

# Tutorial-9: Parameters and Stack Frames

AbdulWahab Kabani



[ 1 ]

## Objectives

- Review Passing Parameters to a subroutine
- Difference Between Passing by value and by reference
- Defining a dynamic workspace for a subroutine (stack frame).



[ 2 ]

## Outline

- Passing Parameters
- Value vs. Reference
- Passing By Value (revision)
- Passing By Reference
- Stack Frame
- Conclusion



[ 3 ]

## Outline

- **Passing Parameters**
- Value vs. Reference
- Passing By Value (revision)
- Passing By Reference
- Stack Frame
- Conclusion



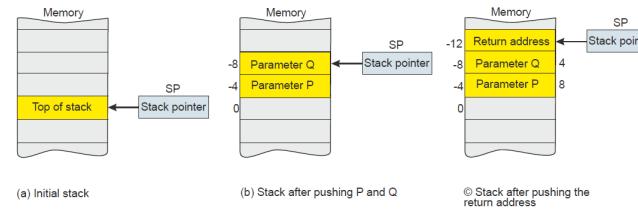
[ 4 ]

## Passing Parameters

- **Question:** How do we pass parameters to a subroutine?

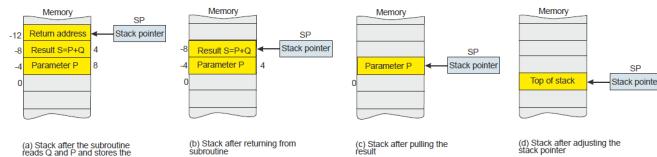


## Passing Parameters



[ 6 ]

## Passing Parameters

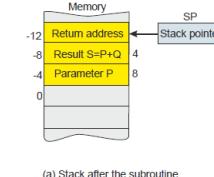


[ 7 ]

## Passing Parameters

- **Steps:**

1. In the caller, push parameters to the stack
2. In the caller, Push Return Address
3. In the subroutine, calculate subroutine result
4. In the subroutine, return the value
5. In the subroutine, change the value of PC (return to call)



## Outline

- **Passing Parameters**
- **Value vs. Reference**
- Passing By Value (revision)
- Passing By Reference
- Stack Frame
- Conclusion



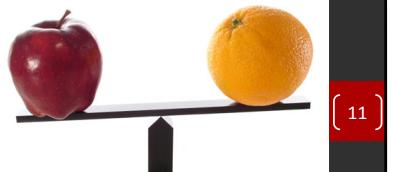
## Value vs. Reference

- **Question:** what is the difference between passing parameters by value and passing by reference?



## Value vs. Reference

- Passing By Value:
  - anything passed into a function call is unchanged in the caller's scope
- Passing By Reference:
  - A reference (memory address) to a variable is passed
  - The function can modify the parameter



## Outline

- **Passing Parameters**
- **Value vs. Reference**
- **Passing By Value (revision)**
- Passing By Reference
- Stack Frame
- Conclusion



## Passing By Value

```

1 AREA Parameter, CODE, READWRITE :make readable because of the stack
2 ADD sp,$base ;point to the base of the stack
3 MOV r1,$base ;dummy value for Q in r1
4 MOV r2,$base ;dummy value for P in r2
5 LDR r3,[sp] ;push the link register on the stack
6 STP r1,[sp,-4]! ;push Q
7 ADD r2,[sp,-4]! ;push P
8 LDR r4,[sp] ;pull the stack pointer
9 ADD sp,r4 ;adjust the stack pointer
10
11 Loop :Lloop
12 ADD r5,[sp,-4]! ;push the link register on the stack
13 STP r1,[sp,-4]! ;push Q (dusted under the return address and Q)
14 LDR r6,[sp] ;push the link register on the stack
15 STP r2,[sp,-4]! ;push P (dusted under the return address)
16 ADD r7,[sp,-4]! ;push the addition
17 STP r3,[sp,-4]! ;push the stack under return address (overwrite Q)
18 LDR r8,[sp] ;pull return address off the stack
19 ADD sp,r8 ;adjust the stack pointer
20
21 Base :DCD $AAAAAA,0 ;stack area
22 Base :DCD $AAAAAA,0 ;stack area and dummy data as marker

```

(13)

## Passing By Value

```

1 AREA Parameter, CODE, READWRITE :make readable because of the stack
2 ADD sp,$base ;point to the base of the stack
3 MOV r1,$base ;dummy value for Q in r1
4 MOV r2,$base ;dummy value for P in r2
5 LDR r3,[sp] ;push the link register on the stack
6 STP r1,[sp,-4]! ;push Q
7 ADD r2,[sp,-4]! ;push P
8 LDR r4,[sp] ;pull the stack pointer
9 ADD sp,r4 ;adjust the stack pointer
10
11 Loop :Lloop
12 ADD r5,[sp,-4]! ;push the link register on the stack
13 STP r1,[sp,-4]! ;push Q (dusted under the return address and Q)
14 LDR r6,[sp] ;push the link register on the stack
15 STP r2,[sp,-4]! ;push P (dusted under the return address)
16 ADD r7,[sp,-4]! ;push the addition
17 STP r3,[sp,-4]! ;push the stack under return address (overwrite Q)
18 LDR r8,[sp] ;pull return address off the stack
19 ADD sp,r8 ;adjust the stack pointer
20
21 Base :DCD $AAAAAA,0 ;stack area
22 Base :DCD $AAAAAA,0 ;stack area and dummy data as marker

```

(14)

## Passing By Value

```

1 AREA Parameter, CODE, READWRITE :make readable because of the stack
2 ADD sp,$base ;point to the base of the stack
3 MOV r1,$base ;dummy value for Q in r1
4 MOV r2,$base ;dummy value for P in r2
5 LDR r3,[sp] ;push the link register on the stack
6 STP r1,[sp,-4]! ;push Q
7 ADD r2,[sp,-4]! ;push P
8 LDR r4,[sp] ;pull the addition
9 ADD sp,r4 ;adjust the stack pointer
10
11 Loop :Lloop
12 ADD r5,[sp,-4]! ;push the link register on the stack
13 STP r1,[sp,-4]! ;push Q (dusted under the return address and Q)
14 LDR r6,[sp] ;push the link register on the stack
15 STP r2,[sp,-4]! ;push P (dusted under the return address)
16 ADD r7,[sp,-4]! ;push the addition
17 STP r3,[sp,-4]! ;push the stack under return address (overwrite Q)
18 LDR r8,[sp] ;pull return address off the stack
19 ADD sp,r8 ;adjust the stack pointer
20
21 Base :DCD $AAAAAA,0 ;stack area
22 Base :DCD $AAAAAA,0 ;stack area and dummy data as marker

```

(15)

## Passing By Value

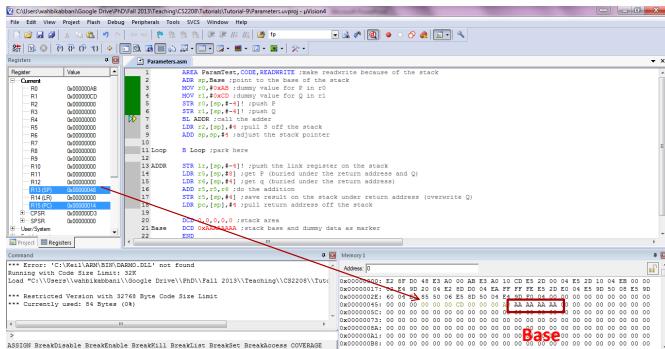
```

1 AREA Parameter, CODE, READWRITE :make readable because of the stack
2 ADD sp,$base ;point to the base of the stack
3 MOV r1,$base ;dummy value for Q in r1
4 MOV r2,$base ;dummy value for P in r2
5 LDR r3,[sp] ;push the link register on the stack
6 STP r1,[sp,-4]! ;push Q
7 ADD r2,[sp,-4]! ;push P
8 LDR r4,[sp] ;pull the addition
9 ADD sp,r4 ;adjust the stack pointer
10
11 Loop :Lloop
12 ADD r5,[sp,-4]! ;push the link register on the stack
13 STP r1,[sp,-4]! ;push Q (dusted under the return address and Q)
14 LDR r6,[sp] ;push the link register on the stack
15 STP r2,[sp,-4]! ;push P (dusted under the return address)
16 ADD r7,[sp,-4]! ;push the addition
17 STP r3,[sp,-4]! ;push the stack under return address (overwrite Q)
18 LDR r8,[sp] ;pull return address off the stack
19 ADD sp,r8 ;adjust the stack pointer
20
21 Base :DCD $AAAAAA,0 ;stack area
22 Base :DCD $AAAAAA,0 ;stack area and dummy data as marker

```

(16)

## Passing By Value



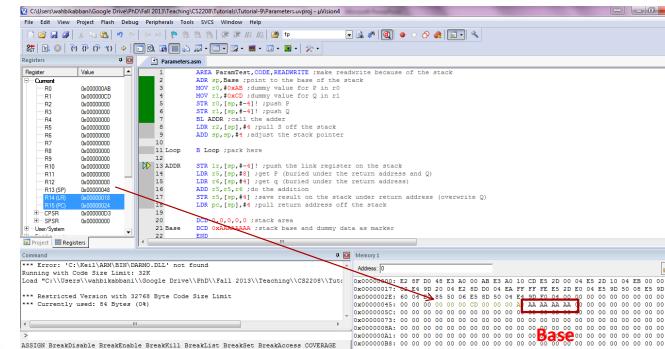
```

1 AREA Parameters,CODE,READWRITE :make read/write because of the stack
2 ADD sp,$base ;point to the base of the stack
3 MOV r0,$base ;dummy value for Q in r0
4 MOV r1,$base ;dummy value for P in r1
5 ADD r2,r0,r1 ;add the two values
6 STP r1,[sp,$base+4]! ;push Q
7 BL ADD ;call the adder
8 LDR r1,[sp,$base+4]! ;pull result on the stack under return address (overwrite Q)
9 ADD r1,r0,r2 ;do the addition
10 ADD sp,$r1,4! ;pull return address off the stack
11 ADD sp,$r1,4! ;adjust the stack pointer
12 Lop :spark here
13 ADDQ r1,[sp,$base+4]! ;push the link register on the stack
14 LDR r1,[sp,$base+4]! ;load the link register under the return address and Q
15 ADD r1,r0,r1 ;add the addition
16 ADD r1,r1,r2 ;do the addition
17 STP r1,[sp,$base+4]! ;push result on the stack under return address (overwrite Q)
18 LDR r1,[sp,$base+4]! ;pull return address off the stack
19 ADD r1,r0,r2 ;do the addition
20 ADD r1,r1,r2 ;do the addition
21 Base DCD 0x00000000 ;stack base and dummy data as marker
22 End

