Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

Chapter 8
Passing Parameters to Subroutines via Registers

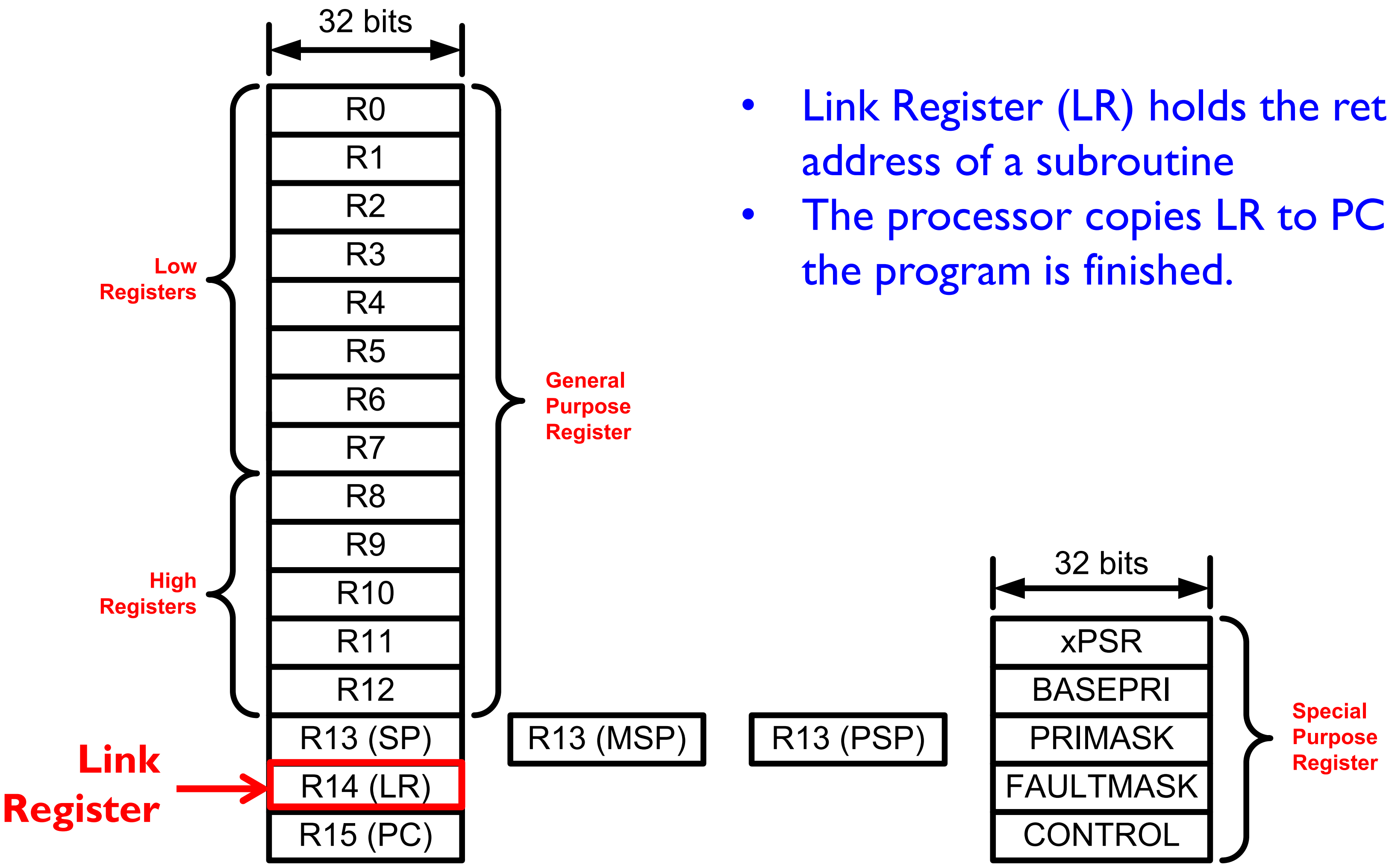
Z. Gu

Fall 2025

Overview

- How to call a subroutine?
- How to return the control back to the caller?
- How to pass arguments into a subroutine?
- How to return a value in a subroutine?
- How to preserve the running environment for the caller?

Link Register (LR)



- Link Register (LR) holds the return address of a subroutine
- The processor copies LR to PC after the program is finished.

Register Bank

Special Registers

Call a Subroutine (BL)

Branch with Link

BL *label*

- `Step 1: LR = PC + 4`
- `Step 2: PC = label`

- Notes:
  - *label* is name of subroutine
  - Compiler translates label to memory address
  - After call, LR holds return address (the instruction following the call)

Caller Program

```
    MOV r4, #100
    ...
    BL  foo
    ...
```

Subroutine/Callee

```
foo PROC
     ...
     MOV   r4, #10
     ...
     BX    LR
     ENDP
```

Return from a Subroutine (BX LR)

Caller Program

```
   MOV r4, #100
   ...
   BL  foo
   ...
```

Subroutine/Callee

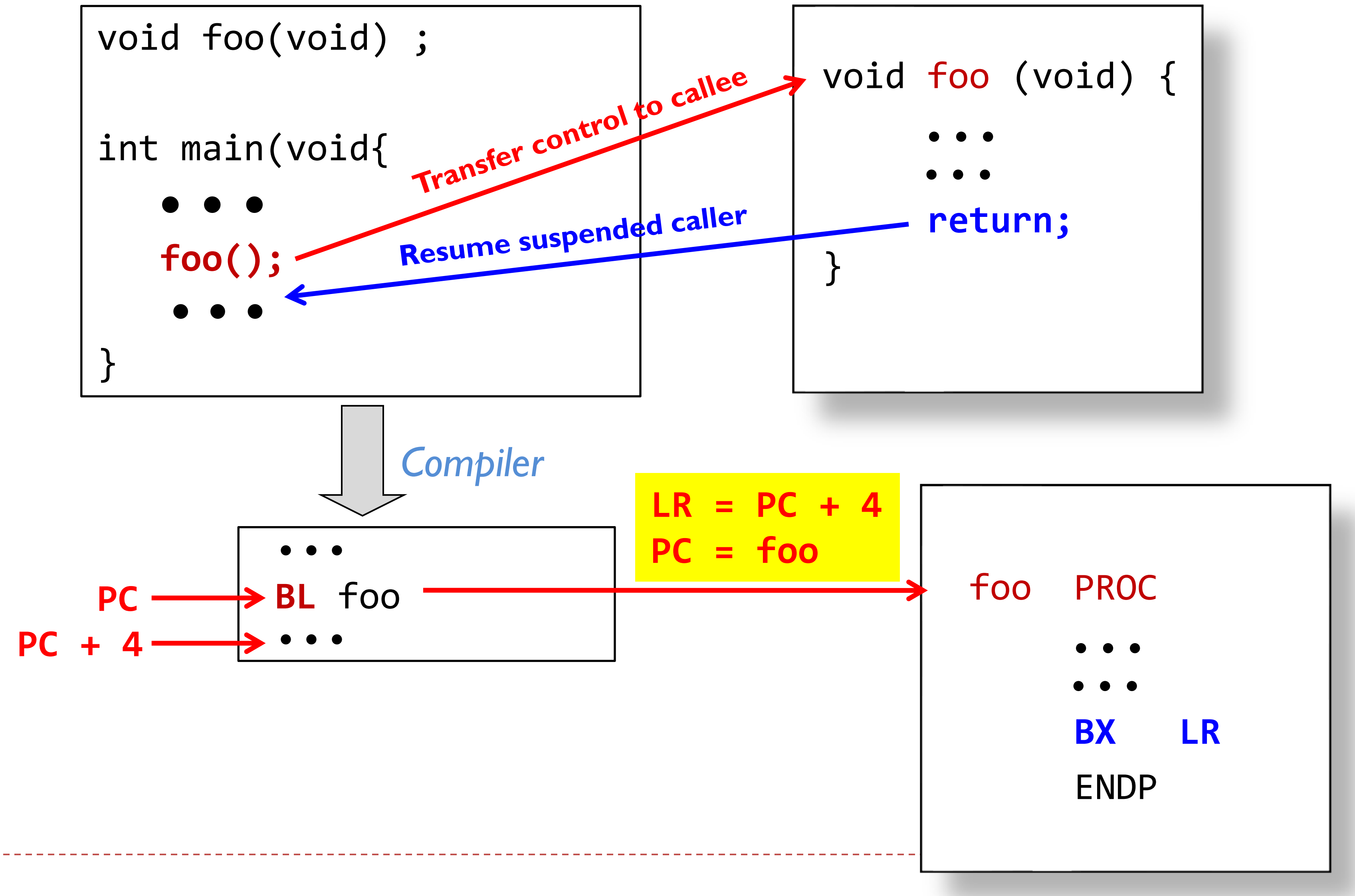
```
foo PROC
    ...
    MOV   r4, #10
    ...
    BX    LR
    ENDP
```

Branch and Exchange

BX *LR*

- PC = LR

BL and BX



```
void foo(void) ;

int main(void{
    • • •
    foo();
    • • •
}
```

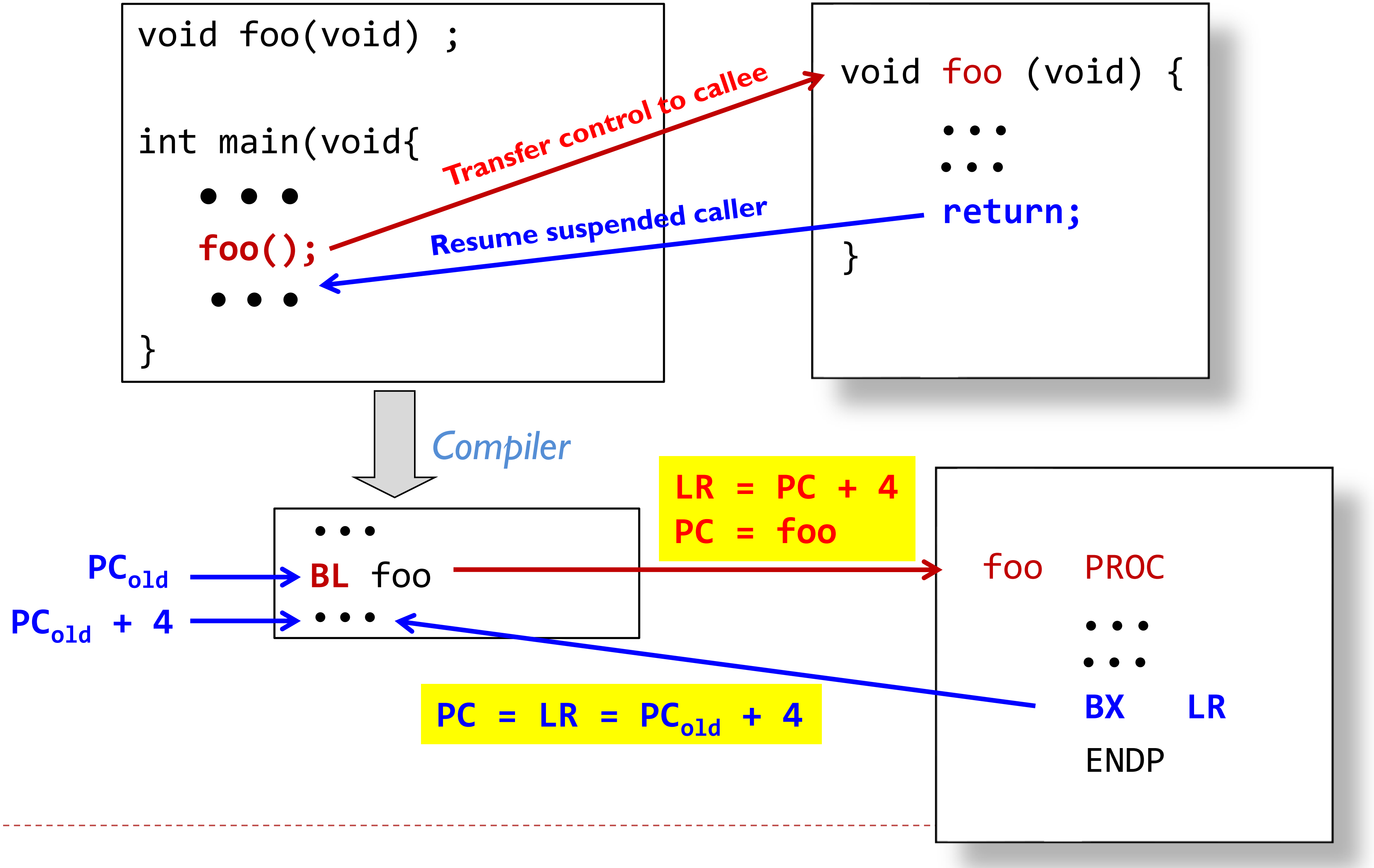
```
void foo (void) {
    • • •
    • • •
    return;
}
```

```
• • •
BL foo
• • •
```

```
LR = PC + 4
PC = foo
```

```
foo   PROC
      • • •
      • • •
      BX    LR
      ENDP
```

