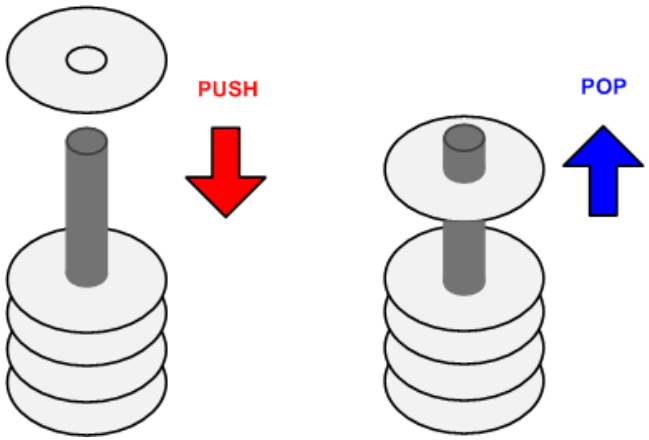
Chapter 8
Preserve Environment via Stack

Dr. Yifeng Zhu
Electrical and Computer Engineering
University of Maine

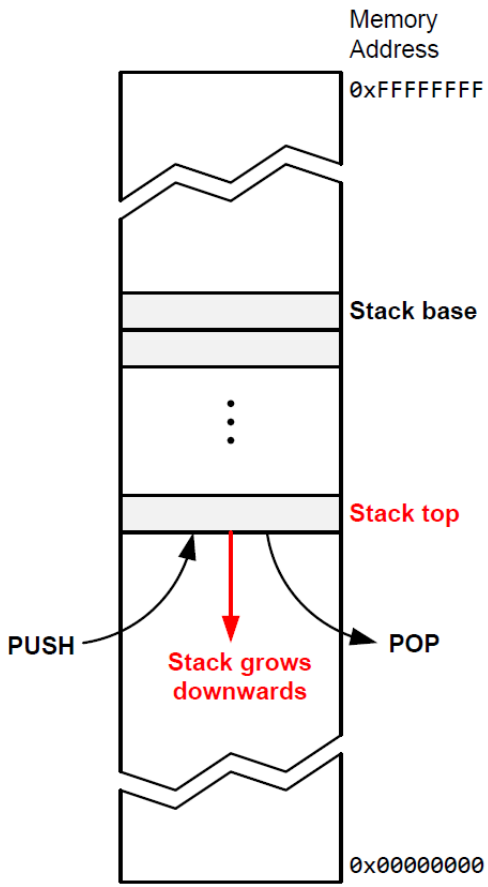
Spring 2015

Stack

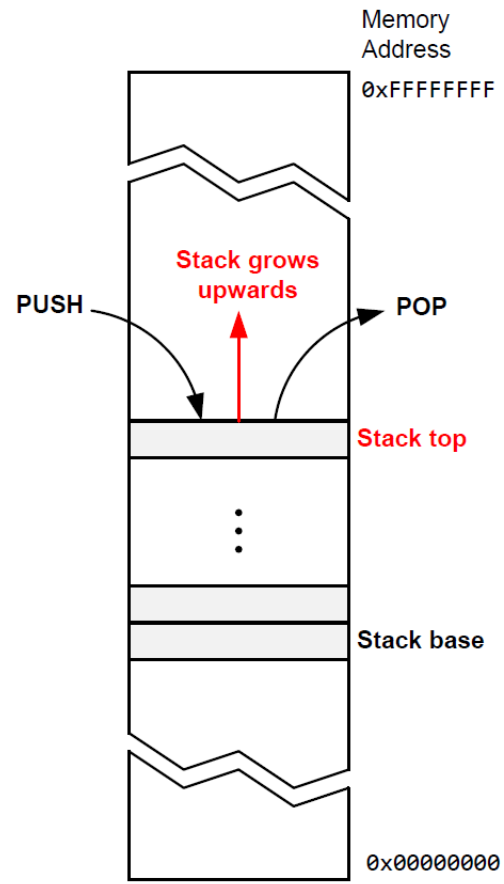
- A **Last-In-First-Out** data structure
- Only allow to access the most recently added item
  - Also called the top of the stack
- Key operations:
  - push (add item to stack)
  - pop (remove top item from stack)



Stack Growth Convention: Ascending *vs* Descending

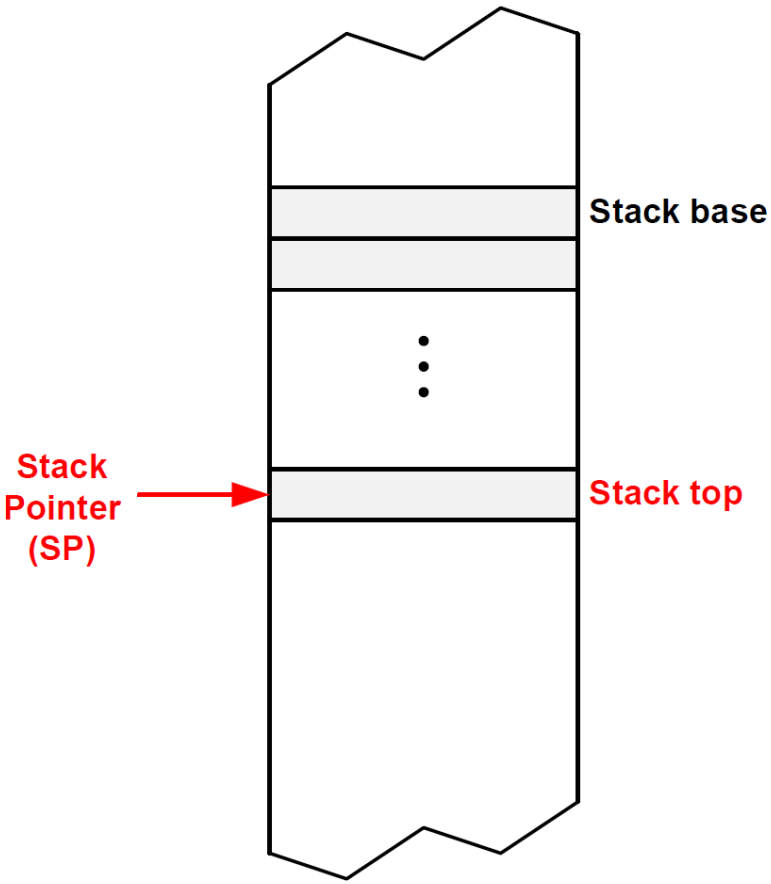


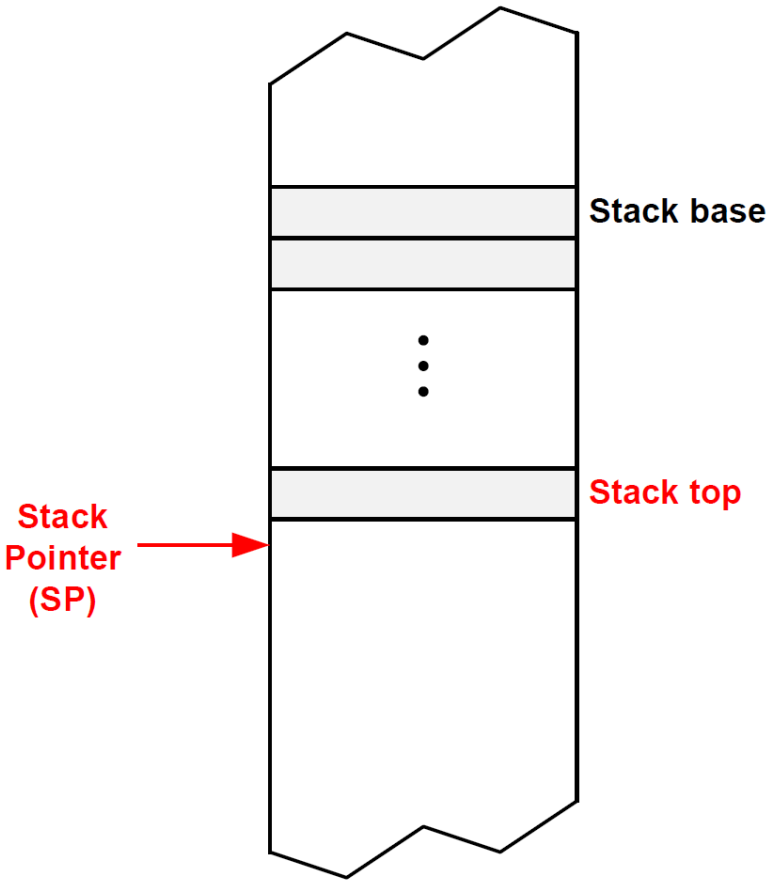
Descending stack: Stack grows towards low memory address



