Stack Frames (12A)

Young Won Lim
2/26/21

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

ARM System-on-Chip Architecture, 2nd ed, Steve Furber

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

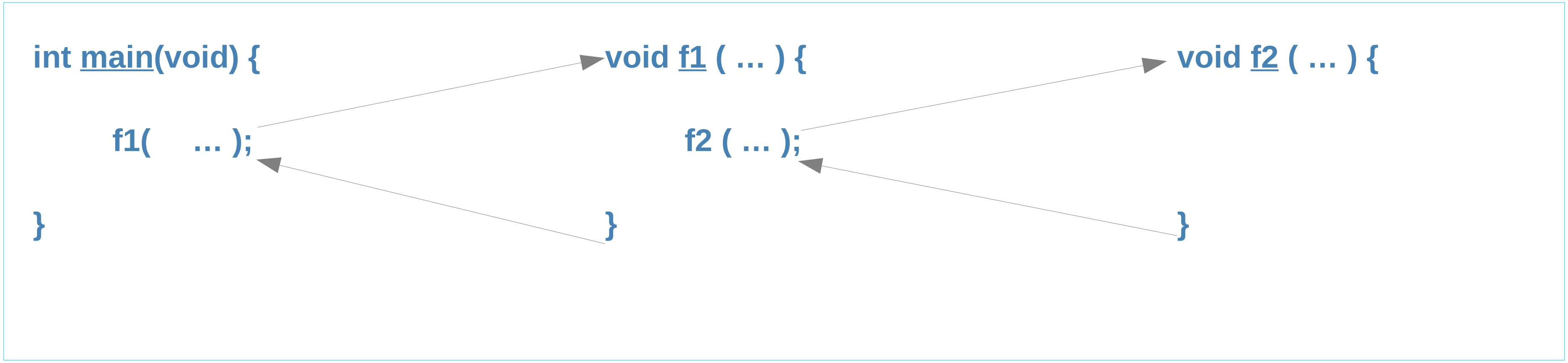
Digital Design and Computer Architecture,
D. M. Harris and S. L. Harris

ARM assembler in Raspberry Pi
Roger Ferrer Ibáñez

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

Nested and recursive function calls

Nested function call

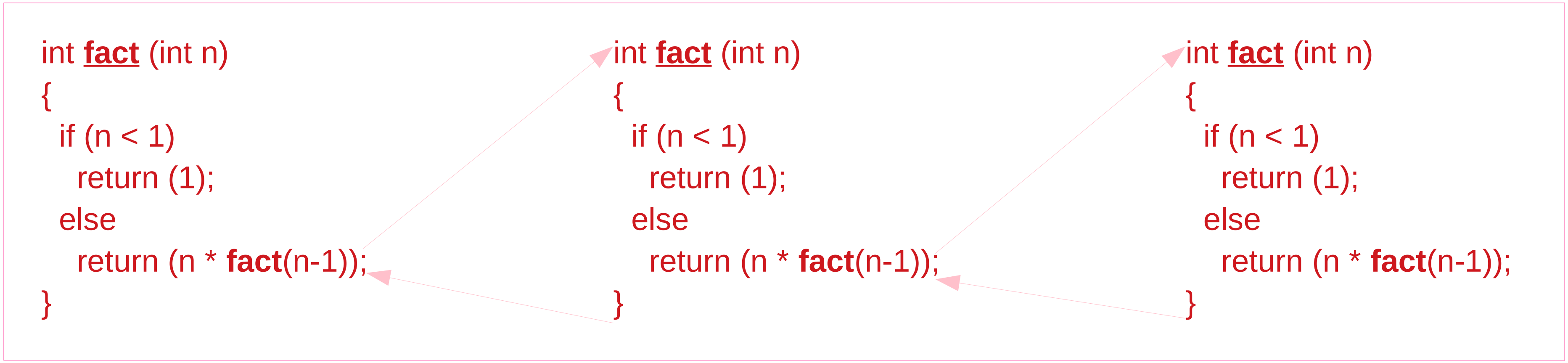


```
int main(void) {

        f1(    … );

}
```

```
void f1 ( … ) {

        f2 ( … );

}
```

```
void f2 ( … ) {

}
```

Recursive function call



```
int fact (int n)
{
  if (n < 1)
    return (1);
  else
    return (n * fact(n-1));
}
```

```
int fact (int n)
{
  if (n < 1)
    return (1);
  else
    return (n * fact(n-1));
}
```

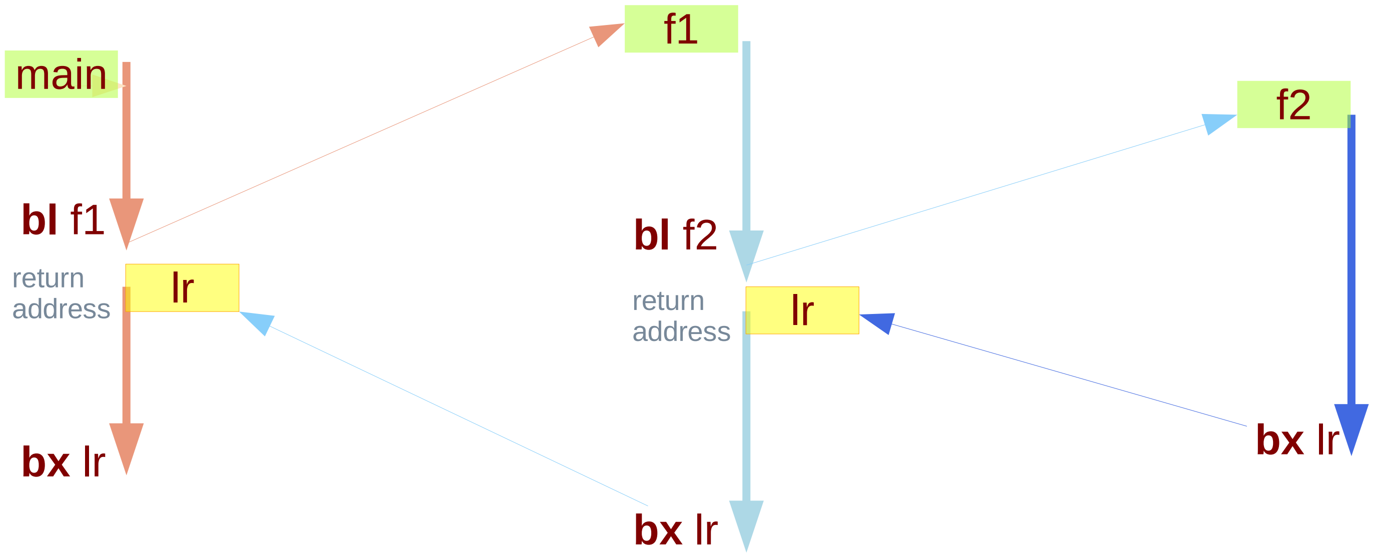
```
int fact (int n)
{
  if (n < 1)
    return (1);
  else
    return (n * fact(n-1));
}
```

Nested and recursive function calls

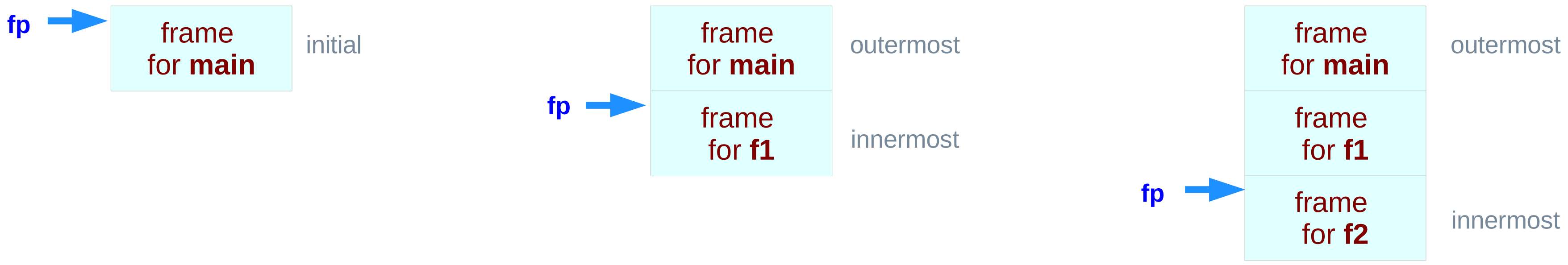
At least, **LR** must not be overwritten

save the followings in a **frame**

- return address
- arguments
- local variables.