```

[17]

## Passing By Value



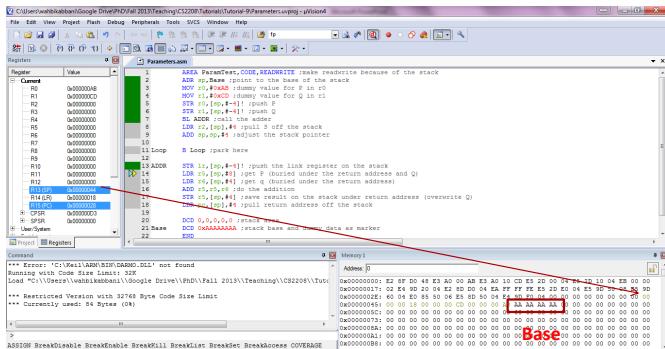
```

1 AREA Parameters,CODE,READWRITE :make read/write because of the stack
2 ADD sp,$base ;point to the base of the stack
3 MOV r1,$base ;dummy value for Q in r1
4 MOV r2,$base ;dummy value for P in r2
5 ADD r3,r1,r2 ;add the two values
6 STP r1,[sp,$base+4]! ;push Q
7 BL ADD ;call the adder
8 LDR r1,[sp,$base+4]! ;pull result on the stack under return address (overwrite Q)
9 ADD r1,r1,r3 ;do the addition
10 ADD sp,$r1,4! ;pull return address off the stack
11 ADD sp,$r1,4! ;adjust the stack pointer
12 Lop :spark here
13 ADDQ r1,[sp,$base+4]! ;push the link register on the stack
14 LDR r1,[sp,$base+4]! ;load the link register under the return address and Q
15 ADD r1,r1,r3 ;do the addition
16 ADD r1,r1,r3 ;do the addition
17 STP r1,[sp,$base+4]! ;push result on the stack under return address (overwrite Q)
18 LDR r1,[sp,$base+4]! ;pull return address off the stack
19 ADD r1,r1,r3 ;do the addition
20 ADD r1,r1,r3 ;do the addition
21 Base DCD 0x00000000 ;stack base and dummy data as marker
22 End

```

[18]

## Passing By Value



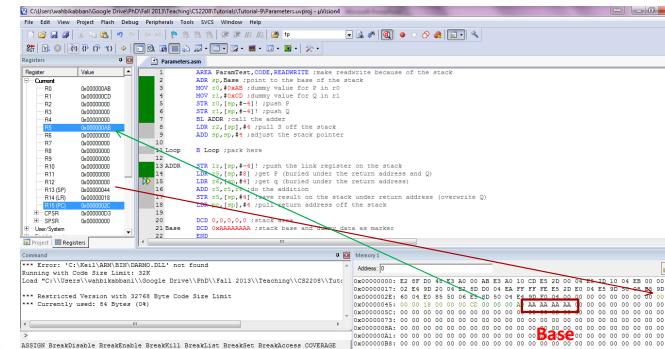
```

1 AREA Parameters,CODE,READWRITE :make read/write because of the stack
2 ADD sp,$base ;point to the base of the stack
3 MOV r0,$base ;dummy value for P in r0
4 MOV r1,$base ;dummy value for Q in r1
5 ADD r2,r0,r1 ;add the two values
6 STP r1,[sp,$base+4]! ;push Q
7 BL ADD ;call the adder
8 LDR r1,[sp,$base+4]! ;pull result on the stack under return address (overwrite Q)
9 ADD r1,r0,r2 ;do the addition
10 ADD sp,$r1,4! ;pull return address off the stack
11 ADD sp,$r1,4! ;adjust the stack pointer
12 Lop :spark here
13 ADDQ r1,[sp,$base+4]! ;push the link register on the stack
14 LDR r1,[sp,$base+4]! ;load the link register under the return address and Q
15 ADD r1,r0,r1 ;add the addition
16 ADD r1,r1,r2 ;do the addition
17 STP r1,[sp,$base+4]! ;push result on the stack under return address (overwrite Q)
18 LDR r1,[sp,$base+4]! ;pull return address off the stack
19 ADD r1,r0,r2 ;do the addition
20 ADD r1,r1,r2 ;do the addition
21 Base DCD 0x00000000 ;stack base and dummy data as marker
22 End

```

[19]

## Passing By Value



```

1 AREA Parameters,CODE,READWRITE :make read/write because of the stack
2 ADD sp,$base ;point to the base of the stack
3 MOV r1,$base ;dummy value for Q in r1
4 MOV r2,$base ;dummy value for P in r2
5 ADD r3,r1,r2 ;add the two values
6 STP r1,[sp,$base+4]! ;push Q
7 BL ADD ;call the adder
8 LDR r1,[sp,$base+4]! ;pull result on the stack under return address (overwrite Q)
9 ADD r1,r1,r3 ;do the addition
10 ADD sp,$r1,4! ;pull return address off the stack
11 ADD sp,$r1,4! ;adjust the stack pointer
12 Lop :spark here
13 ADDQ r1,[sp,$base+4]! ;push the link register on the stack
14 LDR r1,[sp,$base+4]! ;load the link register under the return address and Q
15 ADD r1,r1,r3 ;do the addition
16 ADD r1,r1,r3 ;do the addition
17 STP r1,[sp,$base+4]! ;push result on the stack under return address (overwrite Q)
18 LDR r1,[sp,$base+4]! ;pull return address off the stack
19 ADD r1,r1,r3 ;do the addition
20 ADD r1,r1,r3 ;do the addition
21 Base DCD 0x00000000 ;stack base and dummy data as marker
22 End

```

[20]

## Passing By Value

```

1 AREA Parameters, CODE, READWRITE :make readable because of the stack
2 ADD sp,$base, rpoint to the base of the stack
3 MOV r0,$base
4 MOV r1,$Q    ; dummy value for Q in r1
5 ADD r0,r1,r0
6 STP r0,[sp,-4]! push P
7 STP r1,[sp,-4]! push Q
8 BL ADD    ;call the adder
9 LDR r0,[sp]    ; pull result off the stack
10 ADD sp,r0,4#  ;pull 8 off the stack
11 ADD sp,r0,4#  ;adjust the stack pointer
12 Loop    ; loop
13 STP lr,[sp,-4]! push the link register on the stack
14 ADD r0,r1,r0
15 ADD r0,r1,r0
16 ADD r0,r1,r0
17 ADD r0,r1,r0
18 ADD r0,r1,r0
19 ADD r0,r1,r0
20 ADD r0,r1,r0
21 Base    ; stack has some dummy data as marker
22 End

```

(21)

## Passing By Value

```

1 AREA Parameters, CODE, READWRITE :make readable because of the stack
2 ADD sp,$base, rpoint to the base of the stack
3 MOV r1,$Q    ; dummy value for Q in r1
4 ADD r0,r1,r0
5 STP r1,[sp,-4]! push Q
6 ADD r0,r1,r0
7 STP r1,[sp,-4]! push P
8 BL ADD    ;call the adder
9 LDR r0,[sp]    ; pull result off the stack
10 ADD sp,r0,4#  ;pull 8 off the stack
11 ADD sp,r0,4#  ;adjust the stack pointer
12 Loop    ; loop
13 STP lr,[sp,-4]! push the link register on the stack
14 ADD r0,r1,r0
15 ADD r0,r1,r0
16 ADD r0,r1,r0
17 ADD r0,r1,r0
18 ADD r0,r1,r0
19 ADD r0,r1,r0
20 ADD r0,r1,r0
21 Base    ; stack has some dummy data as marker
22 End

```

(22)

## Passing By Value

```

1 AREA Parameters, CODE, READWRITE :make readable because of the stack
2 ADD sp,$base, rpoint to the base of the stack
3 MOV r0,$base
4 MOV r1,$Q    ; dummy value for Q in r1
5 ADD r0,r1,r0
6 STP r0,[sp,-4]! push P
7 STP r1,[sp,-4]! push Q
8 BL ADD    ;call the adder
9 LDR r0,[sp]    ; pull result off the stack
10 ADD sp,r0,4#  ;pull 8 off the stack
11 ADD sp,r0,4#  ;adjust the stack pointer
12 Loop    ; loop
13 STP lr,[sp,-4]! push the link register on the stack
14 ADD r0,r1,r0
15 ADD r0,r1,r0
16 ADD r0,r1,r0
17 ADD r0,r1,r0
18 ADD r0,r1,r0
19 ADD r0,r1,r0
20 ADD r0,r1,r0
21 Base    ; stack has some dummy data as marker
22 End

```

(23)

## Passing By Value

```

1 AREA Parameters, CODE, READWRITE :make readable because of the stack
2 ADD sp,$base, rpoint to the base of the stack
3 MOV r1,$Q    ; dummy value for Q in r1
4 ADD r0,r1,r0
5 STP r1,[sp,-4]! push Q
6 ADD r0,r1,r0
7 STP r1,[sp,-4]! push P
8 BL ADD    ;call the adder
9 LDR r0,[sp]    ; pull result off the stack
10 ADD sp,r0,4#  ;pull 8 off the stack
11 ADD sp,r0,4#  ;adjust the stack pointer
12 Loop    ; loop
13 STP lr,[sp,-4]! push the link register on the stack
14 ADD r0,r1,r0
15 ADD r0,r1,r0
16 ADD r0,r1,r0
17 ADD r0,r1,r0
18 ADD r0,r1,r0
19 ADD r0,r1,r0
20 ADD r0,r1,r0
21 Base    ; stack has some dummy data as marker
22 End

```

(24)

