

# Stack Frames (12A)

---

---

Copyright (c) 2014 - 2020 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).

This document was produced by using LibreOffice.

# Based on

---

ARM System-on-Chip Architecture, 2<sup>nd</sup> ed, Steve Furber

Introduction to ARM Cortex-M Microcontrollers  
– Embedded Systems, Jonathan W. Valvano

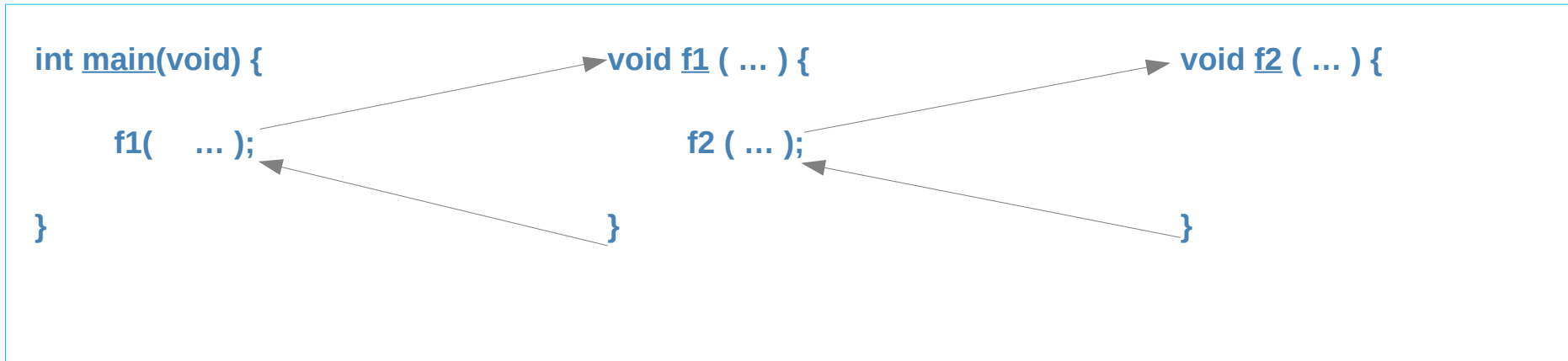
Digital Design and Computer Architecture,  
D. M. Harris and S. L. Harris

ARM assembler in Raspberry Pi  
Roger Ferrer Ibáñez

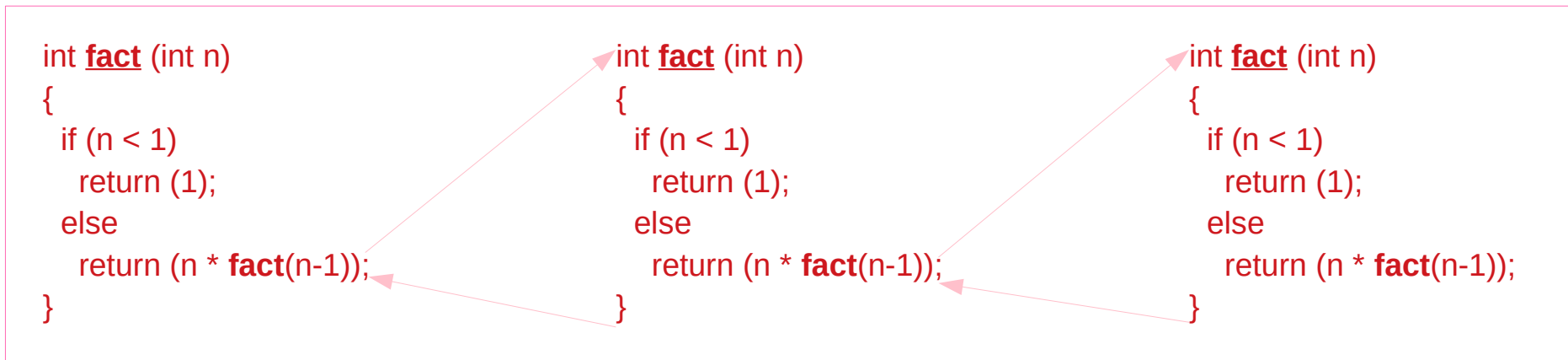
<https://thinkingeek.com/arm-assembler-raspberry-pi/>

# Nested and recursive function calls

## Nested function call



## Recursive function call

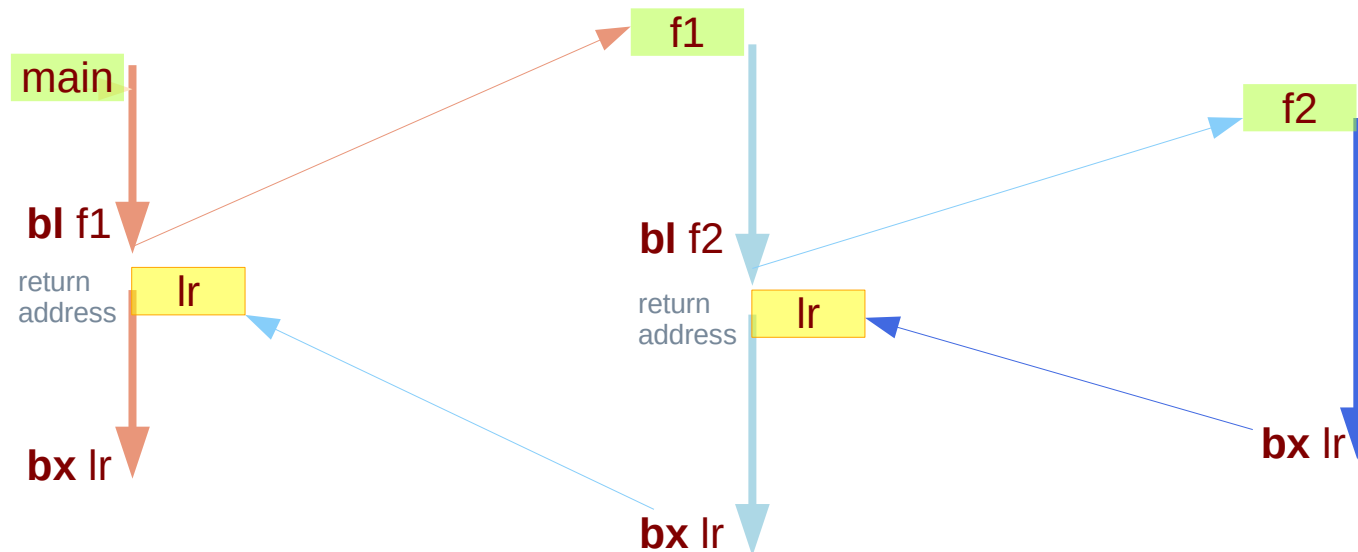


# Nested and recursive function calls

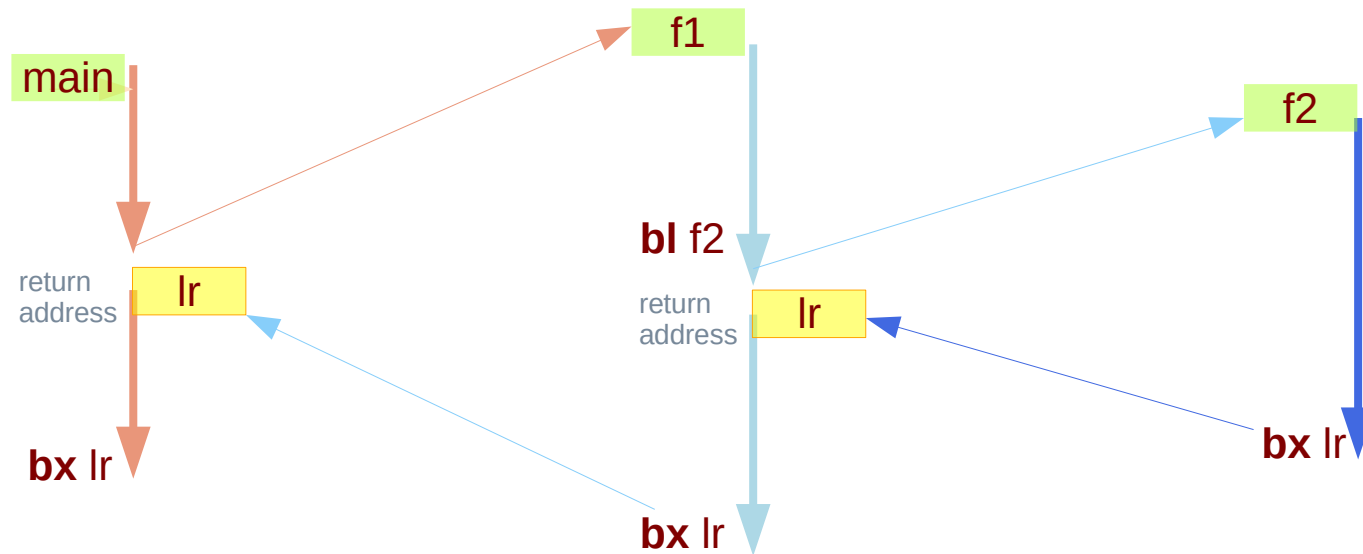
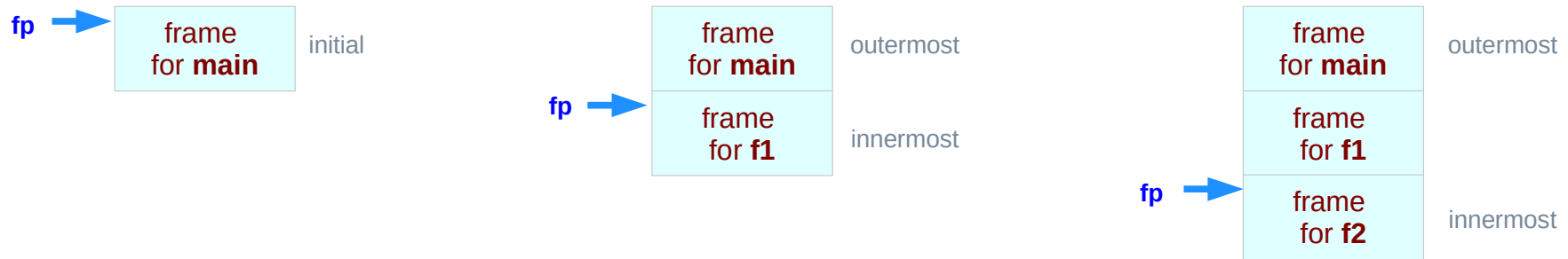
At least, **LR** must not be overwritten