Nested and recursive function calls



fp

frame for **main**

initial

frame for **main**

outermost

fp

frame for **f1**

innermost

frame for **main**

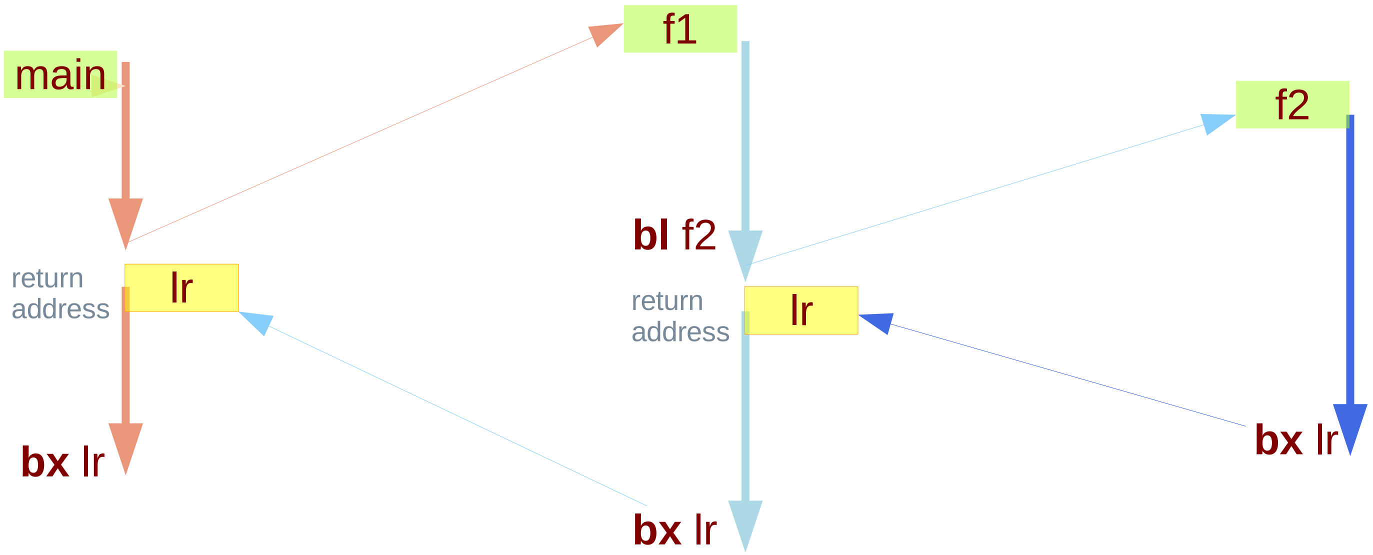
outermost

frame for **f1**

fp

frame for **f2**

innermost



main

return address

lr

bx lr

f1

bl f2

return address

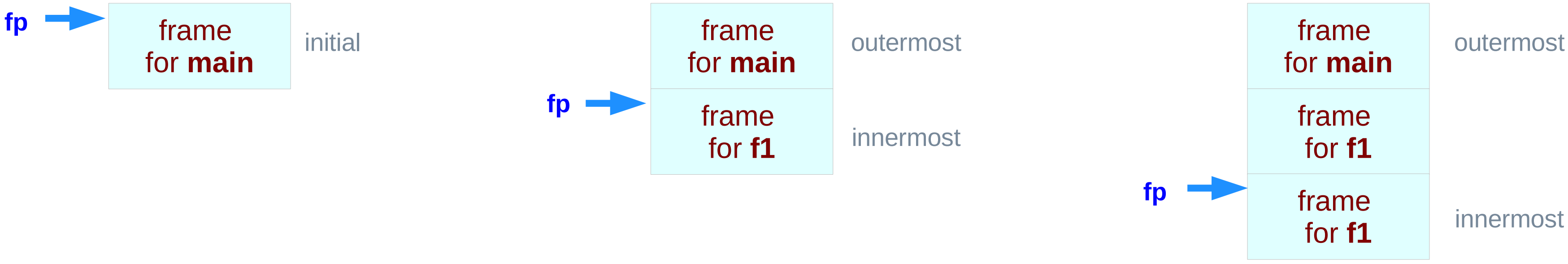
lr

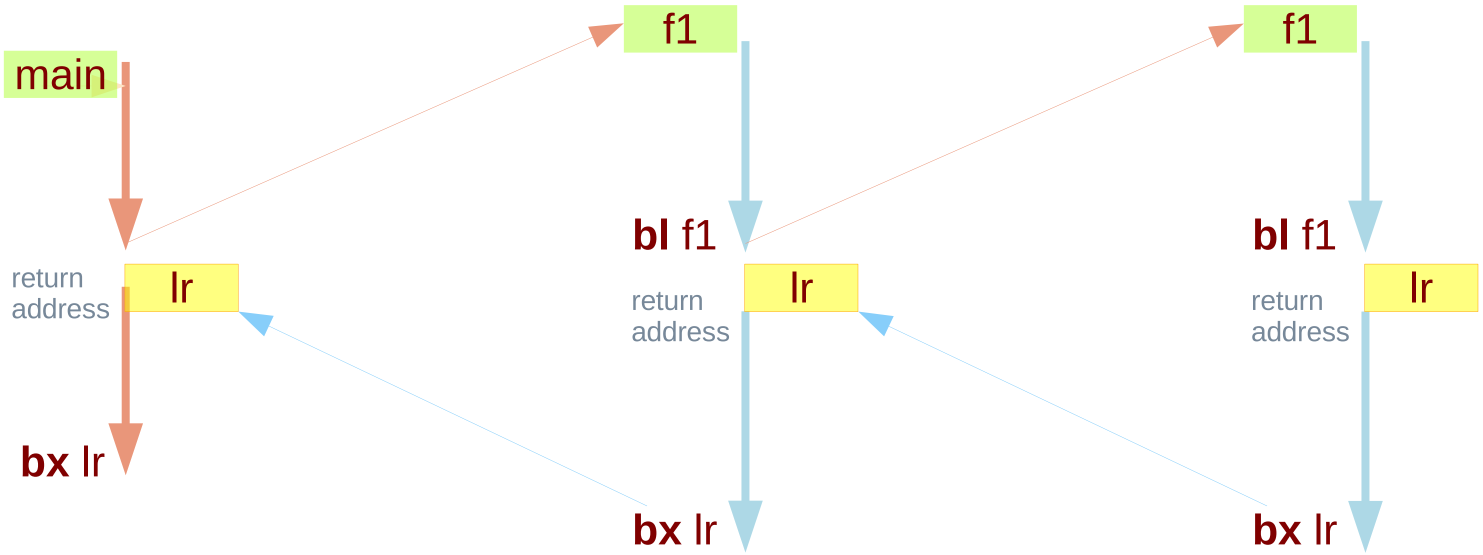
bx lr

f2

bx lr

Recursive function calls





Stack frames (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

Stack frames (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

Stack frames (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

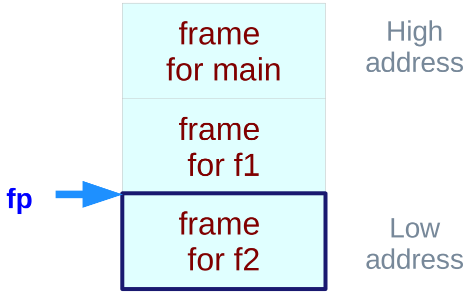
Stack frames (4)

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.

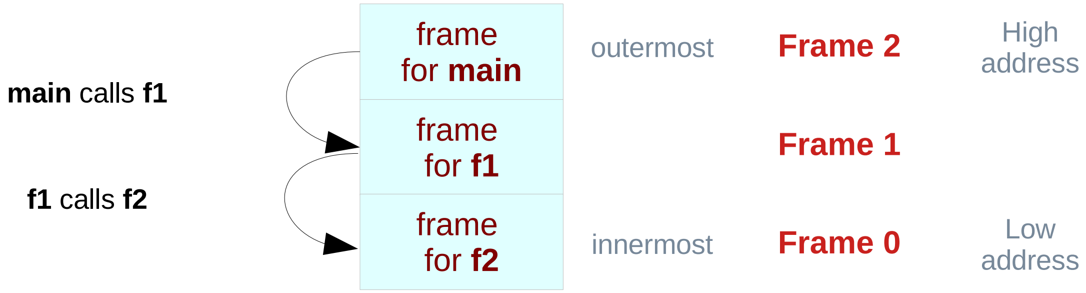


http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.prog%2Ftopic%2Fusing_gdb_StackFrames.html

Stack frames (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

Stack frames (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

Stack frames (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

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**

Computer Organization and Design ARM Edition: The Hardware Software Interface by D. A. Patterson and J. L. Hennessy

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

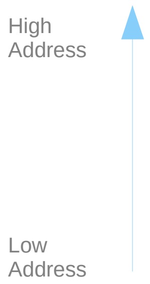
Computer Organization and Design ARM Edition: The Hardware Software Interface by D. A. Patterson and J. L. Hennessy

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



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

Frame pointer and stack pointer registers (4)

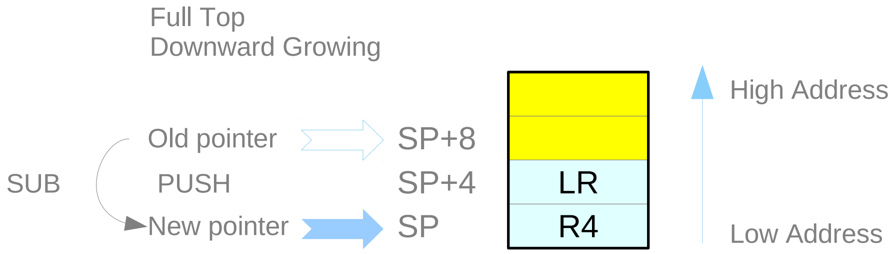
The basic frame layout is,

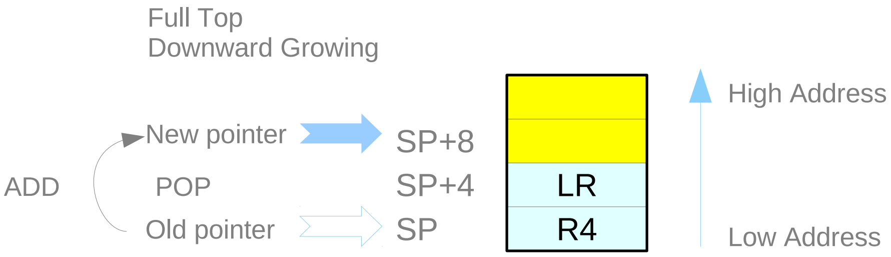
fp[-0] saved pc, where we stored this frame.
fp[-1] saved lr, the return address for this function.
fp[-2] previous sp, before this function eats stack.
fp[-3] previous fp, the last stack frame.
many optional registers...

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

Stack frame skeleton (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
```





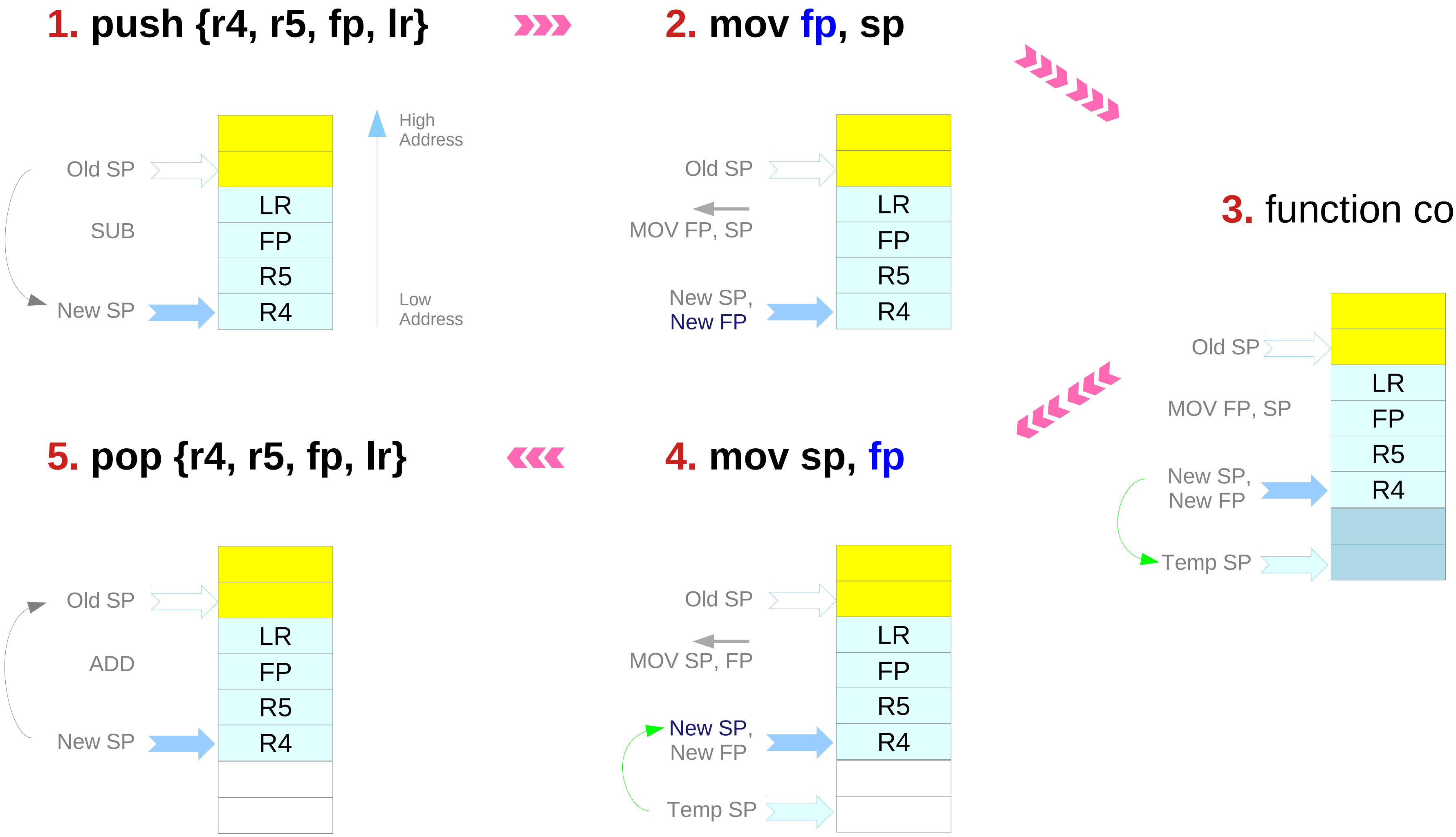
https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/

Stack frame skeleton (2)