Ascending stack: Stack grows towards high memory address

Stack Growth Convention: Full *vs* Empty



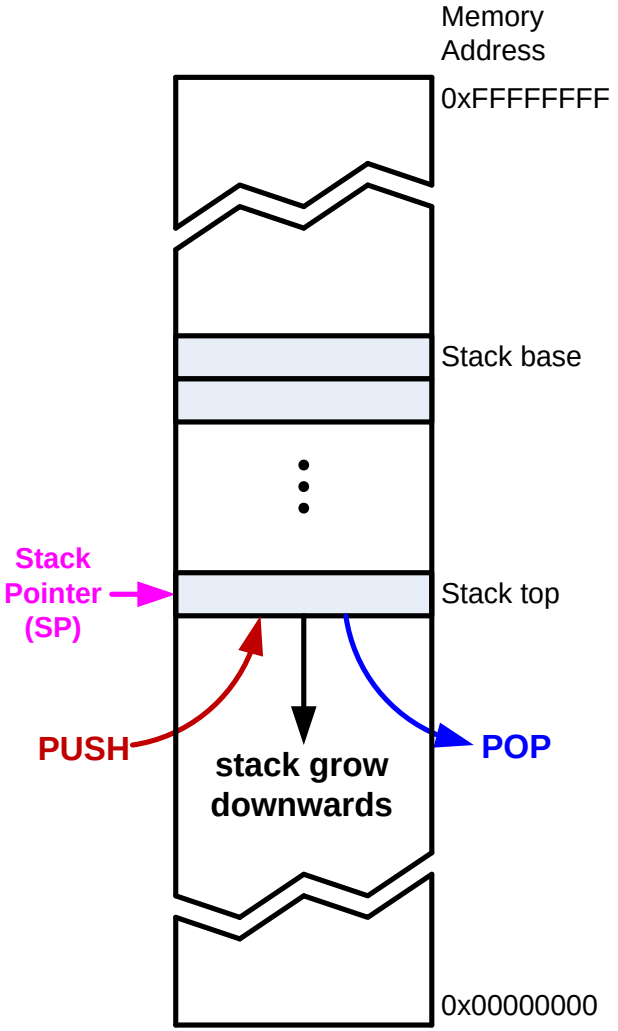


Full stack: SP points to the last item pushed onto the stack

Empty stack: SP points to the next free space on the stack

Cortex-M3 Stack

- stack pointer (SP) = R13
- Cortex-M3 uses full descending stack!
- stack pointer
  - decremented on **PUSH**
  - incremented on **POP**
  - SP starts at **`0x20000200`** for STM32-Discovery



Stack

PUSH {*Rd*}

- SP = SP-4 ⟶ descending stack
- (*SP) = *Rd* ⟶ full stack

Push multiple registers

They are equivalent.

```
PUSH {r6, r7, r8}  ⟺  PUSH {r8, r7, r6}  ⟺  PUSH {r8}
                                             PUSH {r7}
                                             PUSH {r6}
```

- The order in which registers listed in the register list does not matter.
- When pushing multiple registers, these registers are automatically sorted by name and the lowest-numbered register is stored to the lowest memory address.

Stack

POP {*Rd*}

- *Rd* = (*SP) ⟶ full stack
- SP = SP + 4 ⟶ Stack shrinks

Pop multiple registers

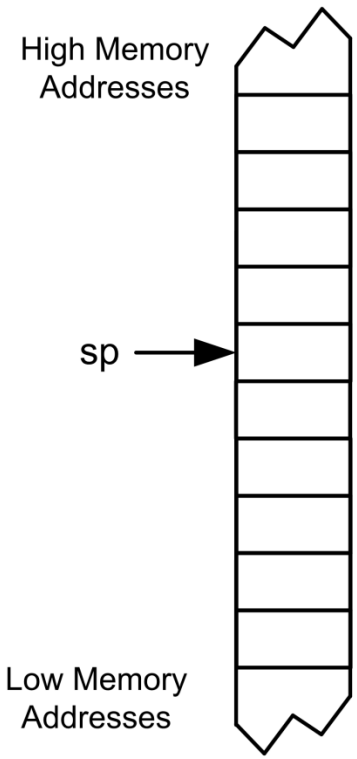
They are equivalent.

```
POP {r6, r7, r8}  ⟺  POP {r8, r7, r6}  ⟺  POP {r6}
                                           POP {r7}
                                           POP {r8}
```

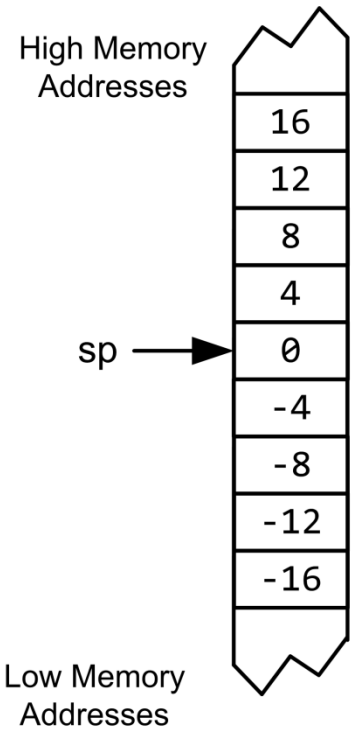
- The order in which registers listed in the register list does not matter.
- When popping multiple registers, these registers are automatically sorted by name and the lowest-numbered register is loaded from the lowest memory address.