save the followings in a **frame**

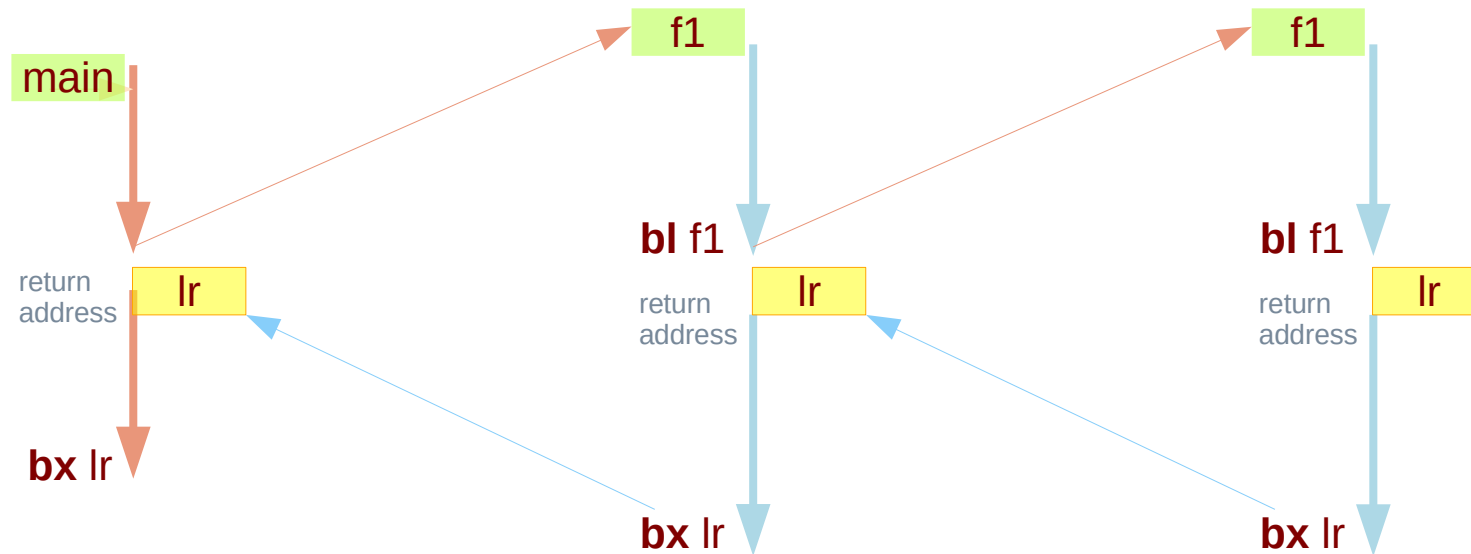
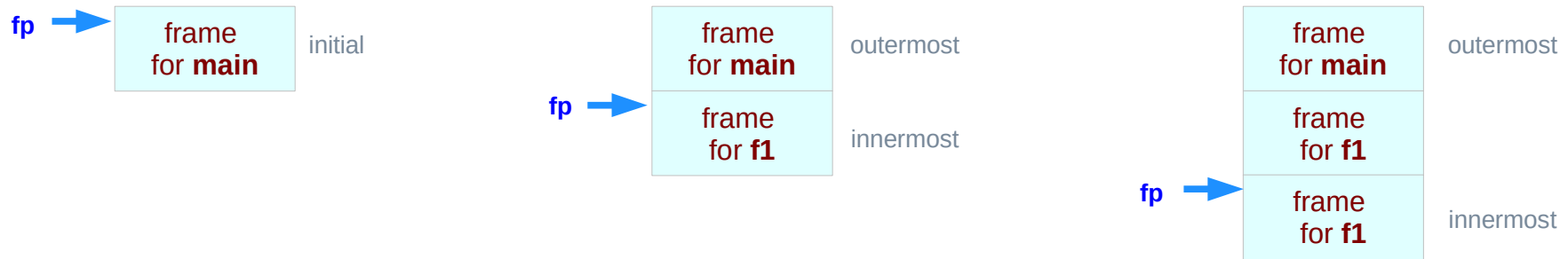
- return address
- arguments
- local variables.



# Nested and recursive function calls



# Recursive function calls



# Activation records (1)

## local variables

- created upon entry to **function**.
- destroyed when function returns.

each **invocation** of a function has its own instantiation of **local variables**.

- recursive and nest calls to a function require several instantiations to exist simultaneously.
- functions return only after all functions it calls have returned last-in-first-out(**LIFO**) behavior.
- a **LIFO** structure called a **stack** is used to hold each instantiation.

the portion of the stack used for an **invocation** of a function is called the function's **stack frame** or **activation record**

<https://www.cs.princeton.edu/courses/archive/spring03/cs320/notes/7-1.pdf>



# Activation records (2)

## a stack frame

a frame of data that gets pushed onto the stack.

## a call stack

divided up into contiguous pieces called **stack frames** which represent a **function call** and its **argument** data.

- **return address**
- **arguments**
- **local variables.**

architecture-dependent.

processor knows the size of each frame and moves the **stack pointer** accordingly as **frames** are pushed and popped off the stack.

<https://stackoverflow.com/questions/10057443/explain-the-concept-of-a-stack-frame-in-a-nutshell>

# Activation records (3)

when your program is started,  
the **call stack** has only one frame,  
that of the function **main()**.  
the **initial frame** or the **outermost frame**.

each time a function is called,  
a new frame is added.  
each time a function returns,  
the frame for that function call is eliminated.

for a recursive function,  
there can be many frames for the same function.

the frame for the currently executing function  
is called the **innermost frame**.  
the most recently created frame

[http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.prog%2Ftopic%2Fusing\\_gdb\\_StackFrames.html](http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.prog%2Ftopic%2Fusing_gdb_StackFrames.html)

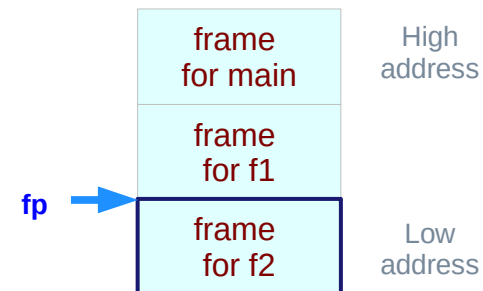
# Activation records (4)

A **stack frame** consists of many bytes

**stack frames** are identified by their addresses.

**the address of the frame** depends on architectures

Usually this address is kept in a register called the **frame pointer register fp** while execution is going on in that frame.

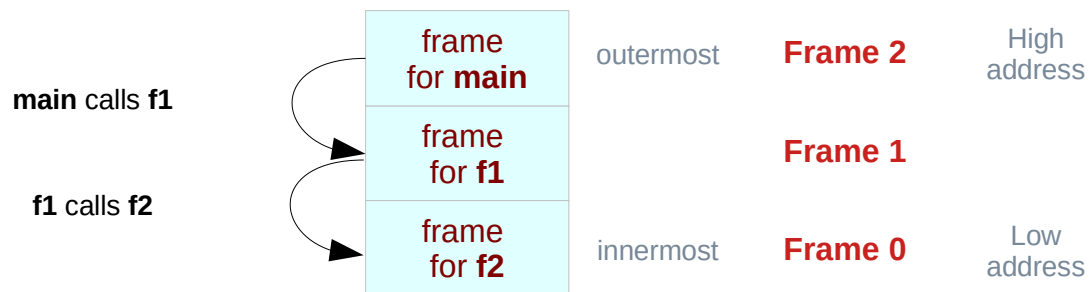


[http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.prog%2Ftopic%2Fusing\\_gdb\\_StackFrames.html](http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.prog%2Ftopic%2Fusing_gdb_StackFrames.html)

# Activation records (5)

**GDB** assigns numbers to all existing stack frames, starting with **0** for the **innermost** frame, **1** for the frame that called it, and so on upward.

These numbers don't really exist in your program; they're assigned by GDB to give you a way of designating stack frames in GDB commands.



[http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.prog%2Ftopic%2Fusing\\_gdb\\_stackframes.html](http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.prog%2Ftopic%2Fusing_gdb_stackframes.html)

# Activation records (6)

## **a call stack**

a stack data structure that stores information about the **active subroutines** of a computer program.

Although maintenance of the **call stack** is important for the proper functioning of most software, the details are normally **hidden** and **automatic** in high-level programming languages.

Many computer instruction sets provide **special instructions** for manipulating stacks.

also known as an

- **execution stack**
- **program stack**
- **control stack**
- **run-time stack**
- **machine stack**

[https://en.wikipedia.org/wiki/Call\\_stack](https://en.wikipedia.org/wiki/Call_stack)

# Activation records (7)

A **call stack** is used for several related purposes, but the main reason for having one is to keep track of the point to which each **active subroutine** should return control when it finishes executing.

An **active subroutine** is one that has been called, but is yet to complete execution, after which control should be handed back to the point of call.

Such **activations** of subroutines may be nested to any level (recursive as a special case), hence the **stack structure**.