```
function:                      ; keep callee-saved registers
                               ; keep the callee saved registers.
                               ; we added r5 to keep the stack 8-byte aligned
    push {r4, r5, fp, lr}      ; but the important thing here is fp
    mov fp, sp                 ; fp ← sp. Keep dynamic link in fp
    ...                        ; code of the function
    mov sp, fp                 ; sp ← fp. Restore dynamic link in fp
    pop {r4, r5, fp, lr}       ; restore the callee saved registers.
                               ; this will restore fp as well
    bx lr                      ; return from the function
```

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

Stack frame skeleton (3)



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

Old SP
SUB
New SP
LR
FP
R5
R4
High Address
Low Address

2. mov fp, sp

Old SP
MOV FP, SP
New SP,
New FP
LR
FP
R5
R4

3. function code

Old SP
MOV FP, SP
New SP,
New FP
Temp SP
LR
FP
R5
R4

4. mov sp, fp

Old SP
MOV SP, FP
New SP,
New FP
Temp SP
LR
FP
R5
R4

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

Old SP
ADD
New SP
LR
FP
R5
R4

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

Stack frame skeleton (4)

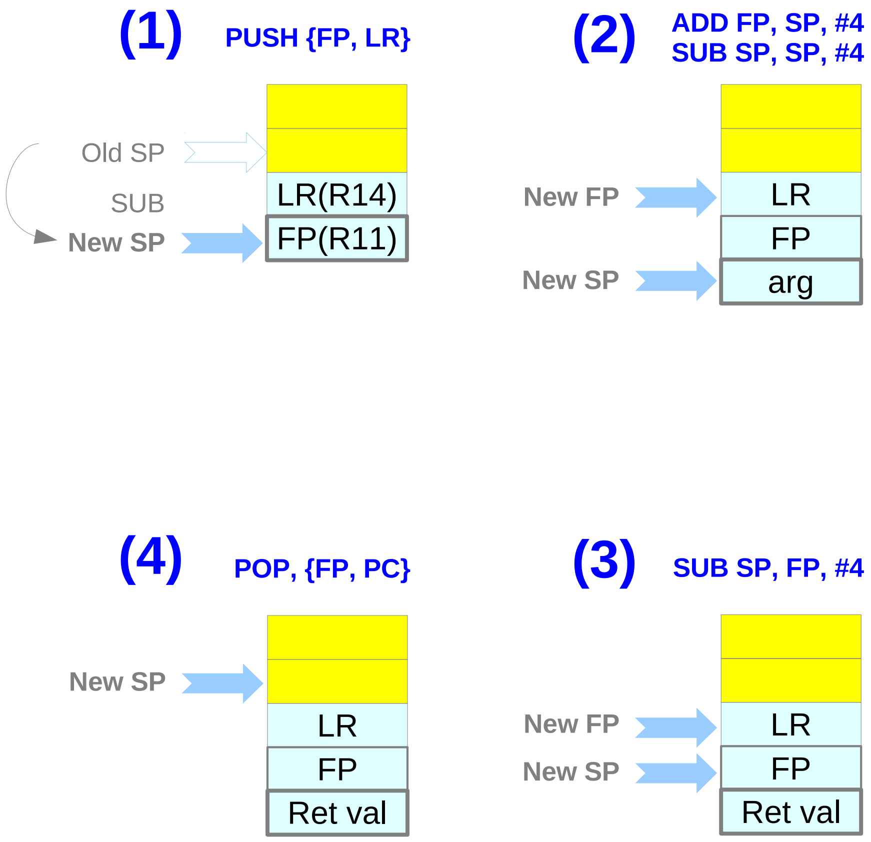
```
function:                     ; keep callee-saved registers
                              ; keep the callee saved registers.
                              ; w added r5 to keep the stack 8-byte aligned
      push{r4, r5, fp, lr}    ; but the important thing here is fp
      mov fp, sp              ; fp ← sp. Keep dynamic link in fp
      sub sp, sp, #8          ; enlarge the stack by 8 bytes
        ...                   ; code of the function
      mov sp, fp              ; sp ← fp. restore dynamic link in fp
      pop {r4, r5, fp, lr}    ; restore the callee saved registers.
                              ; this will restore fp as well
      bx lr                   ; return from the function
```

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

Stack frame example **A**

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

```
0x00010414 <+0>      push    {fp, lr}
0x00010418 <+4>      add     fp, sp, #4
0x0001041c <+8>      sub     sp, sp, #4
0x00010420 <+12>     add     r3, r0, r1
0x00010424 <+16>     str     r3, [fp-#8]
0x00010428 <+20>     bl      some_func
0x0001042c <+24>     str     r0, [fp-#8]
0x00010430 <+28>     sub     sp, fp, #4
0x00010434 <+32>     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(void)
{
  int ia, ib, ic;

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

  return ic;
}
```

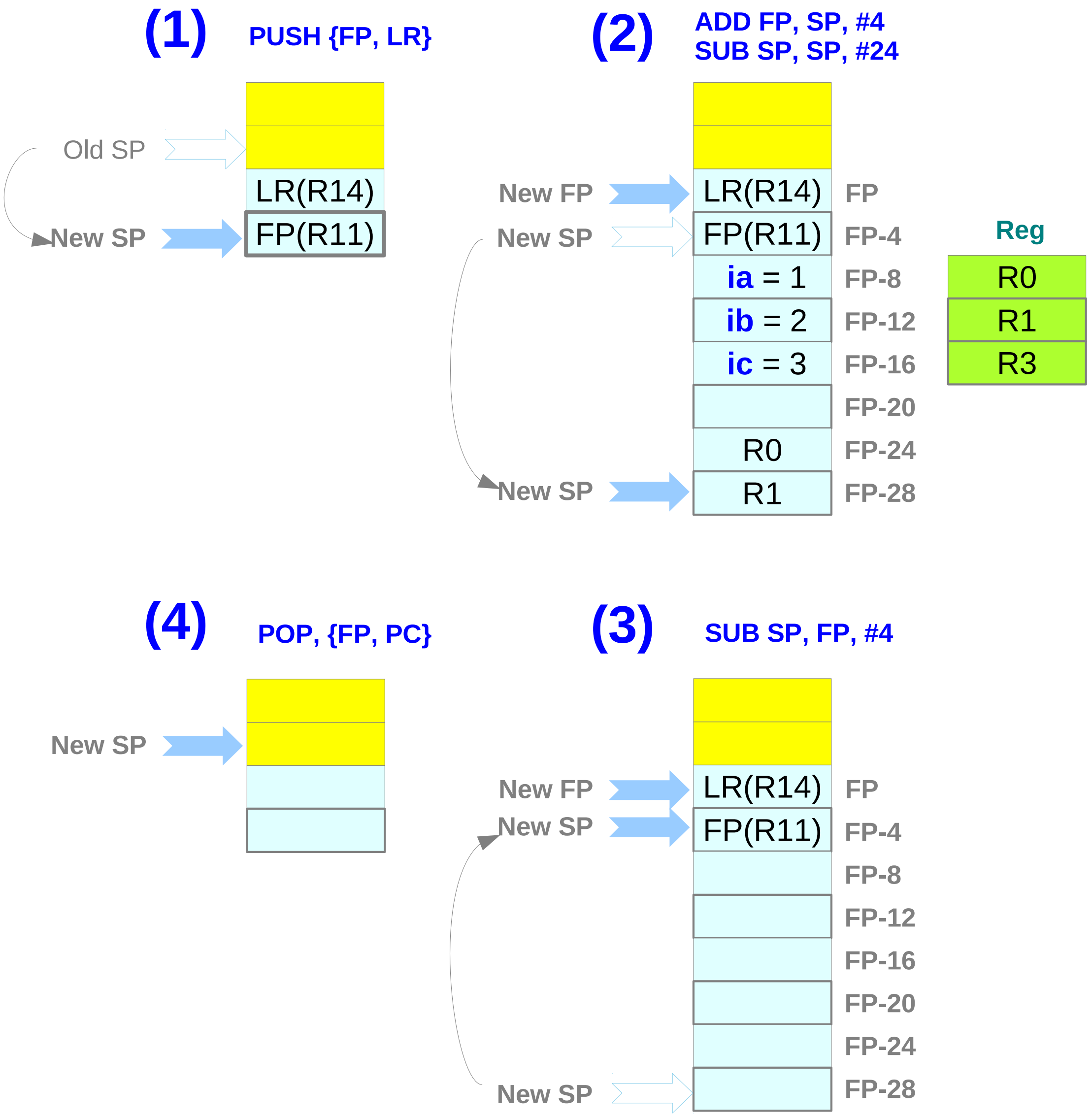
```
Int one(int a, int b)
{
  int c;
  c = two(a,b);
  return c;
}

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

Stack frame for **main**

```
push    {r11, lr}
add     r11, sp, #4
sub     sp, sp, #24
str     r0, [r11, #-24]      Received arg0
str     r1, [r11, #-28]      Received arg1
mov     r3, #1
str     r3, [r11, #-8]       Local var ia
mov     r3, #2
str     r3, [r11, #-12]      Local var ib
ldr     r1, [r11, #-12]      Arg0 for one
ldr     r0, [r11, #-8]       Arg1 for one
bl      <one>
str     r0, [r11, #-16]      Local var ic
ldr     r3, [r11, #-16]
mov     r0, r3               Return value
sub     sp, r11, #4
pop     {r11, pc}
```

```
int main(void) {
  int ia, ib, ic;
  ia = 1;  ib = 2;
  ic = one(ia, ib);
  return ic;       }
```