BL and BX



```
void foo(void) ;

int main(void{
    • • •
    foo();
    • • •
}
```

Transfer control to callee

Resume suspended caller

```
void foo (void) {
    • • •
    • • •
    return;
}
```

Compiler

```
  • • •
  BL  foo
  • • •
```

PC_{old}

PC_{old} + 4

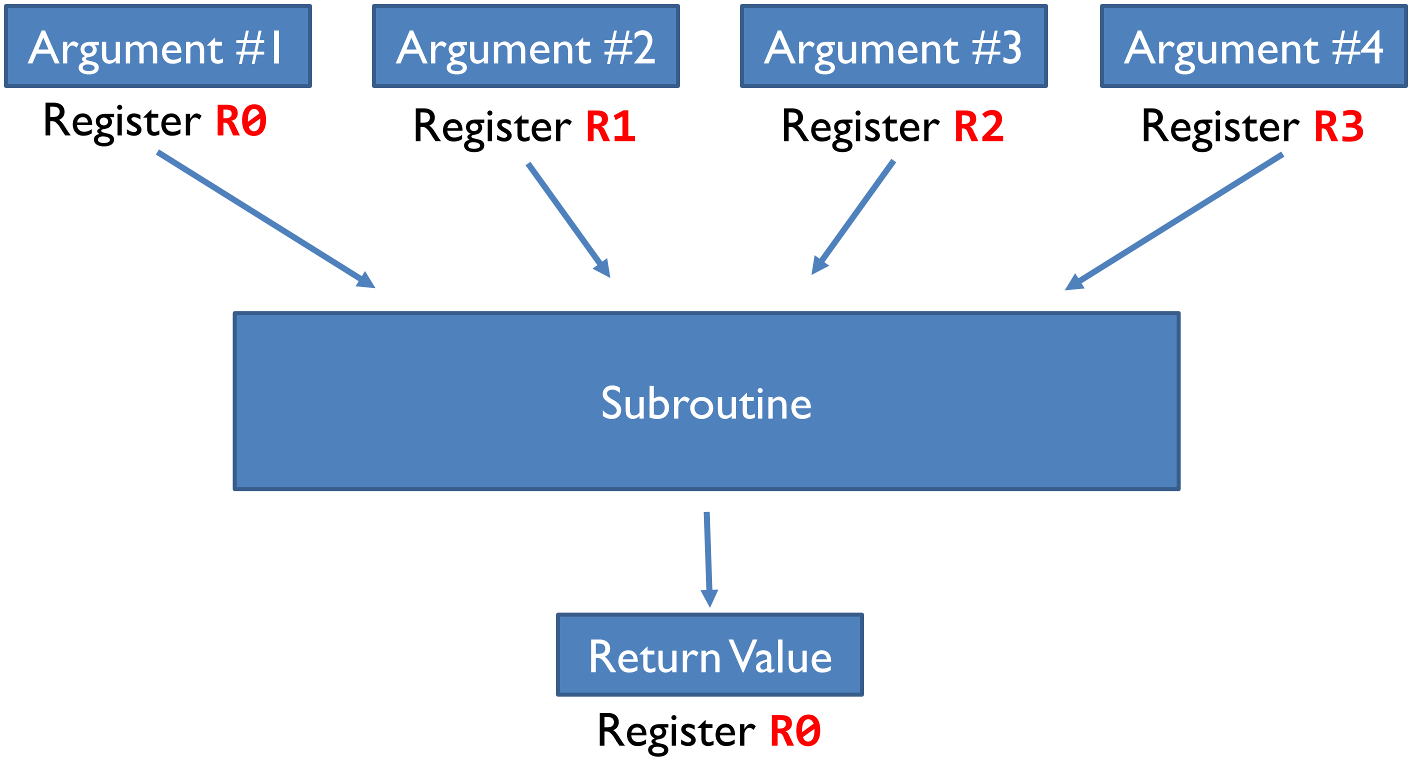
LR = PC + 4
PC = foo

```
foo   PROC
        • • •
        • • •
      BX    LR
      ENDP
```

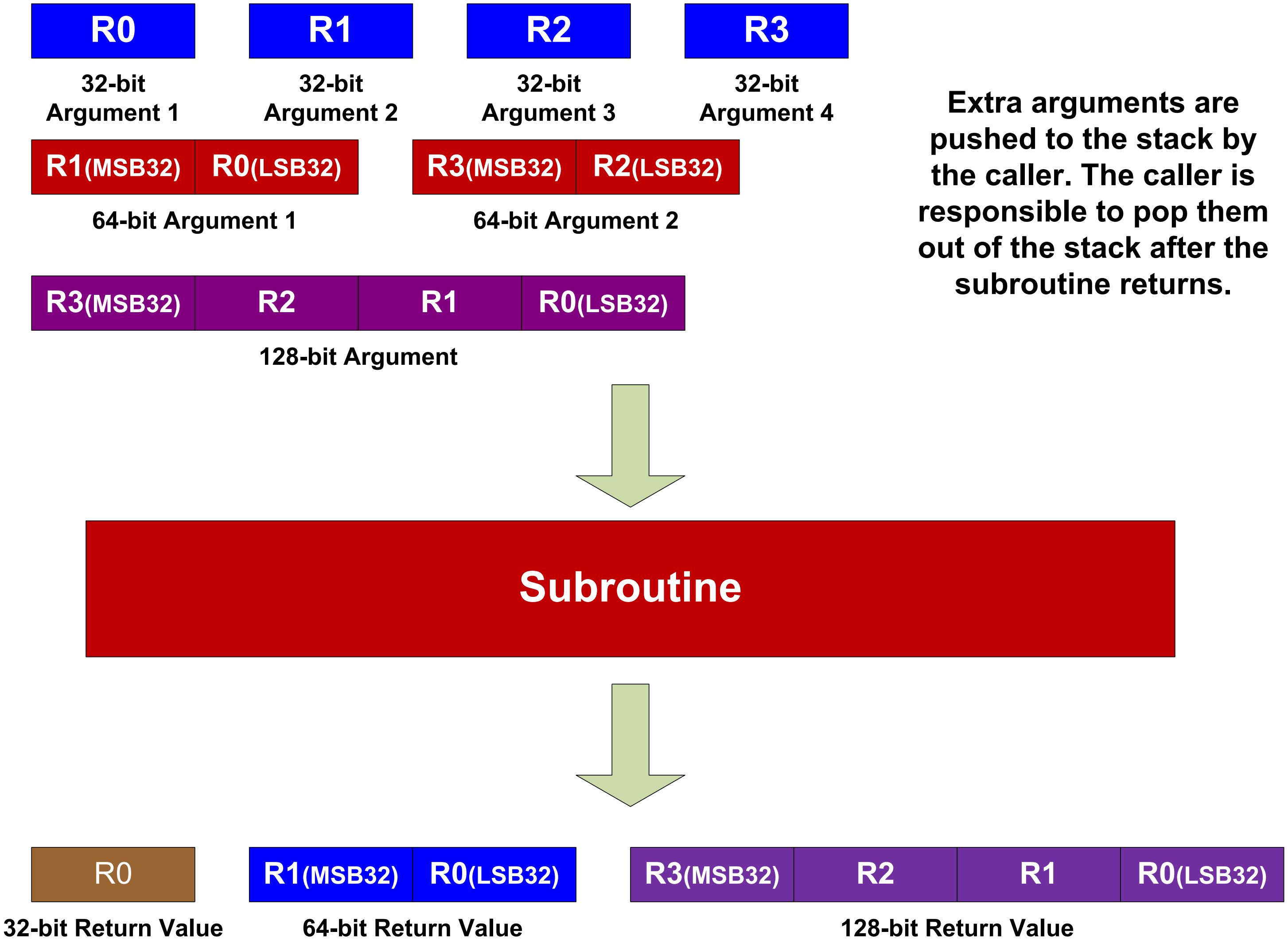
PC = LR = PC_{old} + 4

Passing arguments and Returning Value

- ARM Architecture Procedure Call Standard (AAPCS)
- First four registers are used to pass argument values into a subroutine and to return a value from a subroutine



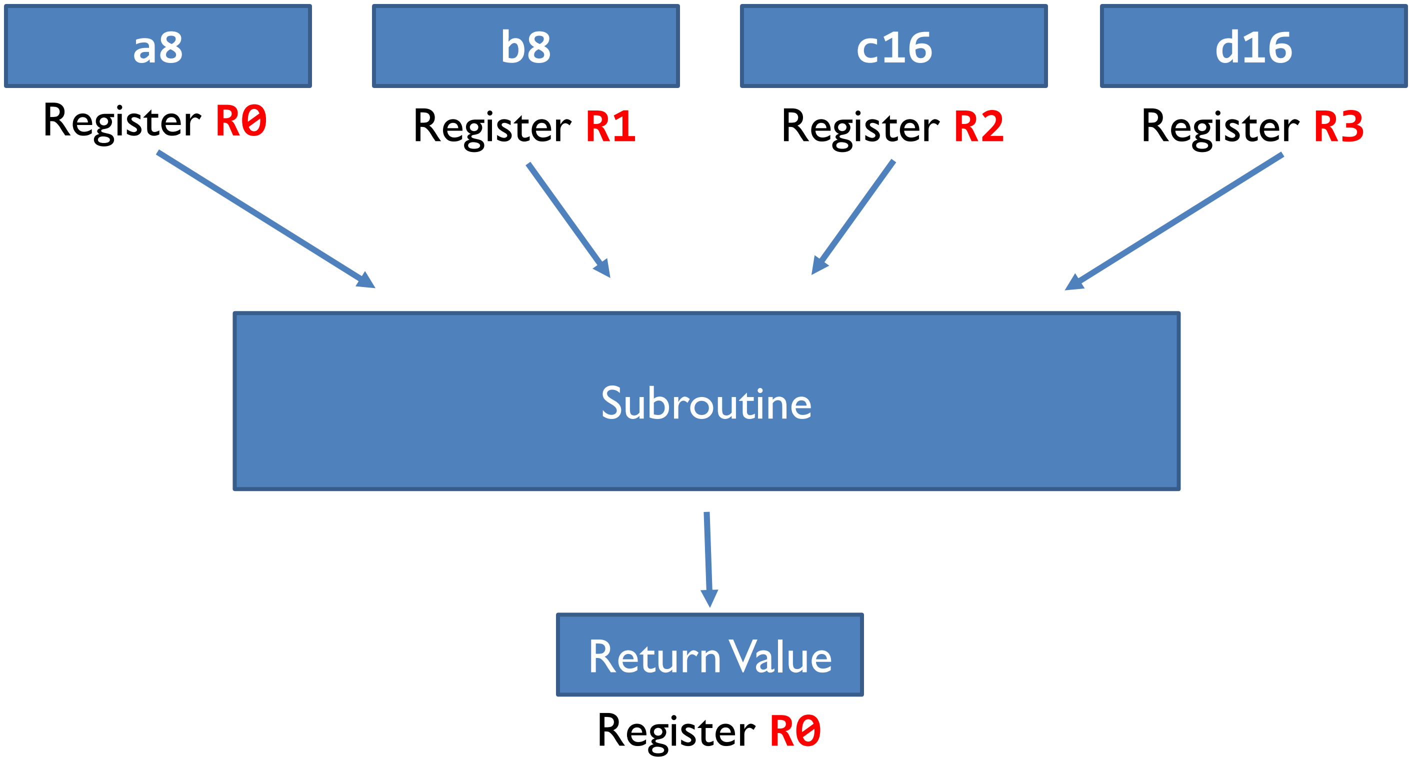
Passing arguments and Returning Value



Extra arguments are pushed to the stack by the caller. The caller is responsible to pop them out of the stack after the subroutine returns.