## Passing By Value

```

1 AREA Parameter, CODE, READWRITE :make readable because of the stack
2 ADD sp, #0x00000000 ;push the base pointer to the base of the stack
3 MOV r1, #0x000000CD ;MOV r1,#0x0000CD dummy value for Q in r1
4 LDR r2, [sp, #0x00000000] ;LDR r2,[sp] push r2 onto the stack
5 STM r1, [sp, #0x00000004]! ;STM r1,[sp, #0x00000004]! push r1
6 LDR r3, [sp, #0x00000004] ;LDR r3,[sp] push r3 onto the stack
7 ADD sp, #0x00000004 ;ADD sp,#0x00000004#4 pull $4 off the stack
8 ADD sp, #0x00000004 ;ADD sp,#0x00000004#4 adjust the stack pointer
9
10
11 Loop B Loop ;loop here
12 ADDQ r4, [sp, #0x00000004]! ;push the link register on the stack
13 STM r4, [sp, #0x00000004]! ;STM r4,[sp, #0x00000004]! push r4 (handed under the return address and Q)
14 LDR r5, [sp, #0x00000004] ;LDR r5,[sp] rget r5 (handed under the return address)
15 STM r5, [sp, #0x00000004]! ;STM r5,[sp, #0x00000004]! push r5 result on the stack under return address (overwrite Q)
16 LDR r6, [sp, #0x00000004] ;LDR r6,[sp] pull return address off the stack
17 ADD sp, #0x00000004 ;ADD sp,#0x00000004#4 pull $4 off the stack
18 ADD sp, #0x00000004 ;ADD sp,#0x00000004#4 adjust the stack pointer
19
20
21 Base DCD 0x00000000 ;set base and dummy data as marker
22 END

```

[ 25 ]

## Passing By Value

```

1 AREA Parameter, CODE, READWRITE :make readable because of the stack
2 ADD sp, #0x00000000 ;push the base pointer to the base of the stack
3 MOV r1, #0x000000CD ;MOV r1,#0x0000CD dummy value for Q in r1
4 LDR r2, [sp, #0x00000000] ;LDR r2,[sp] push r2 onto the stack
5 STM r1, [sp, #0x00000004]! ;STM r1,[sp, #0x00000004]! push r1
6 LDR r3, [sp, #0x00000004] ;LDR r3,[sp] push r3 onto the stack
7 ADD sp, #0x00000004 ;ADD sp,#0x00000004#4 pull $4 off the stack
8 ADD sp, #0x00000004 ;ADD sp,#0x00000004#4 adjust the stack pointer
9
10
11 Loop B Loop ;loop here
12 ADDQ r4, [sp, #0x00000004]! ;push the link register on the stack
13 STM r4, [sp, #0x00000004]! ;STM r4,[sp, #0x00000004]! push r4 (handed under the return address and Q)
14 LDR r5, [sp, #0x00000004] ;LDR r5,[sp] rget r5 (handed under the return address)
15 STM r5, [sp, #0x00000004]! ;STM r5,[sp, #0x00000004]! push r5 result on the stack under return address (overwrite Q)
16 LDR r6, [sp, #0x00000004] ;LDR r6,[sp] pull return address off the stack
17 ADD sp, #0x00000004 ;ADD sp,#0x00000004#4 pull $4 off the stack
18 ADD sp, #0x00000004 ;ADD sp,#0x00000004#4 adjust the stack pointer
19
20
21 Base DCD 0x00000000 ;set base and dummy data as marker
22 END

```

[ 26 ]

## Outline

- Passing Parameters
- Value vs. Reference
- Passing By Value (revision)
- Passing By Reference
- Stack Frame
- Conclusion



[ 27 ]

## Passing By Reference

- The function can modify/change the parameter passed to it
- Can be done by passing a reference (memory address) to the parameter



[ 28 ]

## Passing By Reference

```

1 AREA PassByRef, CODE, READWRITE : Make readable because we locate the stack in this area
2     ADD sp,Base ; Point to the base of the stack
3     ADD sp,F ; Load r5 with the address of parameter F
4     STW sp,[sp+4] ; Push the address of parameter F on the stack (address is in r0)
5     BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r1,r2,r3 ; Infinite loop to end the program
8
9 ABC
10    LDR r1,[sp] ; Save the return address on the stack
11    LDR r2,[sp+4] ; Load r2 with the address of parameter F under the return address
12    ADD r3,r2,r1 ; Add 1 to r2
13    STW r2,[r1+4] ; Save the parameter in the calling environment
14    LDMFD sp!,{r0} ; Return by loading the r0 with the return address from the stack
15
16 P   DCD 0x12345678 ; Location of parameter F and its value
17
18 Base DCD 0x43210000,0,0,0,0 ; Stack area
19 Base DCD 0x43210000,0,0,0,0 ; Stack area and dummy data
20

```

[ 29 ]

## Passing By Reference

```

1 AREA PassByRef, CODE, READWRITE : Make readable because we locate the stack in this area
2     ADD sp,Base ; Point to the base of the stack
3     ADD sp,F ; Load r5 with the address of parameter F
4     STW sp,[sp+4] ; Push the address of parameter F on the stack (address is in r0)
5     BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r1,r2,r3 ; Infinite loop to end the program
8
9 ABC
10    LDR r1,[sp] ; Save the return address on the stack
11    LDR r2,[sp+4] ; Load r2 with the address of parameter F under the return address
12    ADD r3,r2,r1 ; Add 1 to r2
13    STW r2,[r1+4] ; Save the parameter in the calling environment
14    LDMFD sp!,{r0} ; Return by loading the r0 with the return address from the stack
15
16 P   DCD 0x12345678 ; Location of parameter F and its value
17
18 Base DCD 0x43210000,0,0,0,0 ; Stack area
19 Base DCD 0x43210000,0,0,0,0 ; Stack area and dummy data
20

```

Param

[ 30 ]

## Passing By Reference

```

1 AREA PassByRef, CODE, READWRITE : Make readable because we locate the stack in this area
2     ADD sp,Base ; Point to the base of the stack
3     ADD sp,F ; Load r5 with the address of parameter F
4     STW sp,[sp+4] ; Push the address of parameter F on the stack (address is in r0)
5     BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r1,r2,r3 ; Infinite loop to end the program
8
9 ABC
10    LDR r1,[sp] ; Save the return address on the stack
11    LDR r2,[sp+4] ; Load r2 with the address of parameter F under the return address
12    ADD r3,r2,r1 ; Add 1 to r2
13    STW r2,[r1+4] ; Save the parameter in the calling environment
14    LDMFD sp!,{r0} ; Return by loading the r0 with the return address from the stack
15
16 P   DCD 0x12345678 ; Location of parameter F and its value
17
18 Base DCD 0x43210000,0,0,0,0 ; Stack area
19 Base DCD 0x43210000,0,0,0,0 ; Stack area and dummy data
20

```

Param

[ 31 ]

## Passing By Reference

```

1 AREA PassByRef, CODE, READWRITE : Make readable because we locate the stack in this area
2     ADD sp,Base ; Point to the base of the stack
3     ADD sp,F ; Load r5 with the address of parameter F
4     STW sp,[sp+4] ; Push the address of parameter F on the stack (address is in r0)
5     BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r1,r2,r3 ; Infinite loop to end the program
8
9 ABC
10    LDR r1,[sp] ; Save the return address on the stack
11    LDR r2,[sp+4] ; Load r2 with the address of parameter F under the return address
12    ADD r3,r2,r1 ; Add 1 to r2
13    STW r2,[r1+4] ; Save the parameter in the calling environment
14    LDMFD sp!,{r0} ; Return by loading the r0 with the return address from the stack
15
16 P   DCD 0x12345678 ; Location of parameter F and its value
17
18 Base DCD 0x43210000,0,0,0,0 ; Stack area
19 Base DCD 0x43210000,0,0,0,0 ; Stack area and dummy data
20

```

Param

[ 32 ]

## Passing By Reference

```

1 AREA ParamDef,CODE,READWRITE : Make readable because we locate the stack in this area
2 ADD sp,Base ; Point to the base of the stack
3 LDR r0,=Param ; Load r0 with the address of parameter P
4 STP r0,[sp,-4]! ; Push the address of parameter P on the stack (address is in r0)
5 BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r0,r0,r0 ; Infinite loop to end the program
8
9 LDR r1,[sp] ; Save the return address on the stack
10 LDR r1,r1,=4 ; Get the 4 bytes of parameter P
11 ADD r1,r1,r1 ; Add 1 to P
12 ADD r1,r1,r1 ; Add 1 to P
13 ADD r1,r1,r1 ; Add 1 to P
14 ADD r1,r1,r1 ; Add 1 to P
15 LDMFD sp!,{r0} ; Return by loading the r0 in the return address from the stack
16 P DCD 0x12345678 ; Location of parameter P and its value
17
18 Base DCD 0xFFFFFFFF,0,0,0,0 ; Stack base and dummy data
19
20 END

```

(33)

## Passing By Reference

```

1 AREA ParamDef,CODE,READWRITE : Make readable because we locate the stack in this area
2 ADD sp,Base ; Point to the base of the stack
3 LDR r0,=Param ; Load r0 with the address of parameter P
4 STP r0,[sp,-4]! ; Push the address of parameter P on the stack (address is in r0)
5 BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r0,r0,r0 ; Infinite loop to end the program
8
9 LDR r1,[sp] ; Save the return address on the stack
10 LDR r1,r1,=4 ; Get the 4 bytes of parameter P
11 ADD r1,r1,r1 ; Add 1 to P
12 ADD r1,r1,r1 ; Add 1 to P
13 ADD r1,r1,r1 ; Add 1 to P
14 ADD r1,r1,r1 ; Add 1 to P
15 LDMFD sp!,{r0} ; Return by loading the r0 in the return address from the stack
16 P DCD 0x12345678 ; Location of parameter P and its value
17
18 Base DCD 0xFFFFFFFF,0,0,0,0 ; Stack base and dummy data
19
20 END

```

(34)

## Passing By Reference

```

1 AREA ParamDef,CODE,READWRITE : Make readable because we locate the stack in this area
2 ADD sp,Base ; Point to the base of the stack
3 LDR r0,=Param ; Load r0 with the address of parameter P
4 STP r0,[sp,-4]! ; Push the address of parameter P on the stack (address is in r0)
5 BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r0,r0,r0 ; Infinite loop to end the program
8
9 LDR r1,[sp] ; Save the return address on the stack
10 LDR r1,r1,=4 ; Get the 4 bytes of parameter P
11 ADD r1,r1,r1 ; Add 1 to P
12 ADD r1,r1,r1 ; Add 1 to P
13 ADD r1,r1,r1 ; Add 1 to P
14 ADD r1,r1,r1 ; Add 1 to P
15 LDMFD sp!,{r0} ; Return by loading the r0 in the return address from the stack
16 P DCD 0x12345678 ; Location of parameter P and its value
17
18 Base DCD 0xFFFFFFFF,0,0,0,0 ; Stack base and dummy data
19
20 END

```

(35)

