

Chapter 14

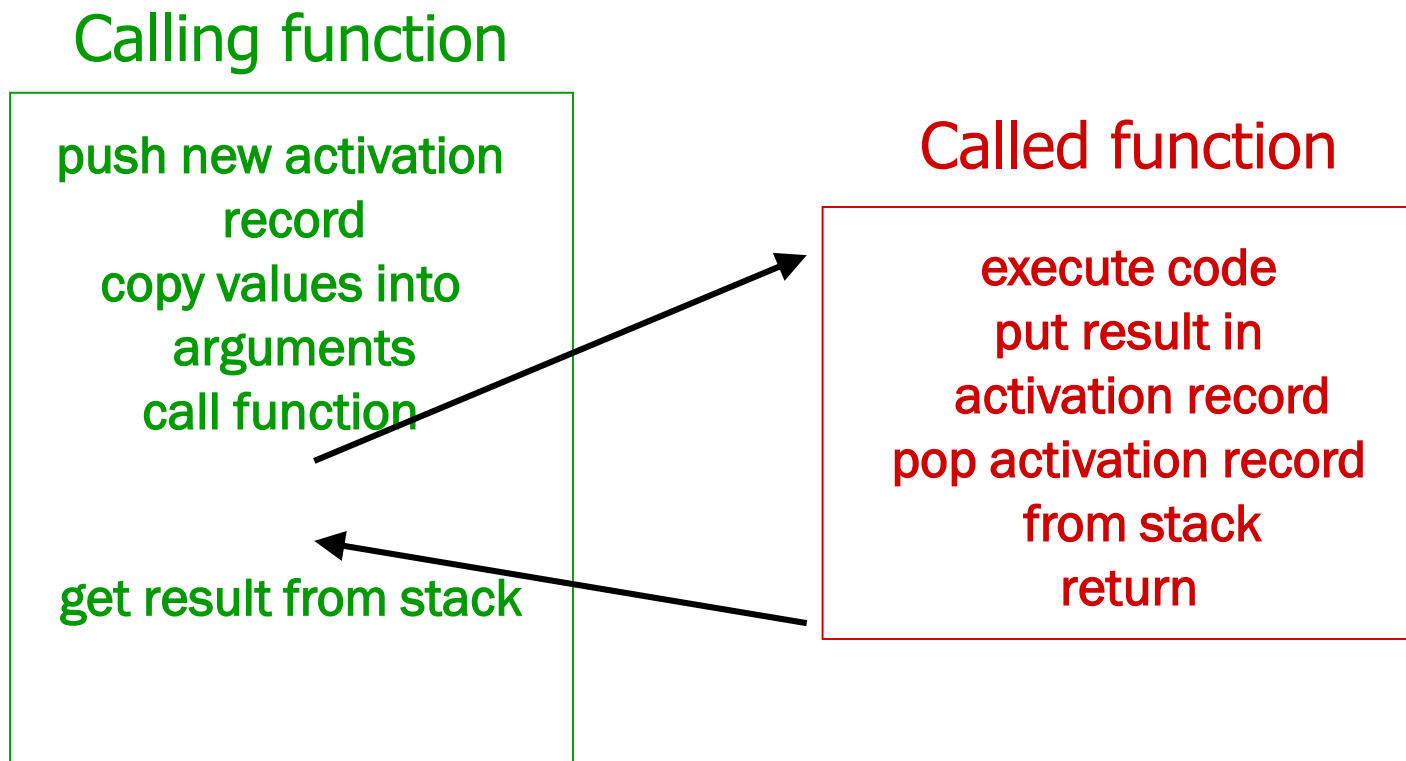
Implementing Functions using Activation Records

(We have already seen
functions in C)

Implementing Functions: Overview

Activation record

- information about each function, including arguments and local variables
- stored on run-time stack



