

# 15 – FUNCTIONS & PARAMETERS

COMP256 – COMPUTING ABSTRACTIONS  
DICKINSON COLLEGE

## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - Values of the **function's parameters (FP)**
  - Return address (RA)**
  - Values of the function's **local variables (LV)**

```
main() {  
    read x;  
    read y;  
    read z;  
  
    m = max3(x,y,z);  
  
    print m;  
}  
  
max3(a,b,c) {  
    x=max(a,b)  
    y=max(x,c)  
    return y;  
}  
  
max(a,b) {  
    x = a;  
    if (b > a)  
        x = b;  
    return x;  
}
```

1

## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values

```
main() {  
    read x;  
    read y;  
    read z;  
  
    A m = max3(x,y,z);  
    print m;  
}  
  
max3(a,b,c) {  
    B x=max(a,b)  
    C y=max(x,c)  
    return y;  
}  
  
max(a,b) {  
    x = a;  
    if (b > a)  
        x = b;  
    return x;  
}
```

LV:  $\emptyset$   
RA: Exit  
FP:  $\emptyset$

2

3

## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values

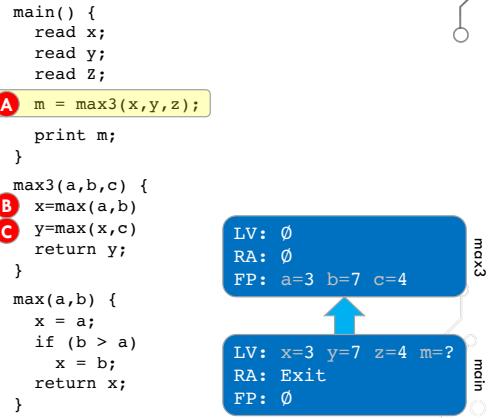
```
main() {  
    read x;  
    read y;  
    read z;  
  
    A m = max3(x,y,z);  
    print m;  
}  
  
max3(a,b,c) {  
    B x=max(a,b)  
    C y=max(x,c)  
    return y;  
}  
  
max(a,b) {  
    x = a;  
    if (b > a)  
        x = b;  
    return x;  
}
```

LV: x=3 y=7 z=4  
RA: Exit  
FP:  $\emptyset$

main

## HLL FUNCTION CALL STACK

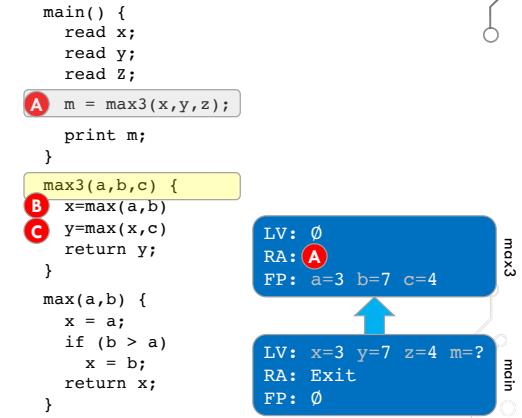
- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values



5

## HLL FUNCTION CALL STACK

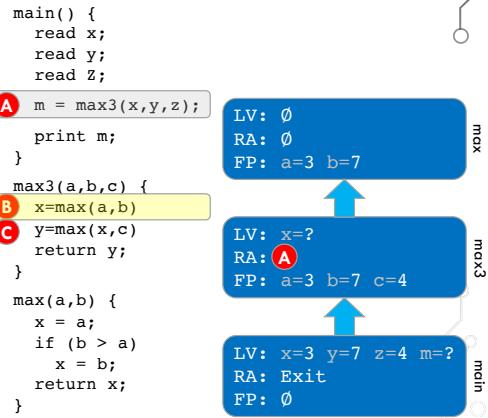
- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values



