COS10004 Computer Systems

Lecture 10.5 – Functions in ARM Assembly - Nested Function Calls

CRICOS provider 00111D

Chris McCarthy

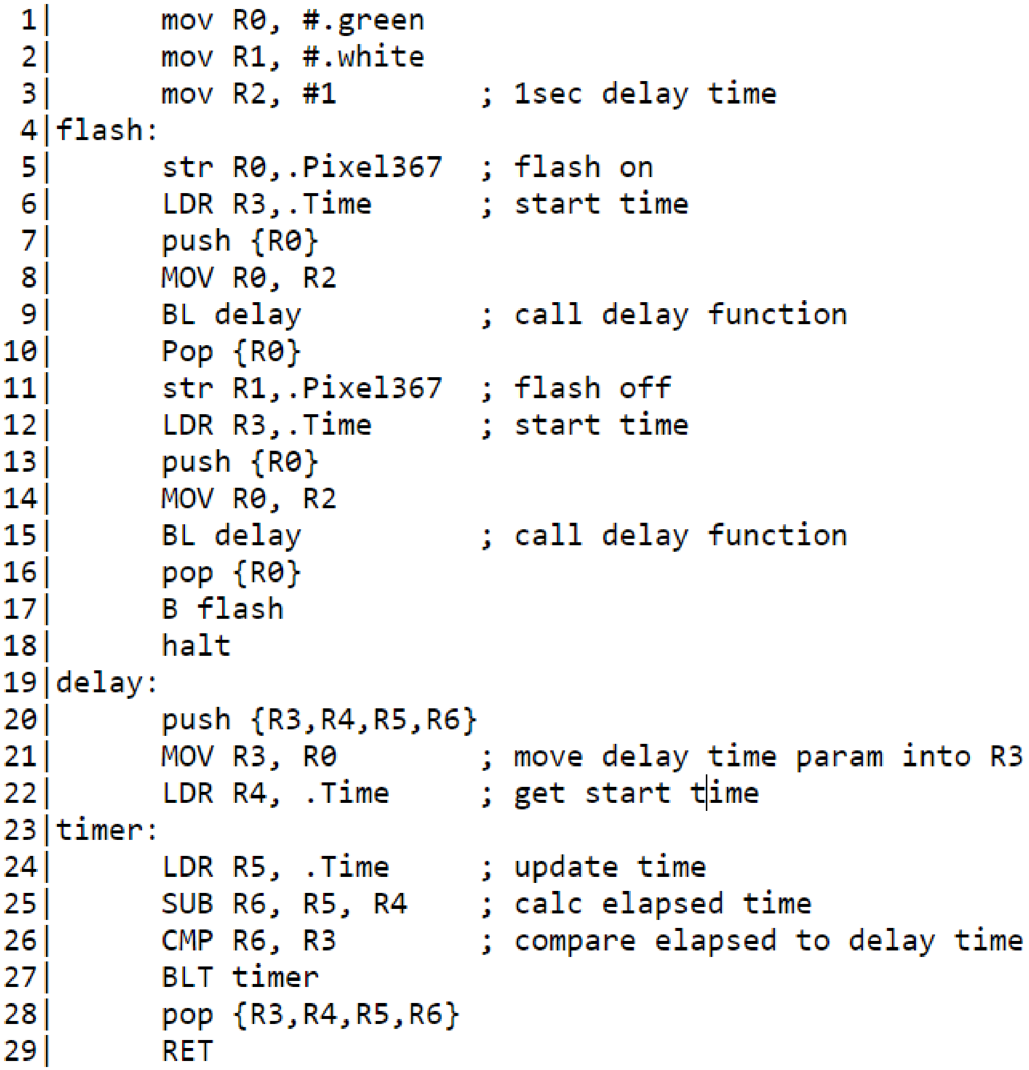
FUNCTIONS IN ASM

- Not 'native' to assembly
  - We need to do a lot of the management ourselves
- Argument passing:
  - How do we pass arguments from one function to another
- Storing and recalling register values
  - each function we call will want to use the same registers (only 13 general purpose registers !)
  - How do we manage this ?
- Managing the program control
  - Jumping from one function to another, and then returning back !