## Passing By Reference

```

1 AREA ParamDef,CODE,READWRITE : Make readable because we locate the stack in this area
2 ADD sp,Base ; Point to the base of the stack
3 LDR r0,=Param ; Load r0 with the address of parameter P
4 STP r0,[sp,-4]! ; Push the address of parameter P on the stack (address is in r0)
5 BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r0,r0,r0 ; Infinite loop to end the program
8
9 LDR r1,[sp] ; Save the return address on the stack
10 LDR r1,r1,=4 ; Get the 4 bytes of parameter P
11 ADD r1,r1,r1 ; Add 1 to P
12 ADD r1,r1,r1 ; Add 1 to P
13 ADD r1,r1,r1 ; Add 1 to P
14 ADD r1,r1,r1 ; Add 1 to P
15 LDMFD sp!,{r0} ; Return by loading the r0 in the return address from the stack
16 P DCD 0x12345678 ; Location of parameter P and its value
17
18 Base DCD 0xFFFFFFFF,0,0,0,0 ; Stack base and dummy data
19
20 END

```

(36)

## Passing By Reference

```

1 DATA .text:$CODE,$READWRITE ; Make readable because we locate the stack in this area
2 ADD sp,Base ; Point to the base of the stack
3 ADD R0,sp,Base ; Load R0 with the address of parameter P
4 STR R0,[sp,4+1] ; Push the address of parameter P on the stack (address is in R0)
5 BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r0,r1,r0 ; Indicate loop to end the program
8
9 ABC ; Subroutine ABC
10 LDR r1,sp ; Load R1 with the base of the stack
11 LDR r2,[r1,4+1] ; Load R2 with the return address
12 LDR r3,[r1,8+1] ; Load R3 with the value of parameter P
13 STR r3,[r1,12] ; Save the parameter in the calling environment
14 LDMFD r0,{r0} ; Load the stack frame onto the return address from the stack
15
16 DCD 0x12345678 ; Location of parameter P and its value
17
18 DCD 0xFFFFFFFF,0,0,0,0 ; Stack base and dummy data
19 Base
20 END

```

[ 37 ]

## Passing By Reference

```

1 DATA .text:$CODE,$READWRITE ; Make readable because we locate the stack in this area
2 ADD sp,Base ; Point to the base of the stack
3 ADD R0,sp,Base ; Load R0 with the address of parameter P
4 STR R0,[sp,4+1] ; Push the address of parameter P on the stack (address is in R0)
5 BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r0,r1,r0 ; Indicate loop to end the program
8
9 ABC ; Subroutine ABC
10 LDR r1,sp ; Load R1 with the base of the stack
11 LDR r2,[r1,4+1] ; Load R2 with the return address
12 LDR r3,[r1,8+1] ; Load R3 with the value of parameter P
13 STR r3,[r1,12] ; Save the parameter in the calling environment
14 LDMFD r0,{r0} ; Load the stack frame onto the return address from the stack
15
16 DCD 0x12345678 ; Location of parameter P and its value
17
18 DCD 0xFFFFFFFF,0,0,0,0 ; Stack base and dummy data
19 Base
20 END

```

[ 38 ]

## Passing By Reference

- In our last example, we forgot to do something important. What is it?
- Hint:** it involves the stack pointer.
- After the subroutine, **sp** was NOT pointing to the base.
- We should adjust the stack pointer. How?
  - ADD sp, sp, #4



## Passing By Reference

```

1 DATA .text:$CODE,$READWRITE ; Make readable because we locate the stack in this area
2 ADD sp,Base ; Point to the base of the stack
3 ADD R0,sp,Base ; Load R0 with the address of parameter P
4 STR R0,[sp,4+1] ; Push the address of parameter P on the stack (address is in R0)
5 BL ABC ; Call subroutine ABC and save the link register
6
7 MUL r0,r1,r0 ; Indicate loop to end the program
8
9 ABC ; Subroutine ABC
10 LDR r1,sp ; Load R1 with the base of the stack
11 LDR r2,[r1,4+1] ; Load R2 with the return address
12 LDR r3,[r1,8+1] ; Load R3 with 1 + P
13 ADD r3,r3,1 ; Add 1 to P
14 STR r3,[r1,12] ; Save the parameter in the calling environment
15 LDMFD r0,{r0} ; Return by loading the FC with the return address from the stack
16
17 DCD 0x12345678 ; Location of parameter P and its value
18
19 DCD 0xFFFFFFFF,0,0,0,0 ; Stack base and dummy data
20 Base

```

[ 40 ]

## Outline

- Passing Parameters
- Value vs. Reference
- Passing By Value (revision)
- Passing By Reference
- Stack Frame
- Conclusion



[ 41 ]

## Stack Frame

- a subroutine sometimes needs local workspace for its temporary variables
- Each time the subroutine is called, a new workspace must be assigned to it
- The stack provides a convenient mechanism for implementing the dynamic allocation of workspace
- This storage allocation is dynamic because it is allocated to variables when they are created and then de-allocated when the variables are no longer required

[ 42 ]

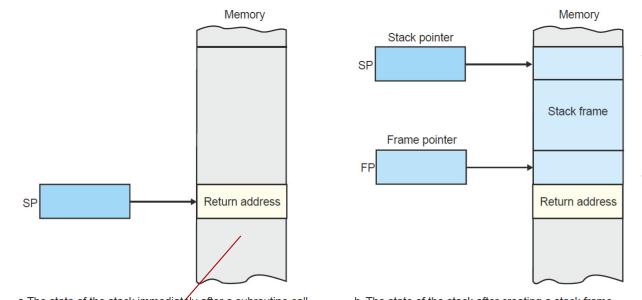
## Stack Frame

- The tools to implement a local workspace are:
  - Stack Frame (SF)
    - a region of temporary storage at the top of the current stack
  - Frame Pointer (FP)
    - an address register
    - points to the bottom of the stack frame



[ 43 ]

## Stack Frame



**Question:** What do we have here?

**Answer:** Parameters (if any) passed by the caller

[ 44 ]

## Stack Frame

- AnySub
- STR fp,[sp,#-4]! Save the old frame pointer on the stack
- MOV fp,sp Make FP point to the base of the stack frame
- SUB sp,sp,#16 Make SP point to the top of the stack frame. **Note: 16 is just an example**
- MOV sp,fp collapse the frame
- LDR fp,[sp],#4 Restore the old FP
- MOV pc,lr Return



[ 45 ]

## Stack Frame

```

1 AREA StackFrame, CODE, READWRITE
2     LDR fp,Stack ;set up the stack pointer
3     LDR r1,=00000000 ;dummy value for r1
4     LDR r2,=00000000 ;dummy value for r2
5     LDR r3,=00000000 ;dummy value for r3
6     LDR r4,=00000000 ;dummy value for r4
7     LDR r5,=00000000 ;dummy value for r5
8     LDR r6,=00000000 ;dummy value for r6
9     LDR r7,=00000000 ;dummy value for r7
10    LDR r8,=00000000 ;dummy value for r8
11    LDR r9,=00000000 ;dummy value for r9
12    LDR r10,=00000000 ;dummy value for r10
13    LDR r11,=00000000 ;dummy value for r11
14    LDR r12,=00000000 ;dummy value for r12
15    LDR r13,=00000000 ;dummy value for r13
16    LDR r14,=00000000 ;dummy value for r14
17    LDR r15,=00000000 ;dummy value for r15
18    LDR r16,=00000000 ;dummy value for r16
19
20    SUB sp,sp,#16 ;if it worked, r0 should contain ?
21
22    LDR r1,fp[sp] ;get parameter A
23    LDR r2,fp[sp+4] ;push the address of A on the stack
24    LDR r3,fp[sp+8] ;push the address of B on the stack
25    LDR r4,fp[sp+12] ;push the address of C on the stack
26    LDR r5,fp[sp+16] ;push the address of D on the stack
27
28    LDR r1,[r2,sp,4-1] ;push the value of A on the stack
29    LDR r2,[r3,sp,4-1] ;push the value of B on the stack
30    LDR r3,[r4,sp,4-1] ;push the value of C on the stack
31    LDR r4,[r5,sp,4-1] ;push the value of D on the stack
32
33    ADD r1,r1,r2 ;add A and B
34    ADD r1,r1,r3 ;add C and D
35
36    STRH r1,[sp,4-1] ;save registers on the stack
37    STR fp,[sp,4-1] ;save the old frame pointer on the stack (pre-indexing)

```

Base  
Literals

[ 46 ]

## Stack Frame

```

1 AREA StackFrame, CODE, READWRITE
2     LDR fp,Stack ;set up the stack pointer
3     LDR fp,=00000000 ;dummy value for fp
4     LDR r1,=00000000 ;dummy value for r1
5     LDR r2,=00000000 ;dummy value for r2
6     LDR r3,=00000000 ;dummy value for r3
7     LDR r4,=00000000 ;dummy value for r4
8     LDR r5,=00000000 ;dummy value for r5
9     LDR r6,=00000000 ;dummy value for r6
10    LDR r7,=00000000 ;dummy value for r7
11    LDR r8,=00000000 ;dummy value for r8
12    LDR r9,=00000000 ;dummy value for r9
13    LDR r10,=00000000 ;dummy value for r10
14    LDR r11,=00000000 ;dummy value for r11
15    LDR r12,=00000000 ;dummy value for r12
16    LDR r13,=00000000 ;dummy value for r13
17    LDR r14,=00000000 ;dummy value for r14
18    LDR r15,=00000000 ;dummy value for r15
19    LDR r16,=00000000 ;dummy value for r16
20
21    SUB sp,sp,#16 ;if it worked, r0 should contain ?
22
23    LDR r1,fp[sp] ;get parameter A
24    LDR r2,fp[sp+4] ;push the address of A on the stack
25    LDR r3,fp[sp+8] ;push the address of B on the stack
26    LDR r4,fp[sp+12] ;push the address of C on the stack
27
28    LDR r1,[r2,sp,4-1] ;push the value of A on the stack
29    LDR r2,[r3,sp,4-1] ;push the value of B on the stack
30    LDR r3,[r4,sp,4-1] ;push the value of C on the stack
31
32    ADD r1,r1,r2 ;add A and B
33    ADD r1,r1,r3 ;add C and D
34
35    STRH r1,[sp,4-1] ;save registers on the stack
36    STR fp,[sp,4-1] ;save the old frame pointer on the stack (pre-indexing)

```

Base  
Literals

[ 47 ]

