

# **COS10004 Computer Systems**

## **Lecture 8.2 Arm ASM – LED Flash (part 2) - the Busy Wait Timer**

CRICOS provider 00111D

*Dr Chris McCarthy*

# INSERT DELAY WITH BUSY WAIT TIMER

- Pseudocode:
  - Program GPIO18 LED for output
  - Loop1:
    - Turn LED on (pull GPIO18 high)
    - Busy wait
    - Turn LED off (pull GPIO18 low)
    - Busy wait
  - branch to loop1

# A DUMB TIMER

- Busy wait:
  - a loop that executes until it reaches a certain value
  - keep CPU occupied

set limit value ;some large number

initialise counter to 0

timer\$:

- Add 1 to counter
- Compare counter to limit value
- branch to timer\$ if counter < limit

# A DUMB TIMER

- Busy wait:
  - a loop that executes until it reaches a certain value
  - keep CPU occupied

set limit value ;some large number

initialise counter to 0

timer\$:

- Add 1 to counter
- Compare counter to limit value
- branch to timer\$ if counter < limit

How ?

# IF TESTS : CMP

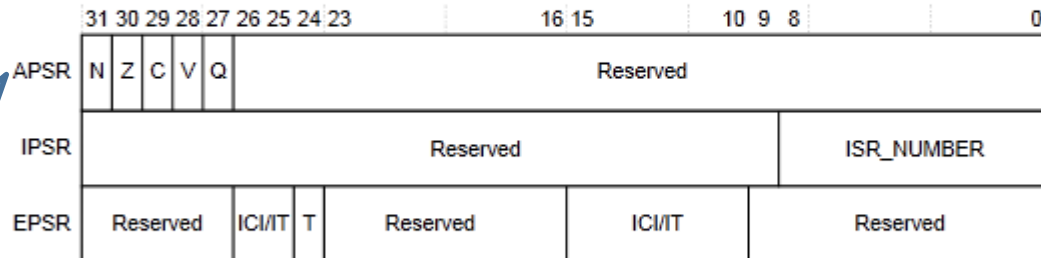
- Called 'Compare' in ARM asm
  - Subtracts 2nd value from first, and sets flags accordingly.
  - Loads the **Application Program Status Register (APSR)** with the results of the comparison (done by the ALU).
  - The APSR flags include:
    - N      ALU result was Negative.
    - Z      ALU result was Zero.
    - C      ALU set the Carry bit.
    - V      ALU result caused overflow.
  - This register can then be inspected by branch commands
    - We'll come to this !

## Program Status Register

The *Program Status Register* (PSR) combines:

- *Application Program Status Register* (APSR)
- *Interrupt Program Status Register* (IPSR)
- *Execution Program Status Register* (EPSR).

These registers are mutually exclusive bitfields in the 32-bit PSR. The bit assignments are:



Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the `MSR` or `MRS` instructions. For example:

- read all of the registers using `PSR` with the `MRS` instruction
- write to the APSR N, Z, C, V, and Q bits using `APSR_nzcvq` with the `MSR` instruction.

**Table 2.4. APSR bit assignments**

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27]	Q	Saturation flag
[26:0]	-	Reserved

<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/CHDBIBGJ.html>

# DETAILS

- `cmp r2,#1234`

compare

register 2

compare r2  
and #0

r2 - #1234;  
set APSR with  
ALU flags

store the ALU  
flags in the APSR

# ACTING ON THE APSR

- Branch (b) reads the APSR and jumps according to the flags and the relevant suffix
  - Really a conditional goto
  - e.g. compares register with a number or 0
- Assume `cmp r1,r2`, then:
  - b - unconditional branch
  - beq branch if Z flag set (i.e.,  $r1 == r2$ )
  - bge - branch if  $r1 \geq r2$  ( $(r1-r2) \geq 0$ )
  - blt – branch if  $r1 < r2$  ( $r1 - r2 < 0$  (N flag set))
  - Many others...