Stack

PUSH {r3, r1, r7, r2}

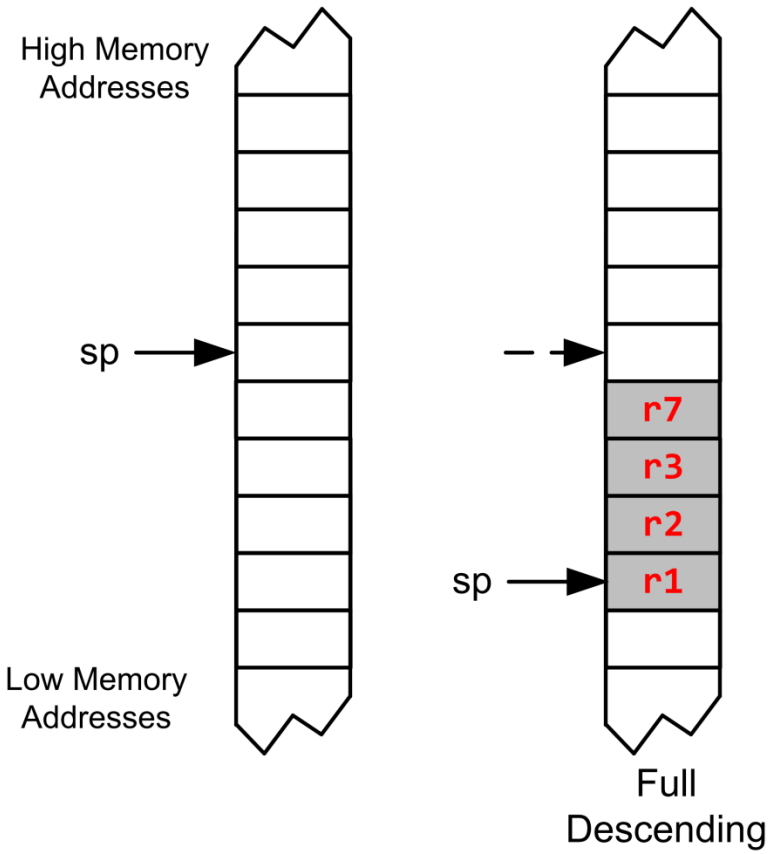


POP {r3, r1, r7, r2}

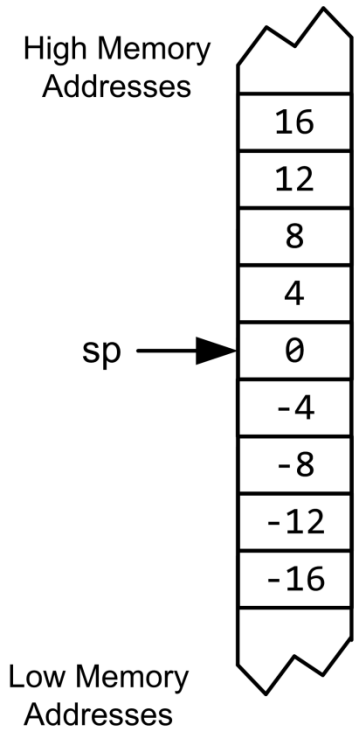


Stack

PUSH {r3, r1, r7, r2}

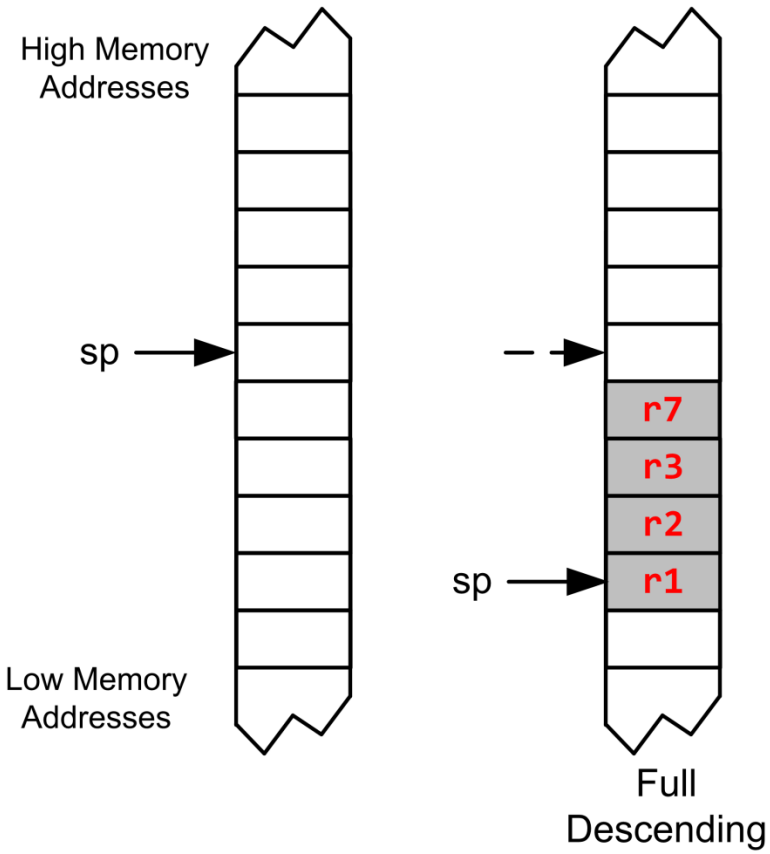


POP {r3, r1, r7, r2}

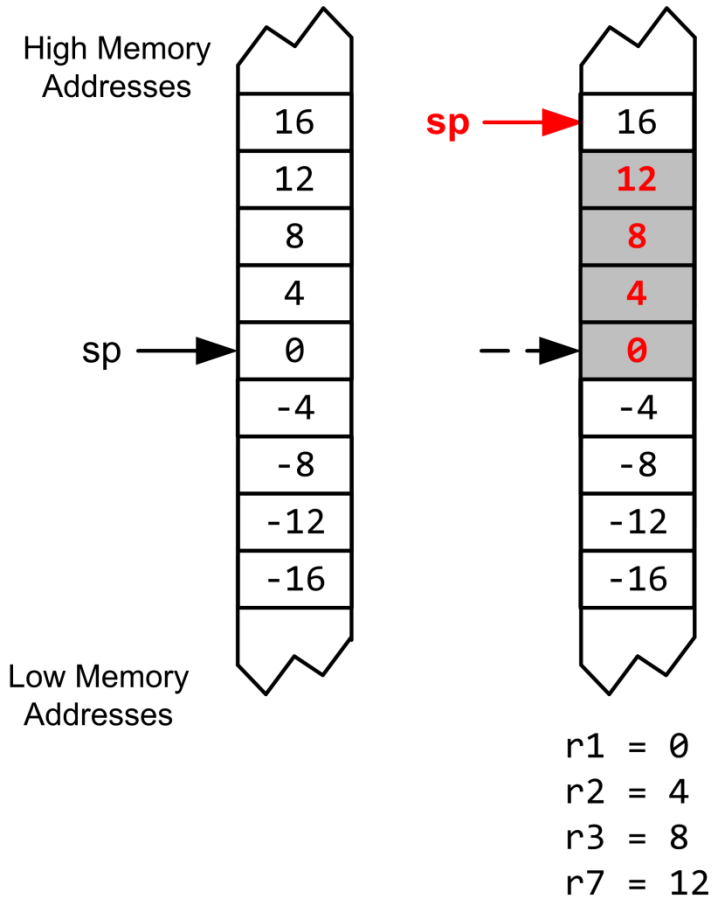