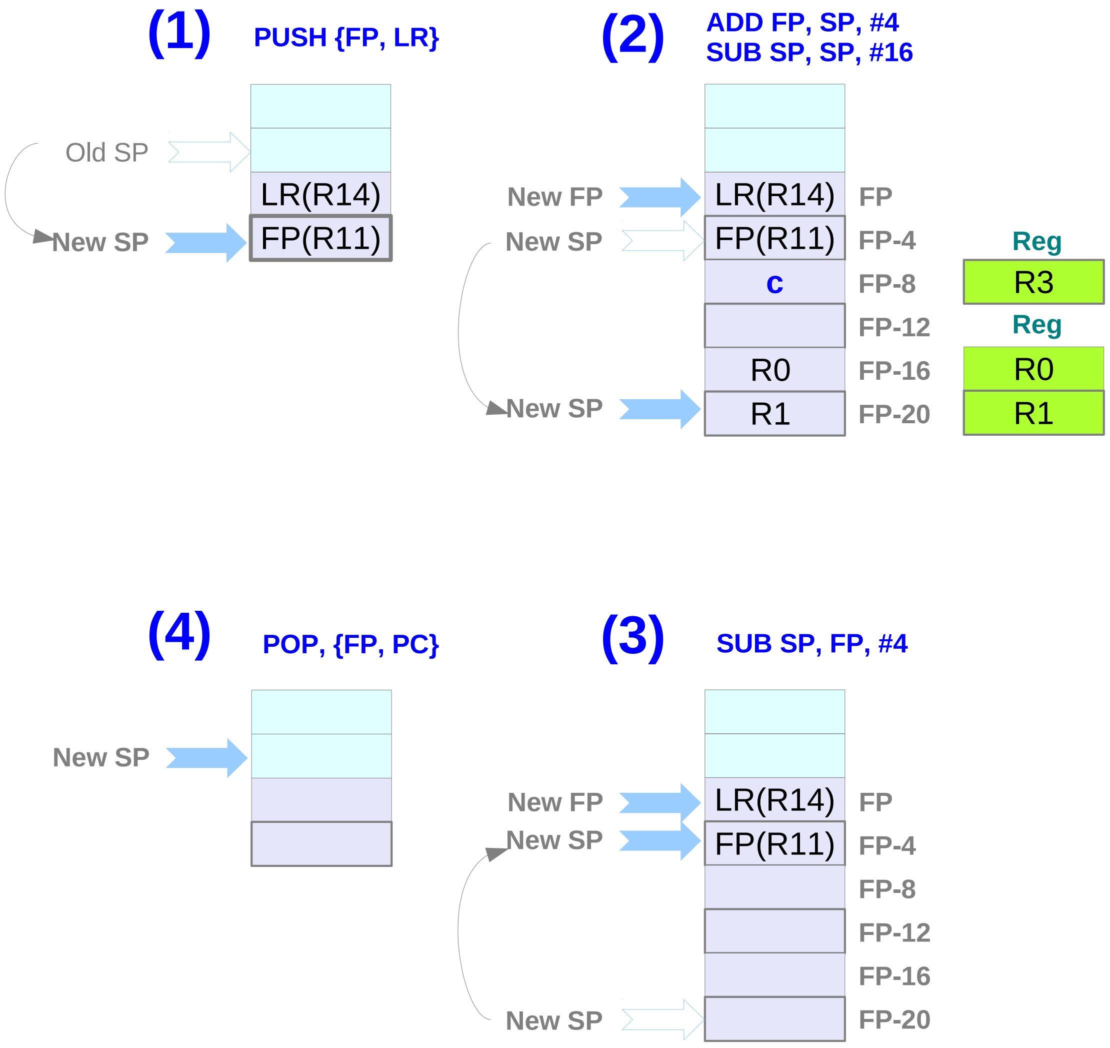


https://lloydrochester.com/post/c/stack-of-frames-arm/

Stack frame for **one**

```
push    {r11, lr}
add     r11, sp, #4
sub     sp, sp, #16
str     r0, [r11, #-16]     Received arg0
str     r1, [r11, #-20]     Received arg1
ldr     r1, [r11, #-20]     Arg0 for two
ldr     r0, [r11, #-16]     Arg1 for two
bl      <two>
str     r0, [r11, #-8]      Local var c
ldr     r3, [r11, #-8]
mov     r0, r3              Return value
sub     sp, r11, #4
pop     {r11, pc}
```

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

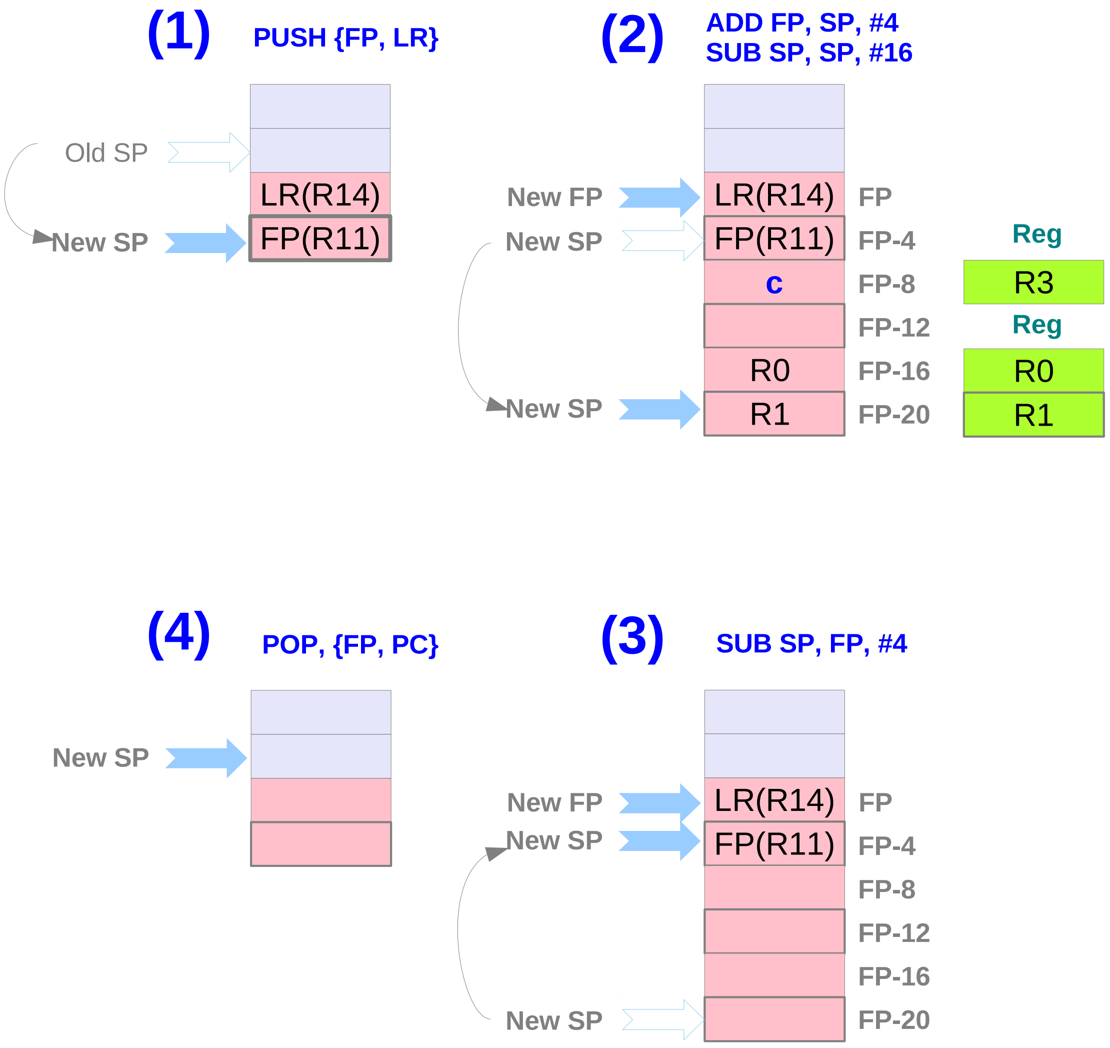


https://lloydrochester.com/post/c/stack-of-frames-arm/

Stack frame for **two**

```
push    {r11, lr}
add     r11, sp, #4
sub     sp, sp, #16
str     r0, [r11, #-16]      Received arg0
str     r1, [r11, #-20]      Received arg1
ldr     r1, [r11, #-20]      Arg0 for three
ldr     r0, [r11, #-16]      Arg1 for three
bl      <three>
str     r0, [r11, #-8]       Local var c
ldr     r3, [r11, #-8]
mov     r0, r3               Return value
sub     sp, r11, #4
pop     {r11, pc}
```

