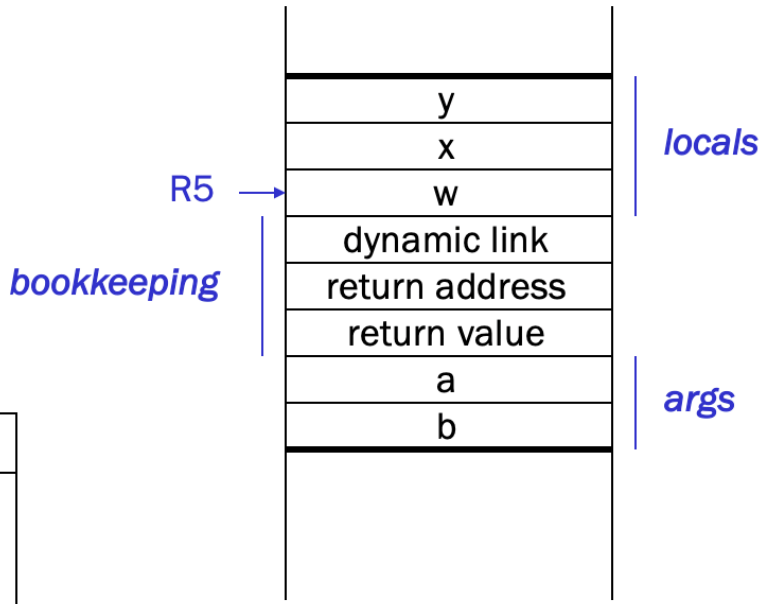
Patt-Ch14 Functions

1 Run-Time Stack

1.1 Activation Record

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

14-11

(1) Arguments

- Pushed by **caller**
- 倒着存，如上图，先存b再存a

(2) 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(R5)**
- **used to pop this activation record from stack**

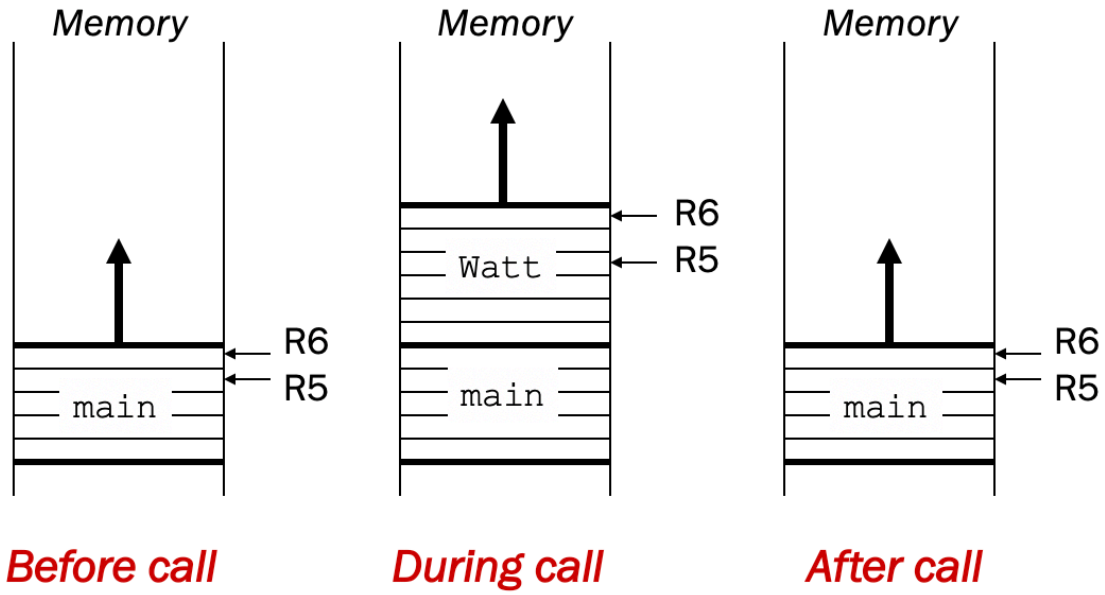
(3) Frame Pointer and Local Variables

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

1.2 Run-Time Stack

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.