Stack

PUSH {r3, r1, r7, r2}

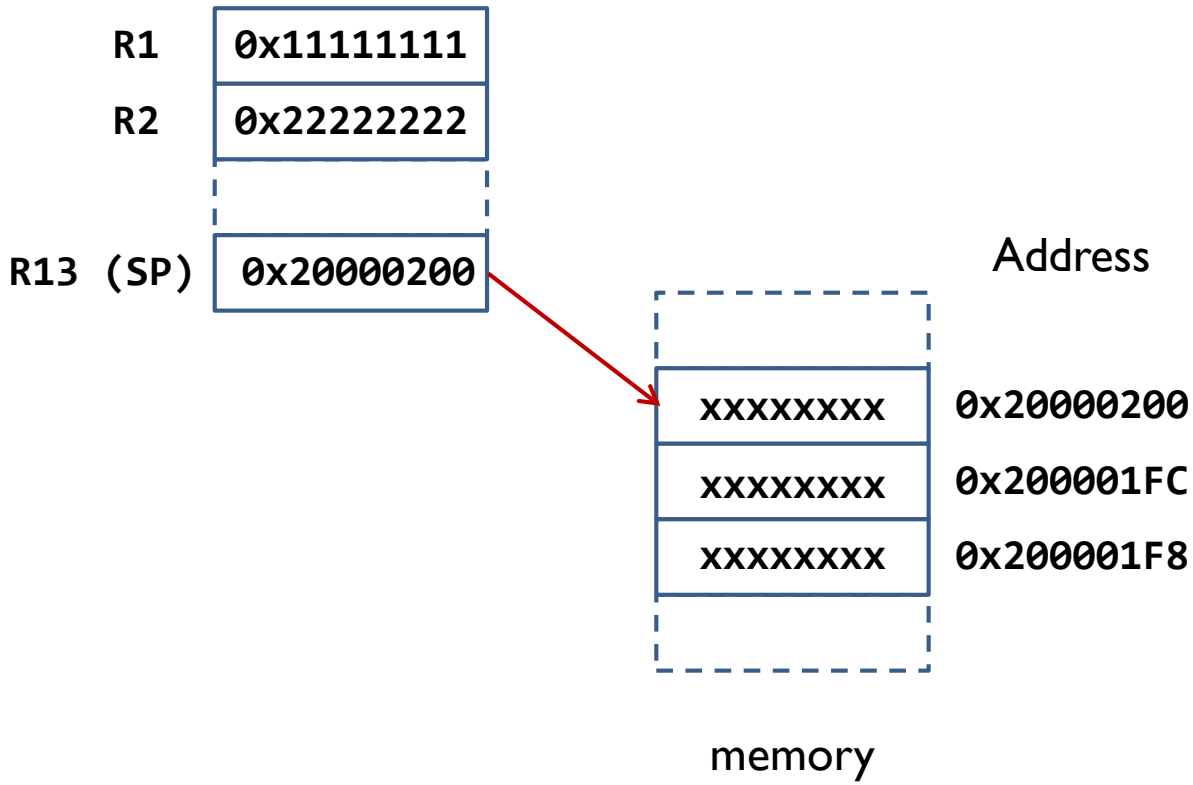


POP {r3, r1, r7, r2}

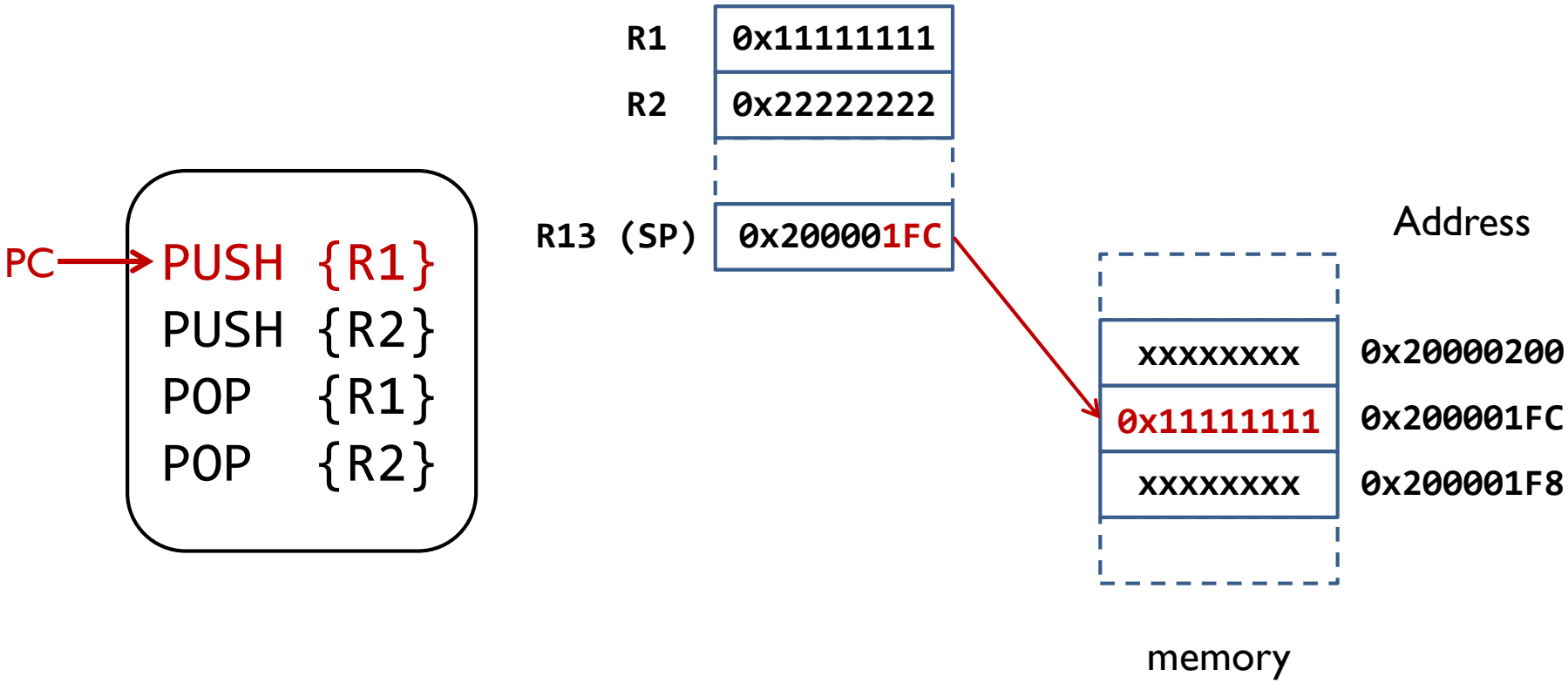


Example: swap R1 & R2

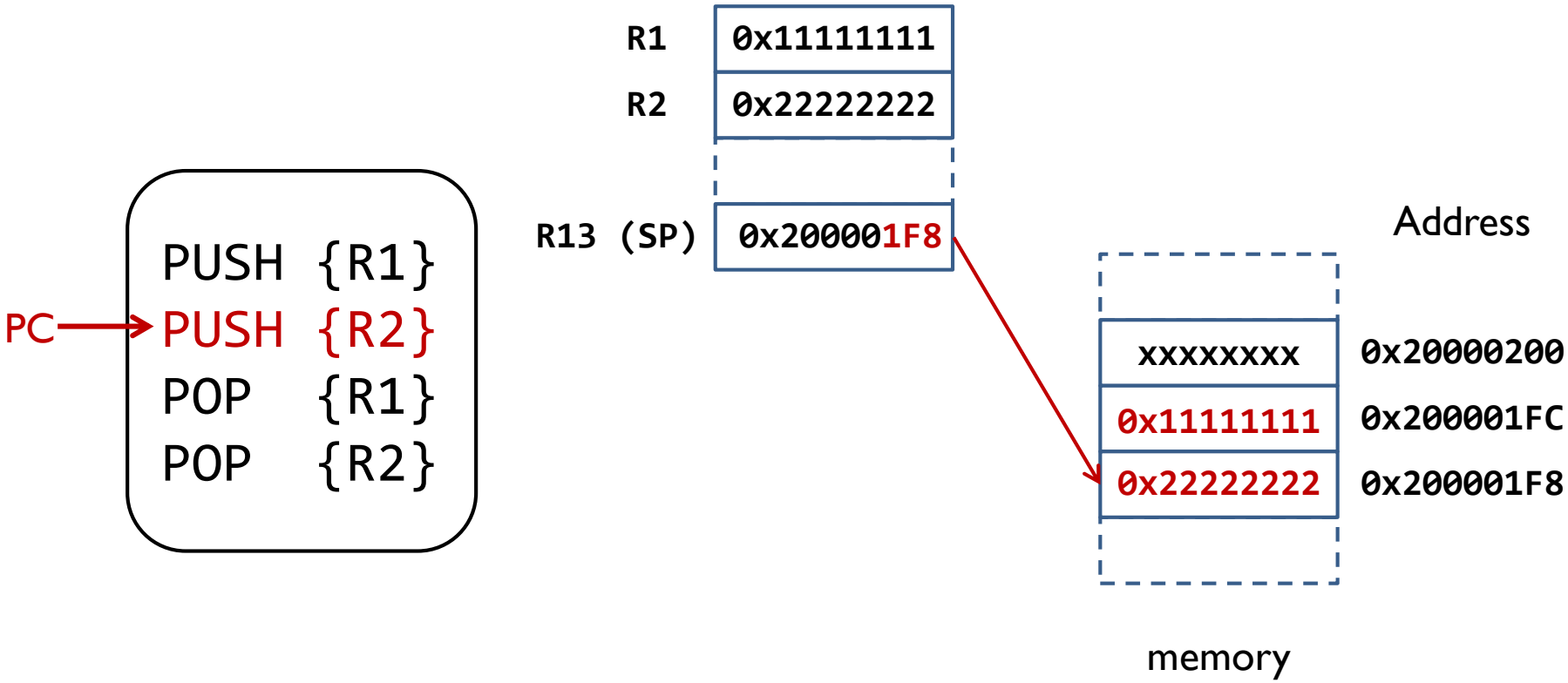
```
PUSH {R1}
PUSH {R2}
POP  {R1}
POP  {R2}
```