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



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

Stack frame for **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
```

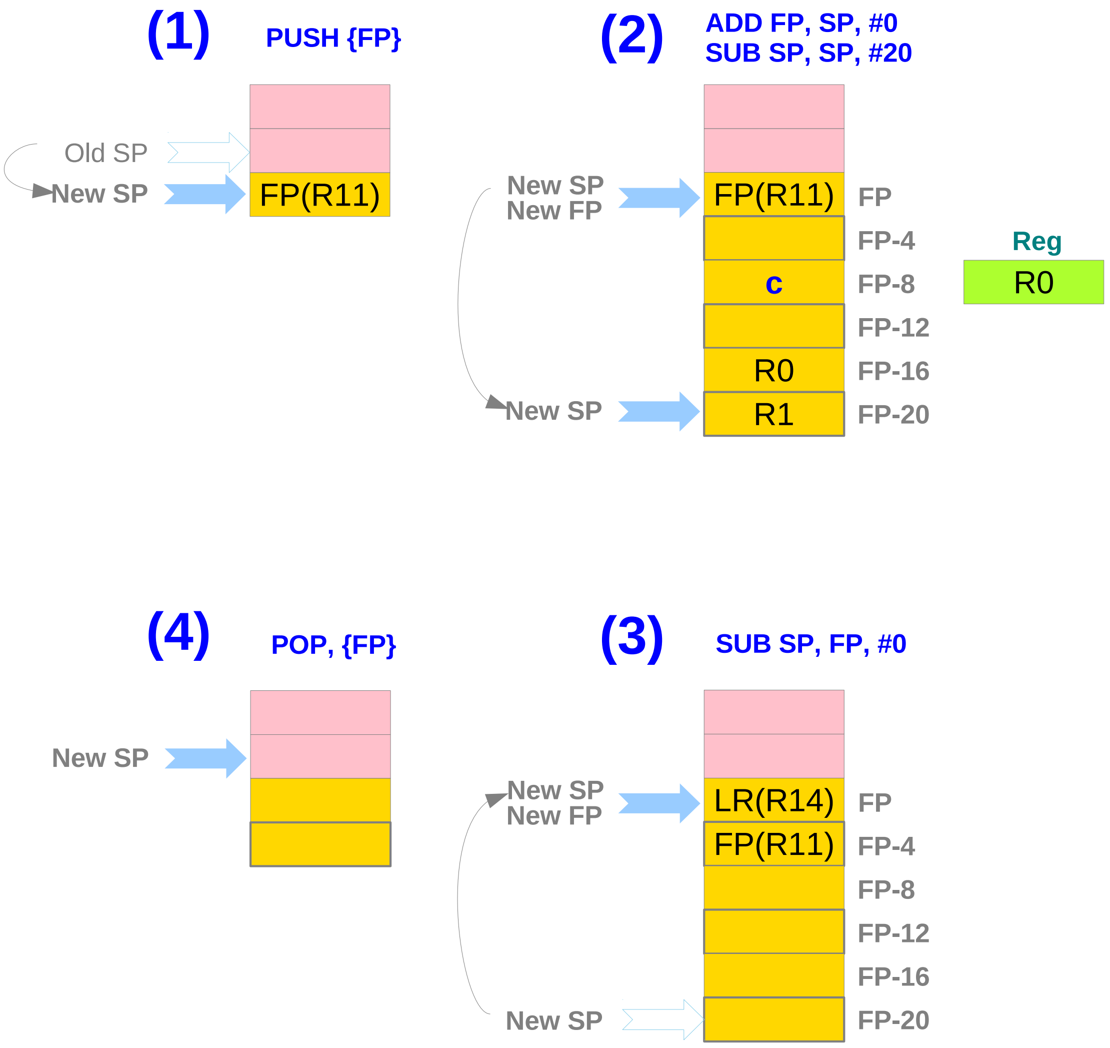
Received arg0

Received arg1

Local var **c**

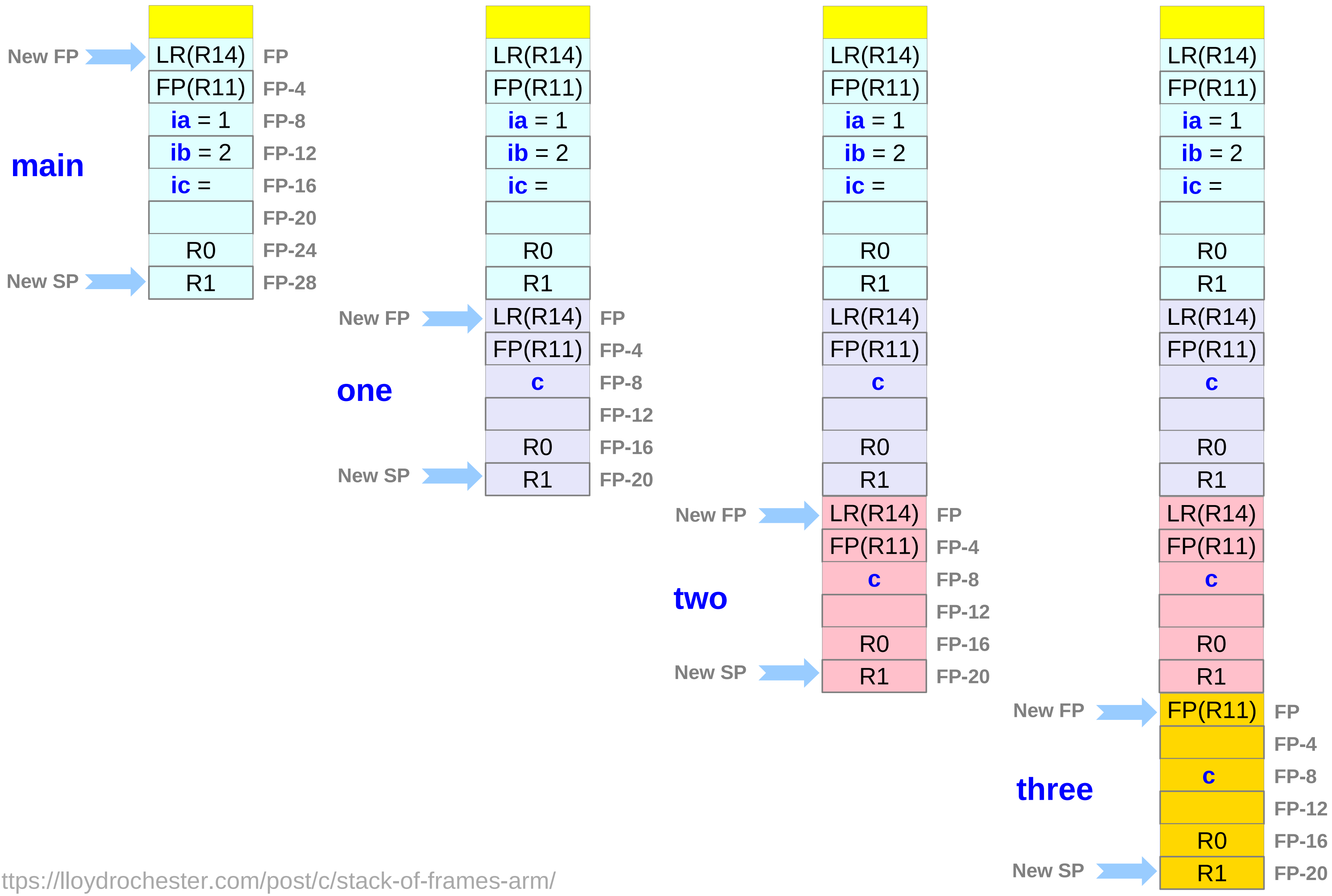
Return value

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



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

Stack frame snapshots



New FP	LR(R14)	FP	main
	FP(R11)	FP-4	
	ia = 1	FP-8	
	ib = 2	FP-12	
	ic =	FP-16	
		FP-20	
	R0	FP-24	
New SP	R1	FP-28	

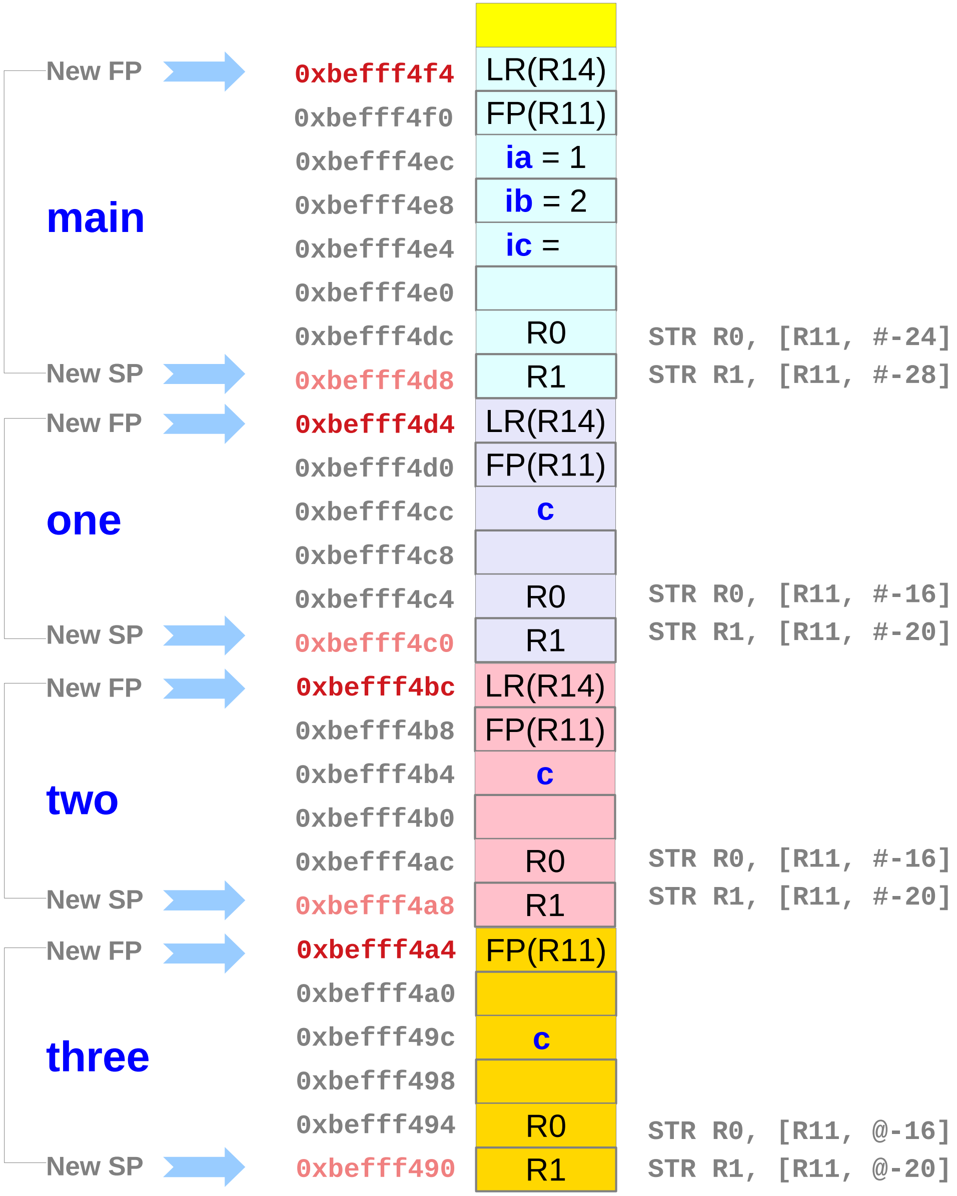
	LR(R14)		main
	FP(R11)		
	ia = 1		
	ib = 2		
	ic =		
	R0		
	R1		
New FP	LR(R14)	FP	one
	FP(R11)	FP-4	
	c	FP-8	
		FP-12	
	R0	FP-16	
New SP	R1	FP-20	

	LR(R14)		main
	FP(R11)		
	ia = 1		
	ib = 2		
	ic =		
	R0		
	R1		
	LR(R14)		one
	FP(R11)		
	c		
	R0		
	R1		
New FP	LR(R14)	FP	two
	FP(R11)	FP-4	
	c	FP-8	
		FP-12	
	R0	FP-16	
New SP	R1	FP-20	

	LR(R14)		main
	FP(R11)		
	ia = 1		
	ib = 2		
	ic =		
	R0		
	R1		
	LR(R14)		one
	FP(R11)		
	c		
	R0		
	R1		
	LR(R14)		two
	FP(R11)		
	c		
	R0		
	R1		
New FP	FP(R11)	FP	three
		FP-4	
	c	FP-8	
		FP-12	
	R0	FP-16	
New SP	R1	FP-20	

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

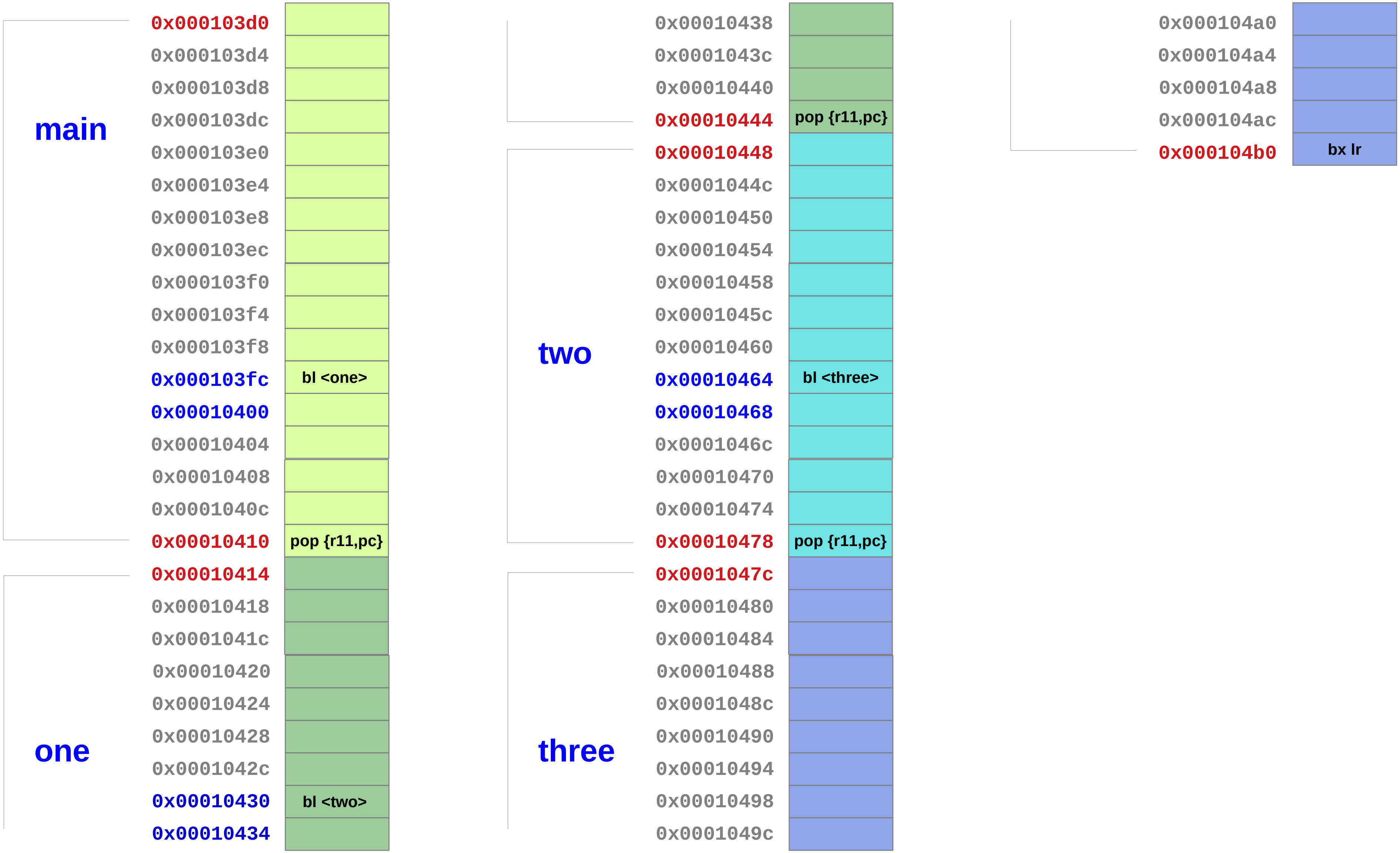
Stack frame memory map



New FP	0xbefff4f4	LR(R14)	
	0xbefff4f0	FP(R11)	
	0xbefff4ec	ia = 1	
main	0xbefff4e8	ib = 2	
	0xbefff4e4	ic =	
	0xbefff4e0		
	0xbefff4dc	R0	STR R0, [R11, #-24]
New SP	0xbefff4d8	R1	STR R1, [R11, #-28]
New FP	0xbefff4d4	LR(R14)	
	0xbefff4d0	FP(R11)	
one	0xbefff4cc	c	
	0xbefff4c8		
	0xbefff4c4	R0	STR R0, [R11, #-16]
New SP	0xbefff4c0	R1	STR R1, [R11, #-20]
New FP	0xbefff4bc	LR(R14)	
	0xbefff4b8	FP(R11)	
	0xbefff4b4	c	
two	0xbefff4b0		
	0xbefff4ac	R0	STR R0, [R11, #-16]
New SP	0xbefff4a8	R1	STR R1, [R11, #-20]
New FP	0xbefff4a4	FP(R11)	
	0xbefff4a0		
three	0xbefff49c	c	
	0xbefff498		
	0xbefff494	R0	STR R0, [R11, @-16]
New SP	0xbefff490	R1	STR R1, [R11, @-20]

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

Text area memory map



main

0x000103d0	
0x000103d4	
0x000103d8	
0x000103dc	
0x000103e0	
0x000103e4	
0x000103e8	
0x000103ec	
0x000103f0	
0x000103f4	
0x000103f8	
0x000103fc	bl <one>
0x00010400	
0x00010404	
0x00010408	
0x0001040c	
0x00010410	pop {r11,pc}

one

0x00010414	
0x00010418	
0x0001041c	
0x00010420	
0x00010424	
0x00010428	
0x0001042c	
0x00010430	bl <two>
0x00010434	
0x00010438	
0x0001043c	
0x00010440	
0x00010444	pop {r11,pc}

two

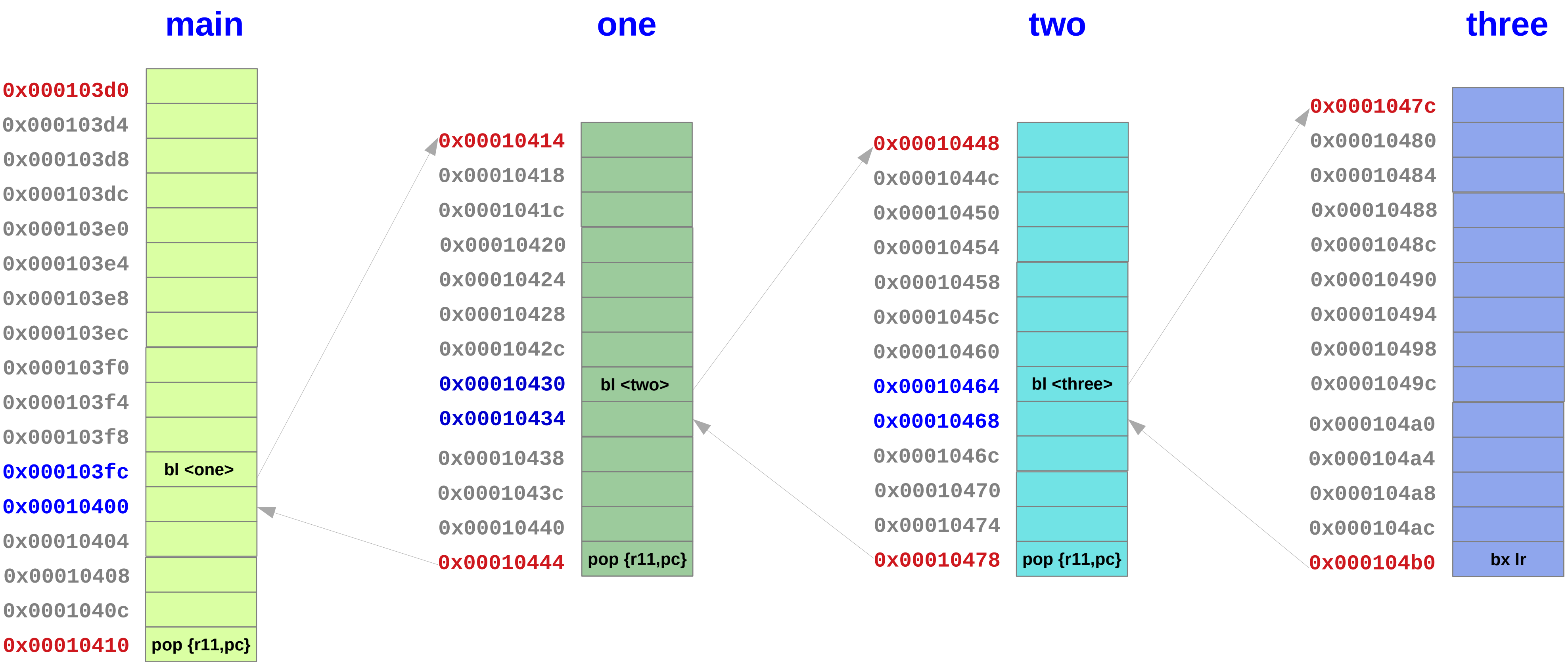
0x00010448	
0x0001044c	
0x00010450	
0x00010454	
0x00010458	
0x0001045c	
0x00010460	
0x00010464	bl <three>
0x00010468	
0x0001046c	
0x00010470	
0x00010474	
0x00010478	pop {r11,pc}

three

0x0001047c	
0x00010480	
0x00010484	
0x00010488	
0x0001048c	
0x00010490	
0x00010494	
0x00010498	
0x0001049c	
0x000104a0	
0x000104a4	
0x000104a8	
0x000104ac	
0x000104b0	bx lr

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

Nested procedure calls



	main
0x000103d0	
0x000103d4	
0x000103d8	
0x000103dc	
0x000103e0	
0x000103e4	
0x000103e8	
0x000103ec	
0x000103f0	
0x000103f4	
0x000103f8	
0x000103fc	bl <one>
0x00010400	
0x00010404	
0x00010408	
0x0001040c	
0x00010410	pop {r11,pc}

	one
0x00010414	
0x00010418	
0x0001041c	
0x00010420	
0x00010424	
0x00010428	
0x0001042c	
0x00010430	bl <two>
0x00010434	
0x00010438	
0x0001043c	
0x00010440	
0x00010444	pop {r11,pc}

	two
0x00010448	
0x0001044c	
0x00010450	
0x00010454	
0x00010458	
0x0001045c	
0x00010460	
0x00010464	bl <three>
0x00010468	
0x0001046c	
0x00010470	
0x00010474	
0x00010478	pop {r11,pc}

	three
0x0001047c	
0x00010480	
0x00010484	
0x00010488	
0x0001048c	
0x00010490	
0x00010494	
0x00010498	
0x0001049c	
0x000104a0	
0x000104a4	
0x000104a8	
0x000104ac	
0x000104b0	bx lr

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

Disassembly of **main**