## Stack Frame

```

1 AREA StackFrame, CODE, READWRITE
2     LDR fp,Stack ;set up the stack pointer
3     LDR fp,=00000000 ;dummy value for fp
4     LDR r1,=00000000 ;dummy value for r1
5     LDR r2,=00000000 ;dummy value for r2
6     LDR r3,=00000000 ;dummy value for r3
7     LDR r4,=00000000 ;dummy value for r4
8     LDR r5,=00000000 ;dummy value for r5
9     LDR r6,=00000000 ;dummy value for r6
10    LDR r7,=00000000 ;dummy value for r7
11    LDR r8,=00000000 ;dummy value for r8
12    LDR r9,=00000000 ;dummy value for r9
13    LDR r10,=00000000 ;dummy value for r10
14    LDR r11,=00000000 ;dummy value for r11
15    LDR r12,=00000000 ;dummy value for r12
16    LDR r13,=00000000 ;dummy value for r13
17    LDR r14,=00000000 ;dummy value for r14
18    LDR r15,=00000000 ;dummy value for r15
19    LDR r16,=00000000 ;dummy value for r16
20
21    SUB sp,sp,#16 ;if it worked, r0 should contain ?
22
23    LDR r1,fp[sp] ;get parameter A
24    LDR r2,fp[sp+4] ;push the address of A on the stack
25    LDR r3,fp[sp+8] ;push the address of B on the stack
26    LDR r4,fp[sp+12] ;push the address of C on the stack
27
28    LDR r1,[r2,sp,4-1] ;push the value of A on the stack
29    LDR r2,[r3,sp,4-1] ;push the value of B on the stack
30    LDR r3,[r4,sp,4-1] ;push the value of C on the stack
31
32    ADD r1,r1,r2 ;add A and B
33    ADD r1,r1,r3 ;add C and D
34
35    STRH r1,[sp,4-1] ;save registers on the stack
36    STR fp,[sp,4-1] ;save the old frame pointer on the stack (pre-indexing)

```

Base  
Literals

[ 48 ]

## Stack Frame

```

1 ADRP fp,FrameParams, CODE, READWRITE
2 ADD sp,fp, #40000000    ; set up the stack pointer
3 LDR r1,=0x11111111      ; dummy value for r1
4 LDR r2,=0x22222222      ; dummy value for r2
5 ADR r3,A            ; r3 is a pointer to A
6 LDR r4,B            ; r4 is a pointer to B
7 STR r5,[sp,#4-1]    ; push the value of A on the stack
8 ADD r6,r1,r3,LSR #2    ; get the address of B on the stack
9 LDR r7,[r6,#4-1]    ; push the address of B on the stack
10 BL subEq            ; call the subroutine
11 LDR r8,r1,r3,LSR #2    ; if it worked, r8 should contain ?
12 LDR r9,r1,r3,LSR #2    ; if it worked, r9 should contain ?
13 LDR r10,r1,r3,LSR #2   ; if it worked, r10 should contain ?
14 AGrün B Again
15 STMDq fp,[r1,r2,1]    ; save registers on the stack
16 STMDq fp,[r4,r5,1]    ; save registers on the stack (pre-indexing)
17 STMDq fp,[r6,r7,1]    ; save the old frame pointer on the stack (pre-indexing)

```

Base Literals

(49)

## Stack Frame

```

1 ADRP fp,FrameParams, CODE, READWRITE
2 ADD sp,fp, #40000000    ; set up the stack pointer
3 LDR r1,=0x11111111      ; dummy value for r1
4 LDR r2,=0x22222222      ; dummy value for r2
5 ADR r3,A            ; r3 is a pointer to A
6 LDR r4,B            ; r4 is a pointer to B
7 STR r5,[sp,#4-1]    ; push the value of A on the stack
8 ADD r6,r1,r3,LSR #2    ; get the address of B on the stack
9 LDR r7,[r6,#4-1]    ; push the address of B on the stack
10 BL subEq            ; call the subroutine
11 LDR r8,r1,r3,LSR #2    ; if it worked, r8 should contain ?
12 LDR r9,r1,r3,LSR #2    ; if it worked, r9 should contain ?
13 LDR r10,r1,r3,LSR #2   ; if it worked, r10 should contain ?
14 AGrün B Again
15 STMDq fp,[r1,r2,1]    ; save registers on the stack
16 STMDq fp,[r4,r5,1]    ; save registers on the stack (pre-indexing)
17 STMDq fp,[r6,r7,1]    ; save the old frame pointer on the stack (pre-indexing)

```

Base Literals

(50)

## Stack Frame

```

1 ADRP fp,FrameParams, CODE, READWRITE
2 ADD sp,fp, #40000000    ; set up the stack pointer
3 LDR r1,=0x11111111      ; dummy value for fp
4 LDR r2,=0x22222222      ; dummy value for r2
5 ADR r3,A            ; r3 is a pointer to A
6 LDR r4,B            ; r4 is a pointer to B
7 LDR r5,[r3,#4-1]    ; get parameter A
8 LDR r6,[r4,#4-1]    ; get the address of A on the stack
9 ADR r7,B            ; r7 is a pointer to B
10 LDR r8,[r7,#4-1]    ; get the address of B on the stack
11 BL subEq            ; call the subroutine
12 LDR r9,r1,r3,LSR #2    ; if it worked, r9 should contain ?
13 LDR r10,r1,r3,LSR #2   ; if it worked, r10 should contain ?
14 AGrün B Again
15 STMDq fp,[r1,r2,1]    ; save registers on the stack
16 STMDq fp,[r4,r5,1]    ; save registers on the stack (pre-indexing)
17 STMDq fp,[r6,r7,1]    ; save the old frame pointer on the stack (pre-indexing)

```

Base Literals

(51)

## Stack Frame

```

1 ADRP fp,FrameParams, CODE, READWRITE
2 ADD sp,fp, #40000000    ; set up the stack pointer
3 LDR r1,=0x11111111      ; dummy value for fp
4 LDR r2,=0x22222222      ; dummy value for r2
5 ADR r3,A            ; r3 is a pointer to A
6 LDR r4,B            ; r4 is a pointer to B
7 LDR r5,[r3,#4-1]    ; get parameter A
8 LDR r6,[r4,#4-1]    ; get the address of A on the stack
9 ADR r7,B            ; r7 is a pointer to B
10 LDR r8,[r7,#4-1]    ; get the address of B on the stack
11 BL subEq            ; call the subroutine
12 LDR r9,r1,r3,LSR #2    ; if it worked, r9 should contain ?
13 LDR r10,r1,r3,LSR #2   ; if it worked, r10 should contain ?
14 AGrün B Again
15 STMDq fp,[r1,r2,1]    ; save registers on the stack
16 STMDq fp,[r4,r5,1]    ; save registers on the stack (pre-indexing)
17 STMDq fp,[r6,r7,1]    ; save the old frame pointer on the stack (pre-indexing)

```

Base Literals

(52)

## Stack Frame

```

1 ADD sp, #4, esp ; set up the stack pointer
2 LDR r1, =x11111111 ; dummy value for r1
3 LDR r2, =x22222222 ; dummy value for r2
4 LDR r3, =x33333333 ; dummy value for r3
5 LDR r4, =x44444444 ; dummy value for r4
6 LDR r5, =x55555555 ; dummy value for r5
7 LDR r6, =x66666666 ; dummy value for r6
8 LDR r7, =x77777777 ; dummy value for r7
9 LDR r8, =x88888888 ; dummy value for r8
10 LDR r9, =x99999999 ; dummy value for r9
11 LDR r10, =x00000000 ; dummy value for r10
12 LDR r11, =xAAAAAAA ; dummy value for r11
13 LDR r12, =xBBBBBBB ; dummy value for r12
14 Again B Again
15 SAndq STMW sp!,{r1,r2,r3,r4}
16 SAndq STMW sp!,{r5,r6,r7,r8}
17 SAndq STMW sp!,{r9,r10,r11,r12} ; save registers on the stack
18 SAndq STMW sp!,{r1,r2,r3,r4} ; save the old frame pointer on the stack (pre-indexing)

```

[ 53 ]

## Stack Frame

```

1 ADD sp, #4, esp ; set up the stack pointer
2 LDR r1, =x11111111 ; dummy value for r1
3 LDR r2, =x22222222 ; dummy value for r2
4 LDR r3, =x33333333 ; dummy value for r3
5 LDR r4, =x44444444 ; dummy value for r4
6 LDR r5, =x55555555 ; dummy value for r5
7 LDR r6, =x66666666 ; dummy value for r6
8 LDR r7, =x77777777 ; dummy value for r7
9 LDR r8, =x88888888 ; dummy value for r8
10 LDR r9, =x99999999 ; dummy value for r9
11 LDR r10, =x00000000 ; dummy value for r10
12 LDR r11, =xAAAAAAA ; dummy value for r11
13 LDR r12, =xBBBBBBB ; dummy value for r12
14 Again B Again
15 SAndq STMW sp!,{r1,r2,r3,r4}
16 SAndq STMW sp!,{r5,r6,r7,r8}
17 SAndq STMW sp!,{r9,r10,r11,r12} ; save registers on the stack
18 SAndq STMW sp!,{r1,r2,r3,r4} ; save the old frame pointer on the stack (pre-indexing)

```

[ 54 ]

## Stack Frame

```

1 ADD sp, #4, esp ; set up the stack pointer
2 LDR r1, =x11111111 ; dummy value for r1
3 LDR r2, =x22222222 ; dummy value for r2
4 LDR r3, =x33333333 ; dummy value for r3
5 LDR r4, =x44444444 ; dummy value for r4
6 LDR r5, =x55555555 ; dummy value for r5
7 LDR r6, =x66666666 ; dummy value for r6
8 LDR r7, =x77777777 ; dummy value for r7
9 LDR r8, =x88888888 ; dummy value for r8
10 LDR r9, =x99999999 ; dummy value for r9
11 LDR r10, =x00000000 ; dummy value for r10
12 LDR r11, =xAAAAAAA ; dummy value for r11
13 LDR r12, =xBBBBBBB ; dummy value for r12
14 Again B Again
15 SAndq STMW sp!,{r1,r2,r3,r4}
16 SAndq STMW sp!,{r5,r6,r7,r8}
17 SAndq STMW sp!,{r9,r10,r11,r12} ; save registers on the stack
18 SAndq STMW sp!,{r1,r2,r3,r4} ; save the old frame pointer on the stack (pre-indexing)

```

[ 55 ]