Passing arguments and Returning Value

```
uint32_t sum(uint8_t a8, uint8_t b8, uint16_t c16, uint16_t d16);
```



Passing arguments and Returning Value

```
uint32_t sum(uint8_t a8, uint8_t b8, uint16_t c16, uint16_t d16);
```

```
s = sum(1, 2, 3, 4);
```

Caller

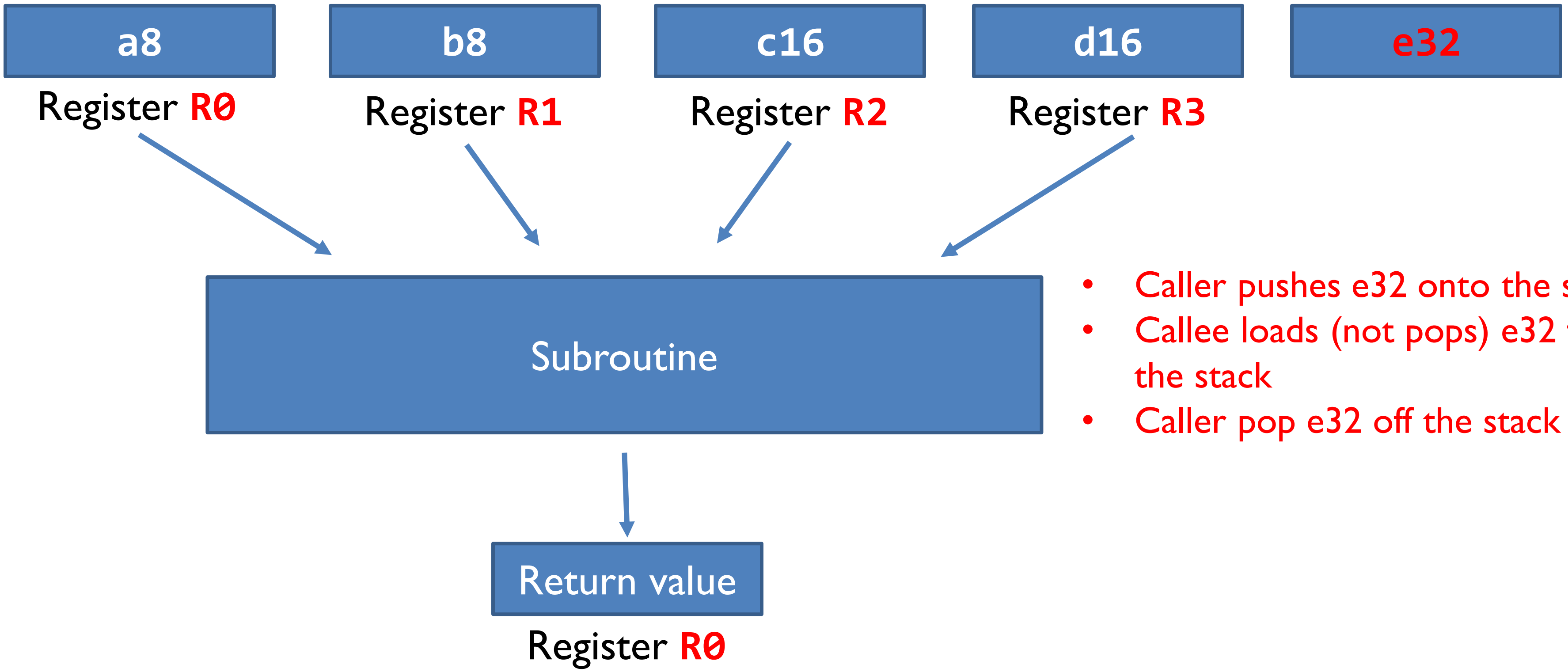
```
   MOVS r0, #1   ; a8
   MOVS r1, #2   ; b8
   MOVS r2, #3   ; c16
   MOVS r3, #4   ; d16
   BL   sum
```

Callee

```
sum PROC
   ADD r0, r0, r1   ; a8 + b8
   ADD r0, r0, r2   ; add c16
   ADD r0, r0, r3   ; add d16
   BX  LR
   ENDP
```

Passing arguments and Returning Value

```
uint32_t sum(uint8_t a, uint8_t b, uint16_t c, uint16_t d, uint32_t e);
```



- Caller pushes e32 onto the stack
- Callee loads (not pops) e32 from the stack
- Caller pop e32 off the stack

Passing arguments and Returning Value

```
uint32_t sum(uint8_t a8, uint8_t b8, uint16_t c16, uint16_t d16,
uint32_t e32);
```

```
s = sum(1, 2, 3, 4, 5);
```

Caller

```
   MOVS r0, #5 ; e32
   PUSH {r0}
   MOVS r0, #1 ; a8
   MOVS r1, #2 ; b8
   MOVS r2, #3 ; c16
   MOVS r3, #4 ; d16
   BL   sum
   ...
   POP {r0}
```

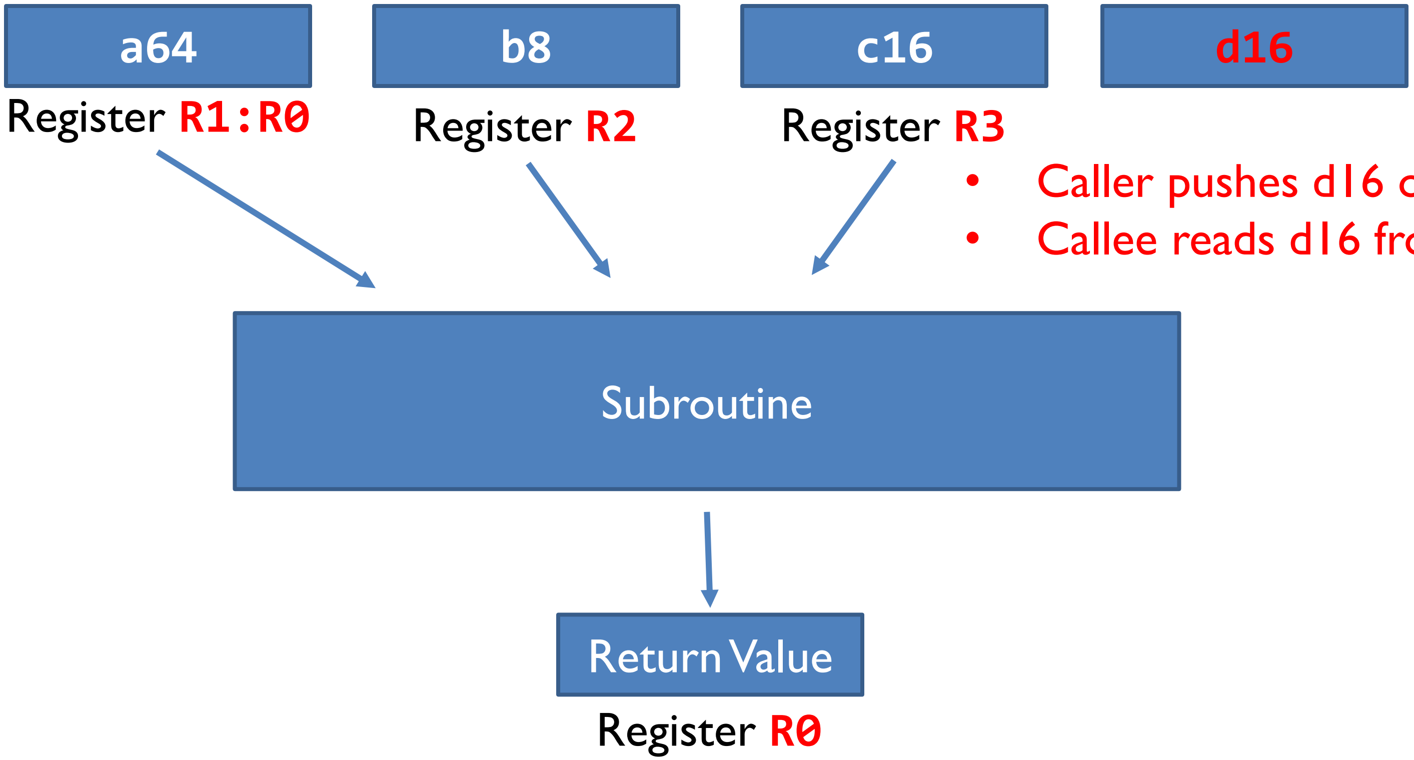
Callee

```
sum PROC
   ADD r0, r0, r1     ; a8 + b8
   ADD r0, r0, r2     ; add c16
   ADD r0, r0, r3     ; add d16
   LDR r1, [sp, #0]   ; read argument e32
   ADD r0, r0, r1     ; add e32
   BX  LR
   ENDP
```

The caller is responsible to pop extra arguments out of the stack after the subroutine returns.

Passing arguments and Returning Value

```
uint64_t sum(uint64_t a64, uint8_t b8, uint16_t c16, uint16_t d16);
```



- Caller pushes d16 onto stack
- Callee reads d16 from stack

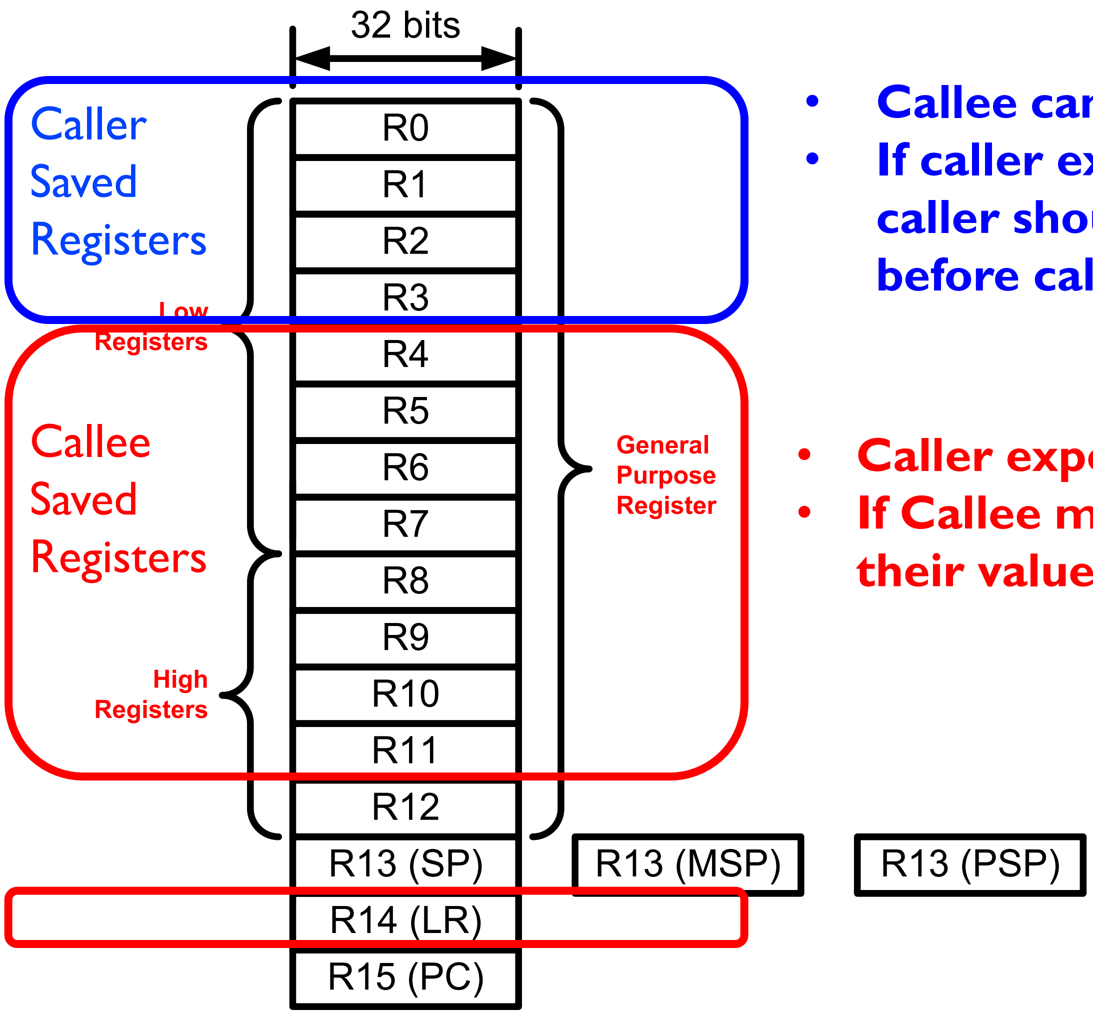
Returning Value

```
uint32_t s32;
uint32_t sum(uint8_t a8, uint8_t b8, uint16_t c16, uint16_t d16);

s32 = sum(1, 2, 3, 4) + 100;
```