```
(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=0xbefff4f4
   0x000103d8 <+8>:    sub    sp, sp, #24         ; sp=0xbefff4d8, frame is size 28=24+4
   0x000103dc <+12>:   str    r0, [r11, #-24]     ; 0xbefff4dc
   0x000103e0 <+16>:   str    r1, [r11, #-28]     ; 0xbefff4d8
   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>:   bl     0x10414 <one>       ; 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/

Disassembly of **one**

```
(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]   ; 0xbefff4c4
   0x00010424 <+16>:   str    r1, [r11, #-20]   ; 0xbefff4c0
   0x00010428 <+20>:   ldr    r1, [r11, #-20]
   0x0001042c <+24>:   ldr    r0, [r11, #-16]
   0x00010430 <+28>:   bl     0x10448 <two>     ; 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/

Disassembly of **two**

```
(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]   ; 0xbefff4ac
   0x00010458 <+16>:   str     r1, [r11, #-20]   ; 0xbefff4a8
   0x0001045c <+20>:   ldr     r1, [r11, #-20]
   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/

Disassembly of **three**

```
(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=0xbefff490
   0x00010488 <+12>:   str     r0, [r11, #-16]    ; 0xbefff494
   0x0001048c <+16>:   str     r1, [r11, #-20]    ; 0xbefff490
   0x00010490 <+20>:   ldr     r2, [r11, #-16]
   0x00010494 <+24>:   ldr     r3, [r11, #-20]
   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=0x00010468
End of assembler dump.
```

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

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;
}
```

callee
function

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

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

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)

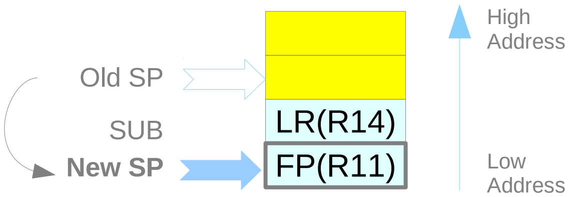
```
int sq_sum5(int a, int b, int c, int d, int e) {
        sq(&a);
        sq(&b);
        sq(&c);
        sq(&d);
        sq(&e);
        return a + b + c + d + e;
}
```

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

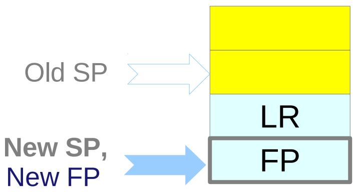
sq_sum5 (2)

