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

Introduction

- 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

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

C++ Implementation - Main Function

```cpp
#include <iostream>

// Function prototypes
int firstFunction(int a);
int secondFunction(int b);
int thirdFunction(int c);
int fourthFunction(int d);

int main() {
    int inputValue = 8;
    std::cout << "Starting with value: " << inputValue << std::endl;
    std::cout << "------------------------" << std::endl;

    int finalResult = firstFunction(inputValue);

    std::cout << "------------------------" << std::endl;
    std::cout << "Final result: " << finalResult << std::endl;

    return 0;
}
```

C++ Implementation - First Function

```cpp
// First function (outermost)
int firstFunction(int a) {
    // Four local variables
    int localVar1 = a + 15;
    int localVar2 = a * 3;
    int localVar3 = a << 3;  // Bit shift left by 3 (multiply by 8)
    int localVar4 = 100;

    // Call the second function
    int nestedResult = secondFunction(localVar2 + a);

    // Final calculation
    int result = localVar1 * localVar4 + nestedResult - localVar3;

    std::cout << "First function: " << a << " -> " << result << std::endl;
    return result;
}
```

C++ Implementation - Second Function

```cpp
// Second function
int secondFunction(int b) {
    // Four local variables
    int localVar1 = b << 1;  // Bit shifting (multiplication by 2)
    int localVar2 = b + b;
    int localVar3 = b * b;
    int localVar4 = 12;

    // Call the third function
    int nestedResult = thirdFunction(localVar1 - localVar4);

    // Some bitwise operations for variety
    int result = nestedResult & localVar3 | localVar2;

    std::cout << "Second function: " << b << " -> " << result << std::endl;
    return result;
}
```

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

C++ Implementation - Third Function

```cpp
// Third function
int thirdFunction(int c) {
    // Four local variables
    int localVar1 = c * 3;
    int localVar2 = c - 5;
    int localVar3 = c + 7;
    int localVar4 = 7;

    // Call the fourth function
    int nestedResult = fourthFunction(localVar1 + localVar2);

    // More math operations
    int result = nestedResult + (localVar3 * localVar4);

    std::cout << "Third function: " << c << " -> " << result << std::endl;
    return result;
}
```

C++ Implementation - Fourth Function

```cpp
// Fourth function (innermost)
int fourthFunction(int d) {
    // Four local variables
    int localVar1 = d * 2;
    int localVar2 = d + 10;
    int localVar3 = d * d;
    int localVar4 = 5;

    // Some interesting math: Compute a polynomial expression
    int result = localVar1 * localVar3 + localVar2 * localVar4;

    std::cout << "Fourth function: " << d << " -> " << result << std::endl;
    return result;
}
```

ARM Assembly - Program Entry

```
@ ARMv7 Assembly implementation of the nested function calls program
@ Using SP for stack pointer and FP (R11) for frame pointer

.text
.global _START

_START:
    LDR SP, =0x2000        // Initialize stack pointer
    MOV R0, #8             // inputValue = 8
    SUB SP, SP, #8         // Allocate space for inputValue + finalResult
    STR R0, [SP, #4]       // Store inputValue at 0x1FFC
    BL FIRSTFUNCTION       // Call FIRSTFUNCTION
    STR R0, [SP]           // Store finalResult at 0x1FF8
DONE:  B DONE              // Infinite loop
```

ARM Assembly - First Function

```
FIRSTFUNCTION:
    PUSH {FP, LR}           // Save FP and LR
    MOV FP, SP              // FP = current SP
    SUB SP, SP, #24         // Allocate 24 bytes for locals

    // CORRECTED: Load inputValue from [FP, #12] instead of [FP, #8]
    LDR R0, [FP, #12]       // Load inputValue from 0x1FFC
    ADD R1, R0, #15         // localVar1 = a + 15
    STR R1, [FP, #-24]      // Store localVar1 at [FP-24]


    // localVar2 = a * 3
    MOV R1, #3              // Load 3 into R1
    MUL R1, R0, R1          // localVar2 = a * 3
    STR R1, [FP, #-20]      // Store localVar2 at [FP-20]

    // localVar3 = a << 3
    LSL R1, R0, #3          // localVar3 = a << 3
    STR R1, [FP, #-16]      // Store localVar3 at [FP-16]

    // localVar4 = 100
    MOV R1, #100            // localVar4 = 100
    STR R1, [FP, #-12]      // Store localVar4 at [FP-12]

    // Call SECONDFUNCTION(localVar2 + a)
    LDR R1, [FP, #-20]      // Load localVar2
    ADD R0, R1, R0          // R0 = localVar2 + a
    BL SECONDFUNCTION       // Call SECONDFUNCTION
    STR R0, [FP, #-8]       // Store nestedResult at [FP-8]
```