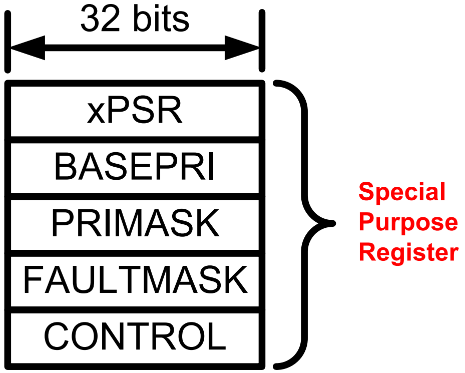
```
MOVS r0, #1   ; 1st argument a8
MOVS r1, #2   ; 2nd argument b8
MOVS r2, #3   ; 3rd argument c16
MOVS r3, #4   ; 4th argument d16
BL   sum      ; result is returned in r0
ADDS r0, r0, #100
LDR  r4, =s32   ; Get memory address of s32
STR  r0, [r4]   ; Save returned result to s32
```

Callee Saved Registers *vs* Caller Saved Registers



- **Callee can freely modify R0, R1, R2, and R3**
- **If caller expects their values are retained, caller should push them onto the stack before calling the callee**

- **Caller expects these values are retained .**
- **If Callee modifies them, callee must restore their values upon leaving the function.**



ARM Procedure Call Standard

Register	Usage	Subroutine Preserved	Notes
r0	Argument 1 and return value	No	If return has 64 bits, then r0:r1 hold it. If argument 1 has 64 bits, r0:r1 hold it.
r1	Argument 2	No	
r2	Argument 3	No	If the return has 128 bits, r0-r3 hold it.
r3	Argument 4	No	If more than 4 arguments, use the stack
r4	General-purpose V1	Yes	Variable register 1 holds a local variable.
r5	General-purpose V2	Yes	Variable register 2 holds a local variable.
r6	General-purpose V3	Yes	Variable register 3 holds a local variable.
r7	General-purpose V4	Yes	Variable register 4 holds a local variable.
r8	General-purpose V5	Yes	Variable register 5 holds a local variable.
r9	Platform specific/V6	Yes/No	Usage is platform-dependent.
r10	General-purpose V7	Yes	Variable register 7 holds a local variable.
r11	General-purpose V8	Yes	Variable register 8 holds a local variable.
r12 (IP)	Intra-procedure-call register	No	It holds intermediate values between a procedure and the sub-procedure it calls.
r13 (SP)	Stack pointer	Yes	SP has to be the same after a subroutine has completed.
r14 (LR)	Link register	No	Receives return address on BL call to procedure
r15 (PC)	Program counter	N/A	Do not directly change PC

Common Coding Patterns

- Callee returns a constant in r0.
  - mov r0,#17   @ r0 is return value register
  - bx lr   @ return from function
- Callee saves some registers, does some arithmetic, and returns the result in r0.
  - push {r4-r7,lr}
  - mov r4, #10
  - mov r5, #100
  - add r0,r4,r5
  - pop {r4-r7,pc}   @ pop saved lr value into PC to return from function
- Callee calls another function (nested function calls)
  - push {lr}   @ must save LR if we call our own function
  - mov r0, #123   @ r0 is first function parameter
  - bl print_int @ call function print_int(123)
  - pop {pc}   @ pop saved lr into PC to return from function
- Callee return: restore previously-pushed LR, then jump to LR (POP {lr}; BX lr), or equivalently, pop previously-pushed LR to PC
  - POP {pc} ≡ POP {lr}; BX lr

Common Coding Patterns

- Memory access: first put memory address into register, then load memory content at that address
  - adr r2, mydata @Compute address of label mydata using a PC-relative add and put that address in r2
  - ldr r0,[r2] @Dereference that address, loading the 32-bit word stored at mydata into r0; after this, r0 = 123.
  - bx lr
  - mydata:
  - .word 123
- Or
  - ldr r2,=mydata @ pseudo-instruction that loads absolute address of mydata from a nearby literal pool into r2
  - ldr r0,[r2]
  - bx lr
  - mydata:
  - .word 123
- adr vs. ldr
  - If mydata is in range for adr, both forms will leave r2 holding the same address at run time.
  - Out-of-range labels: adr may fail; ldr =mydata still works.

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX LR
        ENDP
        ...
```

Sum of Square: $x^2 + y^2$

R1: second argument

R0: first argument

```
int SSQ(int x, int y){
        int z;
        z = x*x + y*y;
        return z;
}
```

R0: Return Value

Example: SSQ(3, 4)

```
; Caller setup (passes x=3, y=4; calls SSQ; uses returned value)
MOV R0, #3        ; Load arg1 x=3 into R0
MOV R1, #4        ; Load arg2 y=4 into R1
BL  SSQ           ; Call SSQ: LR ← return address, PC ← SSQ entry; R0,R1 carry x,y
MOV R2, R0        ; Save returned result z from R0 into caller temp R2
B   ENDL
...

; Callee (SSQ) computes z = x*x + y*y and returns it
SSQ PROC
MUL R2, R0, R0    ; R2 = R0 * R0 = x*x
MUL R3, R1, R1    ; R3 = R1 * R1 = y*y
ADD R2, R2, R3    ; R2 = (x*x) + (y*y) = z
MOV R0, R2        ; Move result z into R0
BX  LR            ; Return to caller: branch to address in LR
ENDP

; Register roles
; R0: x on entry, z on return
; R1: y on entry
; R2: temp for x*x and then z in callee; holds z in caller after MOV R2,R0
; R3: temp for y*y in callee
```

Example: SSQ(3,4)

```
     MOV R0,#3
     MOV R1,#4
     BL  SSQ
     MOV R2,R0
     B  ENDL

SSQ  MUL R2,R0,R0
     MUL R3,R1,R1
     ADD R2,R2,R3
     MOV R0,R2
     BX  LR
ENDL ...
```

R0	
R1	
R2	
R3	

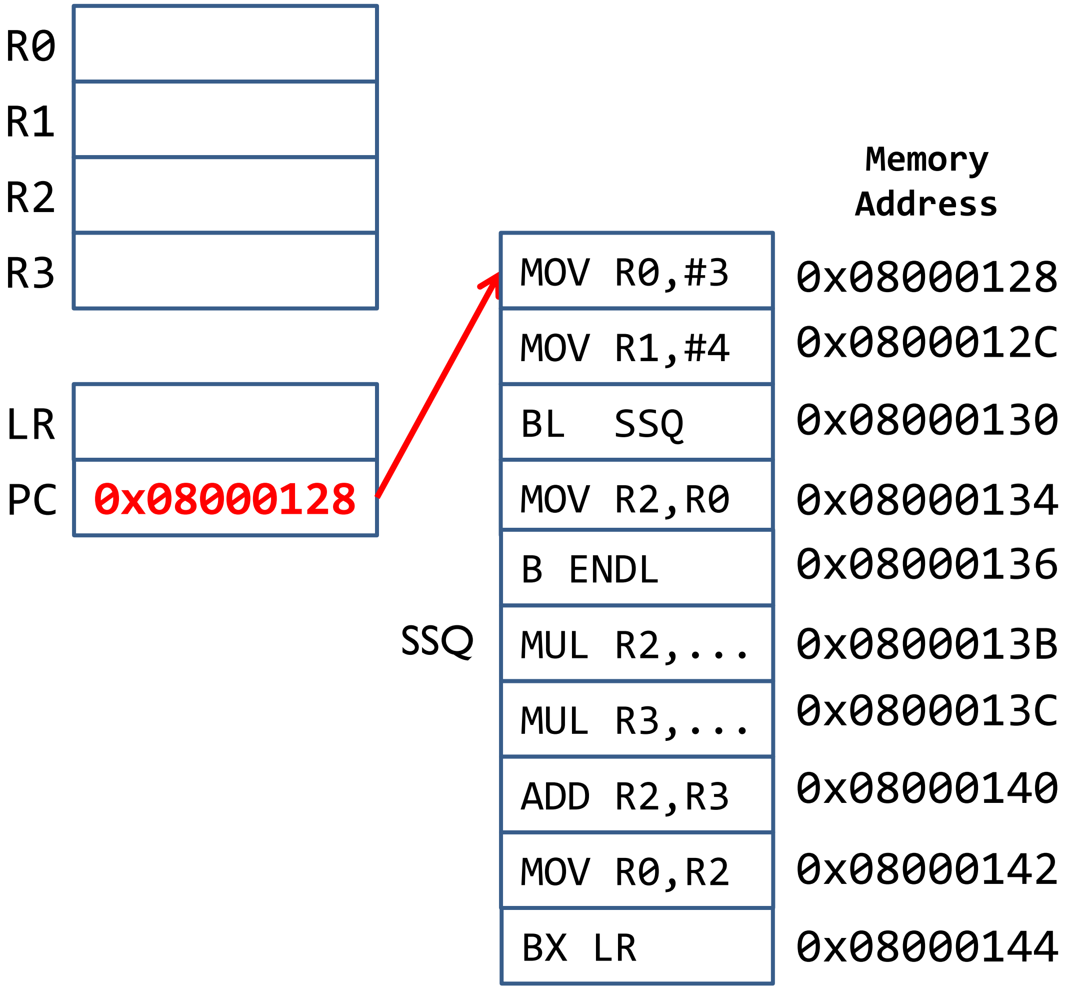
LR	
PC	

pc = 0x08000128

		Memory Address
	MOV R0,#3	0x08000128
	MOV R1,#4	0x0800012C
	BL SSQ	0x08000130
	MOV R2,R0	0x08000134
	B ENDL	0x08000136
SSQ	MUL R2,...	0x0800013A
	MUL R3,...	0x0800013C
	ADD R2,R3	0x08000140
	MOV R0,R2	0x08000142
	BX LR	0x08000144

Example: SSQ(3, 4)

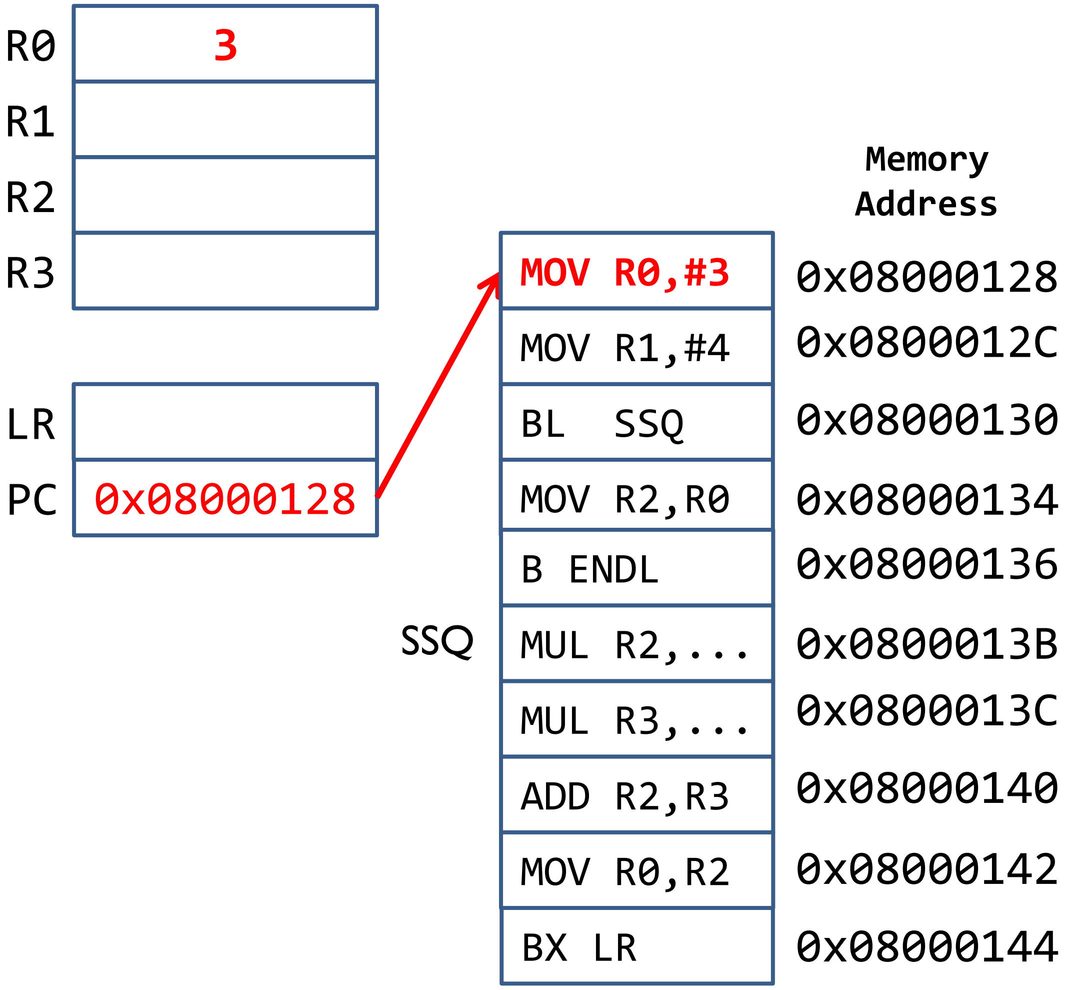
```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