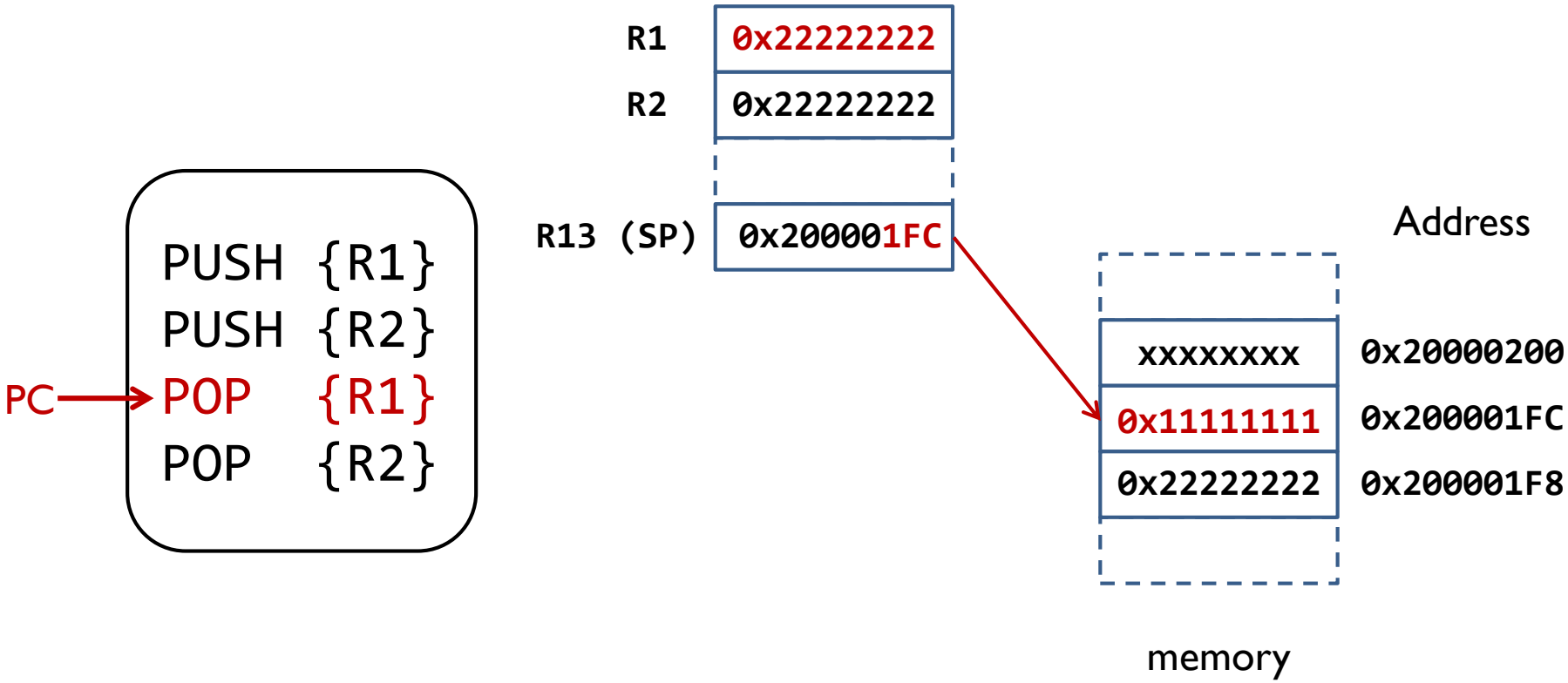
Example: swap R1 & R2



Example: swap R1 & R2



Example: swap R1 & R2



Example: swap R1 & R2

```
PUSH {R1}
PUSH {R2}
POP  {R1}
POP  {R2}
```

PC → POP {R2}

R1	0x22222222
R2	0x11111111
R13 (SP)	0x20000200

	Address
xxxxxxxx	0x20000200
0x11111111	0x200001FC
0x22222222	0x200001F8

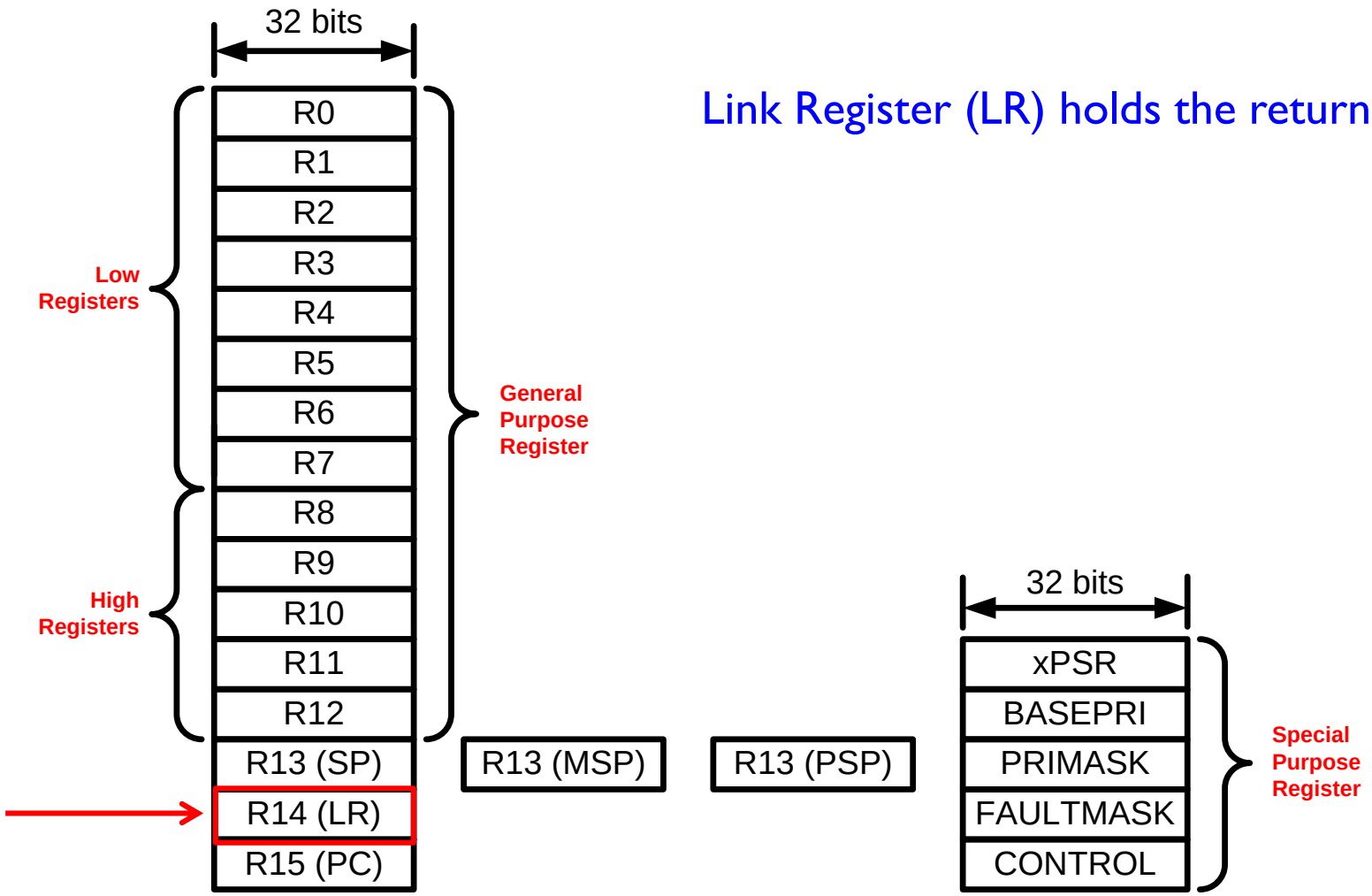
memory

Subroutine

- A subroutines, also called a function or a procedure,
  - single-entry, single-exit
  - Return to caller after it exits
- When a subroutine is called, the Link Register (LR) holds the memory address of the next instruction to be executed when the subroutine exits.

Link Register

Link Register (LR) holds the return address



Call a Subroutine

Caller Program	Subroutine/Callee

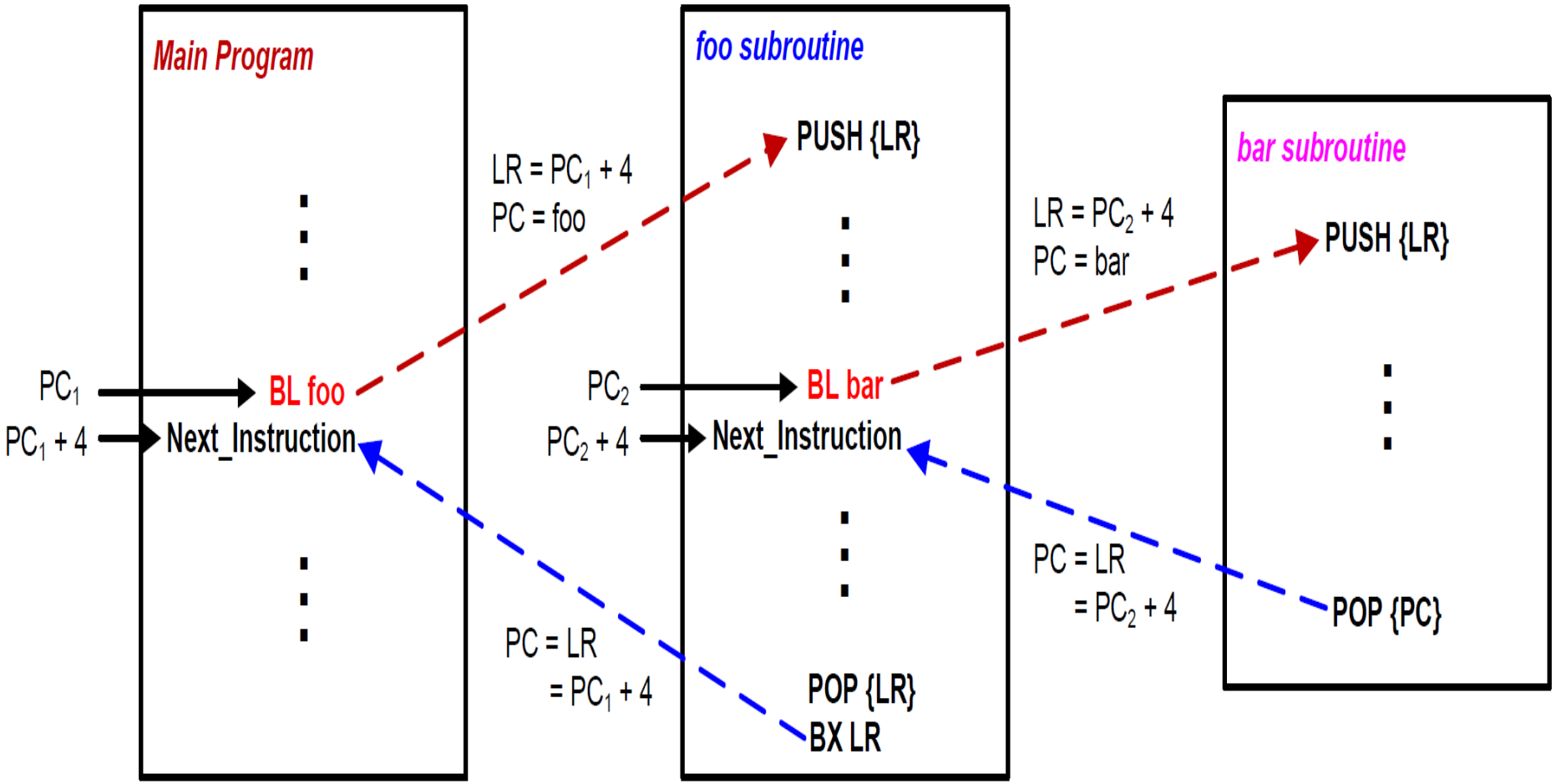
`MOV r4, #100`
`...`
`BL foo`
`...`
`ADD r4, r4, #1 ; r4 = 11, not 101`
	`foo PROC`
	`...`
	`MOV r4, #10 ; foo changes r4`
	`...`
	`BX LR`
	`ENDP`

Preserve Runtime Environment via Stack

Caller Program	Subroutine/Callee

`MOV r4, #100`
`...`
`BL foo`
`...`
`ADD r4, r4, #1 ; r4 = 101, not 11`
	`foo PROC`
	`PUSH {r4} ; preserve r4`
	`...`
	`MOV r4, #10 ; foo changes r4`
	`...`
	`POP {r4} ; Recover r4`
	`BX LR`
	`ENDP`

Stacks and Subroutines



Subroutine calling Another Subroutine

```
MAIN
   MOV R0,#2
   BL QUAD
ENDL    ...
```

Function MAIN

```
QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
```

Function QUAD

```
SQ      MUL R0,R0
        BX LR
```

Function SQ

Subroutine Calling Another Subroutine

```
MAIN PROC
   MOV R0,#2
   BL QUAD
ENDL    ...
   ENDP
```

Function MAIN

```
QUAD    PROC
        PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
        ENDP
```