[https://en.wikipedia.org/wiki/Call\\_stack](https://en.wikipedia.org/wiki/Call_stack)

# Argument, scratch, variable, return result registers

## **R0 – R3, R12 :**

argument or scratch registers

that are not preserved by the **callee** on a procedure call

## **R4 – R11**

8 **variable registers** that must be preserved on a procedure call  
(if used, the **callee** must save and restore them)

## **R0, R1 :**

return result registers

The called performs the calculations,  
places the result (if any) in **R0** and **R1**  
and returns control to the caller using **MOV PC, LR**

# Argument, scratch, variable, return result registers

Registers that is preserved across a procedure

variable registers **R4 – R11**

stack pointer register **sp**

link register **lr**

stack above the stack pointer

Registers that is not preserved across a procedure

argument registers **R0 – R3**

intra procedure call scratch register **r12**

stack below the stack pointer



# Frame pointer and stack pointer registers (1)

**LR (R14, link register, )**

where you were

**PC (R15, program counter)**

where you are

**FP (R11, frame pointer)**

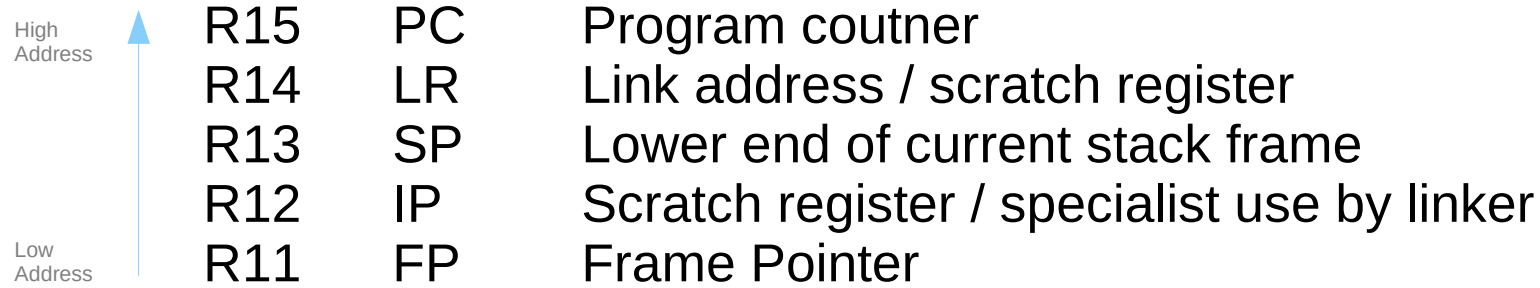
where the stack was

**SP (R13, stack pointer)**

where the stack is

<https://stackoverflow.com/questions/15752188/arm-link-register-and-frame-pointer>

# APCS Register Use Convention



High Address	R15	PC	Program counter
	R14	LR	Link address / scratch register
	R13	SP	Lower end of current stack frame
	R12	IP	Scratch register / specialist use by linker
Low Address	R11	FP	Frame Pointer

# Frame pointer and stack pointer registers (4)

The basic frame layout is,

fp[-0] saved pc, where we stored this frame.

fp[-1] saved lr, the return address for this function.

fp[-2] previous sp, before this function eats stack.

fp[-3] previous fp, the last stack frame.

many optional registers...

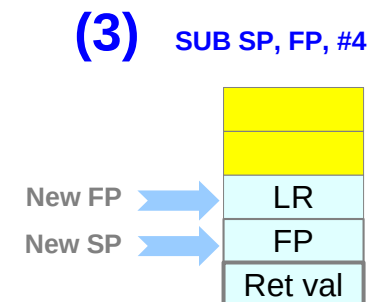
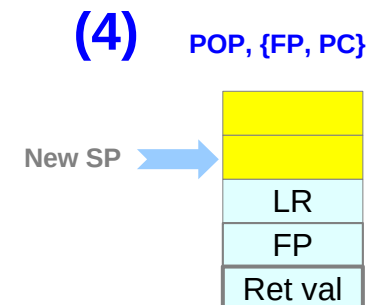
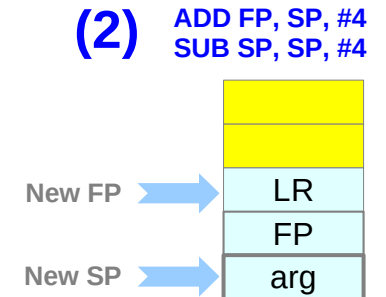
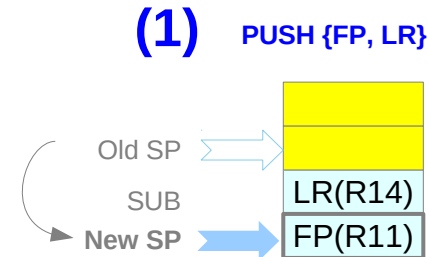
<https://stackoverflow.com/questions/15752188/arm-link-register-and-frame-pointer>

# Stack frame example A

```
Int add(int a, int b) {  
    int c;  
    c = a + b;  
    some_func(a,b);  
    return c;  
}
```

```
0x00010414 <+0>  
0x00010418 <+4>  
0x0001041c <+8>  
0x00010420 <+12>  
0x00010424 <+16>  
0x00010428 <+20>  
0x0001042c <+24>  
0x00010430 <+28>  
0x00010434 <+32>
```

```
push    {fp, lr}  
add     fp, sp, #4  
sub     sp, sp, #4  
add     r3, r0, r1  
str     r3, [fp-#8]  
bl    some_func  
str     r0, [fp-#8]  
sub     sp, fp, #4  
pop     {fp, pc}
```



<https://lloydrochester.com/post/c/stack-of-frames-arm/>

# Stack frame example B

```
int one(int, int);
int two(int, int);
int three(int, int);

Int main(int argc, char *argv[])
{
    int ia, ib, ic;

    ia = 1;
    ib = 2;
    ic = one(ia, ib);

    return ic;
}
```

```
Int one(int a, int b)
{
    int c;
    c = two(a,b);
    return c;
}
```

```
Int two(int a, int b)
{
    int c;
    c = three(a,b);
    return c;
}
```

```
Int three(int a, int b)
{
    int c;
    c = a+b;
    return c;
}
```

<https://lloydchrochester.com/post/c/stack-of-frames-arm/>

# Stack frame example B

(gdb) disassemble **main**

Dump of assembler code for function main:

0x000103d0 <+0>:	push	{r11, lr}	; lr=0xbfe84718 r11 at lowest address
0x000103d4 <+4>:	add	r11, sp, #4	; r11=fp=0x0
0x000103d8 <+8>:	sub	sp, sp, #24	; sp=0xbeff4d8, frame is size 28=24+4
0x000103dc <+12>:	str	r0, [r11, #-24]	; 0xfffffe8
0x000103e0 <+16>:	str	r1, [r11, #-28]	; 0xfffffe4
0x000103e4 <+20>:	mov	r3, #1	
0x000103e8 <+24>:	str	r3, [r11, #-8]	
0x000103ec <+28>:	mov	r3, #2	
0x000103f0 <+32>:	str	r3, [r11, #-12]	
0x000103f4 <+36>:	ldr	r1, [r11, #-12]	
0x000103f8 <+40>:	ldr	r0, [r11, #-8]	
0x000103fc <+44>:	<b>bl</b>	0x10414 < <b>one</b> >	; here the lr will be set to 0x00010400
0x00010400 <+48>:	str	r0, [r11, #-16]	; r0 has the return value from function one
0x00010404 <+52>:	ldr	r3, [r11, #-16]	
0x00010408 <+56>:	mov	r0, r3	; r0 will return with the value of int ic
0x0001040c <+60>:	sub	sp, r11, #4	; point sp one word above fp
0x00010410 <+64>:	pop	{r11, pc}	; pc will be restored to 0xbfe84718

End of assembler dump.

<https://lloydrochester.com/post/c/stack-of-frames-arm/>

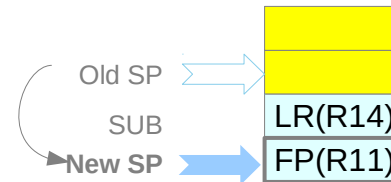
# Stack frame example B

main

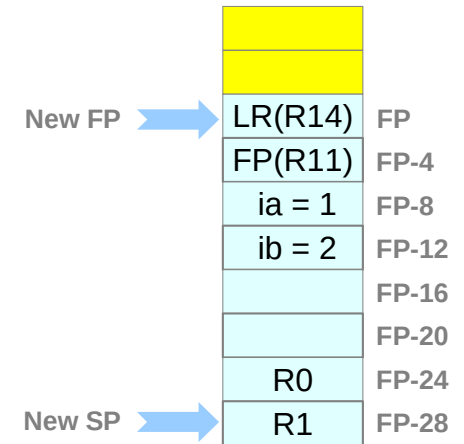
```

push  {r11, lr}
add   r11, sp, #4
sub   sp, sp, #24
str   r0, [r11, #-24]
str   r1, [r11, #-28]
mov   r3, #1
str   r3, [r11, #-8]
mov   r3, #2
str   r3, [r11, #-12]
ldr   r1, [r11, #-12]
ldr   r0, [r11, #-8]
bl  0x10414 <one>
str   r0, [r11, #-16]
ldr   r3, [r11, #-16]
mov   r0, r3
sub   sp, r11, #4
pop   {r11, pc}
    
```

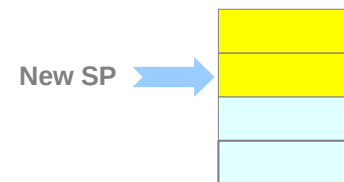
(1) PUSH {FP, LR}



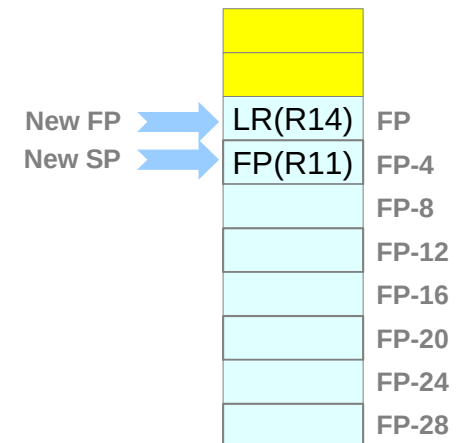
(2) ADD FP, SP, #4  
SUB SP, SP, #24



(4) POP, {FP, PC}



(3) SUB SP, FP, #4



<https://lloydrochester.com/post/c/stack-of-frames-arm/>

# Stack frame example B

(gdb) disassemble **one**

Dump of assembler code for function one:

0x00010414 <+0>:	push	{r11, lr}	; lr=0x00010400 r11=fp=0xbefff4d0
0x00010418 <+4>:	add	r11, sp, #4	; r11=fp=0xbefff4d4
0x0001041c <+8>:	sub	sp, sp, #16	; sp=0xbefff4c0 frame is size 20=16+4
0x00010420 <+12>:	str	r0, [r11, #-16]	
0x00010424 <+16>:	str	r1, [r11, #-20]	; 0xffffffff
0x00010428 <+20>:	ldr	r1, [r11, #-20]	; 0xffffffff
0x0001042c <+24>:	ldr	r0, [r11, #-16]	
0x00010430 <+28>:	<b>bl</b>	0x10448 < <b>two</b> >	; lr will be 0x00010434
0x00010434 <+32>:	str	r0, [r11, #-8]	
0x00010438 <+36>:	ldr	r3, [r11, #-8]	
0x0001043c <+40>:	mov	r0, r3	
0x00010440 <+44>:	sub	sp, r11, #4	; point sp one word above fp
0x00010444 <+48>:	pop	{r11, pc}	; fp=0xbefff4f4, lr=0x00010400

End of assembler dump.

