Subroutines and parameter passing

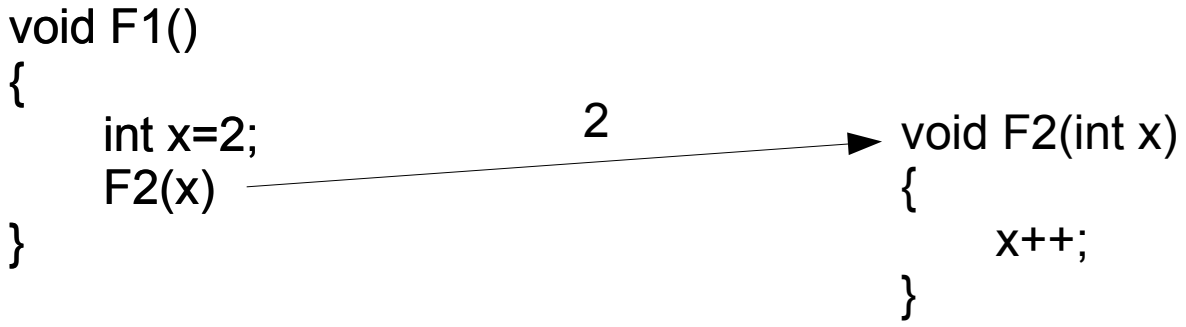
Subroutines and parameter passing

- In C and other high level languages we regularly pass parameters to functions

```
void F1()
{
        int x=2;
        F2(x)
}
```

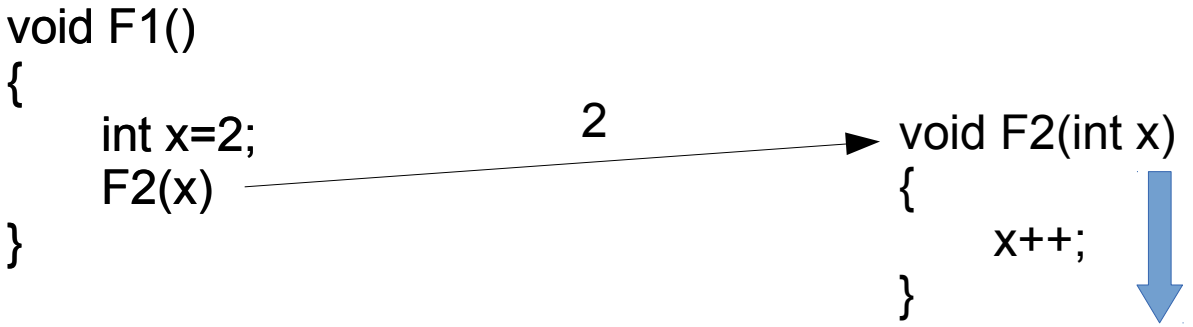
Subroutines and parameter passing

- In C and other high level languages we regularly pass parameters to functions



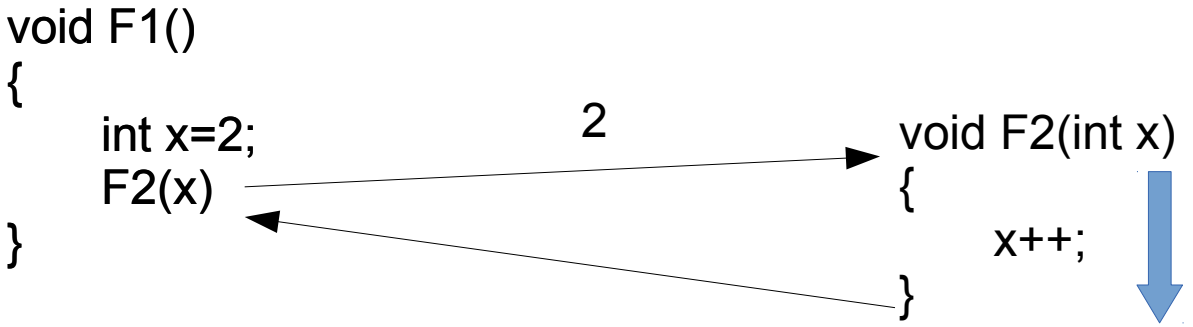
Subroutines and parameter passing

- In C and other high level languages we regularly pass parameters to functions



Subroutines and parameter passing

- In C and other high level languages we regularly pass parameters to functions



What value is the variable x in F1 after F2?

How exactly does the value 2 get passed to F2?

Subroutines and parameter passing

- The value of x in F1 in fact remains unchanged
- Function calls like this pass a disposable copy of the data in parameters.
- If you need F2 to change F1's copy of x you need to pass a pointer to x instead.

Subroutines and parameter passing

- How does the copy of x get to F2?
- It depends....
- In the ARM world there is a standard way of doing this:

  The ARM Achitecture Procedure Call Standard (AAPCS)

http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf

Subroutines and parameter passing

Passing 32 bit values (ints)

Register	Role
R0	Parameter 1
R1	Parameter 2
R2	Parameter 3
R3	Parameter 4

64 bit values are passed using pairs of registers
128 bit values are passed using 4 registers

http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf

Subroutines and parameter passing

- What if you need to pass lots of variables?
- When registers are exhausted the stack is used to pass parameters.
- The target function looks “up” the stack for the incoming parameters.
- Why don't we use the stack for everything?

Subroutines and parameter passing

- How are values returned?
- Again R0 to R3 are used.
- Single result <= 32 bits is returned in R0
- Wider values returned in combinations of R0 to R3

Subroutines and parameter passing

```
    AREA DATA

    ; The following symols are in the CODE section (ROM,Executable
(Thumb), readonly)
    AREA THUMB,CODE,READONLY
    ; EXPORTED Symbols can be linked against
        EXPORT Reset_Handler
        EXPORT __Vectors
    ; Minimal interrupt vector table follows
    ; First entry is initial stack pointer (end of stack)
    ; second entry is the address of the reset handler
__Vectors
        DCD 0x20001000
        DCD Reset_Handler
    ; 'Main' program goes here

Reset_Handler
        MOVS R0,#12
        MOVS R1,#3
        BL pow
stop    B stop
```

Subroutines and parameter passing

```
; functions
; unsigned int pow(unsigned int value, unsigned int power)
; This function takes a two argument in R0,
; and R1.  It raises R0 to the power specified
; in R1.  A special case arises if R1 is zero
; in which case the returned value is 1
; The result is returned in R0
pow
       push {LR}
       push {R2}        ; backup changed registers
       MOVS R2,R0       ; preserve original value in R2
       CMP R1,#0
       BNE do_pow
       MOVS R0,#1       ; power = 0 so return a 1
       B pow_exit
do_pow
       SUBS R1,R1,#1     ; decrement 'power'
       CMP R1, #0        ; finished?
       BEQ pow_exit      ; if so, then exit
       MULS R0,R2,R0     ; otherwise multiply again
       B do_pow
pow_exit
       POP {R2}         ; restore changed registers
       POP {PC}

    end
```