FUNCTIONS IN ASM

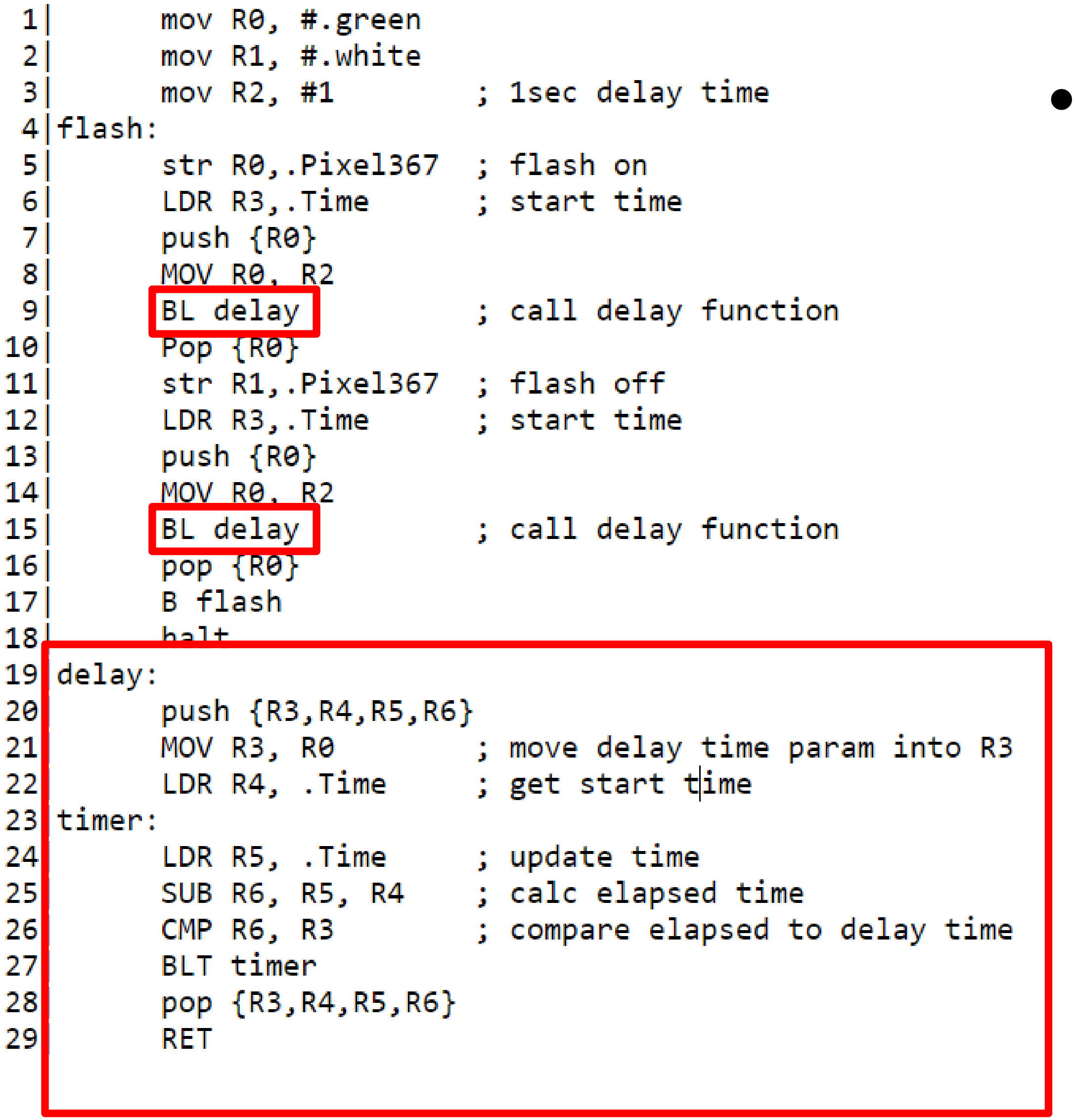
- Not 'native' to assembly
  - We need to do a lot of the management ourselves
- Argument passing:
  - How do we pass arguments from one function to another
- Storing and recalling register values
  - each function we call will want to use the same registers (only 13 general purpose registers !)
  - How do we manage this ?
- Managing the program control
  - Jumping from one function to another, and then returning back !