<https://lloydrochester.com/post/c/stack-of-frames-arm/>



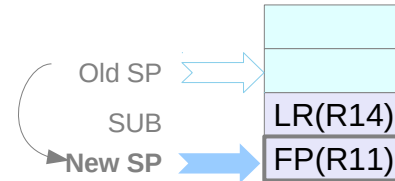
# Stack frame example B

one

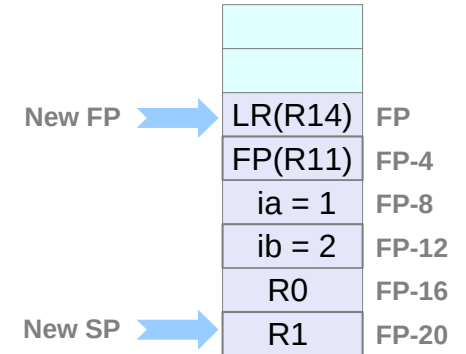
```

push    {r11, lr}
add     r11, sp, #4
sub     sp, sp, #16
str     r0, [r11, #-16]
str     r1, [r11, #-20]
ldr     r1, [r11, #-20]
ldr     r0, [r11, #-16]
bl    0x10448 <two>
str     r0, [r11, #-8]
ldr     r3, [r11, #-8]
mov     r0, r3
sub     sp, r11, #4
pop     {r11, pc}
    
```

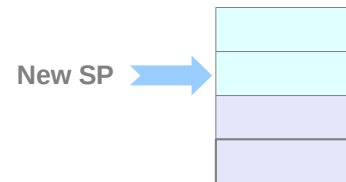
(1) PUSH {FP, LR}



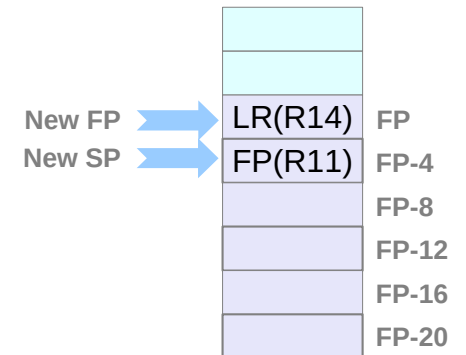
(2) ADD FP, SP, #4  
SUB SP, SP, #24



(4) POP, {FP, PC}



(3) SUB SP, FP, #4



<https://lloydchrochester.com/post/c/stack-of-frames-arm/>

# Stack frame example B

(gdb) disassemble **two**

Dump of assembler code for function two:

```
0x00010448 <+0>:  push    {r11, lr}           ; lr=0x00010434, r11=fp=0xbefff4d4
0x0001044c <+4>:  add     r11, sp, #4         ; fp=0xbefff4bc
0x00010450 <+8>:  sub     sp, sp, #16        ; sp=0xbefff4a8 frame is 20=16+4 words
0x00010454 <+12>: str     r0, [r11, #-16]
0x00010458 <+16>: str     r1, [r11, #-20]    ; 0xffffffff
0x0001045c <+20>: ldr     r1, [r11, #-20]    ; 0xffffffff
0x00010460 <+24>: ldr     r0, [r11, #-16]
0x00010464 <+28>: bl    0x1047c <three>    ; lr will be set to 0x00010468
0x00010468 <+32>: str     r0, [r11, #-8]
0x0001046c <+36>: ldr     r3, [r11, #-8]
0x00010470 <+40>: mov     r0, r3
0x00010474 <+44>: sub     sp, r11, #4
0x00010478 <+48>: pop     {r11, pc}
```

End of assembler dump.

<https://lloydrochester.com/post/c/stack-of-frames-arm/>

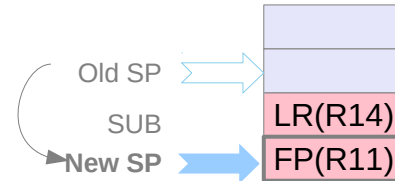
# Stack frame example B

two

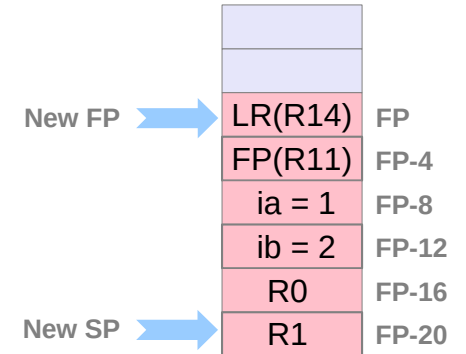
```

push    {r11, lr}
add     r11, sp, #4
sub     sp, sp, #16
str     r0, [r11, #-16]
str     r1, [r11, #-20]
ldr     r1, [r11, #-20]
bl    0x1047c <three>
str     r0, [r11, #-8]
ldr     r3, [r11, #-8]
mov     r0, r3
sub     sp, r11, #4
pop     {r11, pc}
    
```

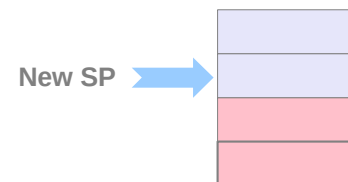
(1) PUSH {FP, LR}



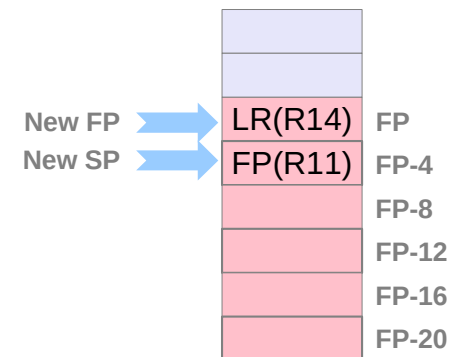
(2) ADD FP, SP, #4  
SUB SP, SP, #24



(4) POP, {FP, PC}



(3) SUB SP, FP, #4



<https://lloydrochester.com/post/c/stack-of-frames-arm/>

# Stack frame example B

(gdb) disassemble **three**

Dump of assembler code for function three:

```
0x0001047c <+0>:  push    {r11}                ; (str r11, [sp, #-4]!) NOTICE no lr!!
0x00010480 <+4>:  add     r11, sp, #0           ; dont add #4 here since no frp=0xbefff4a4
0x00010484 <+8>:  sub     sp, sp, #20          ; stack is size 20 sp=0xbfff490
0x00010488 <+12>: str     r0, [r11, #-16]
0x0001048c <+16>: str     r1, [r11, #-20]      ; 0xfffffec
0x00010490 <+20>: ldr     r2, [r11, #-16]
0x00010494 <+24>: ldr     r3, [r11, #-20]      ; 0xfffffec
0x00010498 <+28>: add     r3, r2, r3
0x0001049c <+32>: str     r3, [r11, #-8]
0x000104a0 <+36>: ldr     r3, [r11, #-8]
0x000104a4 <+40>: mov     r0, r3
0x000104a8 <+44>: add     sp, r11, #0
0x000104ac <+48>: pop     {r11}                ; (ldr r11, [sp], #4)
0x000104b0 <+52>: bx    lr                    ; lr=0x10468
```

End of assembler dump.

<https://lloydrochester.com/post/c/stack-of-frames-arm/>

# Stack frame example B

three

```
push    {r11}
add     r11, sp, #0
sub     sp, sp, #20
```

```
str     r0, [r11, #-16]
str     r1, [r11, #-20]
ldr     r2, [r11, #-16]
ldr     r3, [r11, #-20]
add     r3, r2, r3
str     r3, [r11, #-8]
ldr     r3, [r11, #-8]
mov     r0, r3
```

```
add     sp, r11, #0
pop     {r11}
```

```
bx     lr
```