Run-Time Stack

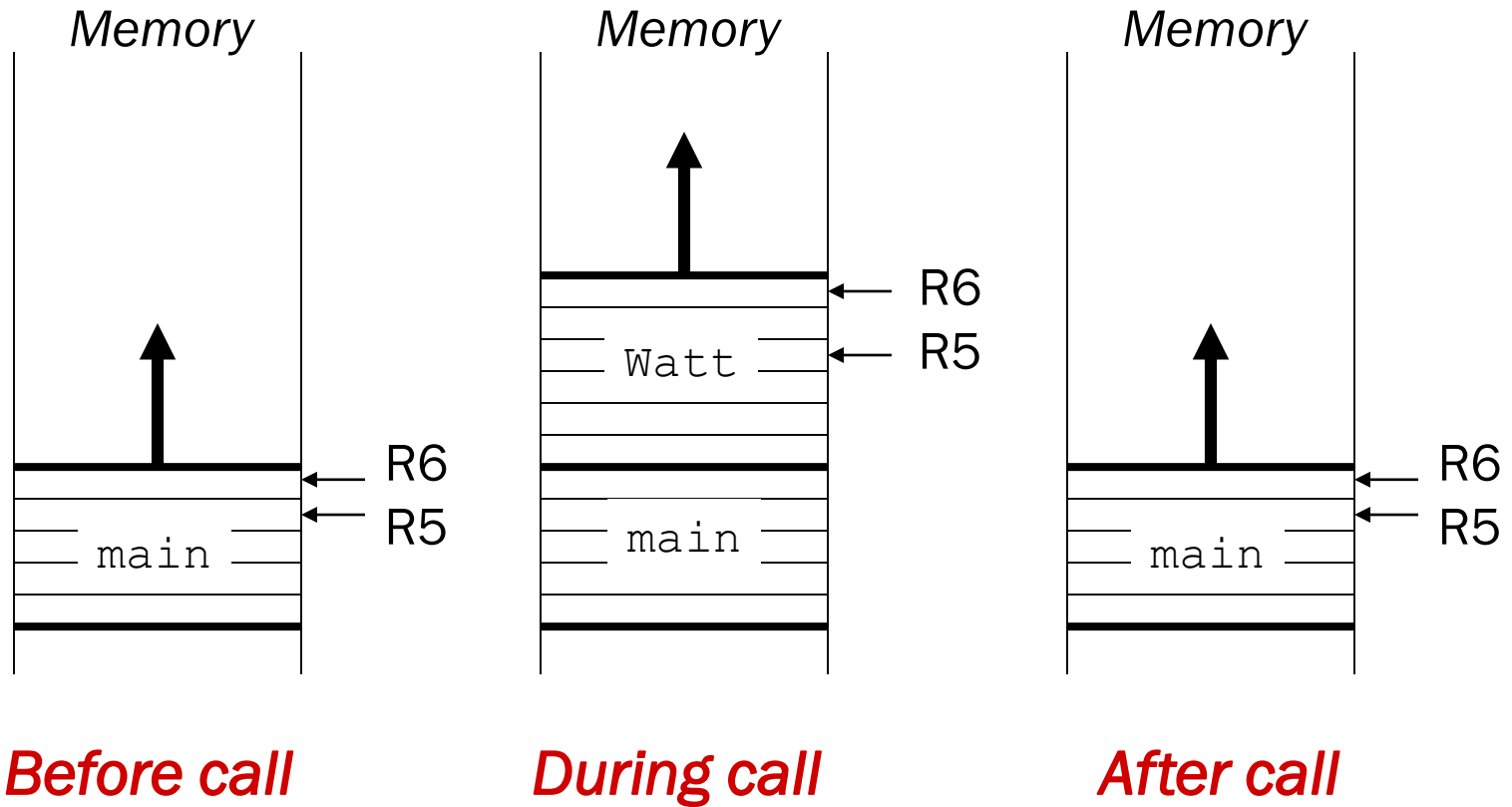
Recall that local variables are stored on the run-time stack in an *activation record*

Frame pointer (R5) points to the beginning of a region of activation record that stores local variables for the current function

When a new function is **called**, its activation record is **pushed** on the stack;

when it **returns**, its activation record is **popped** off of the stack.

