

# Stack Frames (11A)

---

---

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/>

# Local Variables

- Dynamic allocation / release allows for reuse of RAM
- Limited scope of access (making it private) provides for data protection
- Only the program that created the local variable can access it
- Since an interrupt will save registers, the code is reentrant
- Since absolute addressing is not used, the code is relocatable
  
- We can use symbolic names for the variables making it easier to understand
- The number of variables is only limited by the size of the stack
- Because it is more general, it will be easier to add additional variables
- 

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

# Global Variables

```
Void MyFunction (void) {  
    static uint32_t count = 0;  
    count++;  
}
```

```
Static int32_t myPrivateGlobalVariable; // accessible by this file only  
Void static MyPrivateFunction (void) {  
}
```

```
const int16_t Slope=21;  
const uint8_t SinTable[8] = {0, 50, 98, 142, 180, 212, 236, 250};
```

# Callee Saved Registers

function:

```
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 */
```

# Dynamic Link

function:

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

```
mov fp, sp
```

code of the function

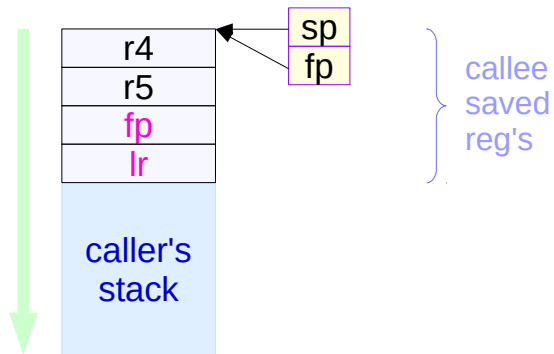
```
mov sp, fp
```

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

```
bx lr
```

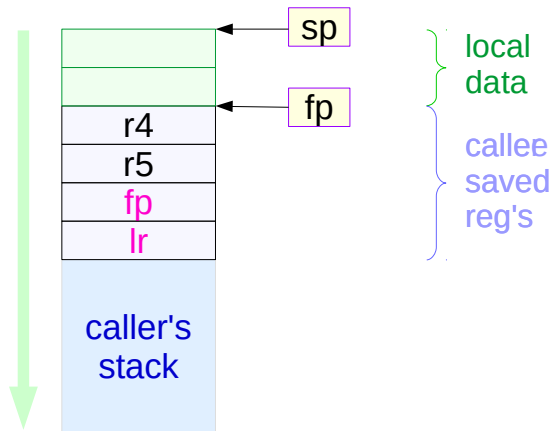
```
/* fp ← sp . Keep dynamic link in fp */
```

```
/* sp ← fp. Restore dynamic link in fp */
```



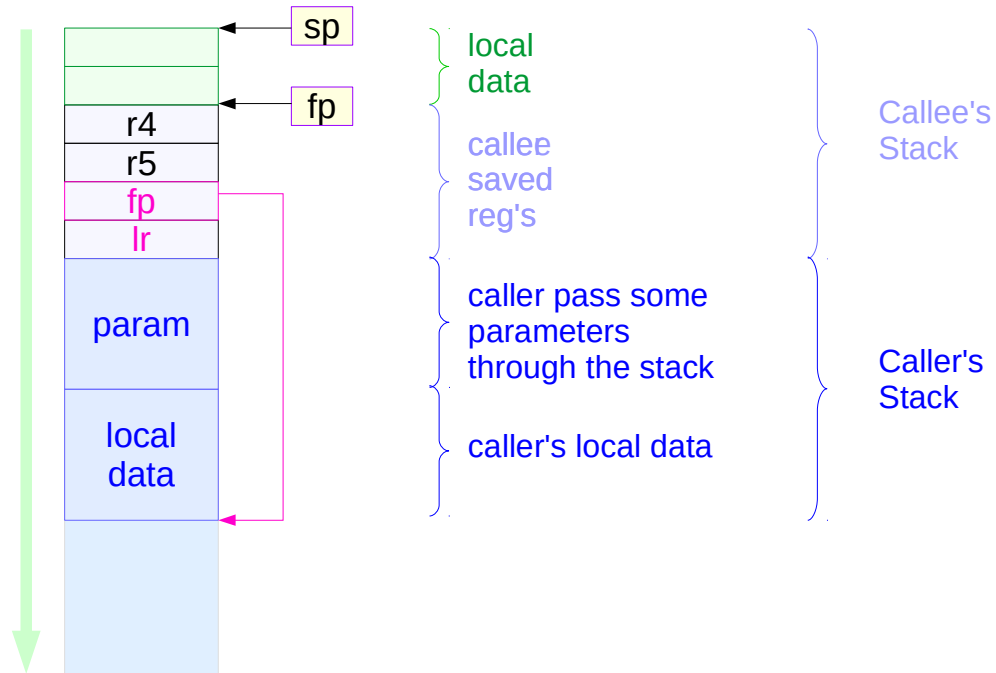
# Local Data

```
function:  
push { r4, r5, fp, lr }  
sub sp, sp, #8      /* 8 bytes local data space */  
mov fp, sp  
    code of the function  
mov sp, fp  
pop { r4, r5, fp, lr }  
bx lr
```





# Local Data and Parameters



# 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 (1)

```
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
```