# DETERMINING THE COMPARISON FROM THE FLAGS

## Condition code suffixes

The **condition code suffix** can be added to many operations. e.g. `movne r1,#12`

<u>Suffix</u>	<u>Flags</u>	<u>Meaning</u>
EQ	Z set	Equal
NE	Z clear	Not equal
CS or HS	C set	Higher or same (unsigned >= )
CC or LO	C clear	Lower (unsigned < )
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Higher (unsigned > )
LS	C clear or Z set	Lower or same (unsigned <=)
GE	N and V the same	Signed >=
LT	N and V differ	Signed <
GT	Z clear, N and V the same	Signed >
LE	Z set, N and V differ	Signed <=

# MAKING A LOOP

- We can use the APSR and a branch to make a loop structure.

```
mov r2,#0
loop1:
    add r2,#1                1,2,3,4,5,6,7,8,9,10
    cmp r2,#10
    bne loop1 ;this counts to 10, but it's wasteful.
```

# MAKING A LOOP

- We can use the APSR and a branch to make a loop structure.
- Better in ASM to count down to 0

```
mov r2,#10
```

```
loop1:
```

```
    sub r2,#1           9,8,7,6,5,4,3,2,1,0
```

```
    cmp r2,#0
```

```
bne loop1 ;runs a bit faster
```

# MAKING THE LED FLASH (OK2)

- pseudocode:

Program the GPIO 18 for output

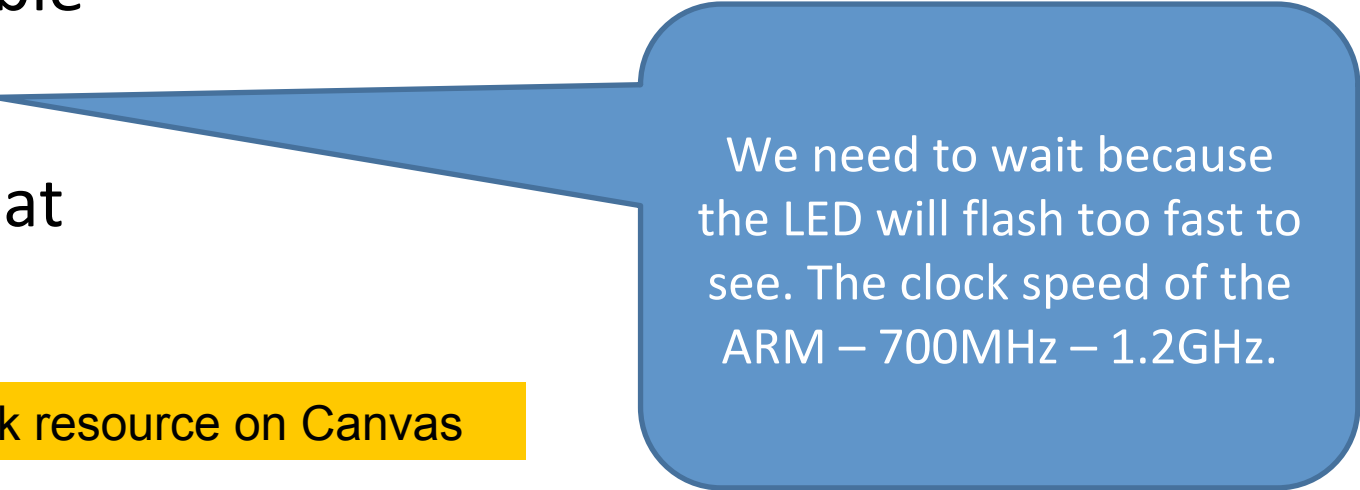
Enable

wait

Disable

wait

repeat

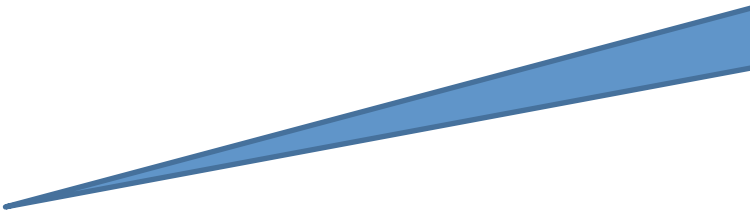


We need to wait because the LED will flash too fast to see. The clock speed of the ARM – 700MHz – 1.2GHz.