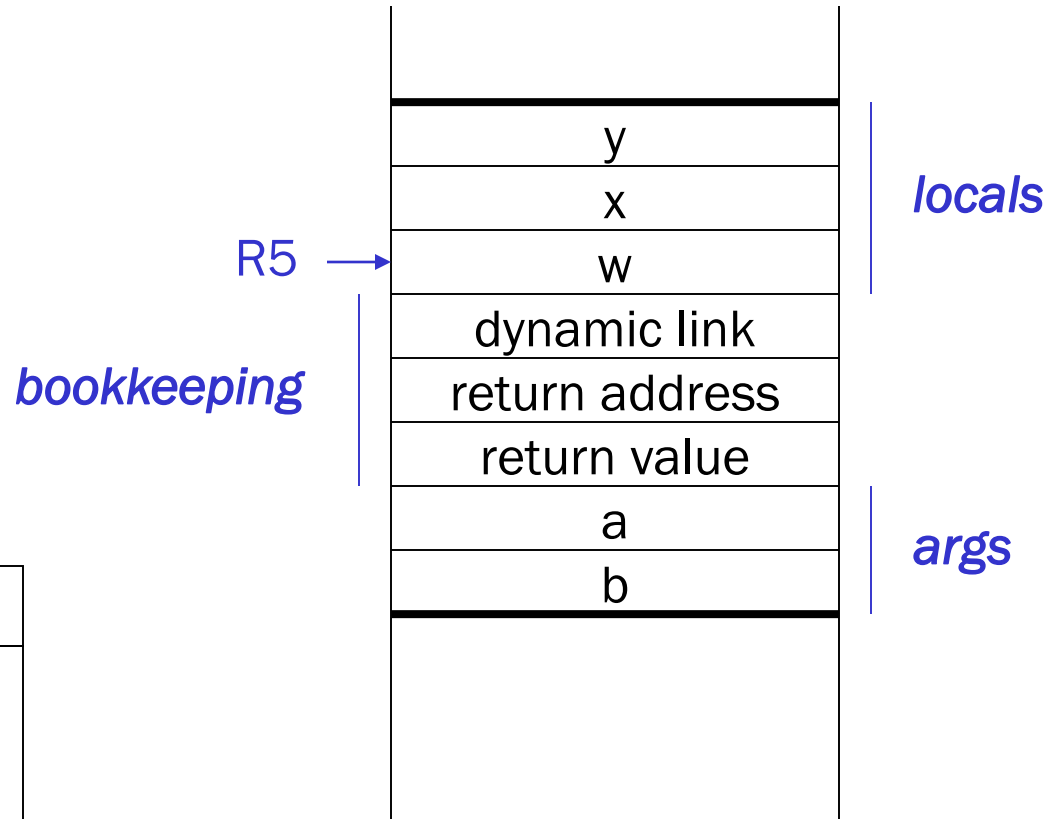
Run-Time Stack



Activation Record

```
int NoName(int a, int b)
{
    int w, x, y;
    .
    .
    .
    return y;
}
```

Name	Type	Offset	Scope
a	int	4	NoName
b	int	5	NoName
w	int	0	NoName
x	int	-1	NoName
y	int	-2	NoName



Activation Record Bookkeeping

Return value

- space for value returned by function
- allocated even if function does not return a value

Return address

- save pointer to next instruction in calling function
- convenient location to store R7 in case another function (JSR) is called

Dynamic link

- caller's frame pointer
- used to pop this activation record from stack

Example Function Call

```
int Volta(int q, int r)
{
    int k;
    int m;
    ...
    return k;
}
```

```
int Watt(int a)
{
    int w;
    ...
    w = Volta(w, 10);
    ...
    return w;
}
```

Calling the Function

```
w = Volta(w, 10);
```

```
; push second arg
```

```
AND R0, R0, #0
```

```
ADD R0, R0, #10
```

```
ADD R6, R6, #-1
```

```
STR R0, R6, #0
```

```
; push first argument
```

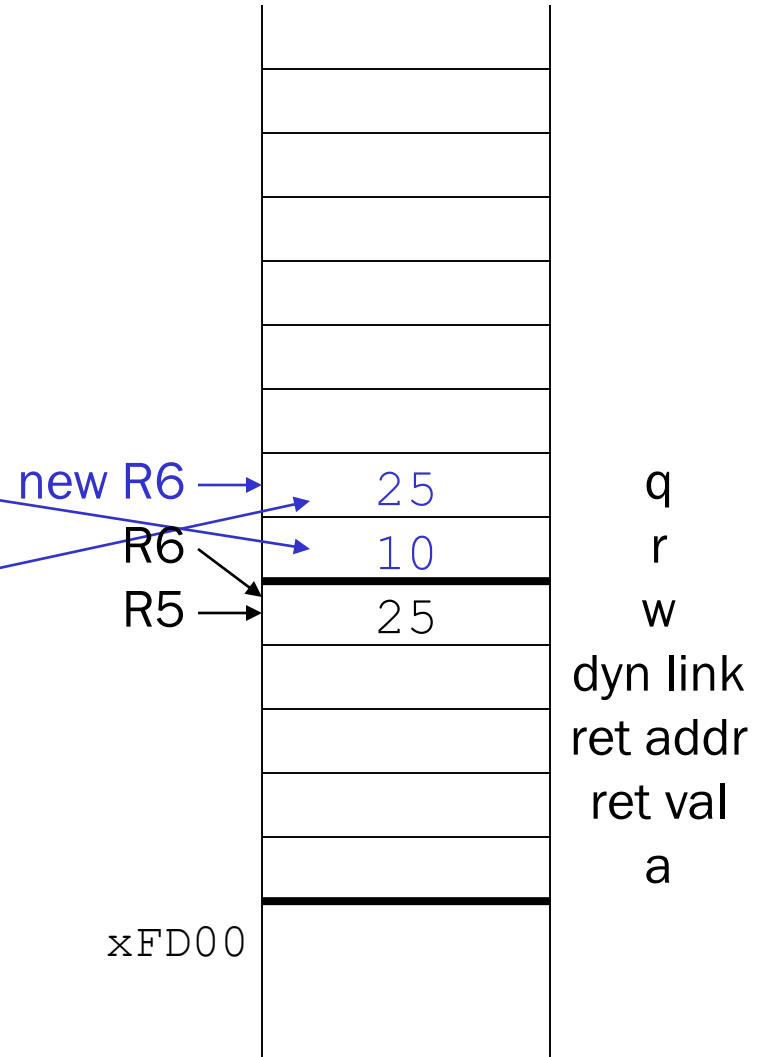
```
LDR R0, R5, #0
```

```
ADD R6, R6, #-1
```

```
STR R0, R6, #0
```

```
; call subroutine
```

```
JSR Volta
```