	Memory Address
MOV R0,#3	0x08000128
MOV R1,#4	0x0800012C
BL SSQ	0x08000130
MOV R2,R0	0x08000134
B ENDL	0x08000136
MUL R2,...	0x0800013B
MUL R3,...	0x0800013C
ADD R2,R3	0x08000140
MOV R0,R2	0x08000142
BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```

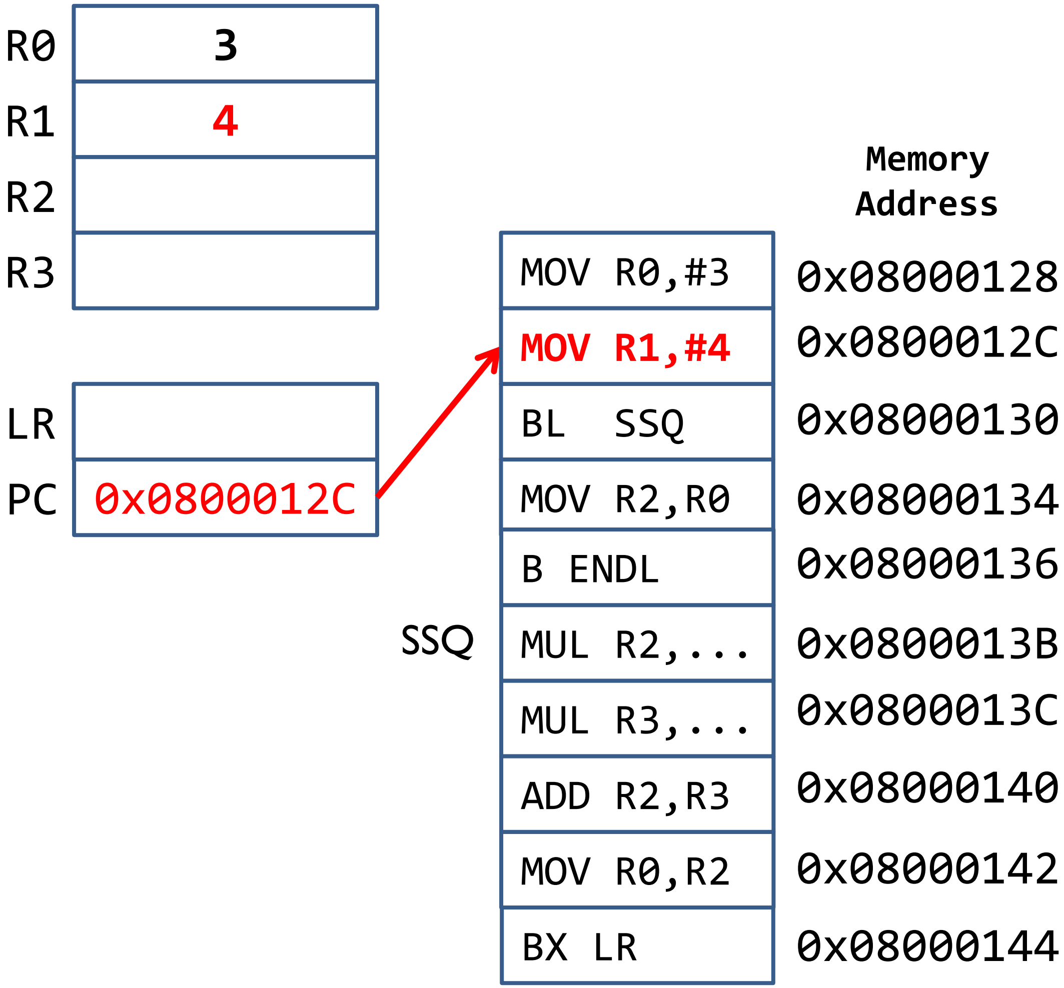


R0	3
R1	
R2	
R3	
LR	
PC	0x08000128

		Memory Address
	MOV R0,#3	0x08000128
	MOV R1,#4	0x0800012C
	BL SSQ	0x08000130
	MOV R2,R0	0x08000134
	B ENDL	0x08000136
SSQ	MUL R2,...	0x0800013B
	MUL R3,...	0x0800013C
	ADD R2,R3	0x08000140
	MOV R0,R2	0x08000142
	BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```