Lab 8 Task resource on Canvas

# A DUMB TIMER

- Variables:
  - r0 = GPIO base address
  - r1 = working memory (for setting bits, registers)
  - use r2 for timing
- ```
mov r2,$3F0000
loop1:
  sub r2,#1
  cmp r2,#0
  bne loop1
```



Does a 'busy wait' - uses 100% of CPU

# DETAILS

- `sub r2,#1`

subtract

to register  
2

subtract 1  
from r2

'dec' in  
some asm

$r2 = r2 - 1$

`r2--`

# DETAILS

- `bne loop1`

branch if not  
equal

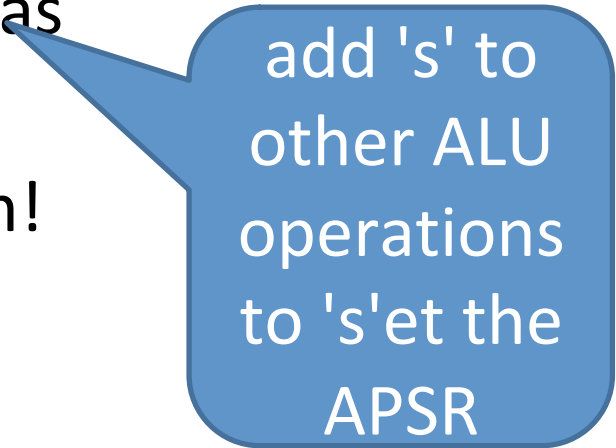
goto this  
label and  
continue  
execution

'jne' in  
some asm

if the z flag is  
clear, goto loop1

# ISSUES

- The APSR is updated after many ALU operations, not just `cmp`!
  - It is not always cleared after it is read.
  - Always treat a **`cmp`** and a **`branch`** as one operation.
  - don't put any code between them!



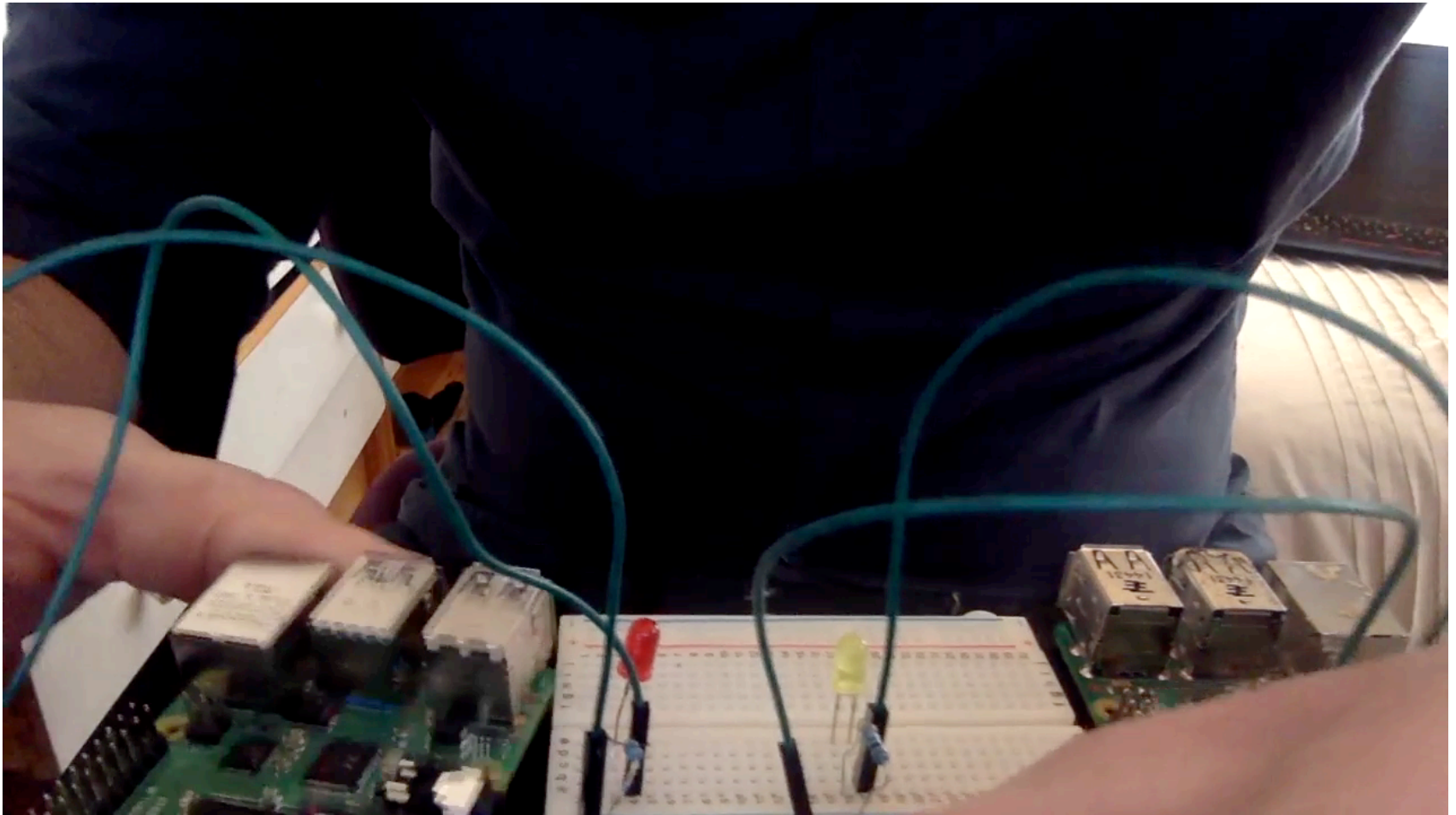
add 's' to  
other ALU  
operations  
to 'set' the  
APSR