(1) PUSH {FP}

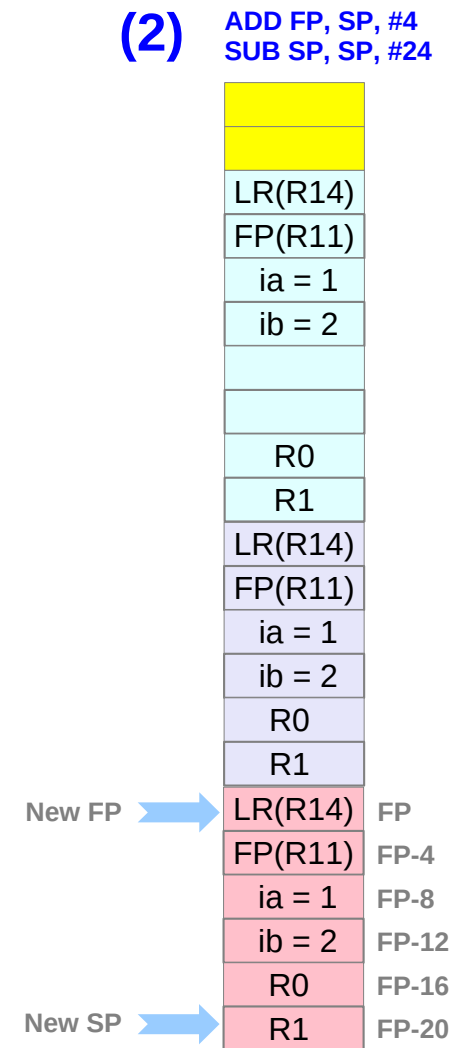
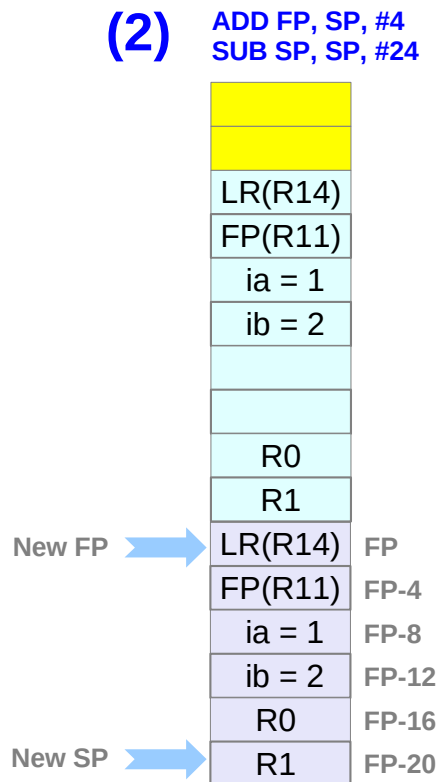
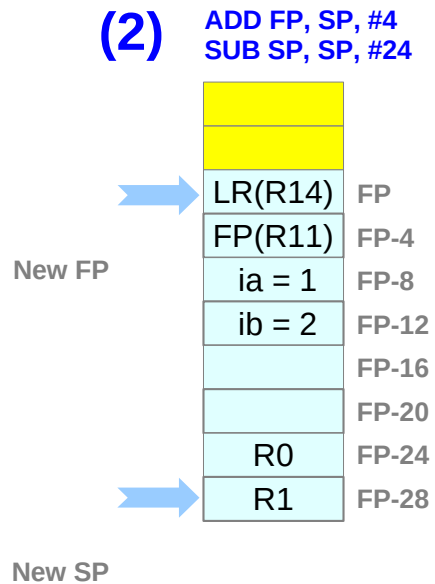
(2) ADD FP, SP, #0  
SUB SP, SP, #20

(4) POP, {FP}

(3) SUB SP, FP, #0

<https://lloydrochester.com/post/c/stack-of-frames-arm/>

# Stack frame example B

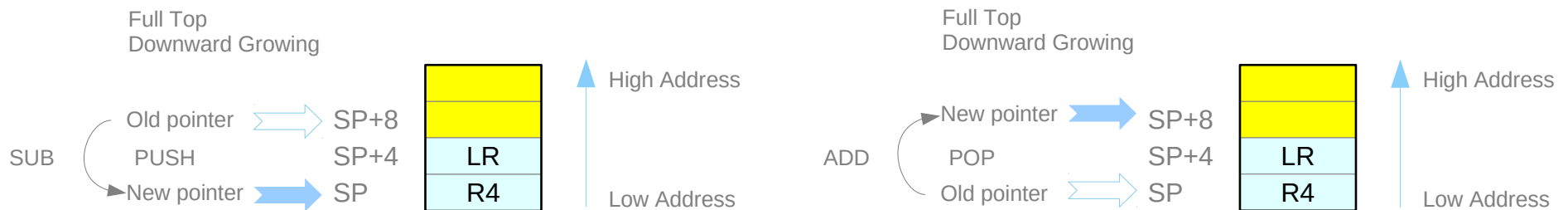


<https://lloydrochester.com/post/c/stack-of-frames-arm/>

# Activation records (1)

```
function:                ; keep callee-saved registers

    push {r4, lr}        ; keep the callee saved registers
    ...                  ; code of the function
    pop {r4, lr}         ; restore the callee saved registers
    bx lr                ; return from the function
```



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

## Activation records (2)

function:

```
push {r4, r5, fp, lr}
```

```
mov fp, sp
```

```
...
```

```
mov sp, fp
```

```
pop {r4, r5, fp, lr}
```

```
bx lr
```

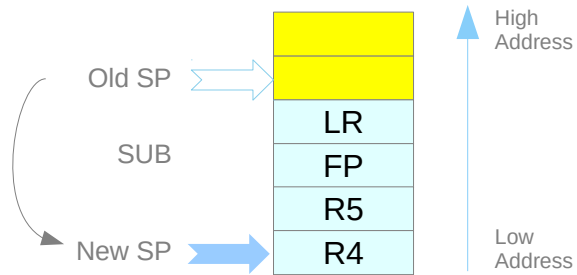
```
; keep callee-saved registers  
; keep the callee saved registers.  
; we added r5 to keep the stack 8-byte aligned  
; but the important thing here is fp  
; fp ← sp. Keep dynamic link in fp  
; code of the function  
; sp ← fp. Restore dynamic link in fp  
; restore the callee saved registers.  
; this will restore fp as well  
; return from the function
```

<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

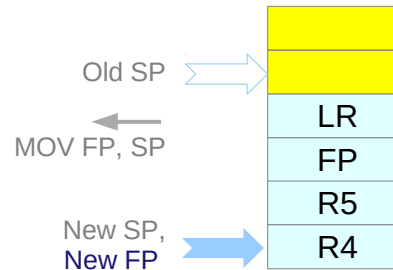


# Activation records (2)

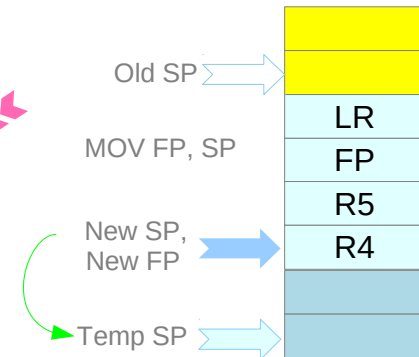
1. push {r4, r5, fp, lr}



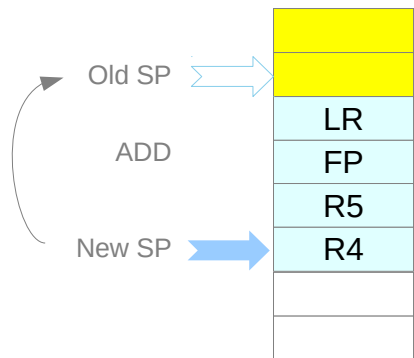
2. mov fp, sp



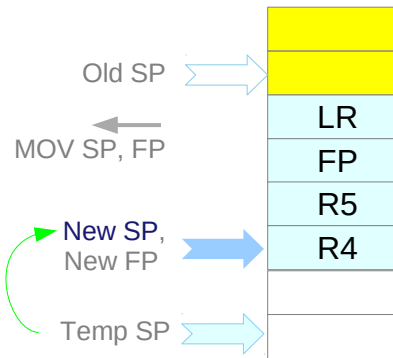
3. function code



5. pop {r4, r5, fp, lr}



4. mov sp, fp



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# Activation records (3)

function:

```
push{r4, r5, fp, lr}
mov fp, sp
sub sp, sp, #8
...
mov sp, fp
pop {r4, r5, fp, lr}

bx lr
```

```
; keep callee-saved registers
; keep the callee saved registers.
; w added r5 to keep the stack 8-byte aligned
; but the important thing here is fp
; fp ← sp. Keep dynamic link in fp
; enlarge the stack by 8 bytes
; code of the function
; sp ← fp. restore dynamic link in fp
; restore the callee saved registers.
; this will restore fp as well
; return from the function
```

<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# Local Data Generating Examples

```
void sq(int *c)
{
    (*c) = (*c) * (*c);
}
```

```
int sq_sum5(int a, int b, int c, int d, int e)
{
    sq(&a);
    sq(&b);
    sq(&c);
    sq(&d);
    sq(&e);
    return a + b + c + d + e;
}
```

```
...
sq_sum5(1, 2, 3, 4, 5);
...
```

callee  
function

- **sq** received a reference
- registers do not have an address
- allocate temporary local storage

caller  
function

# Callee Function Code

```
sq_sum5:  
push { fp, lr }  
mov fp, sp  
sub sp, sp, #16
```

```
str r0, [ fp, #-16 ]    *( fp - 16 ) ← r0  
str r1, [ fp, #-12 ]    *( fp - 12 ) ← r1  
str r2, [ fp, #-8 ]     *( fp - 8 ) ← r2  
str r3, [ fp, #-4 ]     *( fp - 4 ) ← r3
```

```
mov sp, fp  
pop { fp, lr }  
bx lr
```

```
sq:  
ldr r1, [ r0 ]          r1 ← ( *r0 )  
mul r1, r1, r1          r1 ← r1 * r1  
str r1, [ r0 ]          ( *r0 ) ← r1  
bx lr
```

```
sub r0, fp, #16        r0 ← fp - 16  
bl sq                  call sq ( &a )  
sub r0, fp, #12        r0 ← fp - 12  
bl sq                  call sq ( &b )  
sub r0, fp, #8         r0 ← fp - 8  
bl sq                  call sq ( &c )  
sub r0, fp, #4         r0 ← fp - 4  
bl sq                  call sq ( &d )  
add r0, fp, #8         r0 ← fp + 8  
bl sq                  call sq ( &e )
```

```
ldr r0, [ fp, #-16 ]   r0 ← *( fp - 16 ) :a  
ldr r1, [ fp, #-12 ]   r1 ← *( fp - 12 ) :b  
add r0, r0, r1         r0 ← r0 + r1  
ldr r1, [ fp, #-8 ]    r1 ← *( fp - 8 ) :c  
add r0, r0, r1         r0 ← r0 + r1  
ldr r1, [ fp, #-4 ]    r1 ← *( fp - 4 ) :d  
add r0, r0, r1         r0 ← r0 + r1  
ldr r1, [ fp, #8 ]     r1 ← *( fp + 8 ) :e  
add r0, r0, r1         r0 ← r0 + r1
```