Function QUAD

```
SQ      PROC
        MUL R0,R0
        BX LR
        ENDP
```

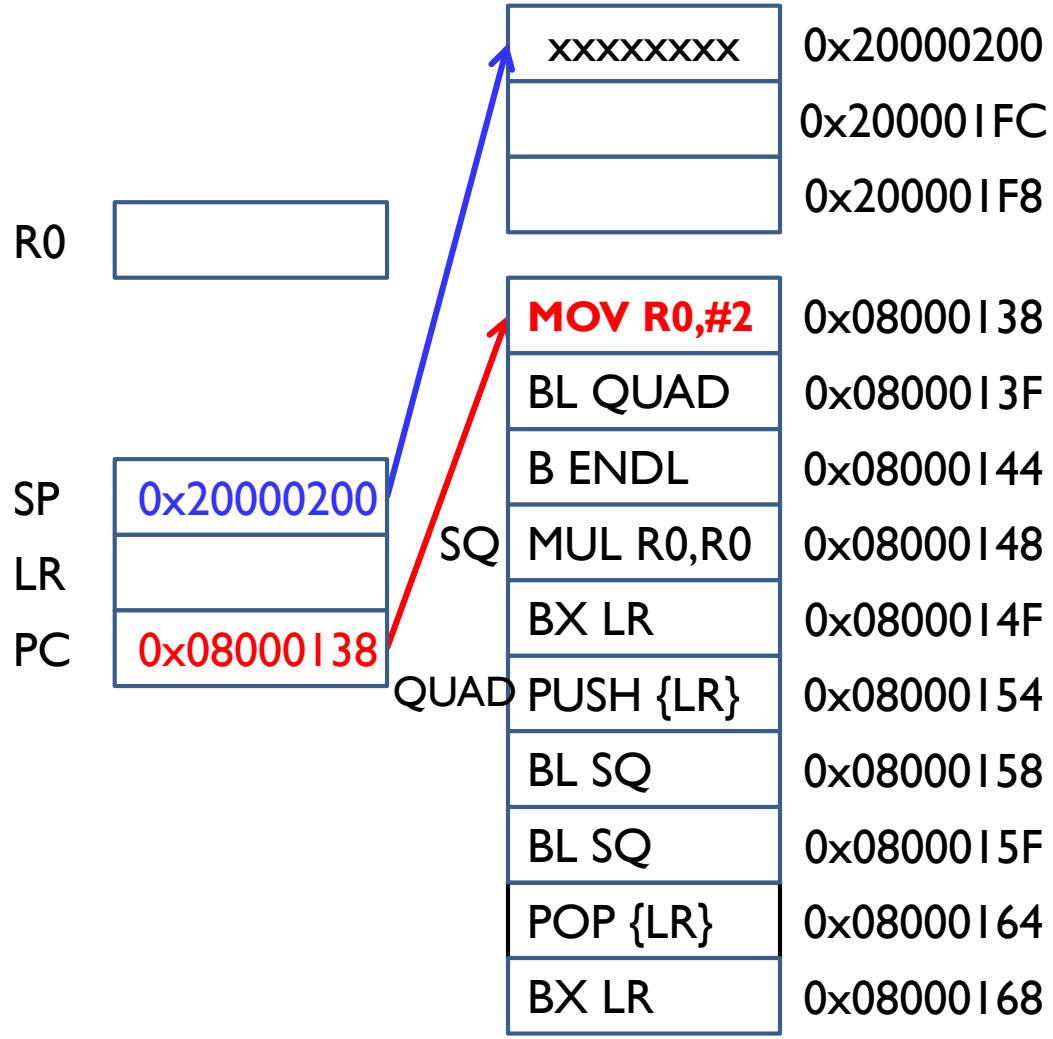
Function SQ

Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

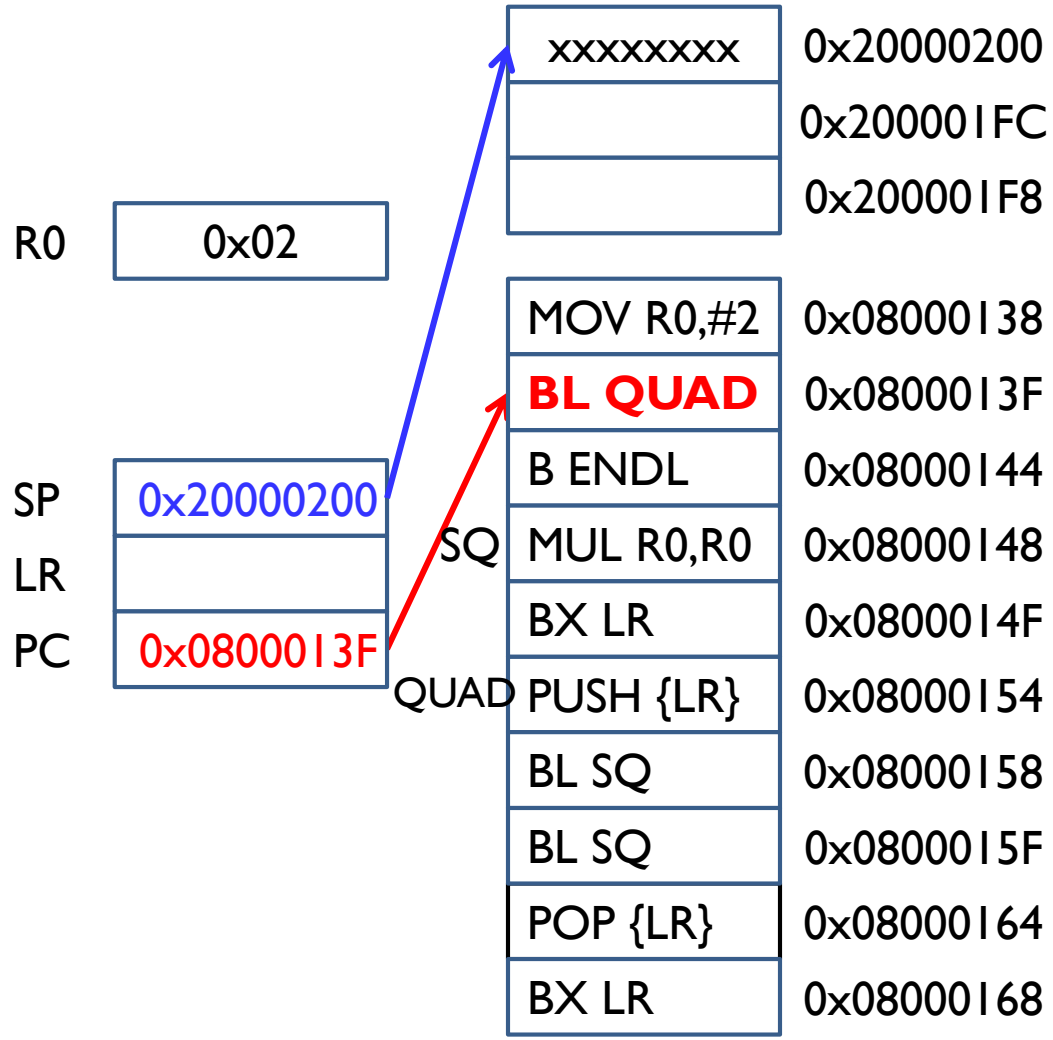