## Stack Frame

```

1 ADD sp, #4, esp ; set up the stack pointer
2 LDR r1, =x11111111 ; dummy value for r1
3 LDR r2, =x22222222 ; dummy value for r2
4 LDR r3, =x33333333 ; dummy value for r3
5 LDR r4, =x44444444 ; dummy value for r4
6 LDR r5, =x55555555 ; dummy value for r5
7 LDR r6, =x66666666 ; dummy value for r6
8 LDR r7, =x77777777 ; dummy value for r7
9 LDR r8, =x88888888 ; dummy value for r8
10 LDR r9, =x99999999 ; dummy value for r9
11 LDR r10, =x00000000 ; dummy value for r10
12 LDR r11, =xAAAAAAA ; dummy value for r11
13 LDR r12, =xBBBBBBB ; dummy value for r12
14 Again B Again
15 SAndq STMW sp!,{r1,r2,r3,r4}
16 SAndq STMW sp!,{r5,r6,r7,r8}
17 SAndq STMW sp!,{r9,r10,r11,r12} ; save registers on the stack
18 SAndq STMW sp!,{r1,r2,r3,r4} ; save the old frame pointer on the stack (pre-indexing)

```

[ 56 ]

## Stack Frame

**Screenshot 57:** Stack Frame (Registers View)

Registers window showing assembly code and memory dump. The assembly code shows the stack frame setup and arithmetic operations. The memory dump shows the stack grows downwards from the base address.

```

12 LDR r0,[fp,1] ; if it worked, r0 should contain ?
13
14 AAgain B Again
15
16 LDR r1,[fp,1] ; save registers on the stack
17 STMDA sp!,{r1,fp,1,2,1c} ; save the old frame pointer to the stack (pre-indexing)
18 MOV fp,sp ; set the frame pointer to point to the top of the stack
19 MOV r2,sp+4 ; move the stack pointer to the top of the one-word stack frame
20
21 LDR r3,[fp,4] ; get the value of A off the stack in r3
22 STR r3,[fp,4] ; store the value of A squared in the stack frame
23 LDR r4,[fp,4] ; get the address of B off the stack in r4 (reuse r2)
24 LDR r5,[fp,4] ; get the value of B in r5 (reuse r1)
25 LDR r6,[fp,4] ; get the value of B squared in r6
26 ADD r7,r5,r6 ; add B to A squared
27 STR r7,[fp,4] ; store the result to the calling environment
28 MOV fp,sp ; restore the stack pointer and collapse the frame

```

Diagram below illustrates the stack frame structure:

Diagram Labels:  
 - Stack pointer: fp -4  
 - Frame pointer pointer: fp  
 - Top of stack: 0xAAAAAAA  
 - The stack frame: Saved on entry to the subroutine  
 - Parameters pushed on the stack: 0x2 Value of A, 0x2 Value of B  
 - The stack base: 0xFFFFFFF  
 - Direction of growth as items are added: Upwards (from base to top)

[57]

## Stack Frame

**Screenshot 58:** Stack Frame (Registers View)

Registers window showing assembly code and memory dump. The assembly code shows the stack frame setup and arithmetic operations. The memory dump shows the stack grows downwards from the base address.

```

12 LDR r0,[fp,1] ; if it worked, r0 should contain ?
13
14 AAgain B Again
15
16 LDR r1,[fp,1] ; save registers on the stack
17 STMDA sp!,{r1,fp,1,2,1c} ; save the old frame pointer on the stack (pre-indexing)
18 MOV fp,sp ; set the frame pointer to point to the top of the stack
19 MOV r2,sp+4 ; move the stack pointer to the top of the one-word stack frame
20
21 LDR r3,[fp,4] ; get the value of A off the stack in r3
22 STR r3,[fp,4] ; store the value of A squared in the stack frame
23 LDR r4,[fp,4] ; get the address of B off the stack in r4 (reuse r2)
24 LDR r5,[fp,4] ; get the value of B in r5 (reuse r1)
25 LDR r6,[fp,4] ; get the value of B squared in r6
26 ADD r7,r5,r6 ; add B to A squared
27 STR r7,[fp,4] ; store the result to the calling environment
28 MOV fp,sp ; restore the stack pointer and collapse the frame

```

Diagram below illustrates the stack frame structure:

Diagram Labels:  
 - Stack pointer: fp -4  
 - Frame pointer pointer: fp  
 - Top of stack: 0xAAAAAAA  
 - The stack frame: Saved on entry to the subroutine  
 - Parameters pushed on the stack: 0x2 Value of A, 0x2 Value of B  
 - The stack base: 0xFFFFFFF  
 - Direction of growth as items are added: Upwards (from base to top)

[58]

## Stack Frame

**Screenshot 59:** Stack Frame (Registers View)

Registers window showing assembly code and memory dump. The assembly code shows the stack frame setup and arithmetic operations. The memory dump shows the stack grows downwards from the base address.

```

12 LDR r0,[fp,1] ; if it worked, r0 should contain ?
13
14 AAgain B Again
15
16 LDR r1,[fp,1] ; save registers on the stack
17 STMDA sp!,{r1,fp,1,2,1c} ; save the old frame pointer on the stack (pre-indexing)
18 MOV fp,sp ; set the frame pointer to point to the top of the stack
19 MOV r2,sp+4 ; move the stack pointer to the top of the one-word stack frame
20
21 LDR r3,[fp,4] ; get the value of A off the stack in r3
22 STR r3,[fp,4] ; store the value of A squared in the stack frame
23 LDR r4,[fp,4] ; get the address of B off the stack in r4 (reuse r2)
24 LDR r5,[fp,4] ; get the value of B in r5 (reuse r1)
25 LDR r6,[fp,4] ; get the value of B squared in r6
26 ADD r7,r5,r6 ; add B to A squared
27 STR r7,[fp,4] ; store the result to the calling environment
28 MOV fp,sp ; restore the stack pointer and collapse the frame

```

Diagram below illustrates the stack frame structure:

Diagram Labels:  
 - Stack pointer: fp -4  
 - Frame pointer pointer: fp  
 - Top of stack: 0xAAAAAAA  
 - The stack frame: Saved on entry to the subroutine  
 - Parameters pushed on the stack: 0x2 Value of A, 0x2 Value of B  
 - The stack base: 0xFFFFFFF  
 - Direction of growth as items are added: Upwards (from base to top)

[59]

## Stack Frame

**Screenshot 60:** Stack Frame (Registers View)

Registers window showing assembly code and memory dump. The assembly code shows the stack frame setup and arithmetic operations. The memory dump shows the stack grows downwards from the base address.

```

12 LDR r0,[fp,1] ; if it worked, r0 should contain ?
13
14 AAgain B Again
15
16 LDR r1,[fp,1] ; save registers on the stack
17 STMDA sp!,{r1,fp,1,2,1c} ; save the old frame pointer on the stack (pre-indexing)
18 MOV fp,sp ; set the frame pointer to point to the top of the stack
19 MOV r2,sp+4 ; move the stack pointer to the top of the one-word stack frame
20
21 LDR r3,[fp,4] ; get the value of A off the stack in r3
22 STR r3,[fp,4] ; store the value of A squared in the stack frame
23 LDR r4,[fp,4] ; get the address of B off the stack in r4 (reuse r2)
24 LDR r5,[fp,4] ; get the value of B in r5 (reuse r1)
25 LDR r6,[fp,4] ; get the value of B squared in r6
26 ADD r7,r5,r6 ; add B to A squared
27 STR r7,[fp,4] ; store the result to the calling environment
28 MOV fp,sp ; restore the stack pointer and collapse the frame

```

Diagram below illustrates the stack frame structure:

Diagram Labels:  
 - Stack pointer: fp -4  
 - Frame pointer pointer: fp  
 - Top of stack: 0xAAAAAAA  
 - The stack frame: Saved on entry to the subroutine  
 - Parameters pushed on the stack: 0x2 Value of A, 0x2 Value of B  
 - The stack base: 0xFFFFFFF  
 - Direction of growth as items are added: Upwards (from base to top)

[60]

## Stack Frame

The screenshot shows assembly code and a memory dump for stack frame 61. The assembly code includes instructions like LDR, STR, and ADD. A memory dump shows the stack frame structure with pointers fp, r1, r2, and r3 pointing to specific memory locations. A diagram below illustrates the stack frame layout with the stack base, literals, and parameters pushed onto the stack.

```

    LDR r0,[fp, #1] ; if it worked, r0 should contain ?
    LDR r1,[fp, #4] ; Again
    LDR r2,[fp, #8] ; Save registers on the stack
    STRB r0,(fp, #4) ; save the old frame pointer to the stack (pre-indexing)
    MOV r0,fp ; move the frame pointer to point to the top of the stack
    MOV fp,r0 ; move the stack pointer to the top of the one-word stack frame
    LDR r3,[fp, #4] ; set the value of A off the stack in r1
    LDR r4,[fp, #8] ; choose the value of A squared in the stack frame
    LDR r5,[fp, #12] ; choose the address of B off the stack in r2 (reuse r2)
    LDR r6,[fp, #16] ; set the value of B in r1 (reuse r1)
    LDR r7,[fp, #20] ; set the value of A squared in r0
    ADD r8,r4,r4 ; add A squared
    ADD r9,r5,r5 ; add B squared
    STR r8,[fp, #4] ; restore the result to the calling environment
    STR r9,[fp, #8] ; restore the stack pointer and collapse the frame
  
```

Memory dump details:

- Stack pointer: fp - 4
- Frame pointer pointer: fp
- Top of stack: 0xAAAAAAA
- Base: 0x11111111
- Literals: 0x22222222
- Parameters pushed on the stack:
  - r1 = 0x11111111
  - r2 = 0x22222222
  - r3 = 0x00000008
  - r4 = 0x00000000
  - r5 = 0x00000000
  - r6 = 0x00000000
  - r7 = 0x00000000
  - r8 = 0x00000000
  - r9 = 0x00000000
  - r10 = 0x00000000
  - r11 = 0x00000000
  - r12 = 0x00000000
  - r13 = 0x00000000
  - r14 = 0x00000000
  - r15 = 0x00000000
  - r16 = 0x00000000
  - r17 = 0x00000000
  - r18 = 0x00000000
  - r19 = 0x00000000
  - r20 = 0x00000000
- The stack base: 0xFFFFFFF

Diagram:

Stack pointer → fp - 4 → Top of stack → 0xAAAAAAA → Base → Literals → Parameters pushed on the stack → The stack base

Direction of growth as items are added: Upwards from the stack base.

