



THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

ARM Assembly: Stack Frames and Frame Pointers Demo

CPE 221: The University of Alabama
in Huntsville

Rahul Bhadani

March 16, 2025

C++ Implementation

ARM Assembly Implementation

Stack Frame Analysis During Program Execution

- ▶ ARM Assembly program with 4 nested function calls
- ▶ Assume base address of SP as 0x2000
- ▶ Each function has its own stack frame
- ▶ Visualizing stack and frame pointers during execution

Stack Organization in ARM

- ▶ Stack grows downward in memory
- ▶ SP (Stack Pointer) - Points to the current top of stack
- ▶ FP (R11) - Frame Pointer, maintains reference to current frame
- ▶ Each function:
 - ▶ Saves previous FP and LR (Link Register)
 - ▶ Sets up new frame with local variables
 - ▶ Restores previous frame on exit

C++ Implementation - Main Function

```
1  #include <iostream>
2
3  // Function prototypes
4  int firstFunction(int a);
5  int secondFunction(int b);
6  int thirdFunction(int c);
7  int fourthFunction(int d);
8
9  int main() {
10     int inputValue = 8;
11     std::cout << "Starting with value: " << inputValue << std::endl;
12     std::cout << "-----" << std::endl;
13
14     int finalResult = firstFunction(inputValue);
15
16     std::cout << "-----" << std::endl;
17     std::cout << "Final result: " << finalResult << std::endl;
18
19     return 0;
20 }
```

C++ Implementation - First Function

```
1  // First function (outermost)
2  int firstFunction(int a) {
3      // Four local variables
4      int localVar1 = a + 15;
5      int localVar2 = a * 3;
6      int localVar3 = a << 3; // Bit shift left by 3 (multiply by 8)
7      int localVar4 = 100;
8
9      // Call the second function
10     int nestedResult = secondFunction(localVar2 + a);
11
12     // Final calculation
13     int result = localVar1 * localVar4 + nestedResult - localVar3;
14
15     std::cout << "First function: " << a << " -> " << result << std::endl;
16     return result;
17 }
```

C++ Implementation - Second Function

```
1  // Second function
2  int secondFunction(int b) {
3      // Four local variables
4      int localVar1 = b << 1;  // Bit shifting (multiplication by 2)
5      int localVar2 = b + b;
6      int localVar3 = b * b;
7      int localVar4 = 12;
8
9      // Call the third function
10     int nestedResult = thirdFunction(localVar1 - localVar4);
11
12     // Some bitwise operations for variety
13     int result = nestedResult & localVar3 | localVar2;
14
15     std::cout << "Second function: " << b << " -> " << result << std::endl;
16     return result;
17 }
```

C++ Implementation - Third Function

```
1  // Third function
2  int thirdFunction(int c) {
3      // Four local variables
4      int localVar1 = c * 3;
5      int localVar2 = c - 5;
6      int localVar3 = c + 7;
7      int localVar4 = 7;
8
9      // Call the fourth function
10     int nestedResult = fourthFunction(localVar1 + localVar2);
11
12     // More math operations
13     int result = nestedResult + (localVar3 * localVar4);
14
15     std::cout << "Third function: " << c << " -> " << result << std::endl;
16     return result;
17 }
```


C++ Implementation - Fourth Function

```
1  // Fourth function (innermost)
2  int fourthFunction(int d) {
3      // Four local variables
4      int localVar1 = d * 2;
5      int localVar2 = d + 10;
6      int localVar3 = d * d;
7      int localVar4 = 5;
8
9      // Some interesting math: Compute a polynomial expression
10     int result = localVar1 * localVar3 + localVar2 * localVar4;
11
12     std::cout << "Fourth function: " << d << " -> " << result << std::endl;
13     return result;
14 }
```