Recall our "Flashing LED" Program from last Lecture



```
      mov R0, #.green
      mov R1, #.white
      mov R2, #1        ; 1sec delay time
flash:
      str R0,.Pixel367  ; flash on
      LDR R3,.Time      ; start time
      push {R0}
      MOV R0, R2
      BL delay          ; call delay function
      Pop {R0}
      str R1,.Pixel367  ; flash off
      LDR R3,.Time      ; start time
      push {R0}
      MOV R0, R2
      BL delay          ; call delay function
      pop {R0}
      B flash
      halt
delay:
      push {R3,R4,R5,R6}
      MOV R3, R0        ; move delay time param into R3
      LDR R4, .Time     ; get start time
timer:
      LDR R5, .Time     ; update time
      SUB R6, R5, R4    ; calc elapsed time
      CMP R6, R3        ; compare elapsed to delay time
      BLT timer
      pop {R3,R4,R5,R6}
      RET
```

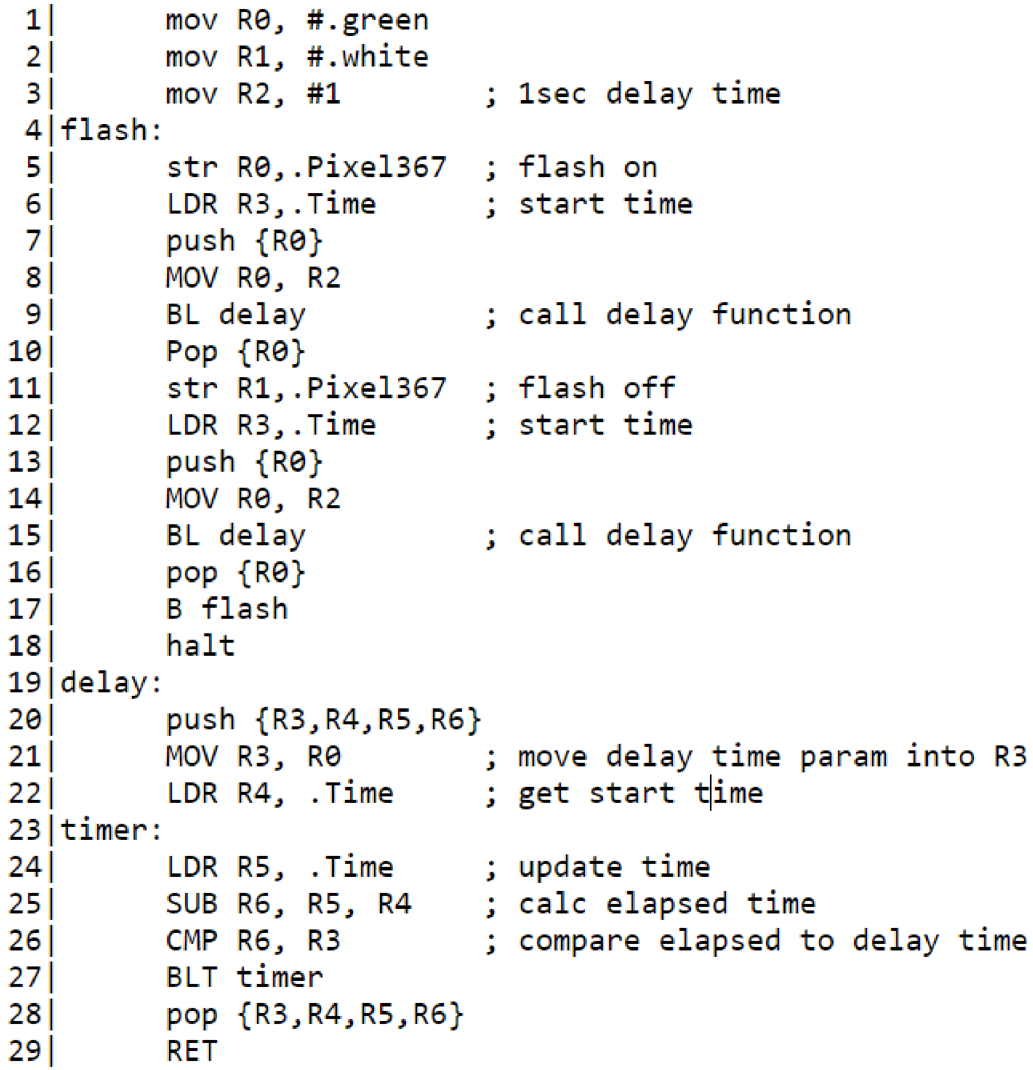
Recall our "Flashing LED" Program from last Lecture



```
        mov R0, #.green
        mov R1, #.white
        mov R2, #1          ; 1sec delay time
flash:
        str R0,.Pixel367    ; flash on
        LDR R3,.Time        ; start time
        push {R0}
        MOV R0, R2
        BL delay            ; call delay function
        Pop {R0}
        str R1,.Pixel367    ; flash off
        LDR R3,.Time        ; start time
        push {R0}
        MOV R0, R2
        BL delay            ; call delay function
        pop {R0}
        B flash
        halt
delay:
        push {R3,R4,R5,R6}
        MOV R3, R0          ; move delay time param into R3
        LDR R4, .Time       ; get start time
timer:
        LDR R5, .Time       ; update time
        SUB R6, R5, R4      ; calc elapsed time
        CMP R6, R3          ; compare elapsed to delay time
        BLT timer
        pop {R3,R4,R5,R6}
        RET
```

- Last lecture we created a function to call for delays.