1.3 Example

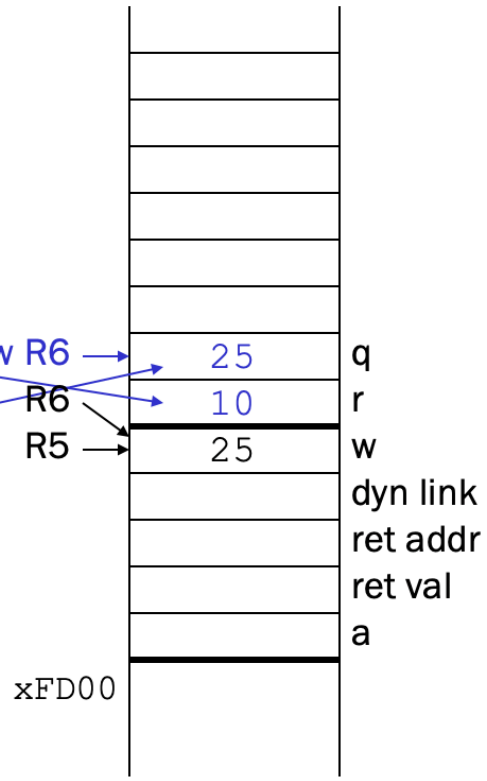
```
int Volta(int q, int r)
{
  int k;
  int m;
  ...
  return k;
}

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

补充新ppt上的形象图片

Calling the Function

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

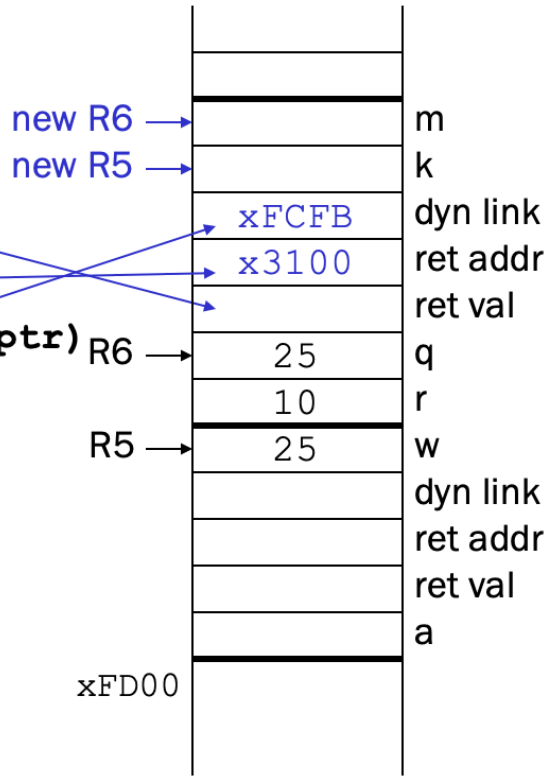
```
; push second arg
AND  R0, R0, #0
ADD  R0, R0, #10
ADD  R6, R6, #-1
STR  R0, R6, #0
; push first argument
LDR  R0, R5, #0
ADD  R6, R6, #-1
STR  R0, R6, #0

; call subroutine
JSR  Volta
```



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

Starting the Callee Function

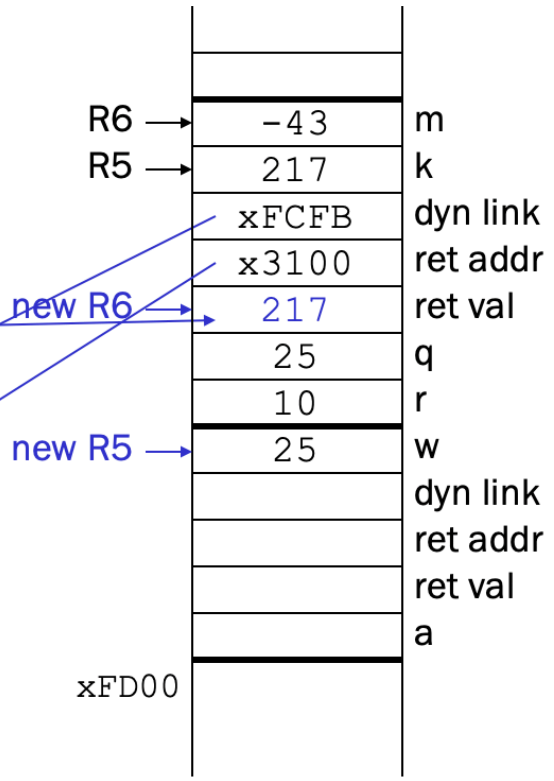
```
; leave space for return value
ADD  R6, R6, #-1
; push return address
ADD  R6, R6, #-1
STR  R7, R6, #0
; push dyn link (caller's frame ptr)
ADD  R6, R6, #-1
STR  R5, R6, #0
; set new frame pointer
ADD  R5, R6, #-1
; allocate space for locals
ADD  R6, R6, #-2
```



Ending the Callee Function

return k;

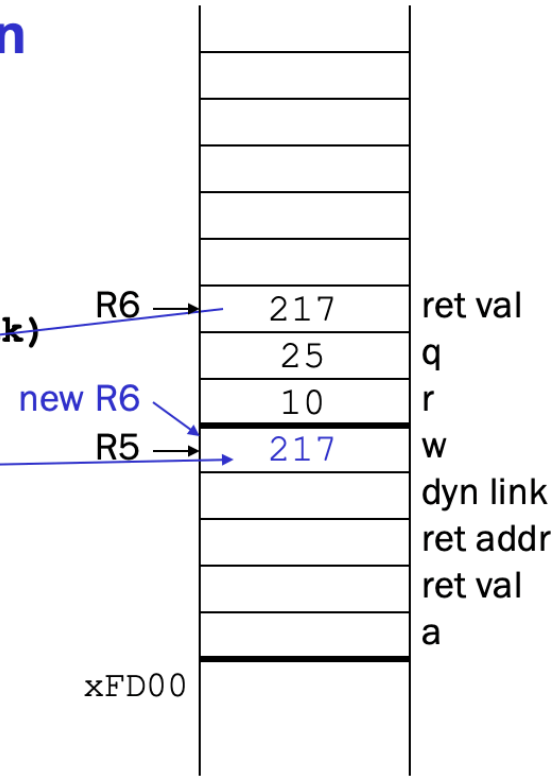
```
; copy k into return value
LDR  R0, R5, #0
STR  R0, R5, #3
; pop local variables
ADD  R6, R5, #1
; pop dynamic link (into R5)
LDR  R5, R6, #0
ADD  R6, R6, #1
; pop return addr (into R7)
LDR  R7, R6, #0
ADD  R6, R6, #1
; return control to caller
RET
```



Resuming the Caller Function

w = Volta(w,10);