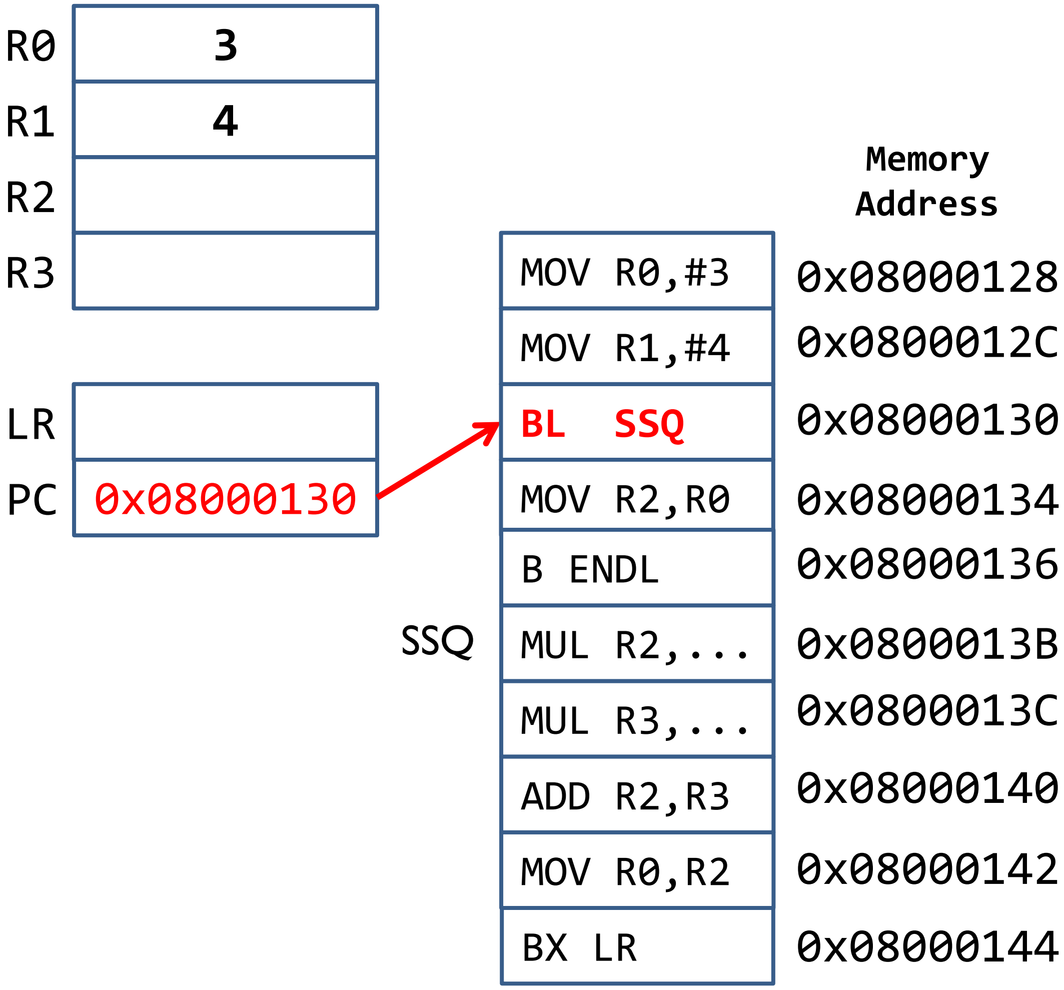


R0	3
R1	4
R2	
R3	
LR	
PC	0x0800012C

		Memory Address
	MOV R0,#3	0x08000128
	MOV R1,#4	0x0800012C
	BL SSQ	0x08000130
	MOV R2,R0	0x08000134
	B ENDL	0x08000136
SSQ	MUL R2,...	0x0800013B
	MUL R3,...	0x0800013C
	ADD R2,R3	0x08000140
	MOV R0,R2	0x08000142
	BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV  R0,#3
        MOV  R1,#4
        BL   SSQ
        MOV  R2,R0
        B  ENDL
        ...
SSQ     PROC
        MUL  R2,R0,R0
        MUL  R3,R1,R1
        ADD  R2,R2,R3
        MOV  R0,R2
        BX   LR
        ENDP
ENDL    ...
```



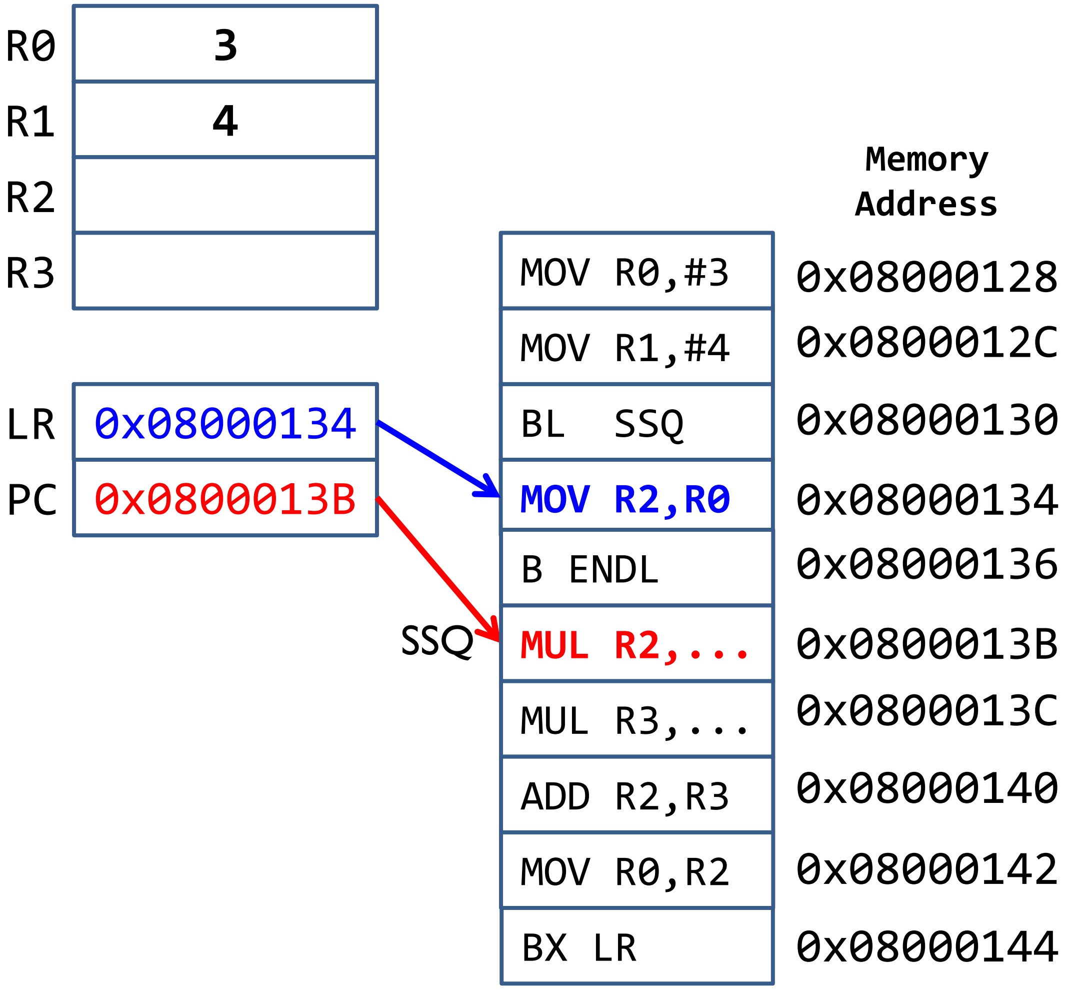
R0	3
R1	4
R2	
R3	

LR	
PC	0x08000130

		Memory Address
	MOV R0,#3	0x08000128
	MOV R1,#4	0x0800012C
	BL SSQ	0x08000130
	MOV R2,R0	0x08000134
	B ENDL	0x08000136
SSQ	MUL R2,...	0x0800013B
	MUL R3,...	0x0800013C
	ADD R2,R3	0x08000140
	MOV R0,R2	0x08000142
	BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



R0	3
R1	4
R2	
R3	

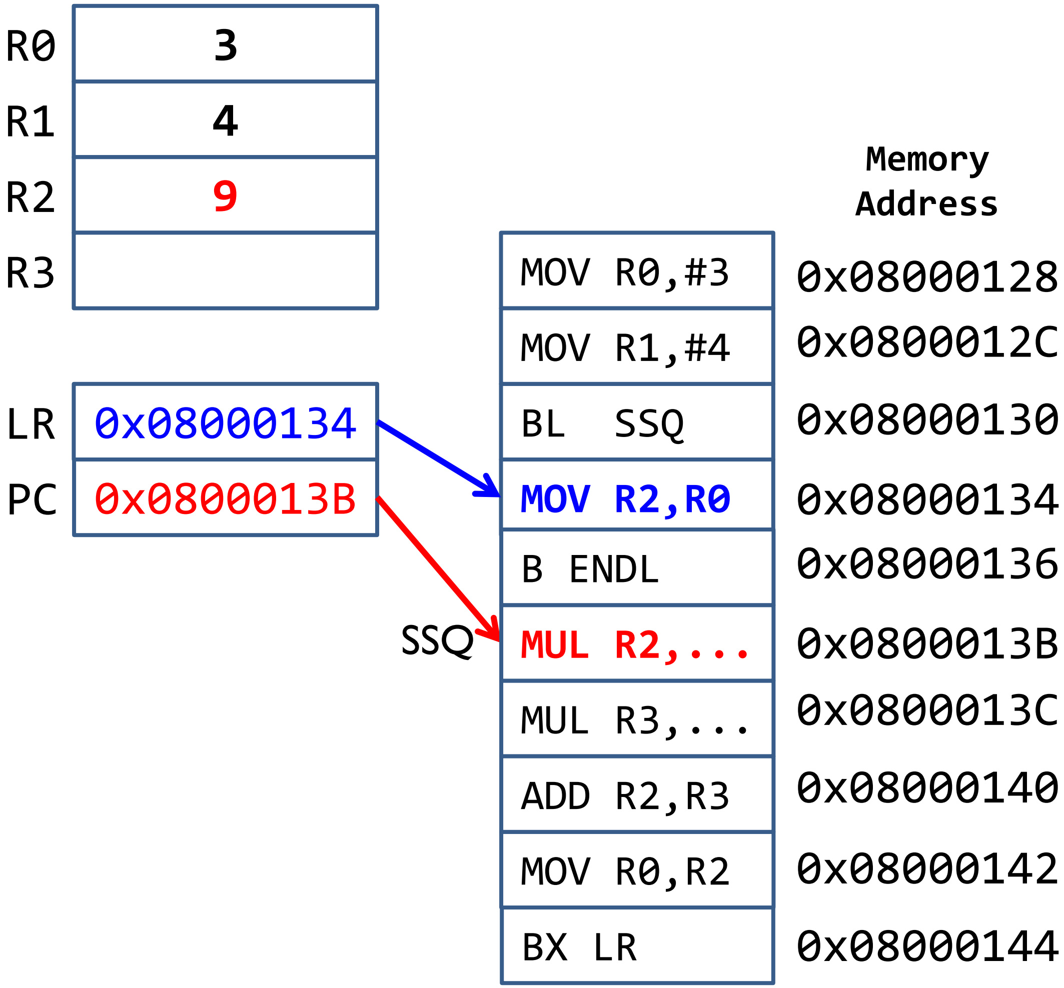
LR	0x08000134
PC	0x0800013B

	Memory Address
MOV R0,#3	0x08000128
MOV R1,#4	0x0800012C
BL SSQ	0x08000130
MOV R2,R0	0x08000134
B ENDL	0x08000136
MUL R2,...	0x0800013B
MUL R3,...	0x0800013C
ADD R2,R3	0x08000140
MOV R0,R2	0x08000142
BX LR	0x08000144

Address of the next instruction after the branch is saved into LR.

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



R0	3
R1	4
R2	9
R3	

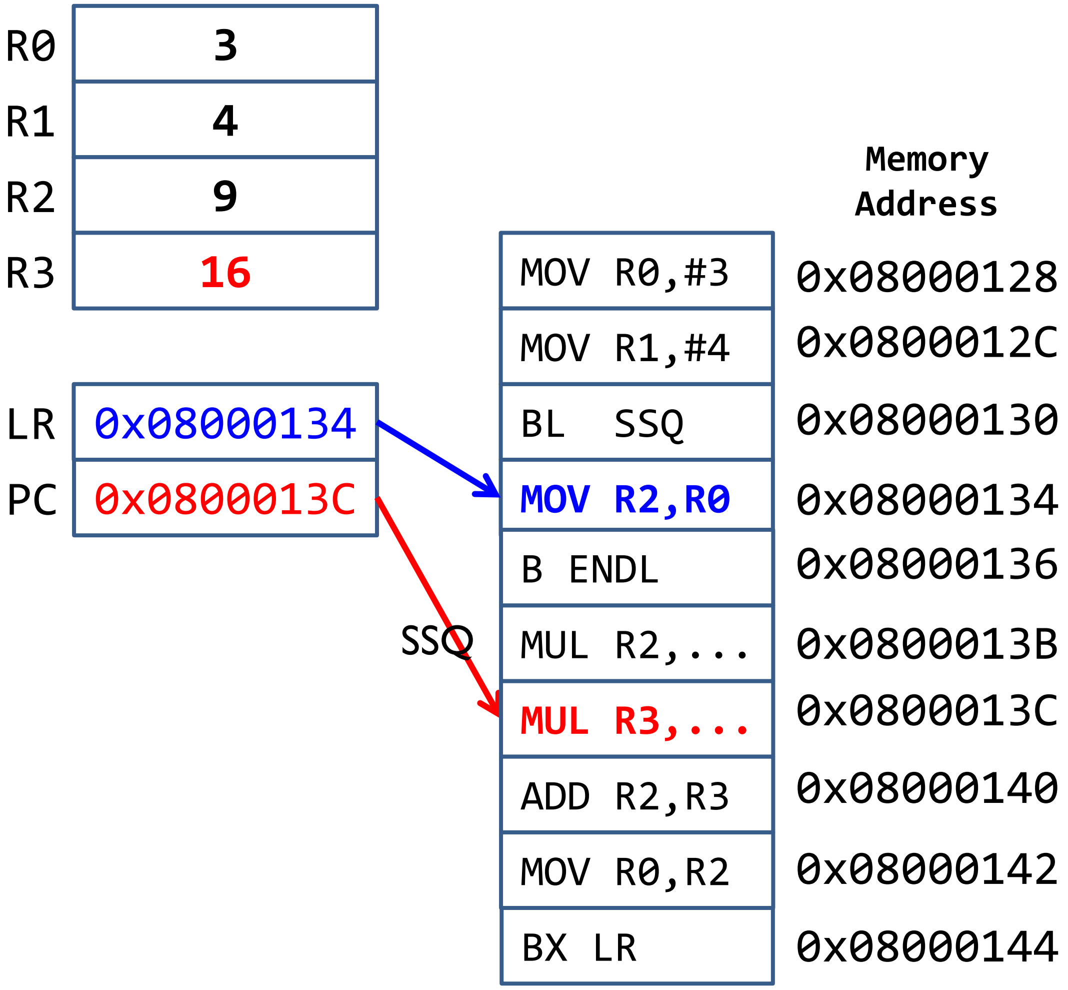
LR	0x08000134
PC	0x0800013B

SSQ

	Memory Address
MOV R0,#3	0x08000128
MOV R1,#4	0x0800012C
BL SSQ	0x08000130
MOV R2,R0	0x08000134
B ENDL	0x08000136
MUL R2,...	0x0800013B
MUL R3,...	0x0800013C
ADD R2,R3	0x08000140
MOV R0,R2	0x08000142
BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



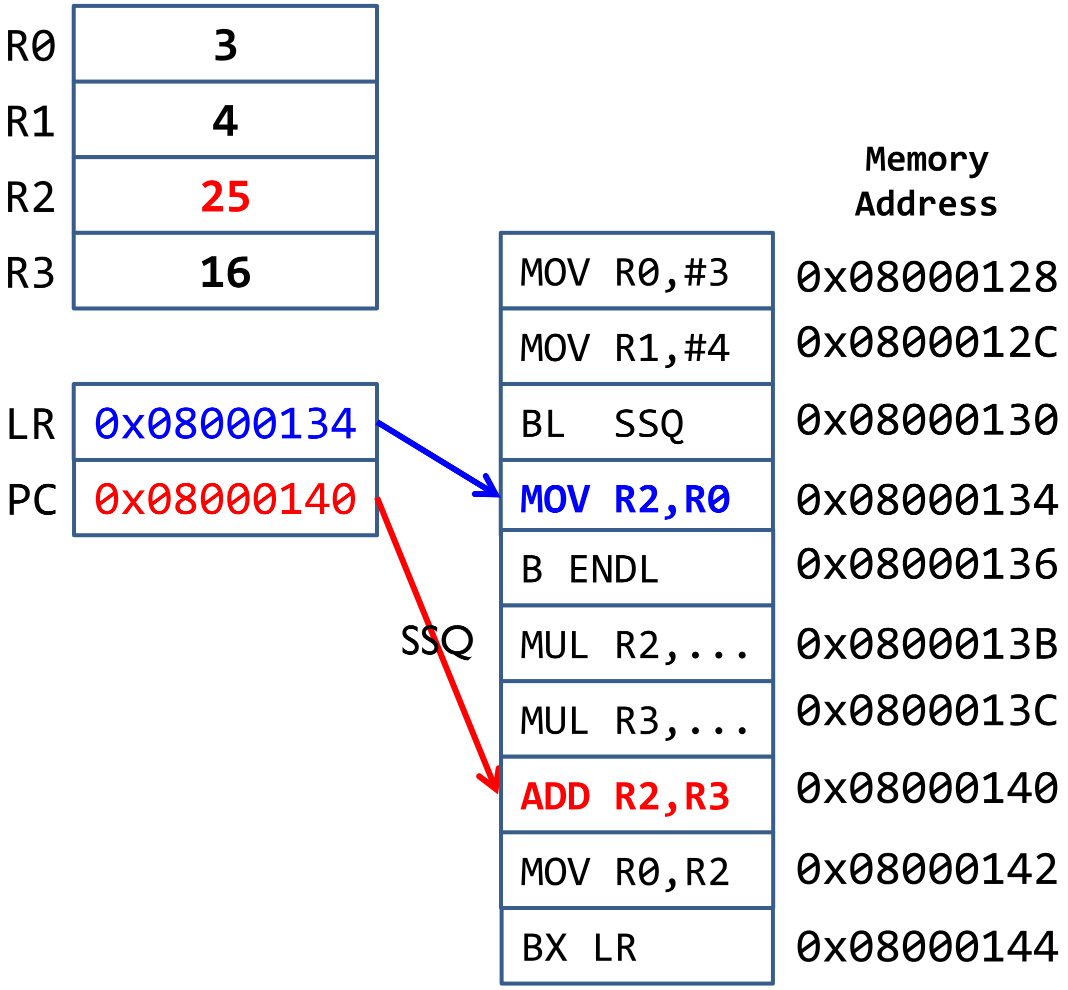
R0	3
R1	4
R2	9
R3	16

LR	0x08000134
PC	0x0800013C

	Memory Address
MOV R0,#3	0x08000128
MOV R1,#4	0x0800012C
BL SSQ	0x08000130
MOV R2,R0	0x08000134
B ENDL	0x08000136
MUL R2,...	0x0800013B