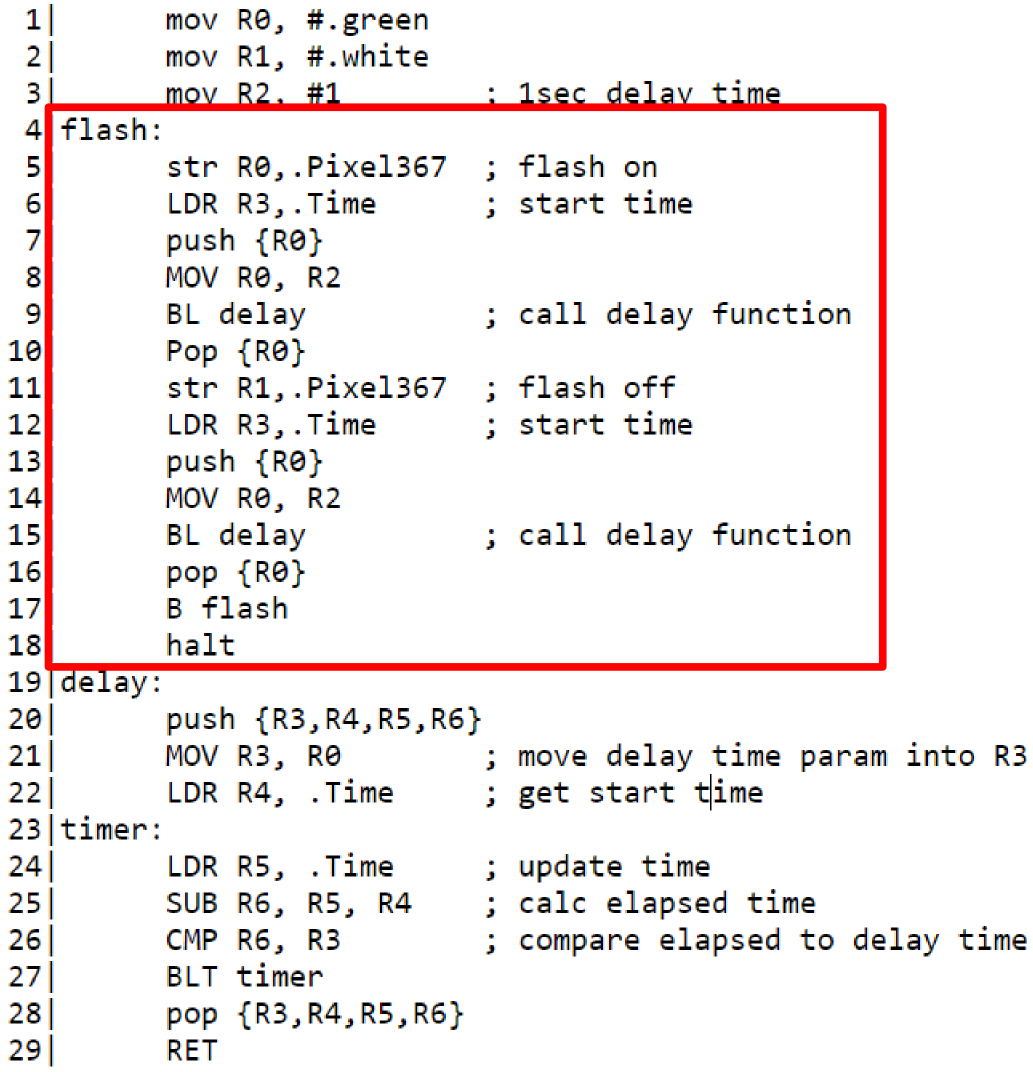
Recall our "Flashing LED" Program from last Lecture



```
      mov R0, #.green
      mov R1, #.white
      mov R2, #1        ; 1sec delay time
flash:
      str R0,.Pixel367  ; flash on
      LDR R3,.Time      ; start time
      push {R0}
      MOV R0, R2
      BL delay          ; call delay function
      Pop {R0}
      str R1,.Pixel367  ; flash off
      LDR R3,.Time      ; start time
      push {R0}
      MOV R0, R2
      BL delay          ; call delay function
      pop {R0}
      B flash
      halt
delay:
      push {R3,R4,R5,R6}
      MOV R3, R0        ; move delay time param into R3
      LDR R4, .Time     ; get start time
timer:
      LDR R5, .Time     ; update time
      SUB R6, R5, R4    ; calc elapsed time
      CMP R6, R3        ; compare elapsed to delay time
      BLT timer
      pop {R3,R4,R5,R6}
      RET
```