```
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
```

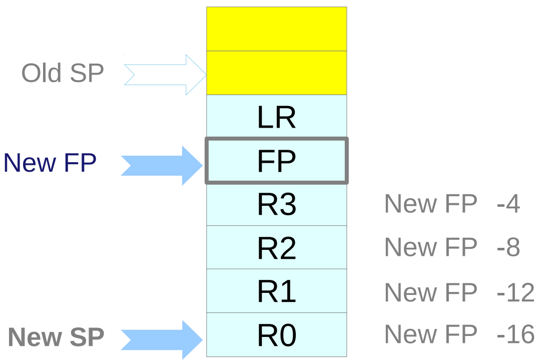
PUSH {FP, LR}



MOV FP, SP



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



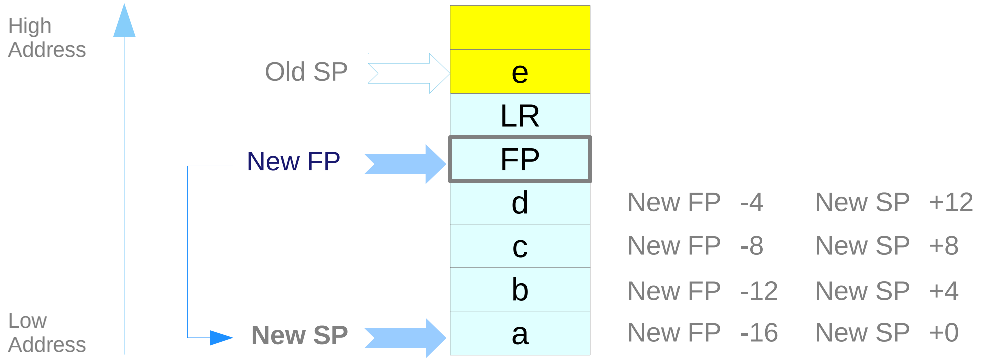
https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/

sq_sum5 (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

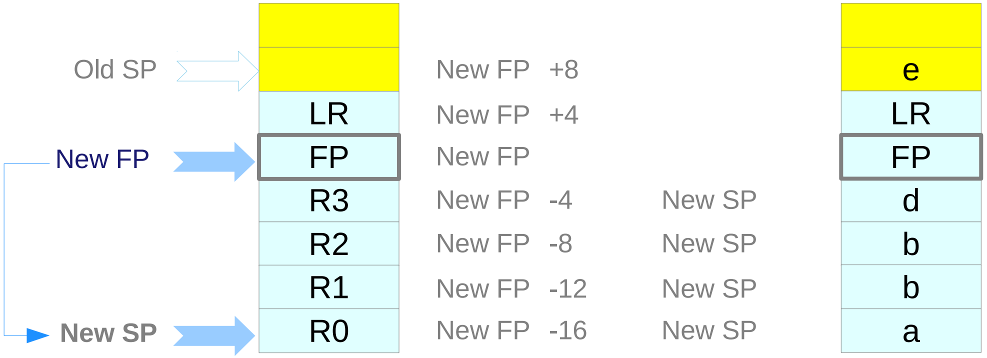
fp[-0] saved pc
fp[-1] saved lr
~~fp[-2] previous sp~~
~~fp[-3] previous fp~~



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

sq_sum5 (4)

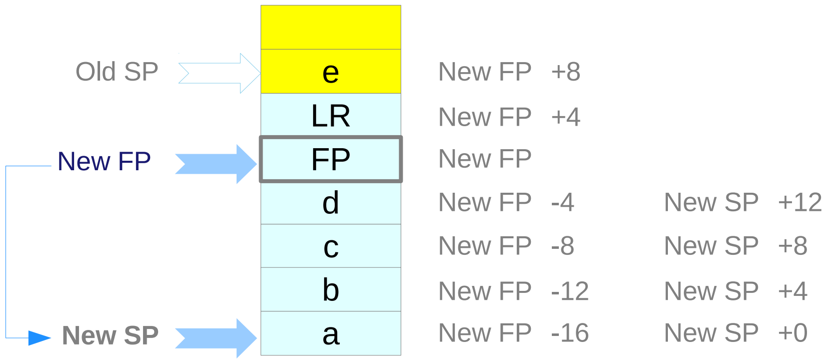
```
sub     r0, fp, #16     ; r0 ← fp - 16
bl      sq              ; call sq(&a);
sub     r0, fp, #12     ; r0 ← fp - 12
bl      sq              ; call sq(&b);
sub     r0, fp, #8      ; r0 ← fp – 8
bl      sq              ; call sq(&c);
sub     r0, fp, #4      ; r0 ← fp - 4
bl      sq              ;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

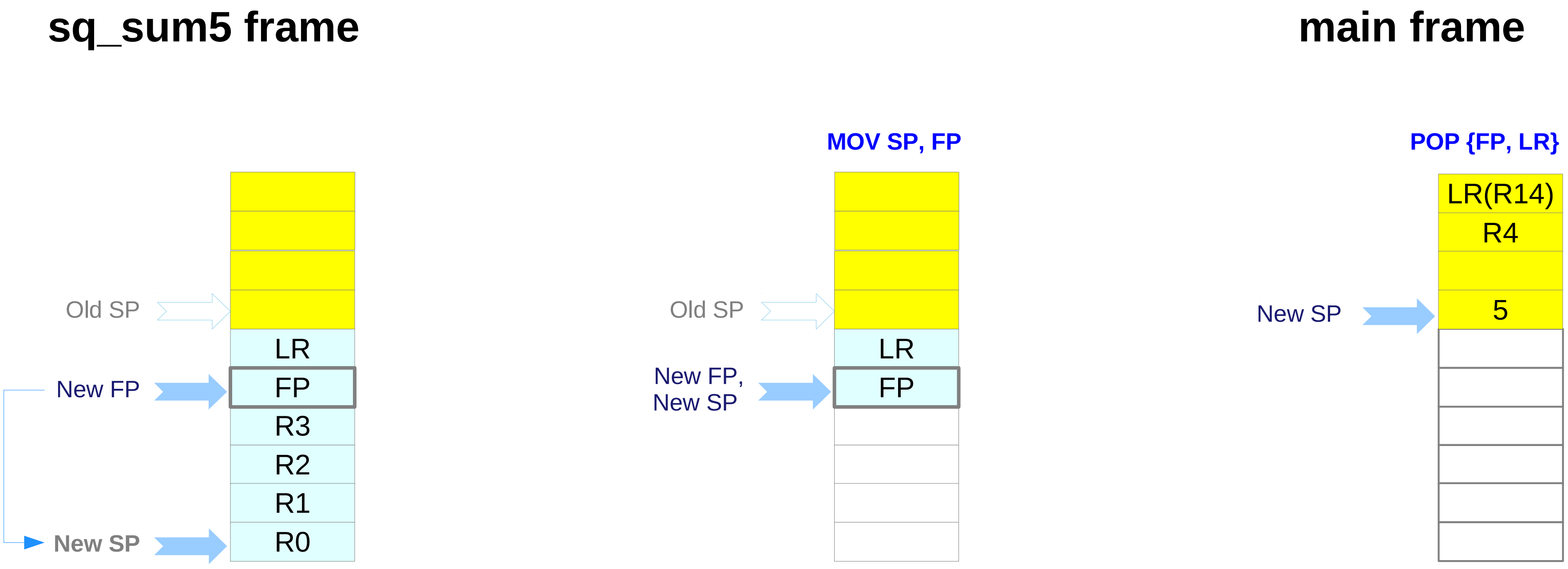
```
ldr     r0, [fp, #-16]     ; r0 ← *(fp - 16).   ; Loads a into r0
ldr     r1, [fp, #-12]     ; r1 ← *(fp - 12).   ; Loads b into r1
add     r0, r0, r1         ; r0 ← r0 + r1       ; (a +b)
ldr     r1, [fp, #-8]      ; r1 ← *(fp -  8).   ; Loads c into r1
add     r0, r0, r1         ; r0 ← r0 + r1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



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

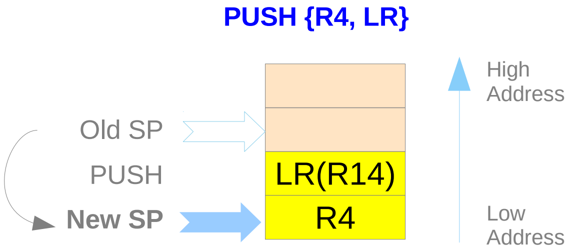
main (1)

```
/* 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)

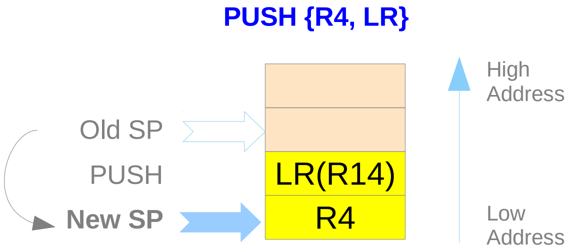
```
main:
        push    {r4, lr}          ; Keep callee-saved registers

        ; 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
```