MUL R3,...	0x0800013C
ADD R2,R3	0x08000140
MOV R0,R2	0x08000142
BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



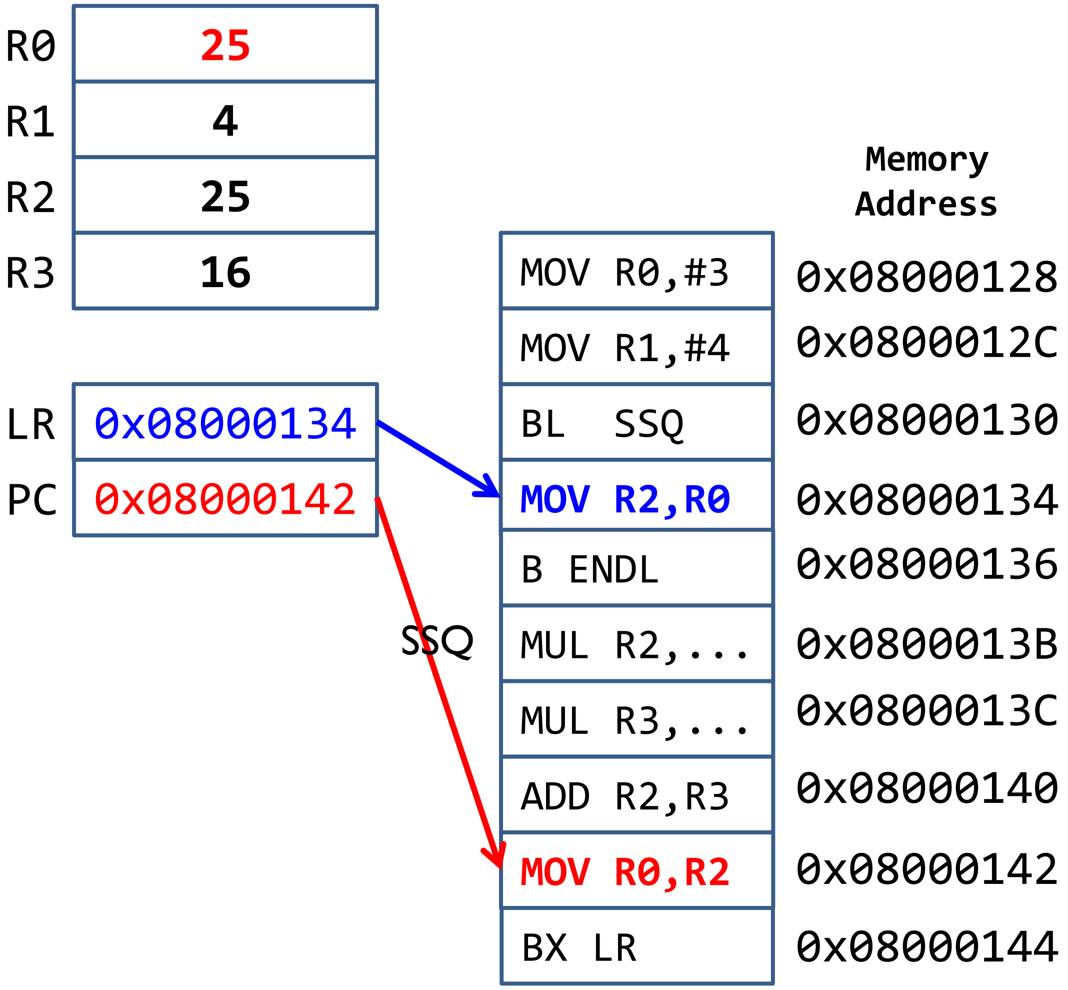
R0	3
R1	4
R2	25
R3	16

LR	0x08000134
PC	0x08000140

	Memory Address
MOV R0,#3	0x08000128
MOV R1,#4	0x0800012C
BL SSQ	0x08000130
MOV R2,R0	0x08000134
B ENDL	0x08000136
MUL R2,...	0x0800013B
MUL R3,...	0x0800013C
ADD R2,R3	0x08000140
MOV R0,R2	0x08000142
BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



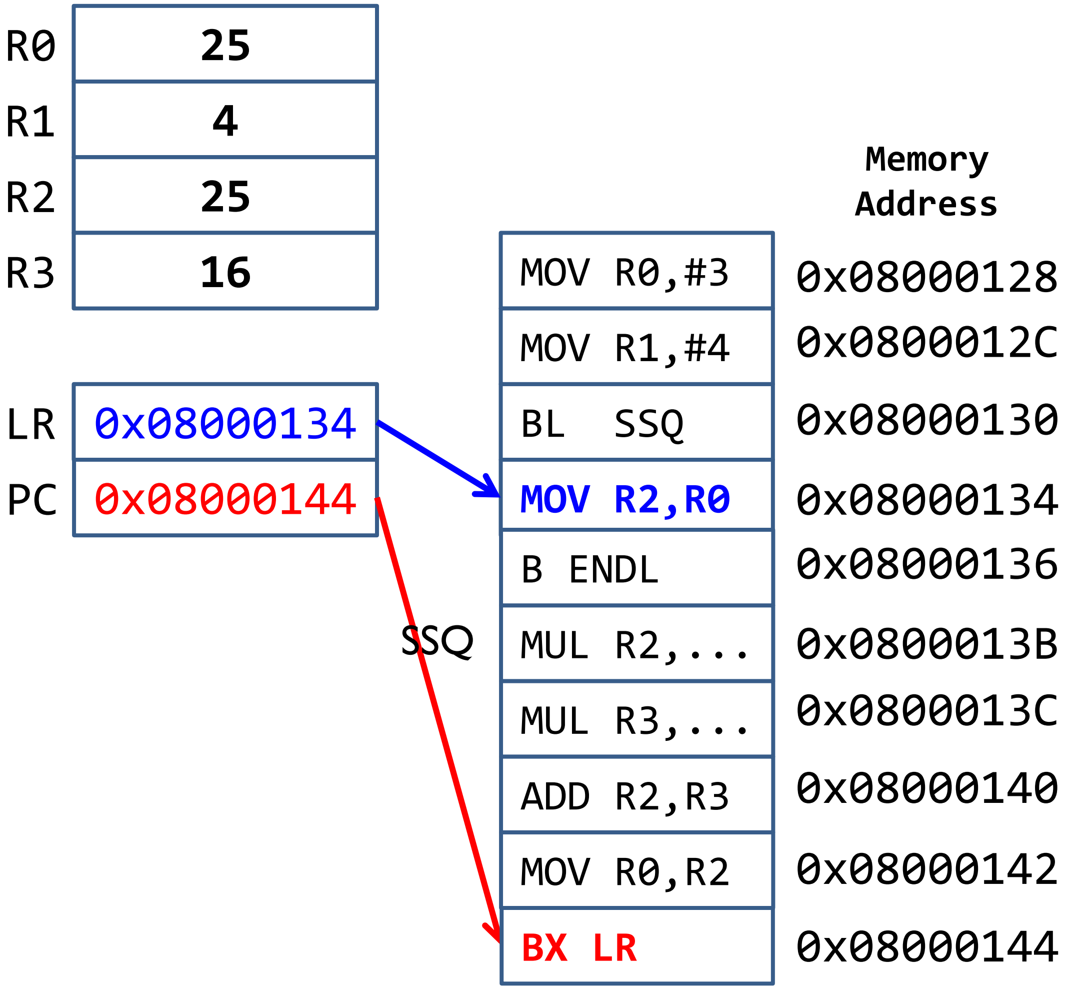
R0	25
R1	4
R2	25
R3	16

LR	0x08000134
PC	0x08000142

		Memory Address
	MOV R0,#3	0x08000128
	MOV R1,#4	0x0800012C
	BL SSQ	0x08000130
	MOV R2,R0	0x08000134
	B ENDL	0x08000136
SSQ	MUL R2,...	0x0800013B
	MUL R3,...	0x0800013C
	ADD R2,R3	0x08000140
	MOV R0,R2	0x08000142
	BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



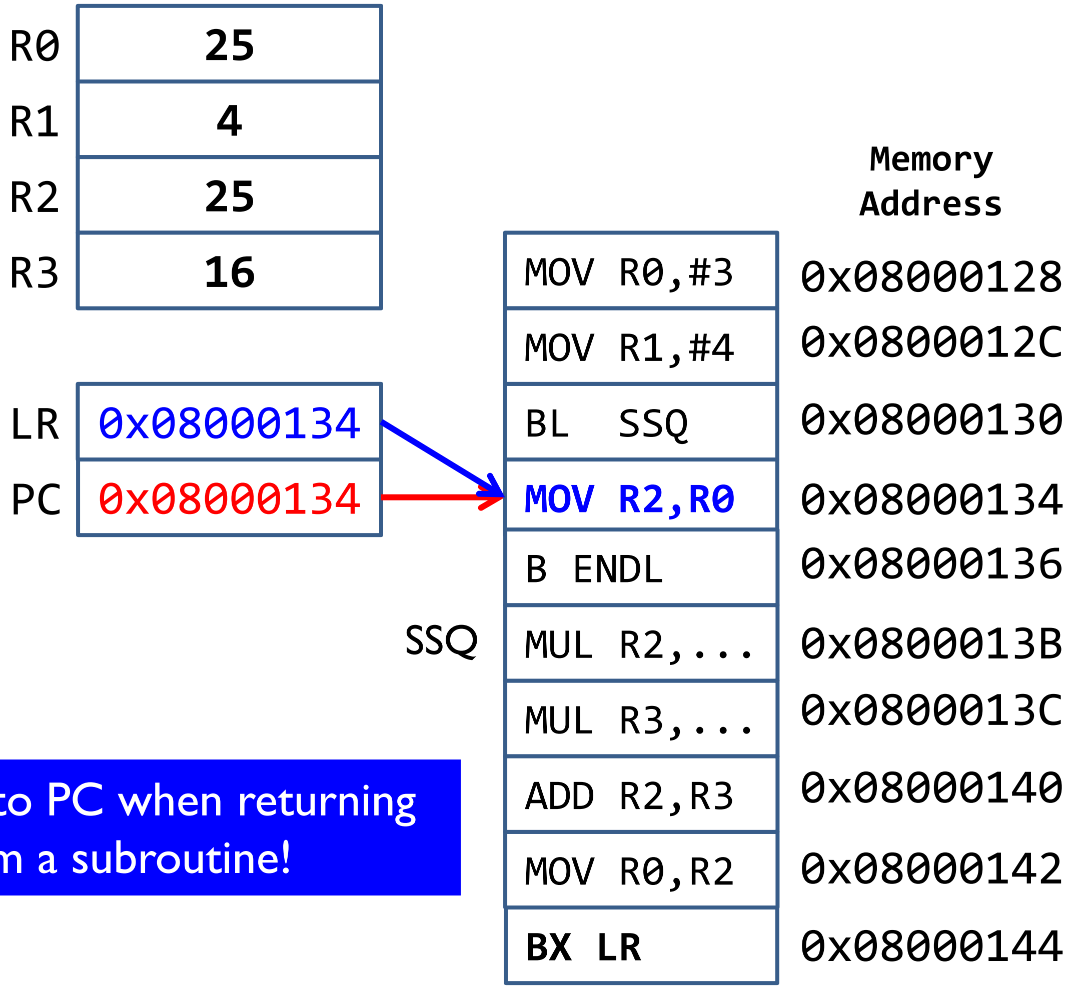
R0	25
R1	4
R2	25
R3	16

LR	0x08000134
PC	0x08000144

	Memory Address
MOV R0,#3	0x08000128
MOV R1,#4	0x0800012C
BL SSQ	0x08000130
MOV R2,R0	0x08000134
B ENDL	0x08000136
MUL R2,...	0x0800013B
MUL R3,...	0x0800013C
ADD R2,R3	0x08000140
MOV R0,R2	0x08000142
BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



R0	25
R1	4
R2	25
R3	16

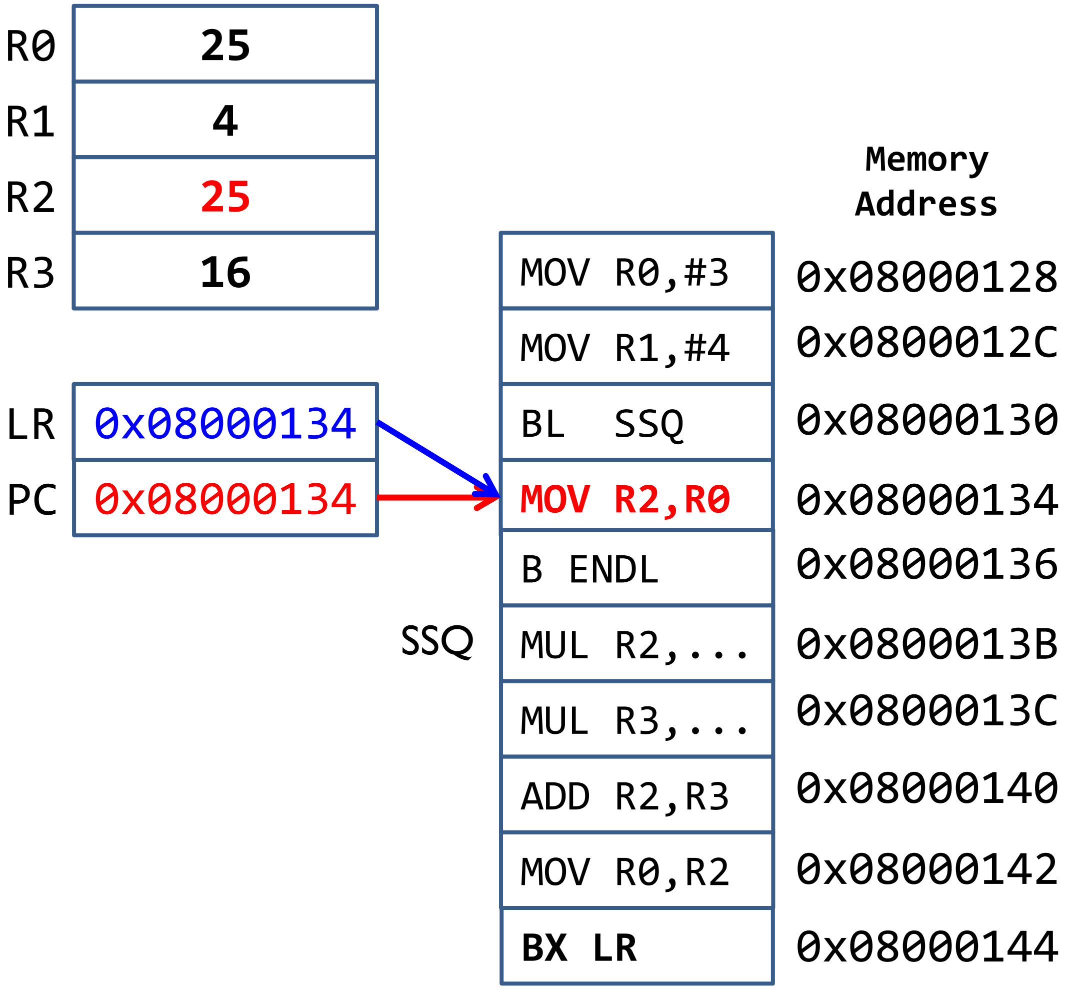
LR	0x08000134
PC	0x08000134

		Memory Address
	MOV R0,#3	0x08000128
	MOV R1,#4	0x0800012C
	BL SSQ	0x08000130
	MOV R2,R0	0x08000134
	B ENDL	0x08000136
SSQ	MUL R2,...	0x0800013B
	MUL R3,...	0x0800013C
	ADD R2,R3	0x08000140
	MOV R0,R2	0x08000142
	BX LR	0x08000144

Copy LR to PC when returning from a subroutine!

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```

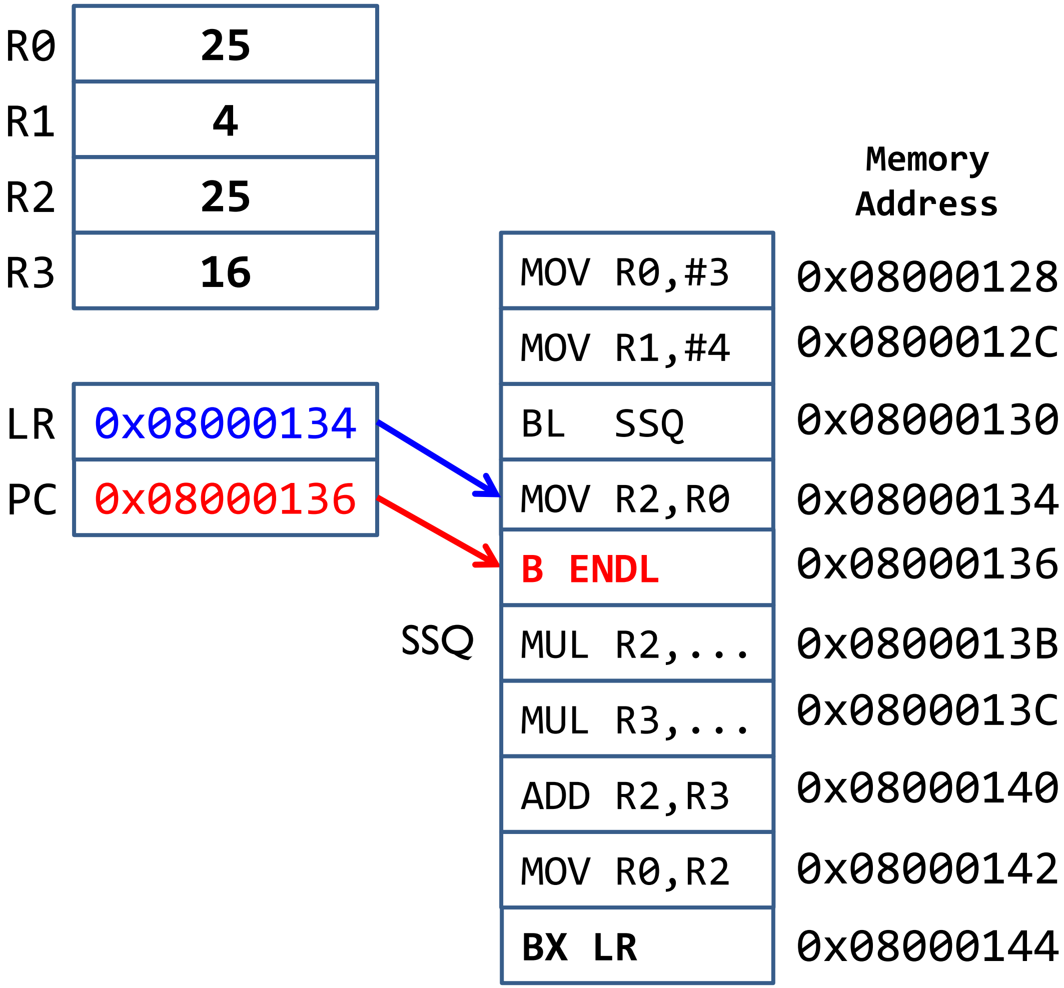


R0	25
R1	4
R2	25
R3	16
LR	0x08000134
PC	0x08000134

		Memory Address
	MOV R0,#3	0x08000128
	MOV R1,#4	0x0800012C
	BL SSQ	0x08000130
	MOV R2,R0	0x08000134
	B ENDL	0x08000136
SSQ	MUL R2,...	0x0800013B
	MUL R3,...	0x0800013C
	ADD R2,R3	0x08000140
	MOV R0,R2	0x08000142
	BX LR	0x08000144

Example: SSQ(3, 4)