- But we can decompose this further!
- Think about the flash code

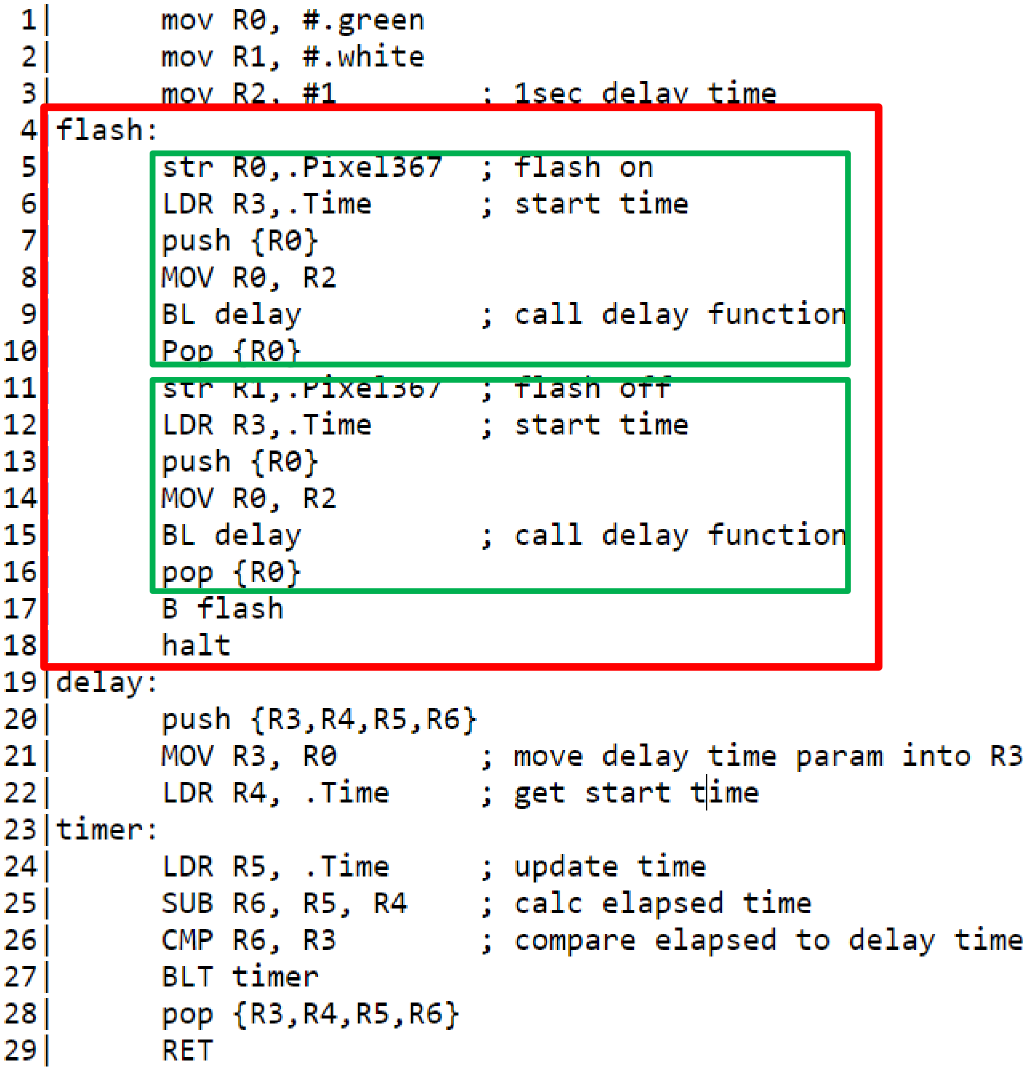
Recall our "Flashing LED" Program from last Lecture



```
      mov R0, #.green
      mov R1, #.white
      mov R2, #1        ; 1sec delay time
flash:
      str R0,.Pixel367  ; flash on
      LDR R3,.Time      ; start time
      push {R0}
      MOV R0, R2
      BL delay          ; call delay function
      Pop {R0}
      str R1,.Pixel367  ; flash off
      LDR R3,.Time      ; start time
      push {R0}
      MOV R0, R2
      BL delay          ; call delay function
      pop {R0}
      B flash
      halt
delay:
      push {R3,R4,R5,R6}
      MOV R3, R0        ; move delay time param into R3
      LDR R4, .Time     ; get start time
timer:
      LDR R5, .Time     ; update time
      SUB R6, R5, R4    ; calc elapsed time
      CMP R6, R3        ; compare elapsed to delay time
      BLT timer
      pop {R3,R4,R5,R6}
      RET
```

- But we can decompose this further!
- Let's write a flash function, and pass it a colour to flash, and a time delay between flashes.

