4 Function Calls and Returns

Compiling Like a Non-optimizing Compiler

```
x = y;
z = x;
```

are translated by a non-optimizing compiler to

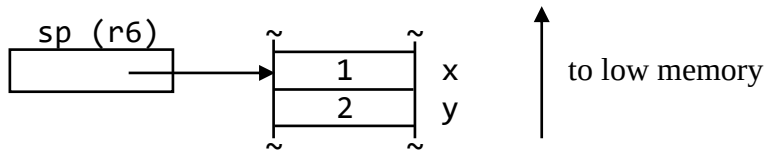
```
ld r0, y
st r0, x
ld r0, x  ; unnecessary ld instruction
st r0, z
```

```
f(x+3, 7);              // Use stack to pass values

x = (a + b) + (c + d);  // evaluate according to parens

x = 1 + 2;              // constant folding
```

Calling a Function

```
f(1, 2);


void f(int x, int y)
{
   ⋮
}
```

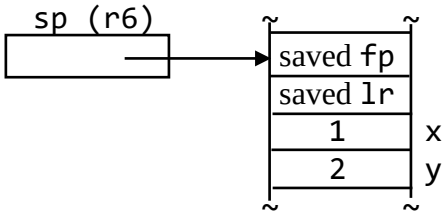
```
1          ; f(1, 2);
2          mov r0, 2
3          push r0          ; creates the parameter y on the stack
4          mov r0, 1
5          push r0          ; creates the parameter x on the stack
6          bl f             ; jump to f
7          add sp, sp, 2    ; pop parameters off the stack
```



In Called Function

```
push lr

push fp
```

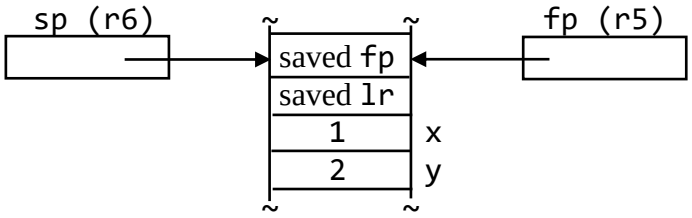
The stack at this point looks like this:



Get `fp` to point to called function's stack frame:

```
mov fp, sp
```

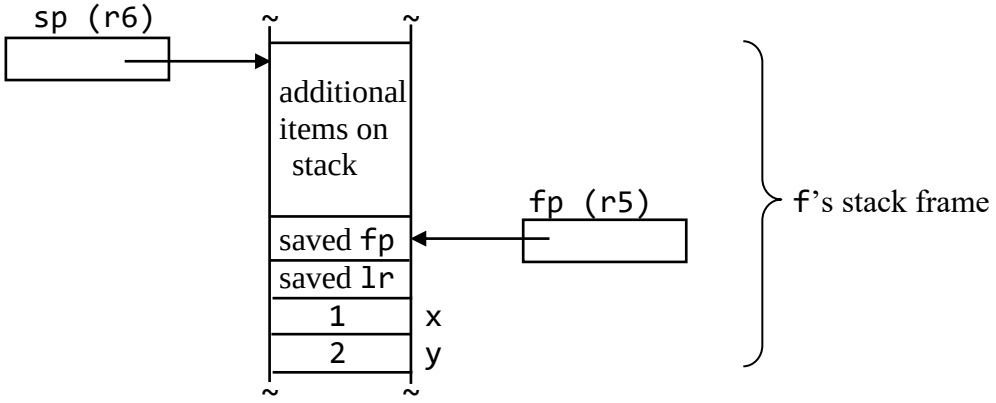
We get the following configuration:



Access first parameter:

```
ldr r0, fp, 2
```

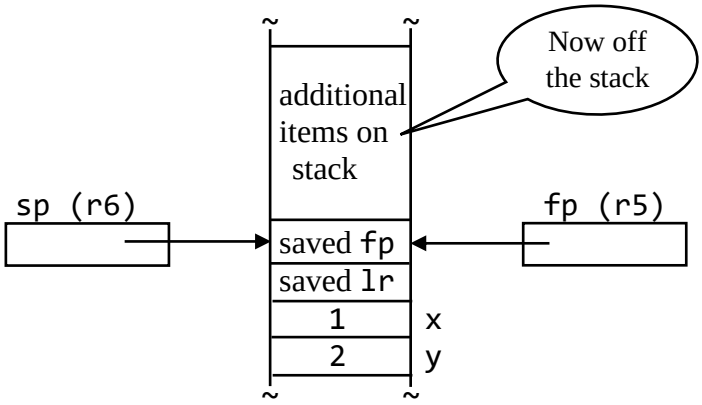
Rule: Use the offset 2 in a `ldr` or `str` instruction to access the first parameter.



To return to the caller

`f` first moves the contents of the `fp` register into the `sp` register:

```
mov sp, fp
```




```
pop fp      ; restore fp with caller's base address
pop lr      ; restore lr with caller's return address
ret
```

Example of a Function Call

```
 1 ; ex0401.a  Calling a function
 2
 3 startup:  bl main
 4           halt            ; back to operating system
 5 ;=============================================================
 6                           ; #include <stdio.h>
 7 f:        push lr         ; void f(int x, int y)
 8           push fp         ; {
 9           mov fp, sp
10
11           ldr r0, fp, 2   ;    printf("%d\n", x + y);
12           ldr r1, fp, 3
13           add r0, r0, r1
14           dout r0
15           nl
16
17           mov sp, fp      ; }
18           pop fp
19           pop lr
20           ret
21 ;=============================================================
22 main:     push lr         ; int main()
23           push fp         ; {
24           mov fp, sp
25
26           mov r0, 2       ;    f(1, 2);
27           push r0
28           mov r0, 1
29           push r0
30           bl f
31           add sp, sp, 2
32
33           mov r0, 0       ;    return 0;
34           mov sp, fp
35           pop fp
36           pop lr
37           ret
38                           ; }
```

Makes `fp` point to `f`'s stack frame

Return code generated even though no `return` statement in C

Parameters created before the `bl`

Parameters destroyed after the `bl`

Summary

1. Argument values in a function call are passed via the stack. Pushing the value of an argument creates it corresponding parameter.

2. Parameters are created by the *calling* function—not the called function—before the `bl` instruction. Parameters are "destroyed" (i.e., removed from the stack) by the *calling* function after the `bl` instruction.

3. Parameters are created *dynamically* (i.e., while the program is executed). Thus, they do not have labels associated with them. They are accessed with the `ldr` and `str` instruction in which an offset—not a label—is specified. Note that in `ex0401.a`, there is no `x` or `y` label.

Returning a Value

```
 1 ; ex0402.a  Returning a value
 2
 3 startup:  bl main
 4           halt              ; back to operating system
 5 ;=============================================================
 6                             ; #include <stdio.h>
 7 f:        push lr           ; int f()
 8           push fp           ; {
 9           mov fp, sp
10
11           mov r0, 5         ;    return 5;
12           mov sp, fp
13           pop fp
14           pop lr
15           ret
16                             ; }
17 ;=============================================================
18 main:     push lr           ; int main()
19           push fp           ; {
20           mov fp, sp
21
22           bl  f             ;    printf("%d\n", f());
23           dout r0
24           nl
25
26           mov r0, 0         ;    return 0;
27           mov sp, fp
28           pop fp
39           pop lr
30           ret
31                             ; }
```