[61]

## Stack Frame

The screenshot shows assembly code and a memory dump for stack frame 62. The assembly code and memory dump are identical to stack frame 61. A memory dump shows the stack frame structure with pointers fp, r1, r2, and r3 pointing to specific memory locations. A diagram below illustrates the stack frame layout with the stack base, literals, and parameters pushed onto the stack.

```

    LDR r0,[fp, #1] ; if it worked, r0 should contain ?
    LDR r1,[fp, #4] ; Again
    LDR r2,[fp, #8] ; Save registers on the stack
    STRB r0,(fp, #4) ; save the old frame pointer to the stack (pre-indexing)
    MOV r0,fp ; move the frame pointer to point to the top of the stack
    MOV fp,r0 ; move the stack pointer to the top of the one-word stack frame
    LDR r3,[fp, #4] ; set the value of A off the stack in r1
    LDR r4,[fp, #8] ; choose the value of A squared in the stack frame
    LDR r5,[fp, #12] ; choose the address of B off the stack in r2 (reuse r2)
    LDR r6,[fp, #16] ; set the value of B in r1 (reuse r1)
    LDR r7,[fp, #20] ; set the value of A squared in r0
    ADD r8,r4,r4 ; add A squared
    ADD r9,r5,r5 ; add B squared
    STR r8,[fp, #4] ; restore the result to the calling environment
    STR r9,[fp, #8] ; restore the stack pointer and collapse the frame
  
```

Memory dump details:

- Stack pointer: fp - 4
- Frame pointer pointer: fp
- Top of stack: 0xAAAAAAA
- Base: 0x11111111
- Literals: 0x22222222
- Parameters pushed on the stack:
  - r1 = 0x11111111
  - r2 = 0x22222222
  - r3 = 0x00000008
  - r4 = 0x00000000
  - r5 = 0x00000000
  - r6 = 0x00000000
  - r7 = 0x00000000
  - r8 = 0x00000000
  - r9 = 0x00000000
  - r10 = 0x00000000
  - r11 = 0x00000000
  - r12 = 0x00000000
  - r13 = 0x00000000
  - r14 = 0x00000000
  - r15 = 0x00000000
  - r16 = 0x00000000
  - r17 = 0x00000000
  - r18 = 0x00000000
  - r19 = 0x00000000
  - r20 = 0x00000000
- The stack base: 0xFFFFFFF

Diagram:

Stack pointer → fp - 4 → Top of stack → 0xAAAAAAA → Base → Literals → Parameters pushed on the stack → The stack base

Direction of growth as items are added: Upwards from the stack base.

[62]

## Stack Frame

The screenshot shows assembly code and a memory dump for stack frame 63. The assembly code and memory dump are identical to stack frame 61. A memory dump shows the stack frame structure with pointers fp, r1, r2, and r3 pointing to specific memory locations. A diagram below illustrates the stack frame layout with the stack base, literals, and parameters pushed onto the stack.

```

    LDR r0,[fp, #1] ; if it worked, r0 should contain ?
    LDR r1,[fp, #4] ; Again
    LDR r2,[fp, #8] ; Save registers on the stack
    STRB r0,(fp, #4) ; save the old frame pointer to the stack (pre-indexing)
    MOV r0,fp ; move the frame pointer to point to the top of the stack
    MOV fp,r0 ; move the stack pointer to the top of the one-word stack frame
    LDR r3,[fp, #4] ; set the value of A off the stack in r1
    LDR r4,[fp, #8] ; choose the value of A squared in the stack frame
    LDR r5,[fp, #12] ; choose the address of B off the stack in r2 (reuse r2)
    LDR r6,[fp, #16] ; set the value of B in r1 (reuse r1)
    LDR r7,[fp, #20] ; set the value of A squared in r0
    ADD r8,r4,r4 ; add A squared
    ADD r9,r5,r5 ; add B squared
    STR r8,[fp, #4] ; restore the result to the calling environment
    STR r9,[fp, #8] ; restore the stack pointer and collapse the frame
  
```

Memory dump details:

- Stack pointer: fp - 4
- Frame pointer pointer: fp
- Top of stack: 0xAAAAAAA
- Base: 0x11111111
- Literals: 0x22222222
- Parameters pushed on the stack:
  - r1 = 0x11111111
  - r2 = 0x22222222
  - r3 = 0x00000008
  - r4 = 0x00000000
  - r5 = 0x00000000
  - r6 = 0x00000000
  - r7 = 0x00000000
  - r8 = 0x00000000
  - r9 = 0x00000000
  - r10 = 0x00000000
  - r11 = 0x00000000
  - r12 = 0x00000000
  - r13 = 0x00000000
  - r14 = 0x00000000
  - r15 = 0x00000000
  - r16 = 0x00000000
  - r17 = 0x00000000
  - r18 = 0x00000000
  - r19 = 0x00000000
  - r20 = 0x00000000
- The stack base: 0xFFFFFFF

Diagram:

Stack pointer → fp - 4 → Top of stack → 0xAAAAAAA → Base → Literals → Parameters pushed on the stack → The stack base

Direction of growth as items are added: Upwards from the stack base.

[63]

## Stack Frame

The screenshot shows assembly code and a memory dump for stack frame 64. The assembly code and memory dump are identical to stack frame 61. A memory dump shows the stack frame structure with pointers fp, r1, r2, and r3 pointing to specific memory locations. A diagram below illustrates the stack frame layout with the stack base, literals, and parameters pushed onto the stack.

```

    LDR r0,[fp, #1] ; if it worked, r0 should contain ?
    LDR r1,[fp, #4] ; Again
    LDR r2,[fp, #8] ; Save registers on the stack
    STRB r0,(fp, #4) ; save the old frame pointer to the stack (pre-indexing)
    MOV r0,fp ; move the frame pointer to point to the top of the stack
    MOV fp,r0 ; move the stack pointer to the top of the one-word stack frame
    LDR r3,[fp, #4] ; set the value of A off the stack in r1
    LDR r4,[fp, #8] ; choose the value of A squared in the stack frame
    LDR r5,[fp, #12] ; choose the address of B off the stack in r2 (reuse r2)
    LDR r6,[fp, #16] ; set the value of B in r1 (reuse r1)
    LDR r7,[fp, #20] ; set the value of A squared in r0
    ADD r8,r4,r4 ; add A squared
    ADD r9,r5,r5 ; add B squared
    STR r8,[fp, #4] ; restore the result to the calling environment
    STR r9,[fp, #8] ; restore the stack pointer and collapse the frame
  
```

Memory dump details:

- Stack pointer: fp - 4
- Frame pointer pointer: fp
- Top of stack: 0xAAAAAAA
- Base: 0x11111111
- Literals: 0x22222222
- Parameters pushed on the stack:
  - r1 = 0x11111111
  - r2 = 0x22222222
  - r3 = 0x00000008
  - r4 = 0x00000000
  - r5 = 0x00000000
  - r6 = 0x00000000
  - r7 = 0x00000000
  - r8 = 0x00000000
  - r9 = 0x00000000
  - r10 = 0x00000000
  - r11 = 0x00000000
  - r12 = 0x00000000
  - r13 = 0x00000000
  - r14 = 0x00000000
  - r15 = 0x00000000
  - r16 = 0x00000000
  - r17 = 0x00000000
  - r18 = 0x00000000
  - r19 = 0x00000000
  - r20 = 0x00000000
- The stack base: 0xFFFFFFF

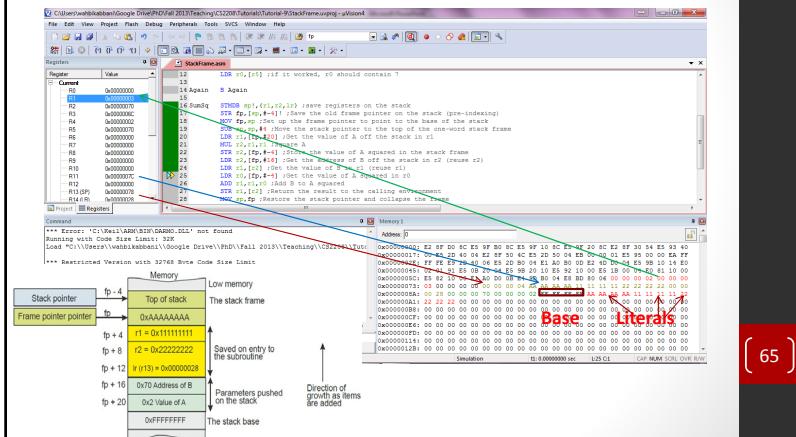
Diagram:

Stack pointer → fp - 4 → Top of stack → 0xAAAAAAA → Base → Literals → Parameters pushed on the stack → The stack base

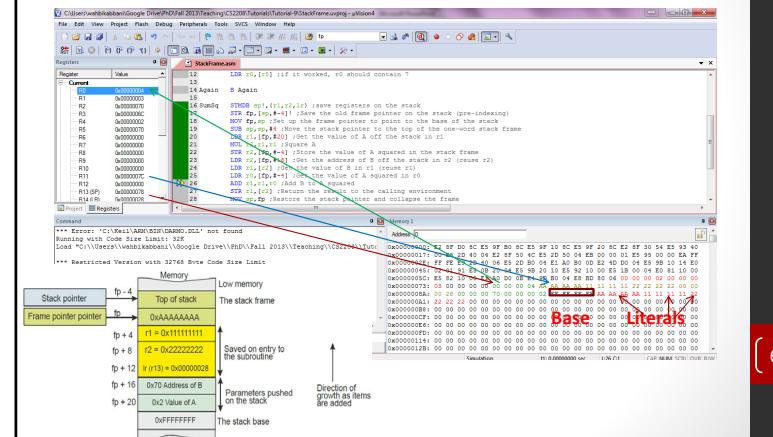
Direction of growth as items are added: Upwards from the stack base.

