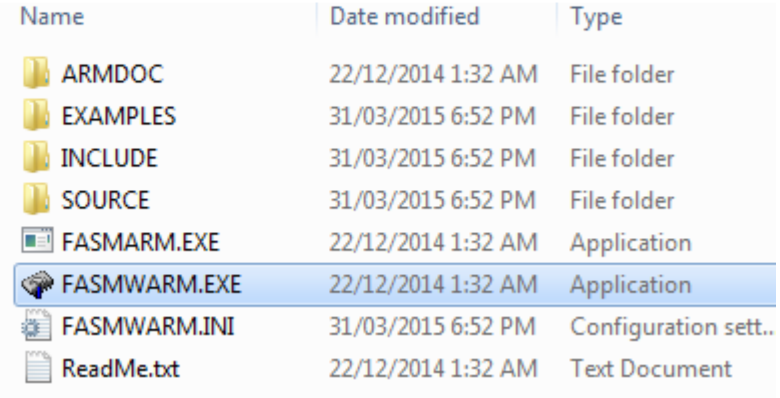
   – str r1,[r0,#32] ;or 32+offset

DESIGN PATTERNS…

- Step 0 would only be done once – setting up the hardware, constants.
- Repeat steps 1 and 2 for each new action:
  - program a GPIO (for input, output, other functions) (registers 0-27)
  - Pull a register high (registers 28-31)
  - Pull a register low (registers 32-

How to run it yourself

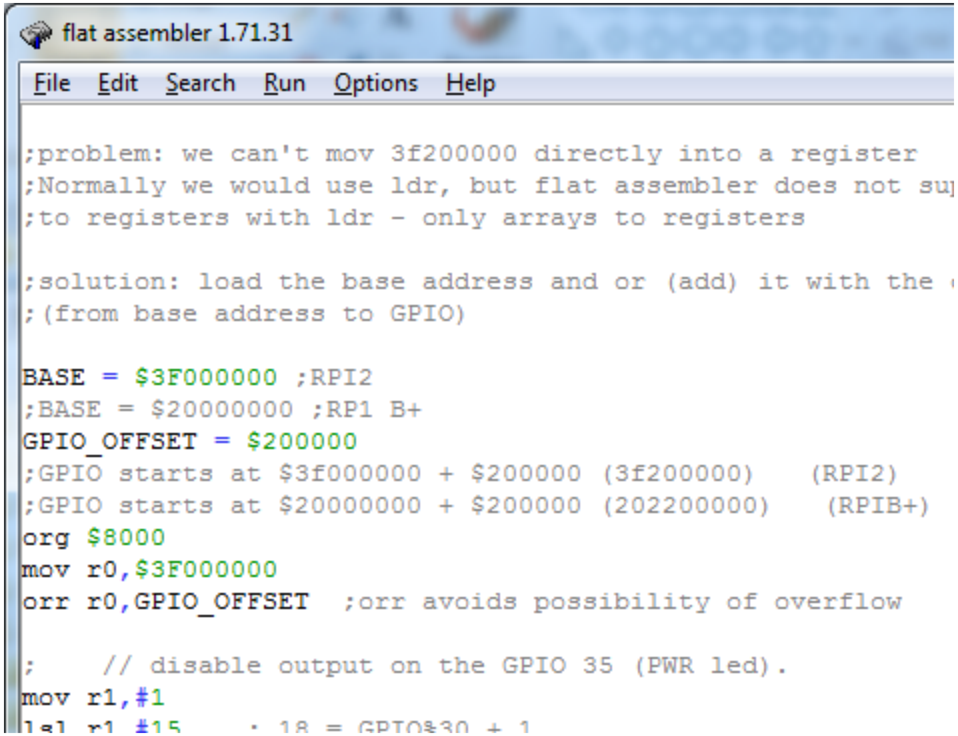
- Launch FASMARM from where ever you installed it (just an exe file)



Name	Date modified	Type
ARMDOC	22/12/2014 1:32 AM	File folder
EXAMPLES	31/03/2015 6:52 PM	File folder
INCLUDE	31/03/2015 6:52 PM	File folder
SOURCE	31/03/2015 6:52 PM	File folder
FASMARM.EXE	22/12/2014 1:32 AM	Application
FASMWARM.EXE	22/12/2014 1:32 AM	Application
FASMWARM.INI	31/03/2015 6:52 PM	Configuration sett..
ReadMe.txt	22/12/2014 1:32 AM	Text Document