# OK2 FLASHING LED (Pi 4)

```
BASE = $FE000000 ;or $3F000000 for 2B and 3B/3B+
RP1 GPIO_OFFSET = $200000
mov r0,BASE
orr r0,GPIO_OFFSET ;start of GPIO
mov r1,#1
lsl r1,#24
str r1,[r0,#4] ;set GPIO18 to output
loop$: ;outer loop - repeat LED on, wait, LED off, wait
    mov r1,#1
    lsl r1,#18
    str r1,[r0,#28] ;turn LED on
    mov r2,$3F0000
    wait1$:
        sub r2,#1
        cmp r2,#0
        bne wait1$ ;count from 4128768 to 0 (busy wait)
    mov r1,#1 ;can be omitted
    lsl r1,#18 ;can be omitted
    str r1,[r0,#40] ;turn LED off (writing to the pull up register)
    mov r2,$3F0000 ;Model 2 is a bit slower in single core mode
    wait2$:
        sub r2,#1
        cmp r2,#0
        bne wait2$ ;count from 4128768 to 0 (busy wait)
b loop$ ;end of outer loop
```

# OK2 RUNNING ON PI 2B AND PI 4B



# PERFORMANCE TESTING

- If we test on five different Pi models we get:

|             | RPI B  | RPI B+ | RPI 2  | RPI 3   | RPI 4   |
|-------------|--------|--------|--------|---------|---------|
| Clock Speed | 700MHz | 700MHz | 900MHz | 1.2 GHz | 1.5 GHz |
| Cores       | 1      | 1      | 4      | 4       | 4       |
| LED flash   | 1.6Hz  | 1.6Hz  | 1.1Hz  | 2.73Hz  | 4.33Hz  |

- Using a busy loop for timing is unreliable.
  - and wastes electrons!

# SUMMARY

- We have now made our LED flash:
  - We can turn LED on and off
  - Using busy wait timer
- Busy wait timer is inefficient
  - 100% CPU usage for nothing!
  - Wastes power
- Can we make a better timer ?
  - Yes – next lecture!