Note: Caller needs to know number and type of arguments, doesn't know about local variables.

Ending the Callee Function

return k;

; copy k into return value

LDR R0, R5, #0

STR R0, R5, #3

; pop local variables

ADD R6, R5, #1

; pop dynamic link (into R5)

LDR R5, R6, #0

ADD R6, R6, #1

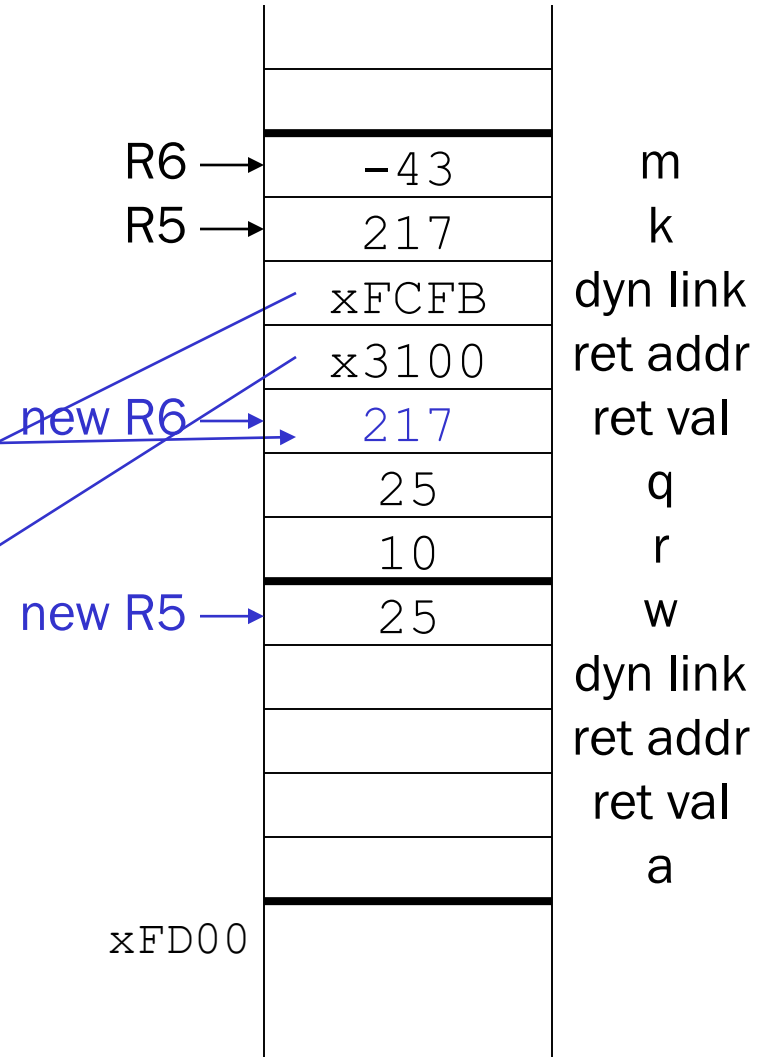
; pop return addr (into R7)

LDR R7, R6, #0

ADD R6, R6, #1

; return control to caller

RET



Summary of LC-3 Function Call Implementation

1. **Caller** pushes arguments (last to first).
2. **Caller** invokes subroutine (JSR).
3. **Callee** allocates return value, pushes R7 and R5.
4. **Callee** allocates space for local variables.
5. **Callee** executes function code.
6. **Callee** stores result into return value slot.
7. **Callee** pops local vars, pops R5, pops R7.
8. **Callee** returns (JMP R7).
9. **Caller** loads return value and pops arguments.
10. **Caller** resumes computation...