ARM Assembly - Program Entry

```
1  @ ARMv7 Assembly implementation of the nested function calls program
2  @ Using SP for stack pointer and FP (R11) for frame pointer
3
4  .text
5  .global _START
6
7  _START:
8      LDR SP, =0x2000          // Initialize stack pointer
9      MOV R0, #8               // inputValue = 8
10     SUB SP, SP, #8           // Allocate space for inputValue + finalResult
11     STR R0, [SP, #4]         // Store inputValue at 0x1FFC
12     BL FIRSTFUNCTION         // Call FIRSTFUNCTION
13     STR R0, [SP]             // Store finalResult at 0x1FF8
14 DONE: B DONE                 // Infinite loop
```

ARM Assembly - First Function

```
1 FIRSTFUNCTION:
2     PUSH {FP, LR}           // Save FP and LR
3     MOV FP, SP              // FP = current SP
4     SUB SP, SP, #24         // Allocate 24 bytes for locals
5
6     // CORRECTED: Load inputValue from [FP, #12] instead of [FP, #8]
7     LDR R0, [FP, #12]       // Load inputValue from 0x1FFC
8     ADD R1, R0, #15         // localVar1 = a + 15
9     STR R1, [FP, #-24]      // Store localVar1 at [FP-24]
10
11
12     // localVar2 = a * 3
13     MOV R1, #3              // Load 3 into R1
14     MUL R1, R0, R1          // localVar2 = a * 3
15     STR R1, [FP, #-20]      // Store localVar2 at [FP-20]
16
17     // localVar3 = a << 3
18     LSL R1, R0, #3          // localVar3 = a << 3
19     STR R1, [FP, #-16]      // Store localVar3 at [FP-16]
20
21     // localVar4 = 100
22     MOV R1, #100            // localVar4 = 100
23     STR R1, [FP, #-12]      // Store localVar4 at [FP-12]
24
25     // Call SECONDFUNCTION(localVar2 + a)
26     LDR R1, [FP, #-20]      // Load localVar2
27     ADD R0, R1, R0          // R0 = localVar2 + a
28     BL SECONDFUNCTION       // Call SECONDFUNCTION
29     STR R0, [FP, #-8]       // Store nestedResult at [FP-8]
```

ARM Assembly - First Function (continued)

```
1 // Calculate final result
2     LDR R1, [FP, #-24]    // Load localVar1
3     LDR R2, [FP, #-12]    // Load localVar4
4     MUL R3, R1, R2        // R3 = localVar1 * localVar4
5     LDR R1, [FP, #-8]     // Load nestedResult
6     ADD R3, R3, R1        // R3 += nestedResult
7     LDR R1, [FP, #-16]    // Load localVar3
8     SUB R0, R3, R1        // R0 = R3 - localVar3
9     STR R0, [FP, #-4]     // Store result at [FP-4]
10    LDR R0, [FP, #-4]     // Load result into R0
11
12    MOV SP, FP            // Epilogue: Restore SP
13    POP {FP, PC}         // Restore FP and return
```

ARM Assembly - Second Function

```
1  SECONDFUNCTION:
2      PUSH {FP, LR}           // Prologue: Save FP and LR
3      MOV FP, SP              // Set FP to current SP
4      SUB SP, SP, #24         // Allocate 24 bytes for locals
5
6      // localVar1 = b << 1
7      LSL R1, R0, #1          // localVar1 = b << 1
8      STR R1, [FP, #-24]      // Store localVar1 at [FP-24]
9
10     // localVar2 = b + b
11     ADD R1, R0, R0           // localVar2 = b + b
12     STR R1, [FP, #-20]      // Store localVar2 at [FP-20]
13
14     // localVar3 = b * b
15     MUL R1, R0, R0           // localVar3 = b * b
16     STR R1, [FP, #-16]      // Store localVar3 at [FP-16]
17
18     // localVar4 = 12
19     MOV R1, #12              // localVar4 = 12
20     STR R1, [FP, #-12]      // Store localVar4 at [FP-12]
```