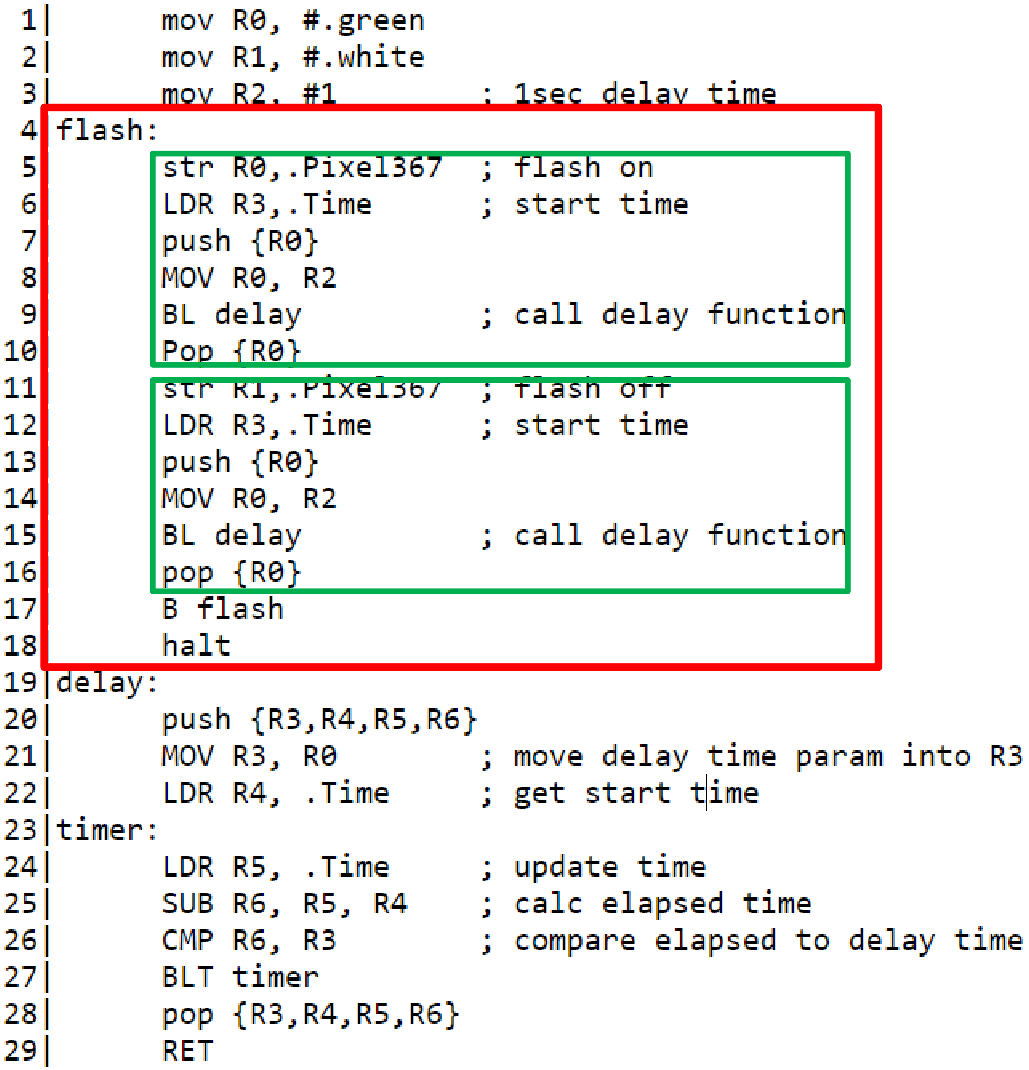
Recall our "Flashing LED" Program from last Lecture



```
      mov R0, #.green
      mov R1, #.white
      mov R2, #1          ; 1sec delay time
flash:
      str R0,.Pixel367    ; flash on
      LDR R3,.Time        ; start time
      push {R0}
      MOV R0, R2
      BL delay            ; call delay function
      Pop {R0}
      str R1,.Pixel367    ; flash off
      LDR R3,.Time        ; start time
      push {R0}
      MOV R0, R2
      BL delay            ; call delay function
      pop {R0}
      B flash
      halt
delay:
      push {R3,R4,R5,R6}
      MOV R3, R0          ; move delay time param into R3
      LDR R4, .Time       ; get start time
timer:
      LDR R5, .Time       ; update time
      SUB R6, R5, R4      ; calc elapsed time
      CMP R6, R3          ; compare elapsed to delay time
      BLT timer
      pop {R3,R4,R5,R6}
      RET
```