ARM Assembly - First Function (continued)

```
// Calculate final result
    LDR R1, [FP, #-24]     // Load localVar1
    LDR R2, [FP, #-12]     // Load localVar4
    MUL R3, R1, R2         // R3 = localVar1 * localVar4
    LDR R1, [FP, #-8]      // Load nestedResult
    ADD R3, R3, R1         // R3 += nestedResult
    LDR R1, [FP, #-16]     // Load localVar3
    SUB R0, R3, R1         // R0 = R3 - localVar3
    STR R0, [FP, #-4]      // Store result at [FP-4]
    LDR R0, [FP, #-4]      // Load result into R0

    MOV SP, FP             // Epilogue: Restore SP
    POP {FP, PC}           // Restore FP and return
```

ARM Assembly - Second Function

```
SECONDFUNCTION:
    PUSH {FP, LR}          // Prologue: Save FP and LR
    MOV FP, SP             // Set FP to current SP
    SUB SP, SP, #24        // Allocate 24 bytes for locals

    // localVar1 = b << 1
    LSL R1, R0, #1         // localVar1 = b << 1
    STR R1, [FP, #-24]     // Store localVar1 at [FP-24]

    // localVar2 = b + b
    ADD R1, R0, R0         // localVar2 = b + b
    STR R1, [FP, #-20]     // Store localVar2 at [FP-20]

    // localVar3 = b * b
    MUL R1, R0, R0         // localVar3 = b * b
    STR R1, [FP, #-16]     // Store localVar3 at [FP-16]

    // localVar4 = 12
    MOV R1, #12            // localVar4 = 12
    STR R1, [FP, #-12]     // Store localVar4 at [FP-12]
```

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

ARM Assembly - Second Function (continued)

```
// Call THIRDFUNCTION(localVar1 - localVar4)
    LDR R1, [FP, #-24]     // Load localVar1
    LDR R2, [FP, #-12]     // Load localVar4
    SUB R0, R1, R2         // R0 = localVar1 - localVar4
    BL THIRDFUNCTION       // Call THIRDFUNCTION
    STR R0, [FP, #-8]      // Store nestedResult at [FP-8]

    // Calculate result
    LDR R1, [FP, #-8]      // Load nestedResult
    LDR R2, [FP, #-16]     // Load localVar3
    AND R3, R1, R2         // R3 = nestedResult & localVar3
    LDR R1, [FP, #-20]     // Load localVar2
    ORR R0, R3, R1         // R0 = R3 | localVar2
    STR R0, [FP, #-4]      // Store result at [FP-4]
    LDR R0, [FP, #-4]      // Load result into R0

    MOV SP, FP             // Epilogue: Restore SP
    POP {FP, PC}           // Restore FP and return
```

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

ARM Assembly - Third Function

```
THIRDFUNCTION:
    PUSH {FP, LR}          // Prologue: Save FP and LR
    MOV FP, SP             // Set FP to current SP
    SUB SP, SP, #24        // Allocate 24 bytes for locals

    // localVar1 = c * 3
    MOV R1, #3             // Load 3 into R1
    MUL R1, R0, R1         // localVar1 = c * 3
    STR R1, [FP, #-24]     // Store localVar1 at [FP-24]

    // localVar2 = c - 5
    SUB R1, R0, #5         // localVar2 = c - 5
    STR R1, [FP, #-20]     // Store localVar2 at [FP-20]

    // localVar3 = c + 7
    ADD R1, R0, #7         // localVar3 = c + 7
    STR R1, [FP, #-16]     // Store localVar3 at [FP-16]

    // localVar4 = 7
    MOV R1, #7             // localVar4 = 7
    STR R1, [FP, #-12]     // Store localVar4 at [FP-12]
```