WRITE THE CODE INTO THE EDITOR

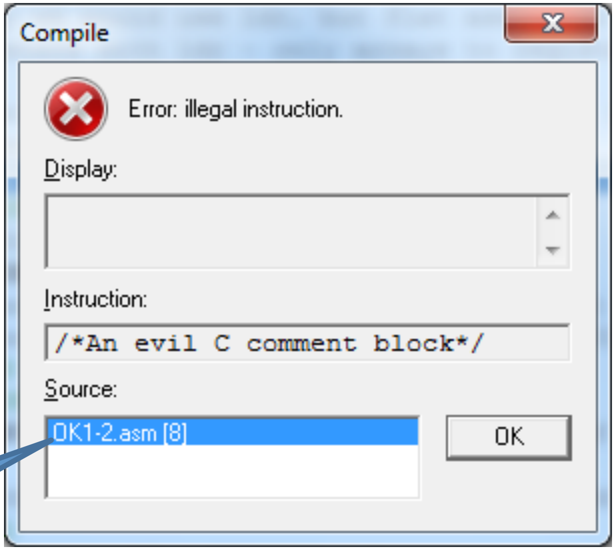
; means comment



```
;problem: we can't mov 3f200000 directly into a register
;Normally we would use ldr, but flat assembler does not su
;to registers with ldr - only arrays to registers

;solution: load the base address and or (add) it with the
;(from base address to GPIO)

BASE = $3F000000 ;RPI2
;BASE = $20000000 ;RP1 B+
GPIO_OFFSET = $200000
;GPIO starts at $3f000000 + $200000 (3f200000)    (RPI2)
;GPIO starts at $20000000 + $200000 (202200000)    (RPIB+)
org $8000
mov r0,$3F000000
orr r0,GPIO_OFFSET  ;orr avoids possibility of overflow

;    // disable output on the GPIO 35 (PWR led).
mov r1,#1
lsl r1,#15     ; 18 = GPIO%30 + 1
```

COMPILE

- Save
- Run / Compile (Ctrl + F9)
- Read any errors and fix


Compile

Error: illegal instruction.

Display:

Instruction:

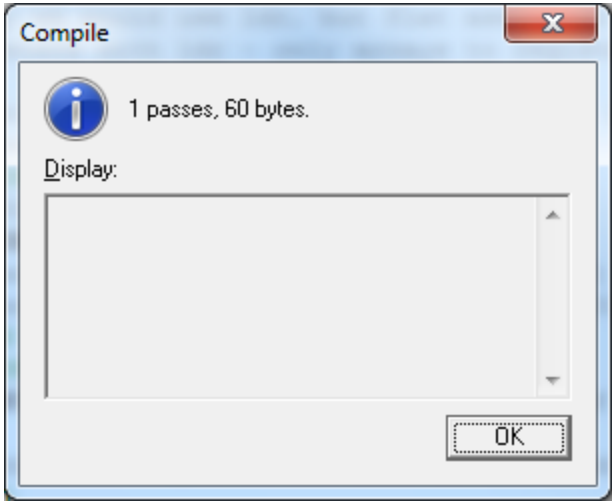
```
/*An evil C comment block*/
```

Source:

OK1-2.asm [8]

line number 8

- Successful compilation


Compile

1 passes, 60 bytes.

Display:

Copy to SD

- Copy <filename>.bin to your correctly formatted micro SD card
- Rename <filename>.bin to kernel7.img
- Wait or dismount card
- Remove card (and adapter)
- Plug micro SD card into Pi
- Power-on Pi
- Be amazed!

THE LAB

- You’re going to do this!
- You will also have more opportunities to get your head around the GPIO programming:
  - We will be doing this for a few weeks!

SUMMARY

- Assembly language is the lowest level of human readable programming
  - Extremely close to native machine code
- RISC versus CISC instruction sets define major differences between CPU architectures:
  - ARM is RISC, Intel is CISC
- ARM asm basics
- GPIO interface
- Oh yeah .. AND we wired up and turned on an LED !