6

## HLL FUNCTION CALL STACK

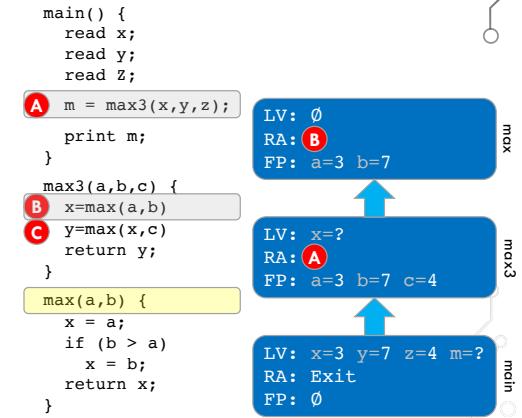
- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values



7

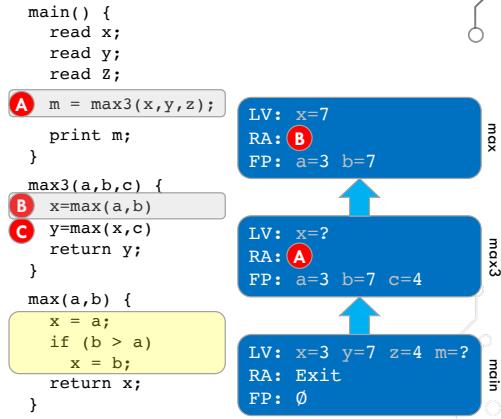
## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values



## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values

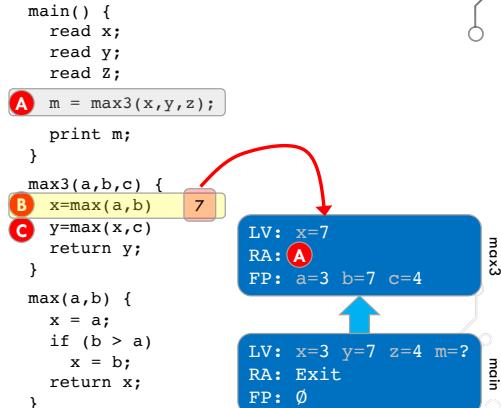


8

9

## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values

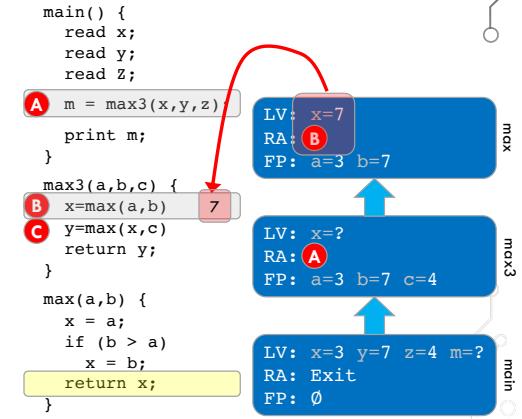


10

11

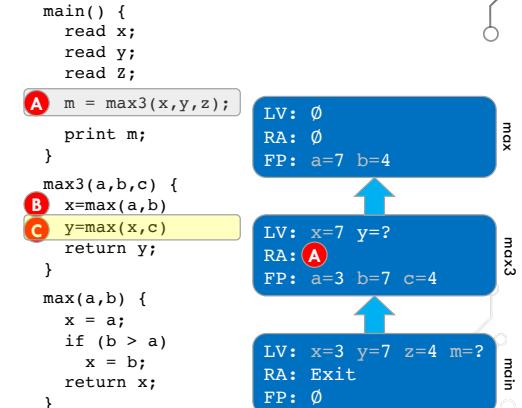
## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values



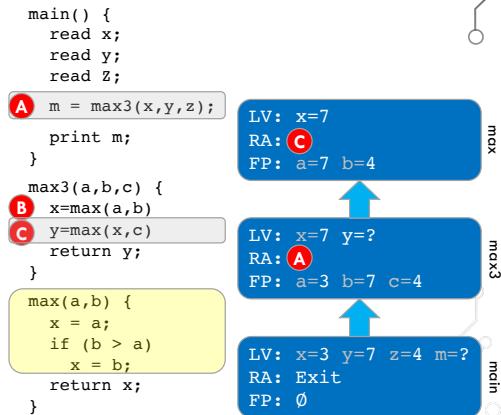
## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values



## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values

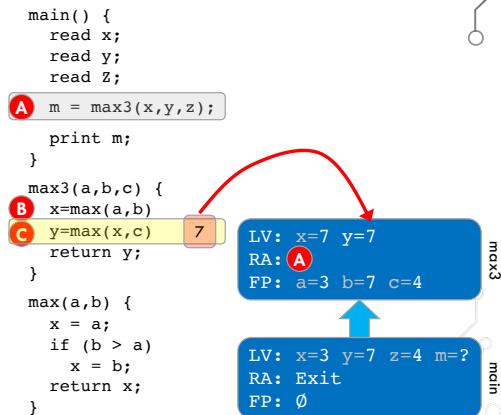


12

13

## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values

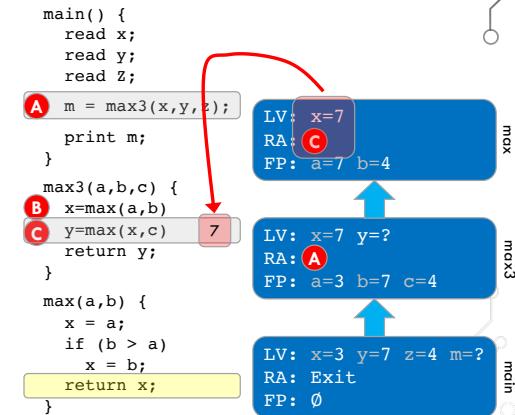


14

15

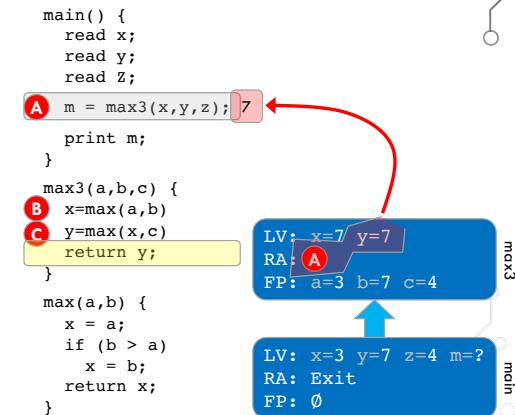
## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values



## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's **local variable (LV)** values



## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's local variable (LV)** values

```

main() {
    read x;
    read y;
    read z;

    A m = max3(x,y,z); 7
    print m;
}

max3(a,b,c) {
    B x=max(a,b)
    C y=max(x,c)
    return y;
}

max(a,b) {
    x = a;
    if (b > a)
        x = b;
    return x;
}

```

LV: x=3 y=7 z=4 m=7  
RA: Exit  
FP: Ø

16

17

7

## HLL FUNCTION CALL STACK

- The (function) **call stack** maintains the state of a running program.
- Each function call adds a **stack frame** to the call stack.
- A stack frame contains the:
  - function's parameter (FP)** values
  - Return address (RA)**
  - function's local variable (LV)** values

```

main() {
    read x;
    read y;
    read z;

    A m = max3(x,y,z);
    print m;
}

max3(a,b,c) {
    B x=max(a,b)
    C y=max(x,c)
    return y;
}

max(a,b) {
    x = a;
    if (b > a)
        x = b;
    return x;
}

```

LV: x=3 y=7 z=4 m=7  
RA: Exit  
FP: Ø

## UNPACKING THE FUNCTION CALL ABSTRACTION

- Function calls in a high level language are an abstraction for the low level instructions in assembly/machine language that explicitly manage the stack frames.

- These instructions:

- Put parameter values into the stack frame
- Call the function
- Preserve the return address into the stack frame
- Preserve used register values in the stack frame
- Retrieve parameters from the stack frame
- Execute the function
- Set the return value
- Restore used register values from stack frame
- Restore the return address from stack frame
- Remove parameters from stack frame
- Return from the function
- Get the return value and continue execution

Calling Code

Function Code

Calling Code

## FUNCTION CALLS IN ASSEMBLY

MAIN:

...  
**CALL FUNC**

Put the parameter values into the stack frame.

...