# Callee Function Code (2)

```
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
```

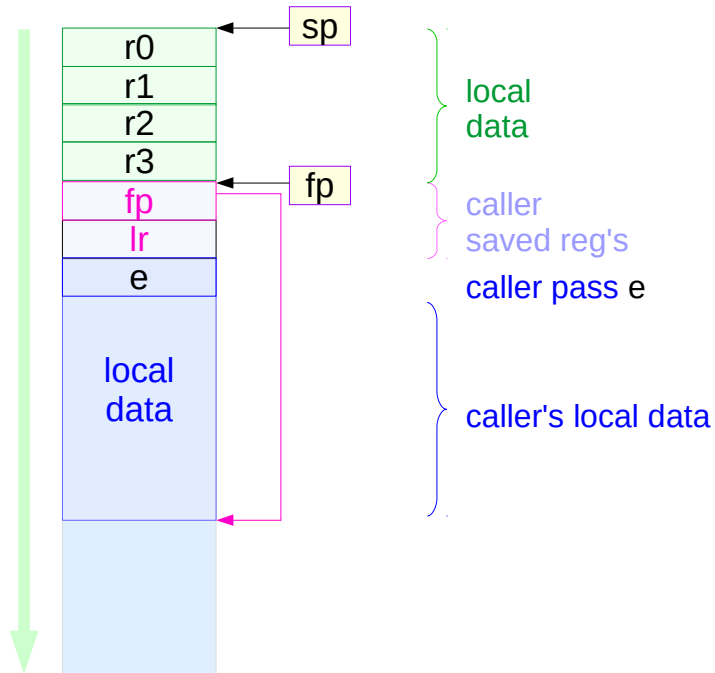
```
mov sp, fp  
pop { fp, lr }  
bx lr
```

At this point the stack looks like this  
| Value | Address (es)

	Value	Address (es)	
	r0	[ fp, #-16 ], [sp]	
	r1	[ fp, #-12 ], [sp, #4]	local data
	r2	[ fp, #-8 ], [sp, #8]	
	r3	[ fp, #-4 ], [sp, #12]	
	fp	[ fp ], [sp, #16]	callee saved registers
	lr	[ fp, #4 ], [sp, #20]	
	e	[ fp, #8 ], [sp, #24]	caller pass e parameters

v  
Higher addresses

# Callee Function Code (3)



At this point the stack looks like this  
 | Value | Address (es)

	Value	Address (es)
r0	[ fp, #-16 ]	[ sp ]
r1	[ fp, #-12 ]	[ sp, #4 ]
r2	[ fp, #-8 ]	[ sp, #8 ]
r3	[ fp, #-4 ]	[ sp, #12 ]
fp	[ fp ]	[ sp, #16 ]
lr	[ fp, #4 ]	[ sp, #20 ]
e	[ fp, #8 ]	[ sp, #24 ]

v  
Higher addresses

local data  
 callee saved registers  
 caller pass e parameters

# 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
```

# APCS Register Use Convention

---

R11	fp	Frame Pointer
R12	ip	Scratch register / specialist use by linker
R13	sp	Lower end of current stack frame
R14	lr	Link address / scratch register
R15	pc	Program counter

# LR and FP Registers

SP    where the stack **is**  
FP    where the stack **was**

PC    where you **are**  
LR    where you **were**

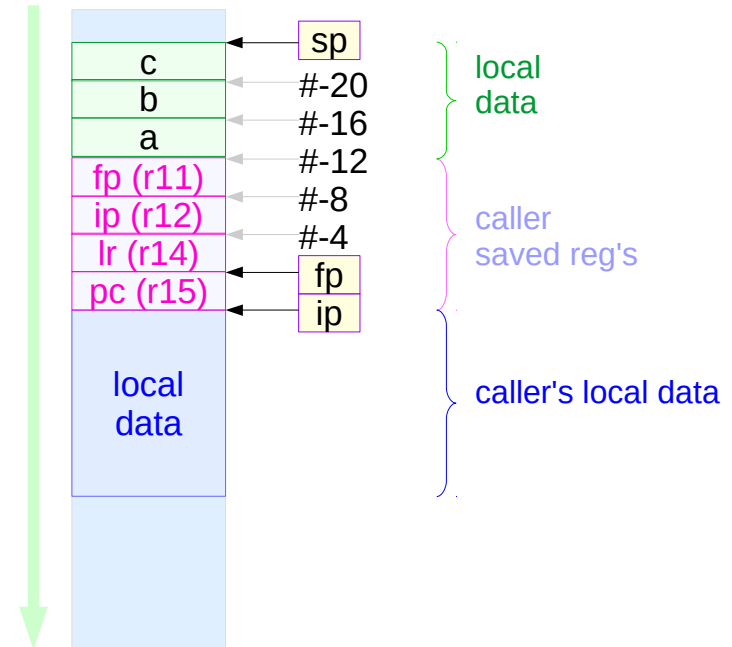
<http://stackoverflow.com/questions/15752188/arm-link-register-and-frame-pointer>



# -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;  
}
```

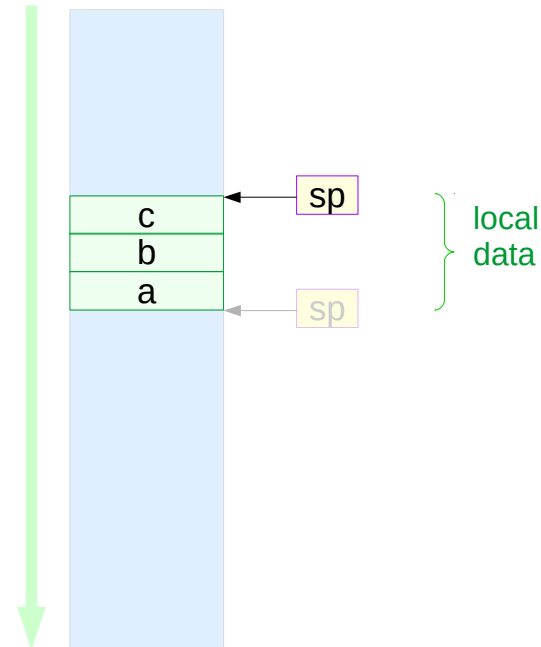


<https://community.arm.com/thread/7092>

# -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>