[64]

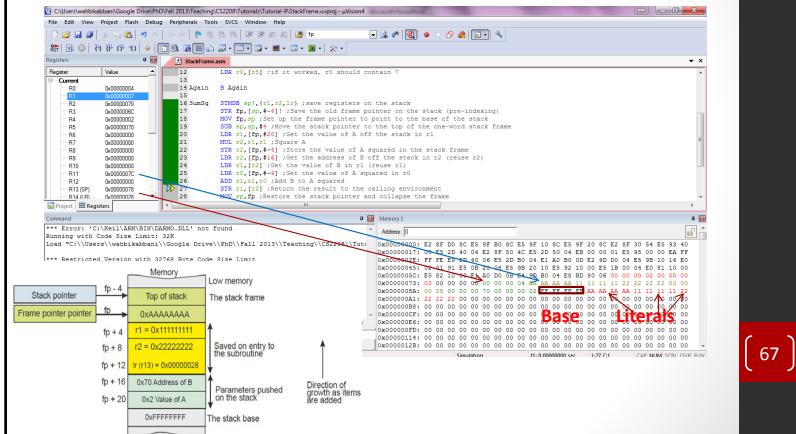
## Stack Frame



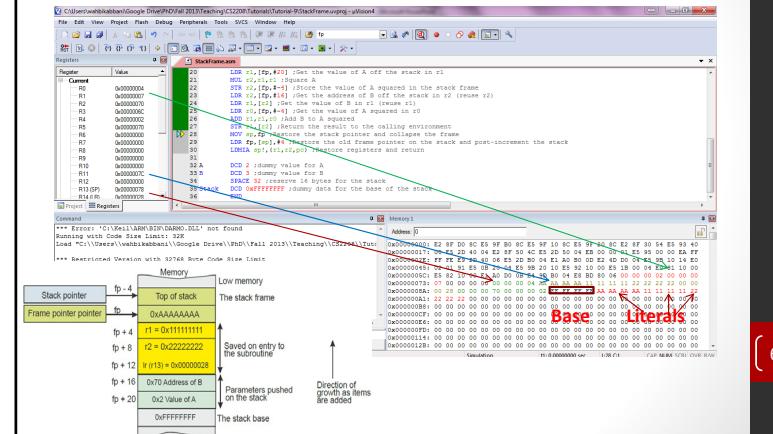
## Stack Frame



## Stack Frame



# Stack Frame



## Stack Frame

The screenshot shows the assembly code for a one-word stack frame:

```

    LD A, [fp+4] ; Get the value of A off the stack in r1
    MUL r1, r1, #2 ; Square the value of A squared in the stack frame
    LD B, [fp+8] ; Get the address of B off the stack in r2 (reuse r2)
    LDR r2, [r2+4] ; Get the value of B off the stack in r2 (reuse r2)
    LDR r2, [r2+4] ; Get the value of A squared in r2
    ADD r2, r2, r1 ; Add B to A squared
    STR r2, [fp+4] ; Store the result to the calling environment
    ADD fp, fp, #4 ; Move the stack pointer and collapse the frame
    LDR fp, [fp+4], #4 ; Restore the stack pointer and post-increment the stack
    LDMIA fp!, {r1,r2,p0} ; Restore registers and return
    ROR r1, r1, #1 ; Dummy value for A
    ROR r1, r1, #1 ; Dummy value for B
    SPACE 32 ; Reserve 16 bytes for the stack
    DCD 0xFFFFFFFF ; Dummy data for the base of the stack
  
```

**Stack Frame Diagram:**

- Stack pointer: fp - 4
- Frame pointer pointer: fp
- Top of stack: 0xAAAAAAA
- Base: 0x11111111
- Literals: 0xFFFFFFF
- Direction of growth: Items are added
- Parameters pushed on the stack: 0x2 Value of A
- Saved on entry to the subroutine: 0x10 Address of B
- Low memory: The stack frame

[69]

## Stack Frame

The screenshot shows the assembly code for a one-word stack frame:

```

    LD A, [fp+4] ; Get the value of A off the stack in r1
    MUL r1, r1, #2 ; Square the value of A squared in the stack frame
    LD B, [fp+8] ; Get the address of B off the stack in r2 (reuse r2)
    LDR r2, [r2+4] ; Get the value of B off the stack in r2 (reuse r2)
    LDR r2, [r2+4] ; Get the value of A squared in r2
    ADD r2, r2, r1 ; Add B to A squared
    STR r2, [fp+4] ; Store the result to the calling environment
    ADD fp, fp, #4 ; Move the stack pointer and collapse the frame
    LDR fp, [fp+4], #4 ; Restore the stack pointer and post-increment the stack
    LDMIA fp!, {r1,r2,p0} ; Restore registers and return
    ROR r1, r1, #1 ; Dummy value for A
    ROR r1, r1, #1 ; Dummy value for B
    SPACE 32 ; Reserve 16 bytes for the stack
    DCD 0xFFFFFFFF ; Dummy data for the base of the stack
  
```

**Stack Frame Diagram:**

- Stack pointer: fp - 4
- Frame pointer pointer: fp
- Top of stack: 0xAAAAAAA
- Base: 0x11111111
- Literals: 0xFFFFFFF
- Direction of growth: Items are added
- Parameters pushed on the stack: 0x2 Value of A
- Saved on entry to the subroutine: 0x10 Address of B
- Low memory: The stack frame

[70]

## Stack Frame

The screenshot shows the assembly code for a one-word stack frame:

```

    LD A, [fp+4] ; Call the subroutine
    ADD r1, r1, #4 ; Set r1 = 4
    LD A, [fp+4] ; r0 should contain ?
    LD B, [fp+8] ; Again
    SUB r0, r0, #4 ; Save registers on the stack
    STMDA fp!, {r1,r2,p0} ; Save old frame pointer on the stack (pre-indexing)
    LD B, [fp+8] ; Move the stack pointer to the top of the one-word stack frame
    LDR r2, [fp+4] ; Get the value of A off the stack in r1
    LDR r2, [fp+4] ; Get the value of A squared in the stack frame
    ADD r2, r2, r1 ; Add B to A squared
    STR r2, [fp+4] ; Store the value of A squared in the stack frame
    LD B, [fp+8] ; Get the value of B off the stack in r2 (reuse r2)
    LDR r2, [fp+4] ; Get the value of B off the stack in r2 (reuse r2)
    ADD r2, r2, r1 ; Add B to A squared
    ADD r1,r1,r2 ; Add B to A squared
    STR r1, [fp+4] ; Return the result to the calling environment
  
```

**Stack Frame Diagram:**

- Stack pointer: fp - 4
- Frame pointer pointer: fp
- Top of stack: 0xAAAAAAA
- Base: 0x11111111
- Literals: 0xFFFFFFF
- Direction of growth: Items are added
- Parameters pushed on the stack: 0x2 Value of A
- Saved on entry to the subroutine: 0x10 Address of B
- Low memory: The stack frame

[71]

## Stack Frame

The screenshot shows the assembly code for a one-word stack frame:

```

    LD A, [fp+4] ; Call the subroutine
    ADD r1, r1, #4 ; Set r1 = 4
    LD A, [fp+4] ; r0 should contain ?
    LD B, [fp+8] ; Again
    SUB r0, r0, #4 ; Save registers on the stack
    STMDA fp!, {r1,r2,p0} ; Save old frame pointer on the stack (pre-indexing)
    LD B, [fp+8] ; Move the stack pointer to the top of the one-word stack frame
    LDR r2, [fp+4] ; Get the value of A off the stack in r1
    LDR r2, [fp+4] ; Get the value of A squared in the stack frame
    ADD r2, r2, r1 ; Add B to A squared
    STR r2, [fp+4] ; Store the value of A squared in the stack frame
    LD B, [fp+8] ; Get the value of B off the stack in r2 (reuse r2)
    LDR r2, [fp+4] ; Get the value of B off the stack in r2 (reuse r2)
    ADD r2, r2, r1 ; Add B to A squared
    ADD r1,r1,r2 ; Add B to A squared
    STR r1, [fp+4] ; Return the result to the calling environment
  
```

**Stack Frame Diagram:**

- Stack pointer: fp - 4
- Frame pointer pointer: fp
- Top of stack: 0xAAAAAAA
- Base: 0x11111111
- Literals: 0xFFFFFFF
- Direction of growth: Items are added
- Parameters pushed on the stack: 0x2 Value of A
- Saved on entry to the subroutine: 0x10 Address of B
- Low memory: The stack frame

[72]

## Stack Frame

- Again, what did we forget to do here?
- Remove the parameters from the stack
- How?
  - We have 2 parameters
  - `ADD sp,sp,#8`



[ 73 ]

## Stack Frame

StackFrame.asm

```

11    E2 80d1  call the subroutine
12    LD 0x1c,[sp] ;set the registers on the stack
13    LD 0x14,[sp] ;set the stack pointer to point to the base of the stack (pre-indexing)
14 Again:
15    LD 0x14,[sp] ;set up the frame pointer to point to the base of the stack
16    LD 0x14,[sp+4] ;set up the stack frame
17    LD 0x14,[sp+8] ;set the value of A off the stack in r0
18    LD 0x14,[sp+12] ;set the value of A squared in r2 (create r2)
19    LD 0x14,[sp+16] ;set the value of B off the stack in r1 (create r1)
20    LD 0x14,[sp+20] ;set the value of B squared in r3 (create r3)
21    LD 0x14,[sp+24] ;pushes A
22    LD 0x14,[sp+28] ;pushes A squared
23    LD 0x14,[sp+32] ;pushes B
24    LD 0x14,[sp+36] ;pushes B squared
25    LD 0x14,[sp+40] ;set the value of A squared in r0
26    LD 0x14,[sp+44] ;set the value of B squared in r1
27    RET 0x1c ;return the result to the calling environment

```

Registers:

Register	Value
R1	0xffffffff
R2	0x00000000
R3	0xffffffff
R4	0xffffffff
R5	0xffffffff
R6	0xffffffff
R7	0xffffffff
R8	0xffffffff
R9	0xffffffff
R10	0xffffffff
R11	0xffffffff
R12	0xffffffff
R13	0xffffffff
R14	0xffffffff
R15	0xffffffff
R16	0xffffffff
R17	0xffffffff
R18	0xffffffff
R19	0xffffffff
R20	0xffffffff
R21	0xffffffff
R22	0xffffffff
R23	0xffffffff
R24	0xffffffff
R25	0xffffffff
R26	0xffffffff
R27	0xffffffff

Memory:

```

Stack pointer: fp - 4
Frame pointer pointer: fp
fp + 4: Top of stack
fp + 8: r1 = 0xffffffff
fp + 12: r2 = 0x00000000
fp + 16: D070 Address of B
fp + 20: 0x2 Value of A
fp + 24: 0xFFFFFFF

```

The stack frame diagram shows the stack grows downwards. The stack base is at fp + 24. Parameters are pushed onto the stack at fp + 20. The base address of B is at fp + 16. The value of A is at fp + 20. The stack grows downwards from fp + 4.

[ 74 ]

## Outline

- Passing Parameters**
- Value vs. Reference**
- Passing By Value (revision)**
- Passing By Reference**
- Stack Frame**
- Conclusion**



[ 75 ]

## Conclusion

- Passing By Value:**
  - anything passed into a function call is unchanged in the caller's scope
- Passing By Reference:**
  - A reference (memory address) to a variable is passed
  - The function can modify the parameter



[ 76 ]

## Conclusion

- a subroutine sometimes needs local workspace for its temporary variables
- Stack Frame (SF)
  - a region of temporary storage at the top of the current stack
- Frame Pointer (FP)
  - points to the bottom of the stack frame



## The End

- Good Luck on your exams.
- Consultation Room : Middlesex College - 342