```
        MOV R0,#3
        MOV R1,#4
        BL  SSQ
        MOV R2,R0
        B ENDL
        ...
SSQ     PROC
        MUL R2,R0,R0
        MUL R3,R1,R1
        ADD R2,R2,R3
        MOV R0,R2
        BX  LR
        ENDP
ENDL    ...
```



R0	25
R1	4
R2	25
R3	16

LR	0x08000134
PC	0x08000136

		Memory Address
	MOV R0,#3	0x08000128
	MOV R1,#4	0x0800012C
	BL SSQ	0x08000130
	MOV R2,R0	0x08000134
	B ENDL	0x08000136
SSQ	MUL R2,...	0x0800013B
	MUL R3,...	0x0800013C
	ADD R2,R3	0x08000140
	MOV R0,R2	0x08000142
	BX LR	0x08000144

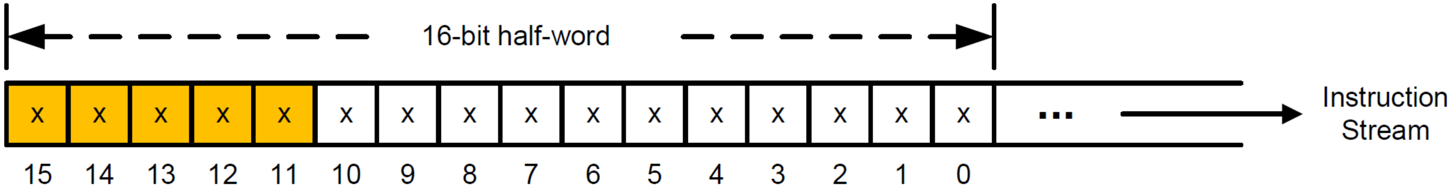
Realities

- In the previous example,
  - PC is incremented by 2 or 4.
  - The least significant bit of LR is always 0.

Well, I lied!

Realities

- PC is always incremented by **4**.
  - Each time, 4 bytes are fetched from the instruction memory
  - It is either two 16-bit instructions or one 32-bit instruction



If bit [15-11] = 11101, 11110, or 11111, then, it is the first half-word of a 32-bit instruction. Otherwise, it is a 16-bit instruction.

- The least significant bit of LR is always **1** for ARM Cortex-M
  - This bit is used to control the processor mode:
    - 0 = ARM, 1 = THUMB
  - Cortex-M only supports THUMB.

Summary

- How to call a subroutine?
  - Branch with link: **`BL subroutine`**
- How to return the control back to the caller?
  - Branch and exchange: **`BX LR`**
- How to pass arguments into a subroutine?
  - Each 8-, 16- or 32-bit variables is passed via `r0`, `r1`, `r2`, `r3`
  - Extra parameters are passed via stack
- How to return a value in a subroutine?
  - Value is returned in `r0`
- How to preserve the running environment for the caller?
  - (to be covered)

References

- Lecture 29. Calling a subroutine
  - https://www.youtube.com/watch?v=xt2Q9n1Udb4&list=PLRJhV4hUhIymmp5CCeIFPyxbknsdcXCc8&index=29
- Lecture 30. Passing Arguments to a Subroutine
  - https://www.youtube.com/watch?v=DGKjFKjxAYs&list=PLRJhV4hUhIymmp5CCeIFPyxbknsdcXCc8&index=31