# Caller Function Code

```
.data
.align 4

message:
.asciz "Sum of 1^2 + 2^2 + 3^2 + 4^2 +
5^2 is %d\n"

.text

sq: <<defined above>>
sq_sum5: <<defined above>>

.globl main
main:

push { r4, lr }

pop { r4, lr }

bx lr
```

```
mov r0, #1      a ← 1
mov r1, #2      b ← 2
mov r2, #3      c ← 3
mov r3, #4      d ← 4

mov r4, #5      r4 ← 5

sub sp, sp, #8
str r4, [sp]    e ← 5

bl sq_sum5     sq_sum5 ( 1, 2, 3, 4, 5 )

add sp, sp, #8

mov r1, r0
ldr r0, address_of_message

bl printf

address_of_message: .word message
```

# sq

```
void sq(int *c) {  
    (*c) = (*c) * (*c);  
}
```

```
sq:  
ldr  r1, [r0]      ; r1 ← (*r0)      ; r0 : argument register  
mul  r1, r1, r1    ; r1 ← r1 * r1  
str  r1, [r0]     ; (*r0) ← r1  
bx   lr           ; return from the function
```

<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# sq\_sum5 (1)

```
int sq_sum5(int a, int b, int c, int d, int e) {  
    sq(&a);  
    sq(&b);  
    sq(&c);  
    sq(&d);  
    sq(&e);  
    return a + b + c + d + e;  
}
```

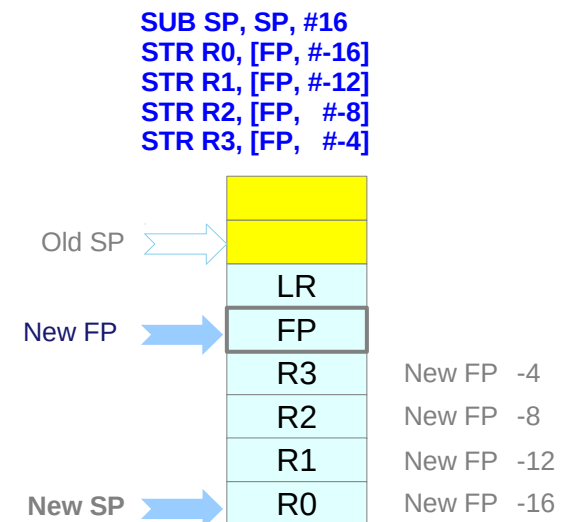
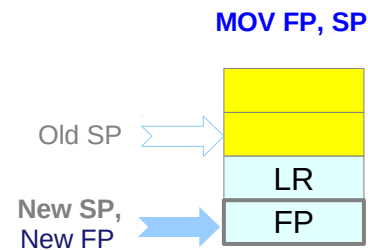
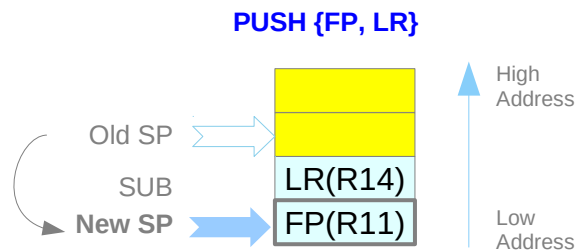
<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# sq\_sum5 (2)

sq\_sum5:

```
push {fp, lr}           ; keep fp and all callee-saved registers.
mov fp, sp              ; set the dynamic link
                        ; allocate space for 4 integers in the stack
                        ; keep parameters in the stack

sub sp, sp, #16         ; sp ← sp - 16.
str r0, [fp, #-16]     ; *(fp - 16) ← r0
str r1, [fp, #-12]     ; *(fp - 12) ← r1
str r2, [fp, #-8]      ; *(fp - 8) ← r2
str r3, [fp, #-4]      ; *(fp - 4) ← r3
```



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

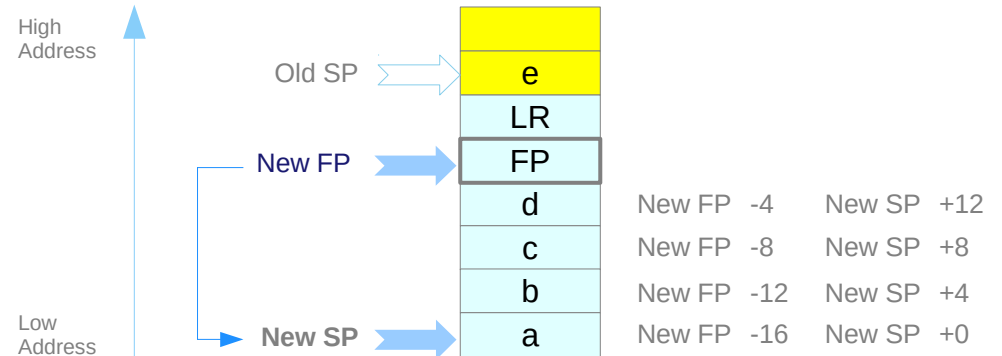


# sq\_sum5 (3)

Value	Address(es)
a	[fp, #-16] [sp]
b	[fp, #-12] [sp, #4]
c	[fp, #-8] [sp, #8]
d	[fp, #-4] [sp, #12]
fp(r11)	[fp] [sp, #16]
lr(r14)	[fp, #4] [sp, #20]
e	[fp, #8] [sp, #24]

High Address  
↓

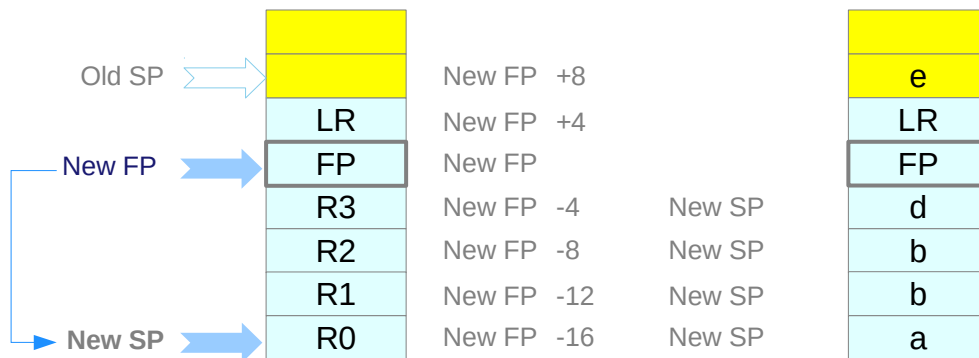
fp[-0] saved pc  
fp[-1] saved lr  
fp[-2] previous sp  
fp[-3] previous fp



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# sq\_sum5 (4)

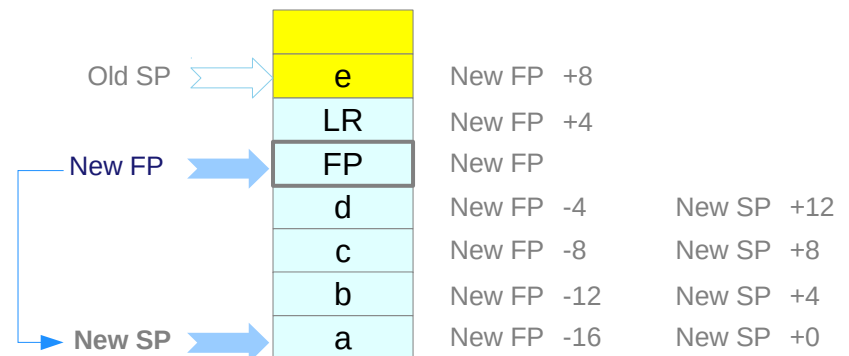
```
sub  r0, fp, #16      ; r0 ← fp - 16
bl   sq              ; call sq(&a);
sub  r0, fp, #12      ; r0 ← fp - 12
bl   sq              ; call sq(&b);
sub  r0, fp, #8       ; r0 ← fp - 8
bl   sq              ; call sq(&c);
sub  r0, fp, #4       ; r0 ← fp - 4
bl   sq              ; call sq(&d)
add  r0, fp, #8       ; r0 ← fp + 8
bl   sq              ; call sq(&e)
```



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# sq\_sum5 (5)

```
ldr  r0, [fp, #-16]    ; r0 ← *(fp - 16). ; Loads a into r0
ldr  r1, [fp, #-12]    ; r1 ← *(fp - 12). ; Loads b into r1
add  r0, r0, r1        ; r0 ← r0 + r1   ; (a + b)
ldr  r1, [fp, #-8]     ; r1 ← *(fp - 8).  ; Loads c into r1
add  r0, r0, r1        ; r0 ← r0 + r1   ; (a + b + c)
ldr  r1, [fp, #-4]     ; r1 ← *(fp - 4).  ; Loads d into r1
add  r0, r0, r1        ; r0 ← r0 + r1   ; (a + b + c + d)
ldr  r1, [fp, #8]      ; r1 ← *(fp + 8).  ; Loads e into r1
add  r0, r0, r1        ; r0 ← r0 + r1   ; (a + b + c + d + e)
```

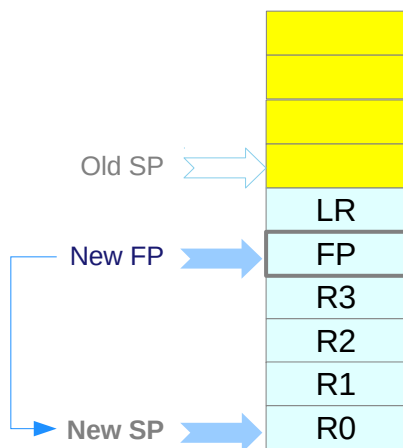


<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

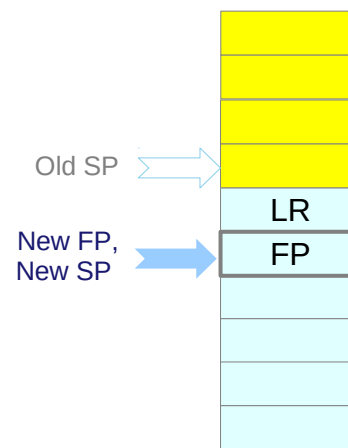
# sq\_sum5 (6)

```
mov sp, fp      ; Undo the dynamic link
pop {fp, lr}    ; Restore fp and callee-saved registers
bx lr          ; Return from the function
```