```
JSR  Volta

; load return value (top of stack)
LDR  R0, R6, #0
; perform assignment
STR  R0, R5, #0
; pop return value
ADD  R6, R6, #1
; pop arguments
ADD  R6, R6, #2
```



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.(Save Rx, Restore Rx)
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…

2 Pointer

Passing Pointers to a Function

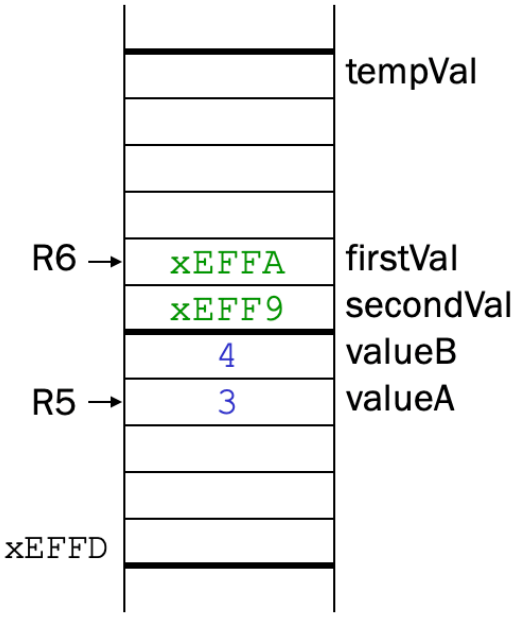
main() wants to swap the values of valueA and valueB

passes the addresses to NewSwap:

```
NewSwap(&valueA, &valueB);
```

Code for passing arguments:

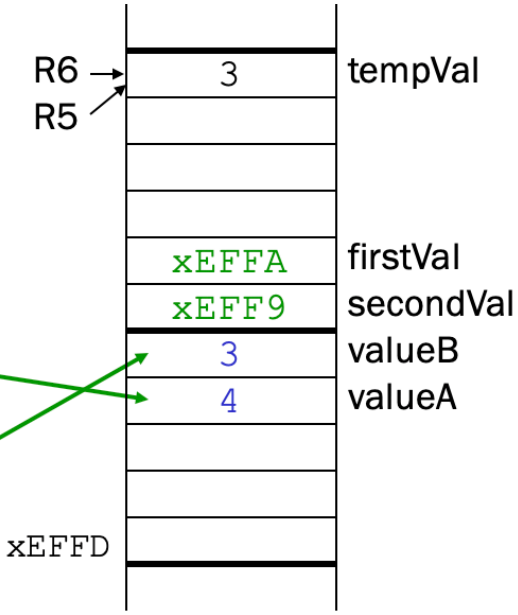
```
ADD R0, R5, #-1 ; addr of valueB
ADD R6, R6, #-1 ; push
STR R0, R6, #0
ADD R0, R5, #0  ; addr of valueA
ADD R6, R6, #-1 ; push
STR R0, R6, #0
```



Code Using Pointers

Inside the NewSwap routine

```
; int tempVal = *firstVal;
LDR  R0, R5, #4  ; R0=xEFFA
LDR  R1, R0, #0  ; R1=M[xEFFA]=3
STR  R1, R5, #0  ; tempVal=3
; *firstVal = *secondVal;
LDR  R1, R5, #5  ; R1=xEFF9
LDR  R2, R1, #0  ; R2=M[xEFF9]=4
STR  R2, R0, #0  ; M[xEFFA]=4
; *secondVal = tempVal;
LDR  R2, R5, #0  ; R2=3
STR  R2, R1, #0  ; M[xEFF9]=3
```



Null Pointer

Sometimes we want a pointer that points to nothing.

In other words, we declare a pointer, but we're not ready to actually point to something yet.

```
int *p;
p = NULL;  /* p is a null pointer */
```

`NULL` is a predefined macro that contains a value that a non-null pointer should never hold.

- **Often, NULL = 0, because Address 0 is not a legal address for most programs on most platforms.**

Array

Array Reference

```
variable[index];
```

i-th element of array (starting with zero);
no limit checking at compile-time or run-time

Array as a Local Variable

Array elements are allocated as part of the activation record.

```
int grid[10];
```

First element (`grid[0]`) is at lowest address of allocated space.

If `grid` is first variable allocated, then R5 will point to `grid[9]`.