HALT

Preserve the return address and used register values in the stack frame.

FUNC:

...

...

...

...

Retrieve parameters from the stack frame.

...

...

...

...

RET

Execute the function. Set the return value.

Restore used registers and return address from stack frame.

## CALLING AND RETURNING

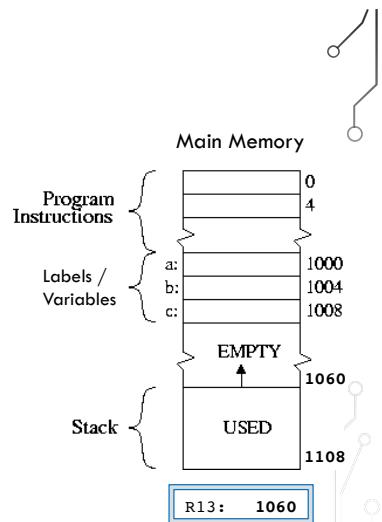
Instruction	Example	Meaning
CALL L RET	CALL FUNC RET	$R12 = PC + 4, PC = FUNC$ $PC = R12$ R12 is a reserved register used for the return address of function calls.
MAIN:	... CALL FUNC ... HALT	
FUNC:	... RET	

## MANAGING STACK FRAMES

Instruction	Example	Meaning
PUSH R POP R	PUSH R2 POP R2	$MM[R13] = R2; R13 = R13 - 4$ $R13 = R13 + 4; R2 = MM[R13]$
.stacksize 100 LOAD R0 #10 PUSH R0 ADD R0 R0 #5 PUSH R0 POP R1 POP R2		.stacksize 100 Set aside 100 bytes, enough for 25 pushes, for the function call stack and initialize R13. R13 is a reserved register that holds the memory address for the next PUSH operation. Register indirect addressing is used to access values in the stack frames.

## WHERE FOR ART THOU STACK?

- The .stacksize directive:
  - Sets aside the specified number of bytes to be used for the call stack.
  - Initializes the value of R13 to the address at which the next PUSH will occur.
    - Known as the *top of the stack*.



## GENERAL PURPOSE & RESERVED REGISTERS

- R0-R11: General purpose registers
  - Use them however you want in your programs.
- R12: The Return Address when a CALL is made
  - Set by CALL, used by RET
- R13: The Stack Pointer (address for the next PUSH)
  - Used by PUSH and POP
- R14: The Return Value from a function.
  - Functions will place their return value into R14, calling code will retrieve the return value from R14.
- R15: Scratch Register
  - Used by the assembler for intermediate results. Do not use.

## AN EXAMPLE

```
main() {
    read x;
    read y;

    m = max(x,y);

    print m;
}
```

```
max(a,b) {
    x = a;
    if (b > a)
        x = b;

    return x;
}
```

## MAIN FUNCTION

```
x: .word
y: .word
m: .word

* Allocate the call stack
.stacksize 100 * 100 bytes/25 words

main() {
    read x;
    read y;

    m = max(x,y);

    print m;
}

MAIN: LOAD R0 STDIN      * read X
      STORE R0 X
      LOAD R1 STDIN      * read Y
      STORE R1 Y

      * Put parameters in stack frame for call
      PUSH R0             * X
      PUSH R1             * Y

      CALL MAX            * Transfer control to MAX

RA:   STORE R14 C          * Get the return value
      STORE R14 STDOUT

      HALT
```

## MAX FUNCTION

```
max(a,b) {
    x = a;
    if (b > a)
        x = b;

    return x;
}
```

```
MAX: PUSH R12      * + 4
     PUSH R1       * + 8
     PUSH R2       * + 12
     PUSH R3       * + 16

     LOAD R2 R13 +20 * R2 is b
     LOAD R1 R13 +24 * R1 is a

     MOV R3 R1      * R3 is x = a
     BGEQ R1 R2 SKIP * a >= b
     MOV R3 R2      * x = b
     SKIP: MOV R14 R3      * RV = x

     POP R3
     POP R2
     POP R1
     POP R12

     ADD R13 R13 #8

     RET
```