ARM Assembly - Second Function (continued)

```
1  // Call THIRDFUNCTION(localVar1 - localVar4)
2  LDR R1, [FP, #-24]    // Load localVar1
3  LDR R2, [FP, #-12]    // Load localVar4
4  SUB R0, R1, R2        // R0 = localVar1 - localVar4
5  BL THIRDFUNCTION      // Call THIRDFUNCTION
6  STR R0, [FP, #-8]     // Store nestedResult at [FP-8]
7
8  // Calculate result
9  LDR R1, [FP, #-8]     // Load nestedResult
10 LDR R2, [FP, #-16]    // Load localVar3
11 AND R3, R1, R2        // R3 = nestedResult & localVar3
12 LDR R1, [FP, #-20]    // Load localVar2
13 ORR R0, R3, R1        // R0 = R3 | localVar2
14 STR R0, [FP, #-4]     // Store result at [FP-4]
15 LDR R0, [FP, #-4]     // Load result into R0
16
17 MOV SP, FP            // Epilogue: Restore SP
18 POP {FP, PC}         // Restore FP and return
```

ARM Assembly - Third Function

```
1  THIRDFUNCTION:
2      PUSH {FP, LR}           // Prologue: Save FP and LR
3      MOV FP, SP              // Set FP to current SP
4      SUB SP, SP, #24         // Allocate 24 bytes for locals
5
6      // localVar1 = c * 3
7      MOV R1, #3              // Load 3 into R1
8      MUL R1, R0, R1          // localVar1 = c * 3
9      STR R1, [FP, #-24]      // Store localVar1 at [FP-24]
10
11     // localVar2 = c - 5
12     SUB R1, R0, #5           // localVar2 = c - 5
13     STR R1, [FP, #-20]      // Store localVar2 at [FP-20]
14
15     // localVar3 = c + 7
16     ADD R1, R0, #7           // localVar3 = c + 7
17     STR R1, [FP, #-16]      // Store localVar3 at [FP-16]
18
19     // localVar4 = 7
20     MOV R1, #7              // localVar4 = 7
21     STR R1, [FP, #-12]      // Store localVar4 at [FP-12]
```

ARM Assembly - Third Function (continued)

```
1 // Call FOURTHFUNCTION(localVar1 + localVar2)
2   LDR R1, [FP, #-24]    // Load localVar1
3   LDR R2, [FP, #-20]    // Load localVar2
4   ADD R0, R1, R2        // R0 = localVar1 + localVar2
5   BL FOURTHFUNCTION     // Call FOURTHFUNCTION
6   STR R0, [FP, #-8]     // Store nestedResult at [FP-8]
7
8   // Calculate result
9   LDR R1, [FP, #-16]    // Load localVar3
10  LDR R2, [FP, #-12]    // Load localVar4
11  MUL R3, R1, R2        // R3 = localVar3 * localVar4
12  LDR R1, [FP, #-8]     // Load nestedResult
13  ADD R0, R1, R3        // R0 = nestedResult + R3
14  STR R0, [FP, #-4]     // Store result at [FP-4]
15  LDR R0, [FP, #-4]     // Load result into R0
16
17  MOV SP, FP            // Epilogue: Restore SP
18  POP {FP, PC}         // Restore FP and return
```


ARM Assembly - Fourth Function

```
1  FOURTHFUNCTION:
2      PUSH {FP, LR}           // Prologue: Save FP and LR
3      MOV FP, SP              // Set FP to current SP
4      SUB SP, SP, #24         // Allocate 24 bytes for locals
5
6      // localVar1 = d * 2
7      LSL R1, R0, #1          // localVar1 = d * 2
8      STR R1, [FP, #-24]      // Store localVar1 at [FP-24]
9
10     // localVar2 = d + 10
11     ADD R1, R0, #10          // localVar2 = d + 10
12     STR R1, [FP, #-20]      // Store localVar2 at [FP-20]
13
14     // localVar3 = d * d
15     MUL R1, R0, R0           // localVar3 = d * d
16     STR R1, [FP, #-16]      // Store localVar3 at [FP-16]
17
18     // localVar4 = 5
19     MOV R1, #5               // localVar4 = 5
20     STR R1, [FP, #-12]      // Store localVar4 at [FP-12]
```