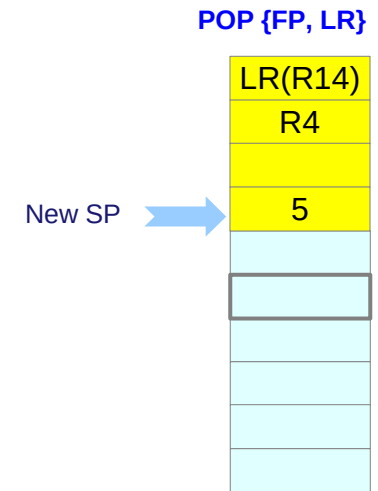
## sq\_sum5 frame



**MOV SP, FP**



## main frame



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# main (1)

```
/* squares.s */  
.data  
  
.align 4  
message: .asciz "Sum of 1^2 + 2^2 + 3^2 + 4^2 + 5^2 is %d\n"  
  
.text  
  
sq:  
    <<defined above>>  
  
sq_sum5:  
    <<defined above>>  
  
.globl main
```

<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# main (2)

main:

```
push {r4, lr}           ; Keep callee-saved registers
```

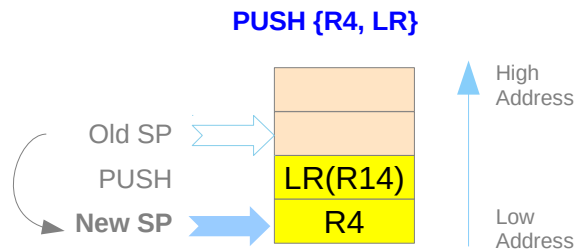
```
; Prepare the call to sq_sum5
```

```
mov r0, #1             ; Parameter r0 ← a=1
```

```
mov r1, #2             ; Parameter r1 ← b=2
```

```
mov r2, #3             ; Parameter r2 ← c=3
```

```
mov r3, #4             ; Parameter r3 ← d=4
```

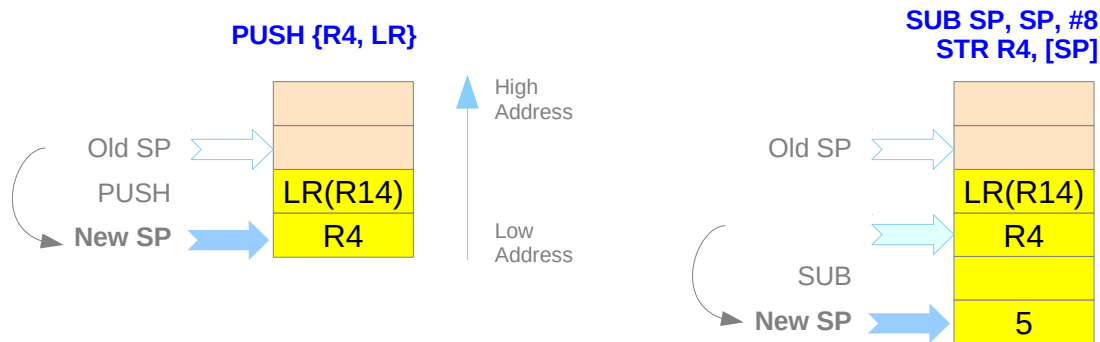


<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# main (3)

; Parameter e goes through the stack,  
; so it requires enlarging the stack

```
mov r4, #5           ; r4 ← 5  
sub sp, sp, #8      ; Enlarge the stack 8 bytes,  
                   ; we will use only the  
                   ; topmost 4 bytes  
str r4, [sp]        ; Parameter e ← 5  
bl sq_sum5          ; call sq_sum5(1, 2, 3, 4, 5)
```



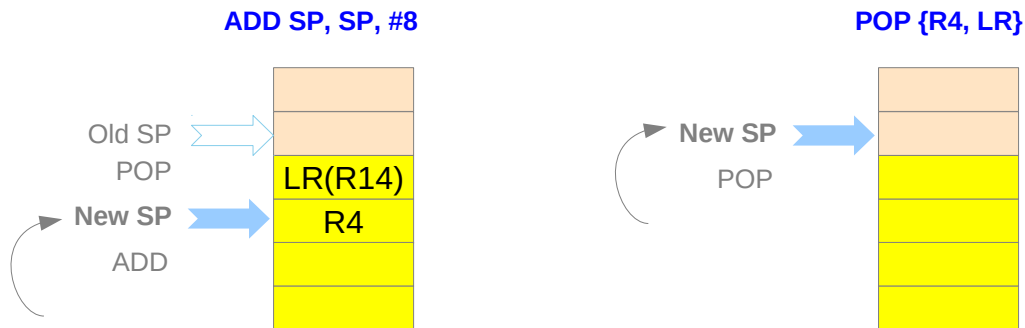
<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# main (4)

```
add sp, sp, #8      ; Shrink back the stack

; Prepare the call to printf
mov r1, r0          ; The result of sq_sum5
ldr r0, address_of_message
bl printf          ; Call printf

pop {r4, lr}       ; Restore callee-saved registers
bx lr
```



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>



# main (6)

```
address_of_message:    .word    message
```

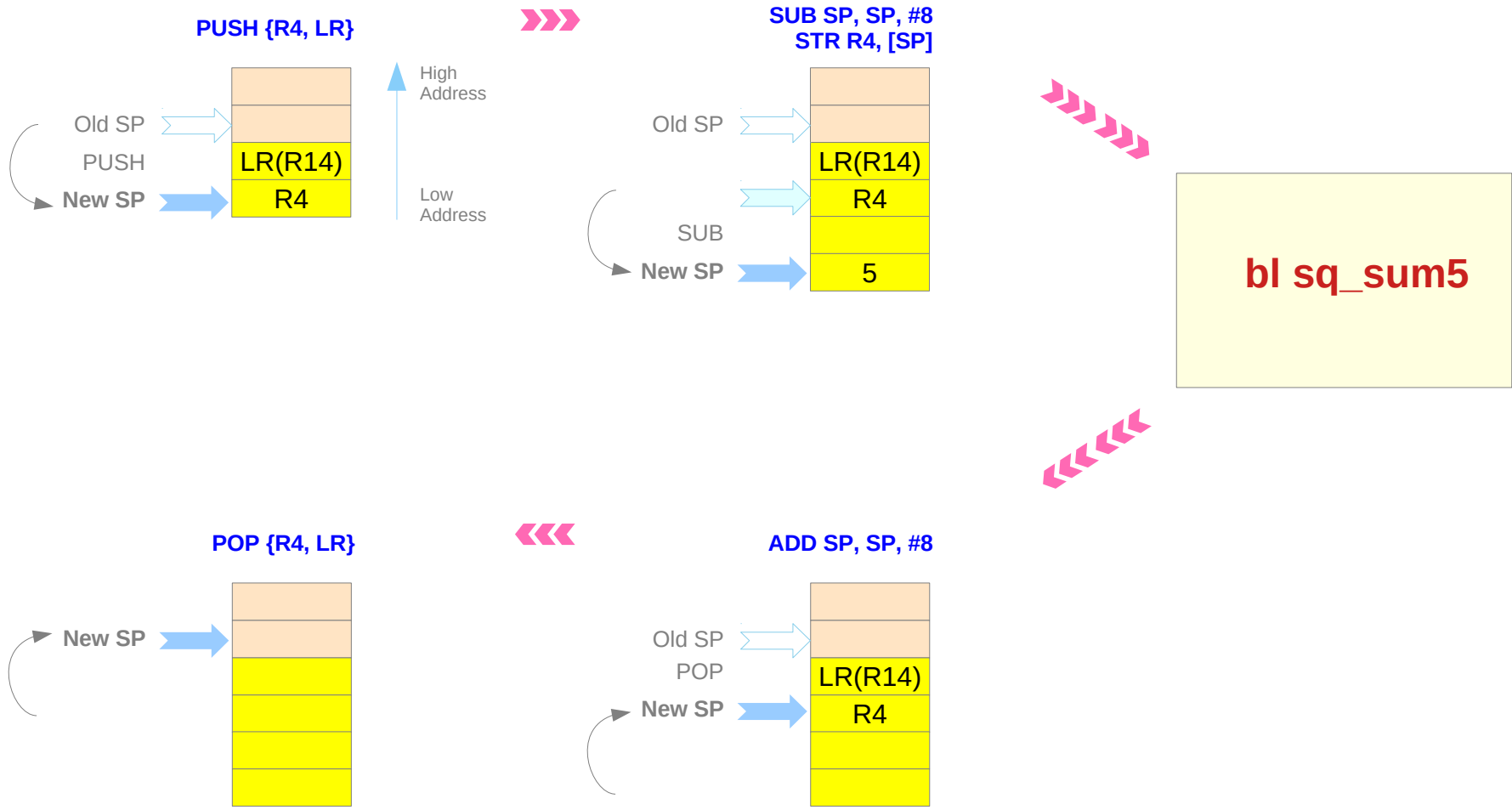
```
message:                .asciz  "Sum of 1^2 + 2^2 + 3^2 + 4^2 + 5^2 is %d\n"
```

```
$ ./square
```

```
Sum of 1^2 + 2^2 + 3^2 + 4^2 + 5^2 is 55
```

<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# main's stack frame



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# sq\_sum5's stack frame (1)

sq\_sum5:

```

push {fp, lr}
mov fp, sp
sub sp, sp, #16
str r0, [fp, #-16]
str r1, [fp, #-12]
str r2, [fp, #-8]
str r3, [fp, #-4]

```

```

sub r0, fp, #16
bl sq
sub r0, fp, #12
bl sq
sub r0, fp, #8
bl sq
sub r0, fp, #4
bl sq
add r0, fp, #8
bl sq