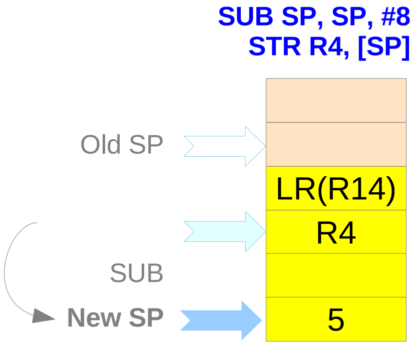


https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/

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)
```

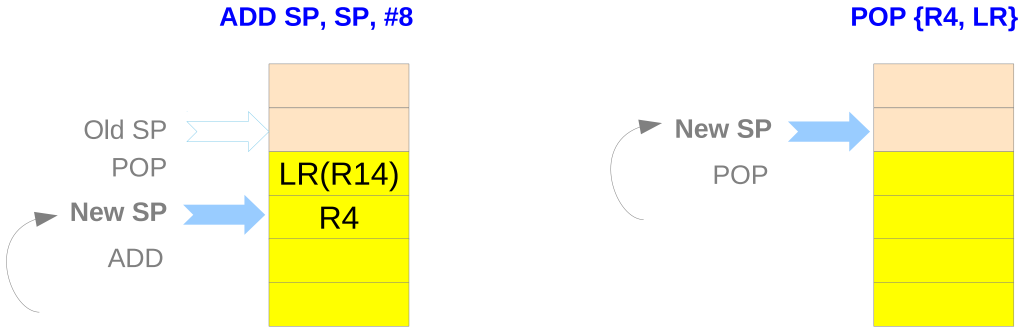




https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/

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/

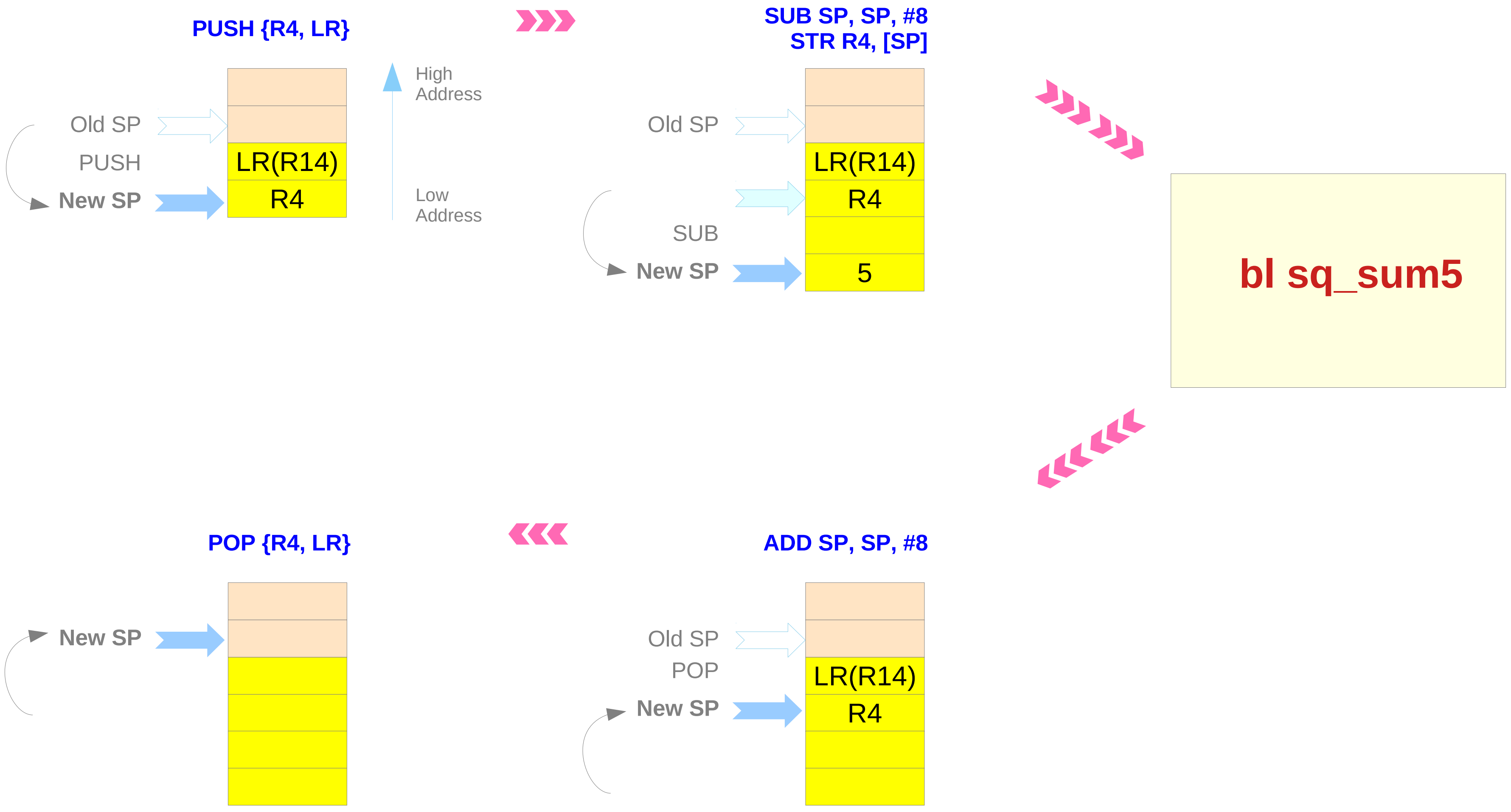
main (6)

```
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

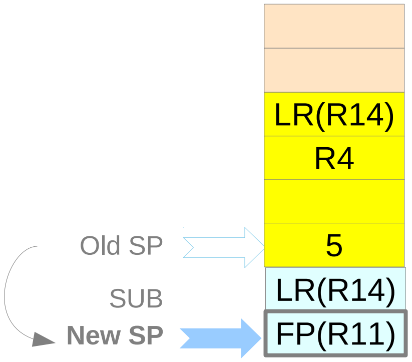


https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/

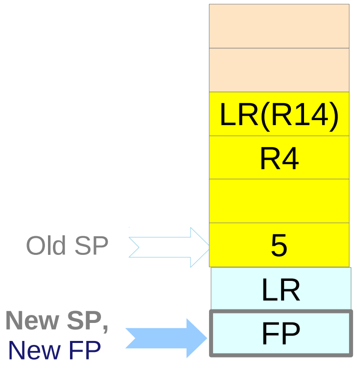
sq_sum5's stack frame (1)

```
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
```

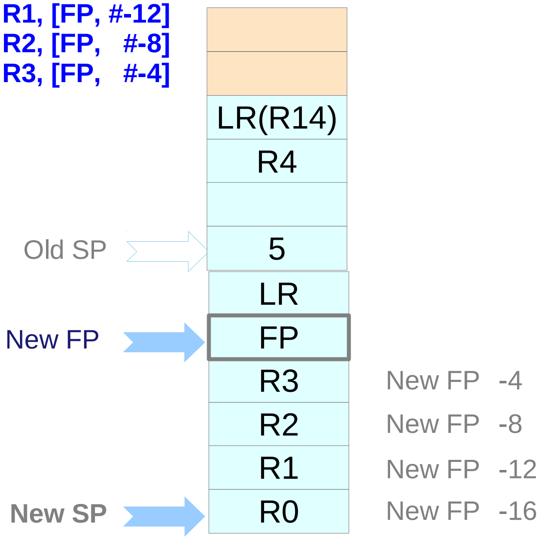
PUSH {FP, LR}



MOV FP, SP



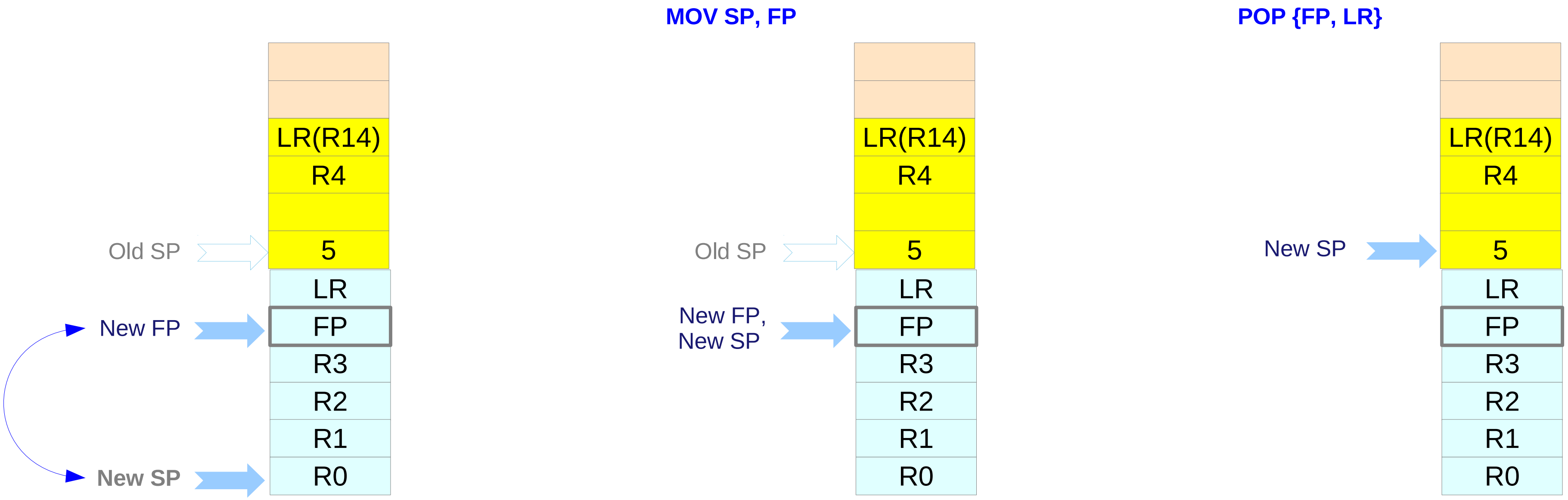
SUB SP, SP, #16
STR R0, [FP, #-16]
STR R1, [FP, #-12]
STR R2, [FP,   #-8]
STR R3, [FP,   #-4]



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
```

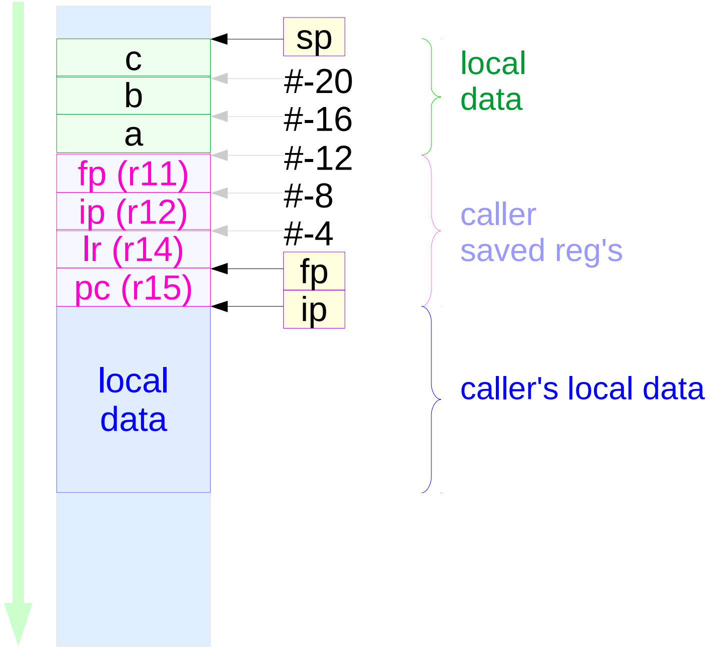


https://thinkingeek.com/2013/02/07/arm-assembler-raspberry-pi-chapter-10/

-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}
```

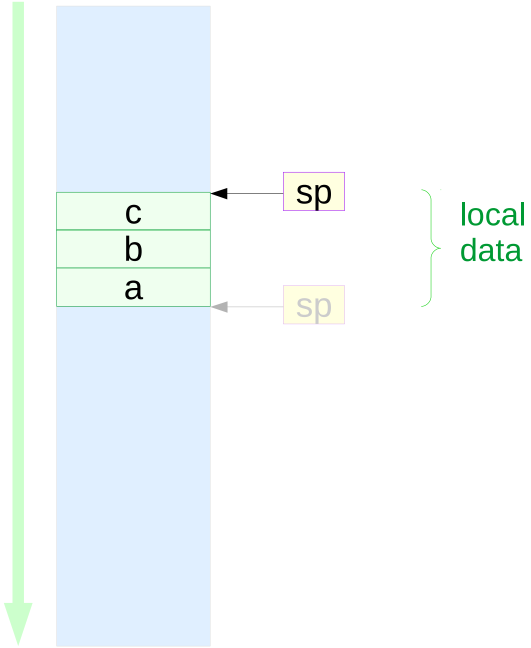
```
main()
{
  volatile int a, b, c;
  c = a + b;
}
```



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
[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