ARM Assembly - Fourth Function (continued)

```
1 // Calculate result
2     LDR R1, [FP, #-24]    // Load localVar1
3     LDR R2, [FP, #-16]    // Load localVar3
4     MUL R3, R1, R2        // R3 = localVar1 * localVar3
5     LDR R1, [FP, #-20]    // Load localVar2
6     LDR R2, [FP, #-12]    // Load localVar4
7     MUL R2, R1, R2        // R2 = localVar2 * localVar4
8     ADD R0, R3, R2        // R0 = R3 + R2
9     STR R0, [FP, #-8]     // Store result at [FP-8]
10    LDR R0, [FP, #-8]     // Load result into R0
11
12    MOV SP, FP            // Epilogue: Restore SP
13    POP {FP, PC}         // Restore FP and return
```

Program Initialization

Address	Content	Description
0x2000	SP_init	Initial stack pointer
0x1FFC	8	inputValue (R0)
0x1FF8	(empty)	Reserved for finalResult

- ▶ SP initialized to 0x2000
- ▶ SP adjusted to 0x1FF8 (SP - 8 bytes)
- ▶ R0 = 8 (inputValue) stored at [SP, #-4]
- ▶ Now ready to call firstFunction with R0 = 8

FIRSTFUNCTION Stack Frame

Address	Content	Description
0x1FF4	LR	Return address
0x1FF0	Old FP	Previous frame pointer
0x1FEC	result	Final calculation
0x1FE8	nestedResult	From SECONDFUNCTION
0x1FE4	100	localVar4
0x1FE0	64	localVar3 $a \ll 3$
0x1FDC	24	$a * 3$ localVar2
0x1FD8	23	$a + 15$ localVar1

SECONDFUNCTION Stack Frame

Address	Content	Description
0x1FD4	LR	Return to FIRSTFUNCTION
0x1FD0	Old FP	Previous frame pointer
0x1FCC		Final result
0x1FC8		nestedResult
0x1FC4		localVar4
0x1FC0		localVar3 ($b * b$)
0x1FBC		localVar2 ($b + b$)
0x1FB8		localVar1 ($b \ll 1$)

- ▶ Bitwise AND/OR operations
- ▶ Receives parameter from FIRSTFUNCTION

THIRDFUNCTION Stack Frame

Address	Content	Description
0x1FB4	LR	Return to SECONDFUNCTION
0x1FB0	Old FP	Previous frame pointer
0x1FAC		Final result
0x1FA8		nestedResult
0x1FA4		localVar4
0x1FA0		localVar3 ($c + 7$)
0x1F9C		localVar2 ($c - 5$)
0x1F98		localVar1 ($c * 3$)

► Calls FOURTHFUNCTION

FOURTHFUNCTION Stack Frame

Address	Content	Description
0x1F94	LR	Return to THIRDFUNCTION
0x1F90	Old FP	Previous frame pointer
0x1F8C		Final result
0x1F88		localVar4
0x1F84		localVar3 ($d * d$)
0x1F80		localVar2 ($d + 10$)
0x1F7C		localVar1 ($d * 2$)

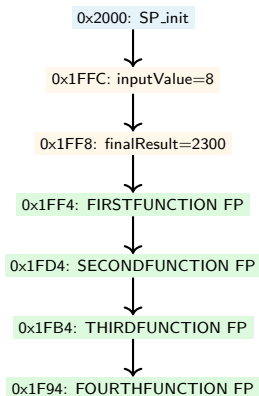
► Innermost function

Final _START Stack State

Address	Content	Description
0x2000	(SP init)	Original stack top
0x1FFC	8	Preserved inputValue
0x1FF8	2300	Final result
0x1FF4	...	Cleaned stack space

- ▶ SP returns to 0x1FF8 after functions unwind
- ▶ 8-byte allocation remains with final result
- ▶ All temporary stack frames cleaned

Complete Stack Growth Diagram



- ▶ Total stack depth: $0x2000 - 0x1F94 = 108$ bytes
- ▶ Each function frame uses 28 bytes (7×4)
- ▶ Stack unwinds completely except for final result