```

```

ldr r0, [fp, #-16]
ldr r1, [fp, #-12]
add r0, r0, r1
ldr r1, [fp, #-8]
add r0, r0, r1
ldr r1, [fp, #-4]
add r0, r0, r1
ldr r1, [fp, #8]
add r0, r0, r1

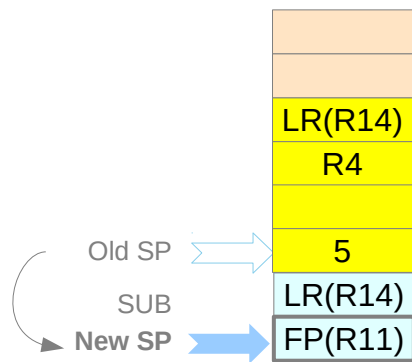
```

```

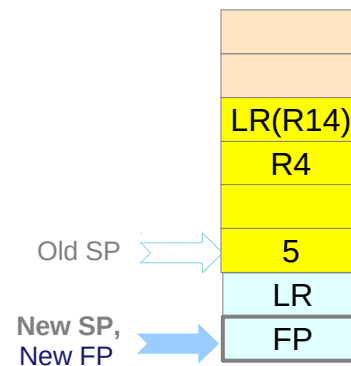
mov sp, fp
pop {fp, lr}
bx lr

```

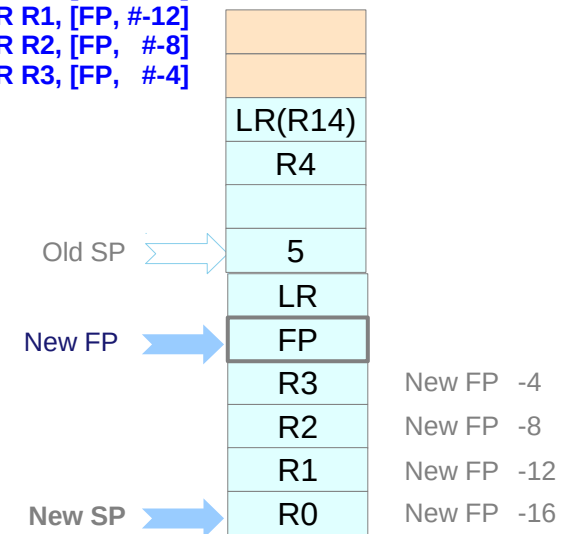
PUSH {FP, LR}



MOV FP, SP



SUB SP, SP, #16  
STR R0, [FP, #-16]  
STR R1, [FP, #-12]  
STR R2, [FP, #-8]  
STR R3, [FP, #-4]



<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# sq\_sum5's stack frame (2)

sq\_sum5:

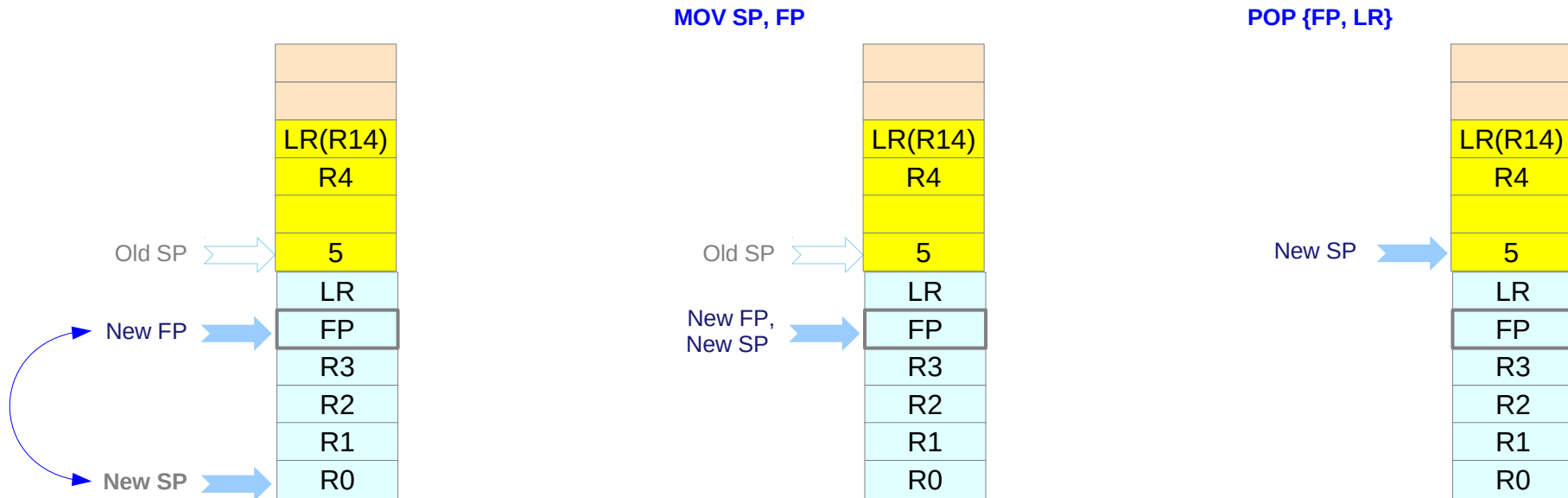
```
push {fp, lr}
mov fp, sp
```

```
sub sp, sp, #16
str r0, [fp, #-16]
str r1, [fp, #-12]
str r2, [fp, #-8]
str r3, [fp, #-4]
```

```
sub r0, fp, #16
bl sq
sub r0, fp, #12
bl sq
sub r0, fp, #8
bl sq
sub r0, fp, #4
bl sq
add r0, fp, #8
bl sq
```

```
ldr r0, [fp, #-16]
ldr r1, [fp, #-12]
add r0, r0, r1
ldr r1, [fp, #-8]
add r0, r0, r1
ldr r1, [fp, #-4]
add r0, r0, r1
ldr r1, [fp, #8]
add r0, r0, r1
```

```
mov sp, fp
pop {fp, lr}
bx lr
```

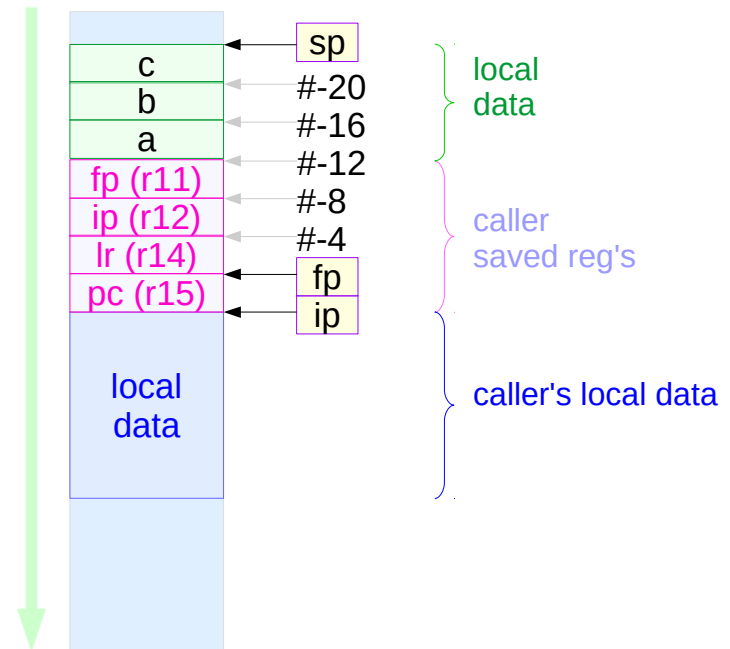


<https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/>

# -fno-omit-frame-pointer

```
main:
mov     ip, sp
stmfd  sp!, { fp, ip, lr, pc }
sub    fp, ip, #4
sub    sp, sp, #12
ldr    r2, [fp, #-16]
ldr    r3, [fp, #-20]
add    r3, r3, r2
str    r3, [fp, #-24]
sub    sp, fp, #12
ldmfd  sp, {fp, sp, pc}
```

```
main()
{
volatile int a, b, c;
c = a + b;
}
```

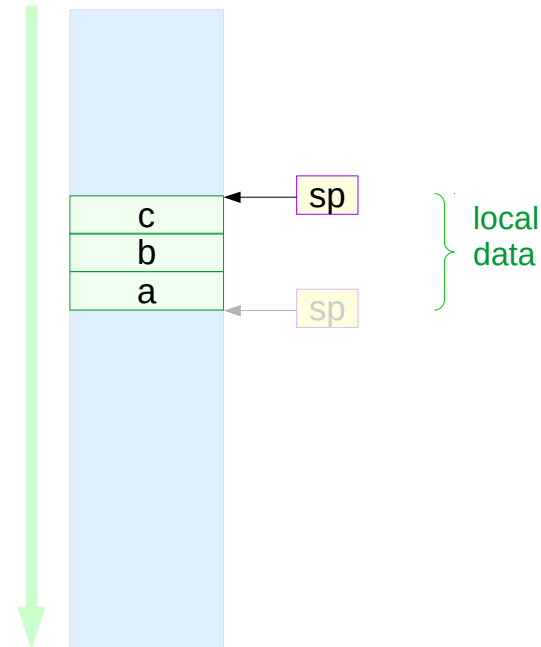


<https://community.arm.com/thread/7092>

# -fomit-frame-pointer

```
main:  
sub    sp, sp, #12  
ldr    r2, [sp, #8]  
ldr    r3, [fp, #4]  
add    r3, r3, r2  
str    r3, [sp, #0]  
sub    sp, sp, #12
```

```
main()  
{  
    volatile int a, b, c;  
    c = a + b;  
}
```



<https://community.arm.com/thread/7092>

---

## References

- [1] [http://wiki.osdev.org/ARM\\_RaspberryPi\\_Tutorial\\_C](http://wiki.osdev.org/ARM_RaspberryPi_Tutorial_C)
- [2] <http://blog.bobuhiro11.net/2014/01-13-baremetal.html>
- [3] <http://www.valvers.com/open-software/raspberry-pi/>
- [4] <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html>