- But we can decompose this further!
- The flash code still has some repetitive code.

Recall our "Flashing LED" Program from last Lecture



```
      mov R0, #.green
      mov R1, #.white
      mov R2, #1          ; 1sec delay time
flash:
      str R0,.Pixel367    ; flash on
      LDR R3,.Time        ; start time
      push {R0}
      MOV R0, R2
      BL delay            ; call delay function
      Pop {R0}
      str R1,.Pixel367    ; flash off
      LDR R3,.Time        ; start time
      push {R0}
      MOV R0, R2
      BL delay            ; call delay function
      pop {R0}
      B flash
      halt
delay:
      push {R3,R4,R5,R6}
      MOV R3, R0          ; move delay time param into R3
      LDR R4, .Time       ; get start time
timer:
      LDR R5, .Time       ; update time
      SUB R6, R5, R4      ; calc elapsed time
      CMP R6, R3          ; compare elapsed to delay time
      BLT timer
      pop {R3,R4,R5,R6}
      RET
```

- The only difference between these blocks is the colour of the pixel being drawn.
- We can write this as a function and pass the colour (and delay time) as parameters

Draw Pixel Function

```
drawpixel:
      PUSH {R3,R4}
      MOV R3, R0        ; copy pixel colour to R3
      MOV R4, R1        ; copy delay time to R4
     STR R3, .Pixel367 ; draw pixel with colour
     PUSH {R0}
     MOV R0, R4         ; pass delay time to delay function
     BL delay           ; call delay function
     Pop {R0}
     RET                ; job done
```

FLASH LOOP (NOW CALLING DRAWPIXEL)

```
mov R2, #1       ; 1sec delay time
flash:
        MOV R0, #.green             ; pass green to drawpixel
        MOV R1, R2                  ; pass time delay
        BL drawpixel                ; "flash on“

        MOV R0, #.white             ; pass white to drawpixel
        MOV R1, R2                  ; pass time delay
        BL drawpixel                ; "flash off"
        B flash
HALT
```

AWESOME – BUT DOES IT WORK ?

- Let’s see it in ARMlite
- Spoiler alert … no … but can you see why ?
  - – Rewind video and take another look at drawpixel
  - – *Hint: think about what happens to LR when functions call other functions*

DRAW PIXEL FUNCTION (FIXED)

```
drawpixel:
     PUSH {R3,R4}
     MOV R3, R0        ; copy pixel colour to R3
     MOV R4, R1        ; copy delay time to R4
    STR R3, .Pixel367 ; draw pixel with colour
    PUSH {R0}
    MOV R0, R4        ; pass delay time to delay function
    PUSH {LR}         ; backup LR before BL delay overwrites it!
    BL delay          ; call delay function
    POP {LR}          ; restore LR after we've returned from delay
    Pop {R0}
    RET               ; job done
```

SUMMARY

- All of this is interesting because this is exactly what compilers have to handle when translating your code in higher level languages
- Critical components include LR and PC registers,
- But also **the stack !**
  - **The stack** allows functions to do what they do using all registers, and then restores what was there previously
  - This is also critical to interrupt handling
    - we’ll come back to this