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

ARM Assembly - Third Function (continued)

```
// Call FOURTHFUNCTION(localVar1 + localVar2)
    LDR R1, [FP, #-24]     // Load localVar1
    LDR R2, [FP, #-20]     // Load localVar2
    ADD R0, R1, R2         // R0 = localVar1 + localVar2
    BL FOURTHFUNCTION      // Call FOURTHFUNCTION
    STR R0, [FP, #-8]      // Store nestedResult at [FP-8]

    // Calculate result
    LDR R1, [FP, #-16]     // Load localVar3
    LDR R2, [FP, #-12]     // Load localVar4
    MUL R3, R1, R2         // R3 = localVar3 * localVar4
    LDR R1, [FP, #-8]      // Load nestedResult
    ADD R0, R1, R3         // R0 = nestedResult + R3
    STR R0, [FP, #-4]      // Store result at [FP-4]
    LDR R0, [FP, #-4]      // Load result into R0

    MOV SP, FP             // Epilogue: Restore SP
    POP {FP, PC}           // Restore FP and return
```

ARM Assembly - Fourth Function

```
FOURTHFUNCTION:
    PUSH {FP, LR}          // Prologue: Save FP and LR
    MOV FP, SP             // Set FP to current SP
    SUB SP, SP, #24        // Allocate 24 bytes for locals

    // localVar1 = d * 2
    LSL R1, R0, #1         // localVar1 = d * 2
    STR R1, [FP, #-24]     // Store localVar1 at [FP-24]

    // localVar2 = d + 10
    ADD R1, R0, #10        // localVar2 = d + 10
    STR R1, [FP, #-20]     // Store localVar2 at [FP-20]

    // localVar3 = d * d
    MUL R1, R0, R0         // localVar3 = d * d
    STR R1, [FP, #-16]     // Store localVar3 at [FP-16]

    // localVar4 = 5
    MOV R1, #5             // localVar4 = 5
    STR R1, [FP, #-12]     // Store localVar4 at [FP-12]
```

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

ARM Assembly - Fourth Function (continued)

```
// Calculate result
    LDR R1, [FP, #-24]     // Load localVar1
    LDR R2, [FP, #-16]     // Load localVar3
    MUL R3, R1, R2         // R3 = localVar1 * localVar3
    LDR R1, [FP, #-20]     // Load localVar2
    LDR R2, [FP, #-12]     // Load localVar4
    MUL R2, R1, R2         // R2 = localVar2 * localVar4
    ADD R0, R3, R2         // R0 = R3 + R2
    STR R0, [FP, #-8]      // Store result at [FP-8]
    LDR R0, [FP, #-8]      // Load result into R0

    MOV SP, FP             // Epilogue: Restore SP
    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<<3
0x1FDC	24	a*3 localVar2
0x1FD8	23	a+15 localVar1

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

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 <<1)

- Bitwise AND/OR operations
- Receives parameter from FIRSTFUNCTION

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

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

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

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

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

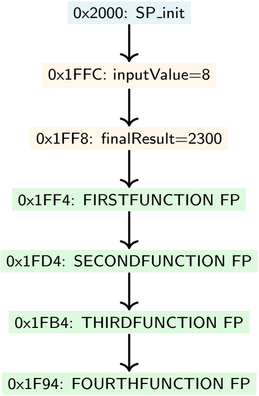
Final _START Stack State

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

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Complete Stack Growth Diagram



0x2000: SP_init

0x1FFC: inputValue=8

0x1FF8: finalResult=2300

0x1FF4: FIRSTFUNCTION FP

0x1FD4: SECONDFUNCTION FP

0x1FB4: THIRDFUNCTION FP

0x1F94: FOURTHFUNCTION FP

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