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

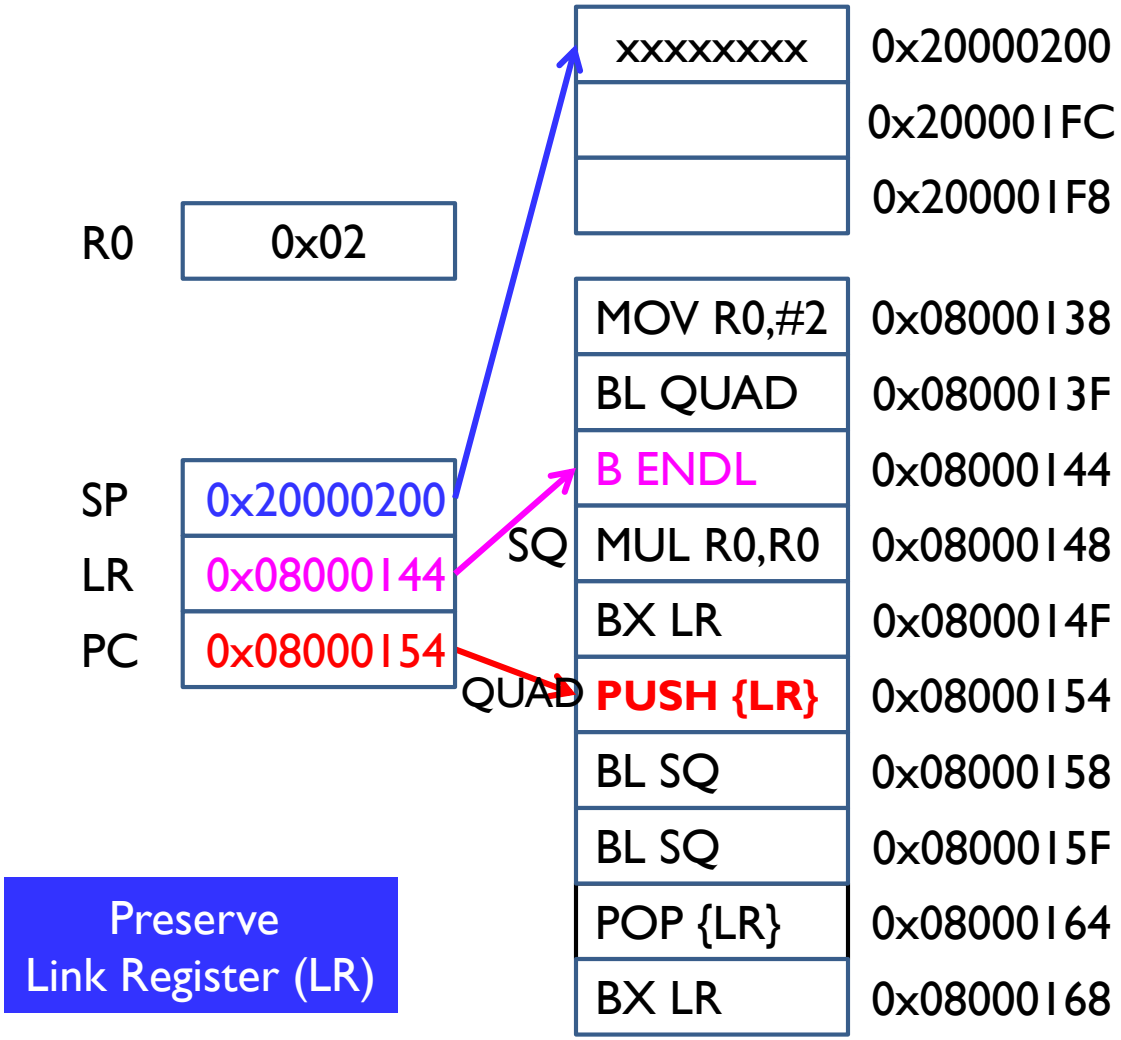


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```



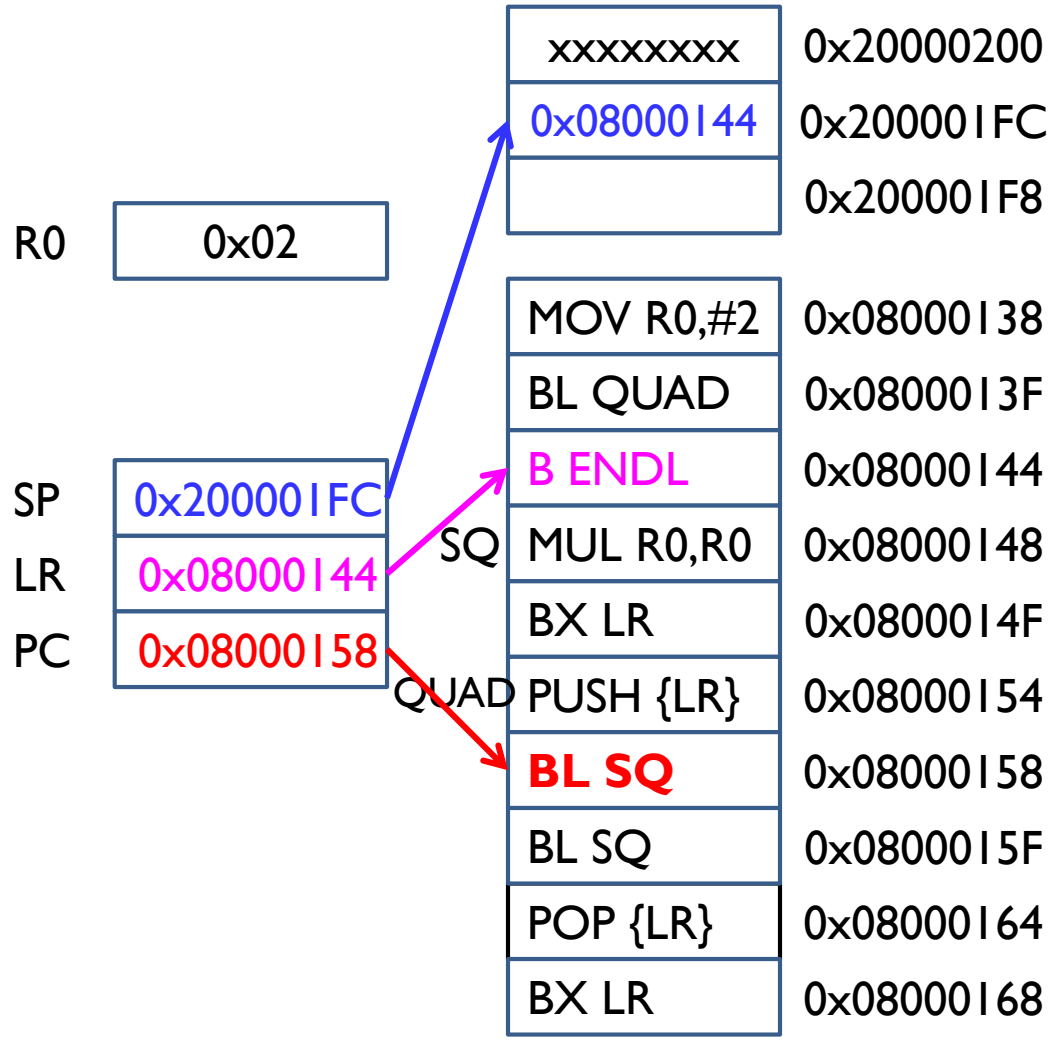
Preserve Link Register (LR)

Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

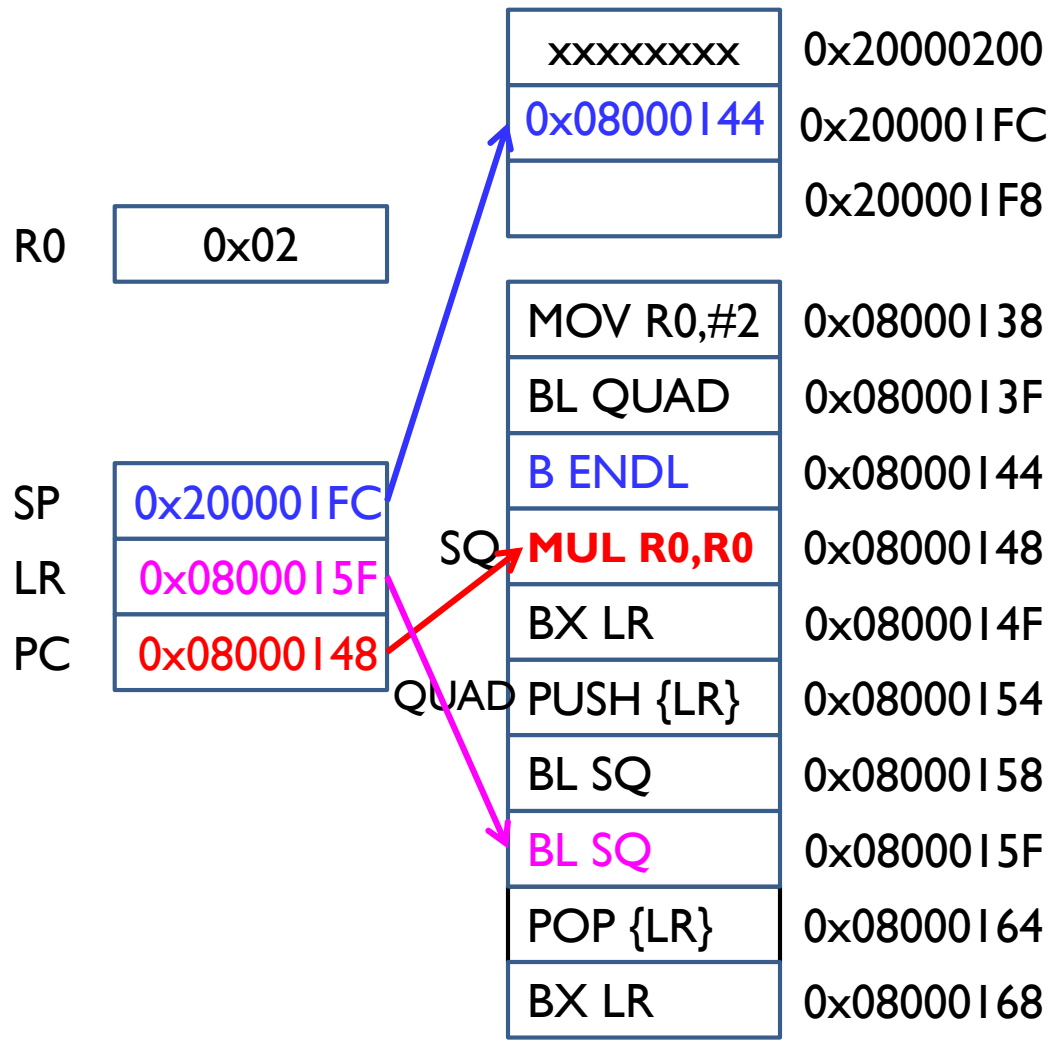


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

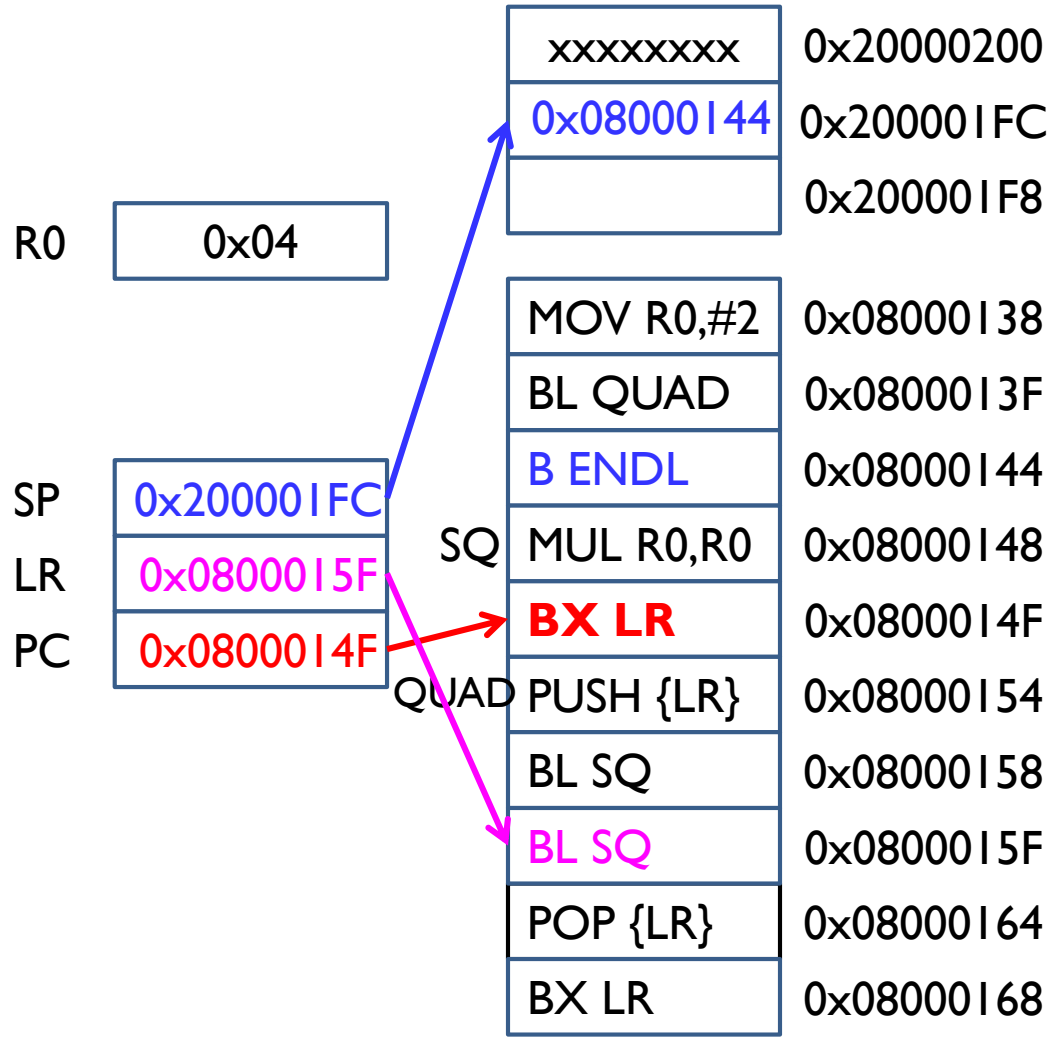


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

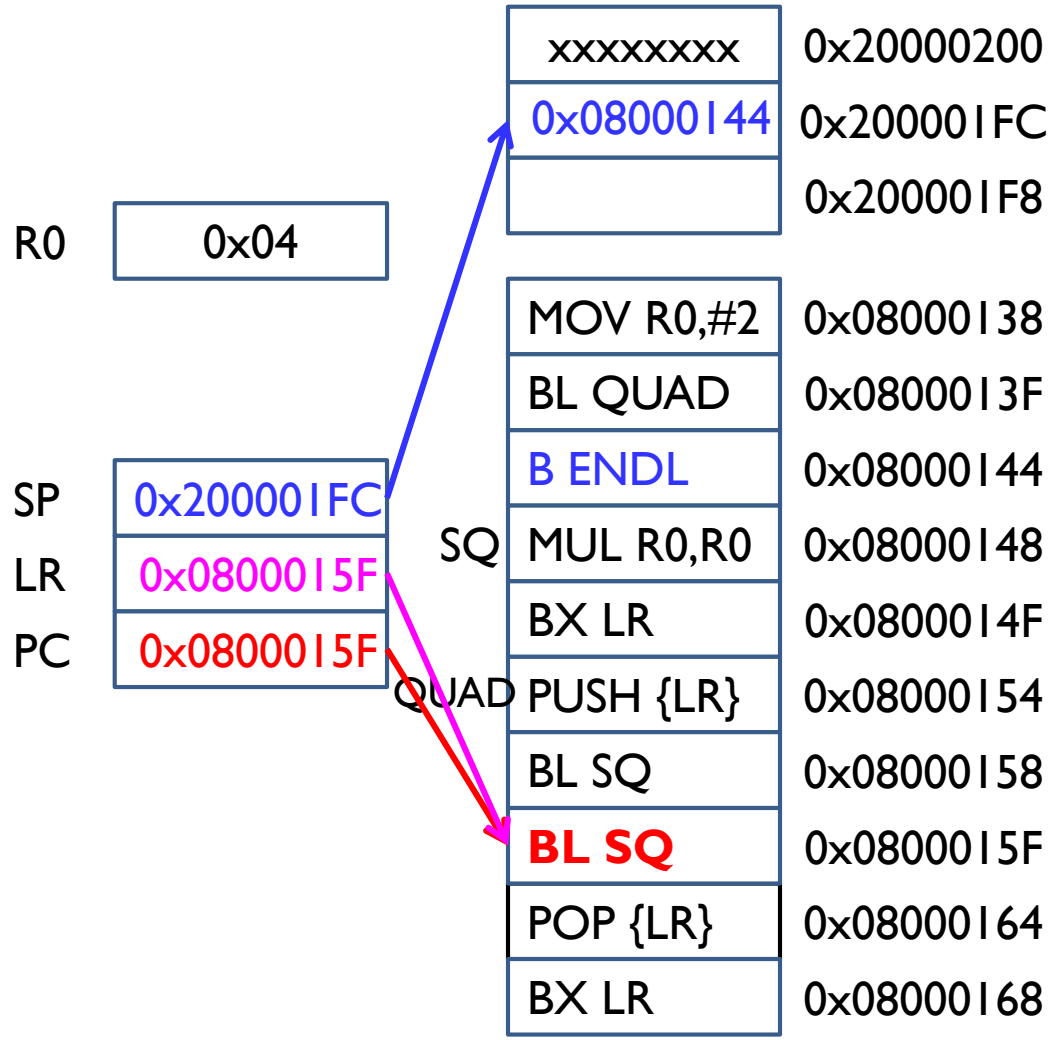


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

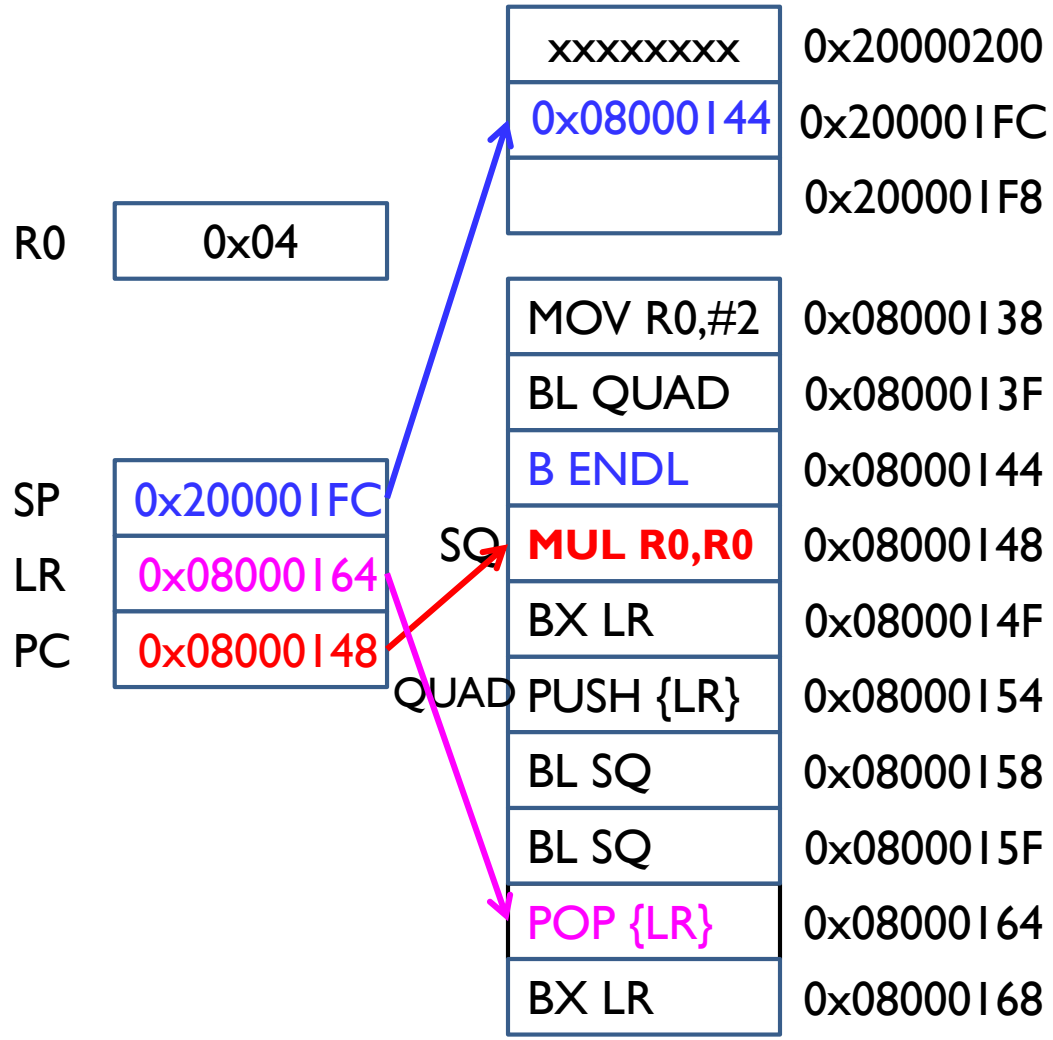


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

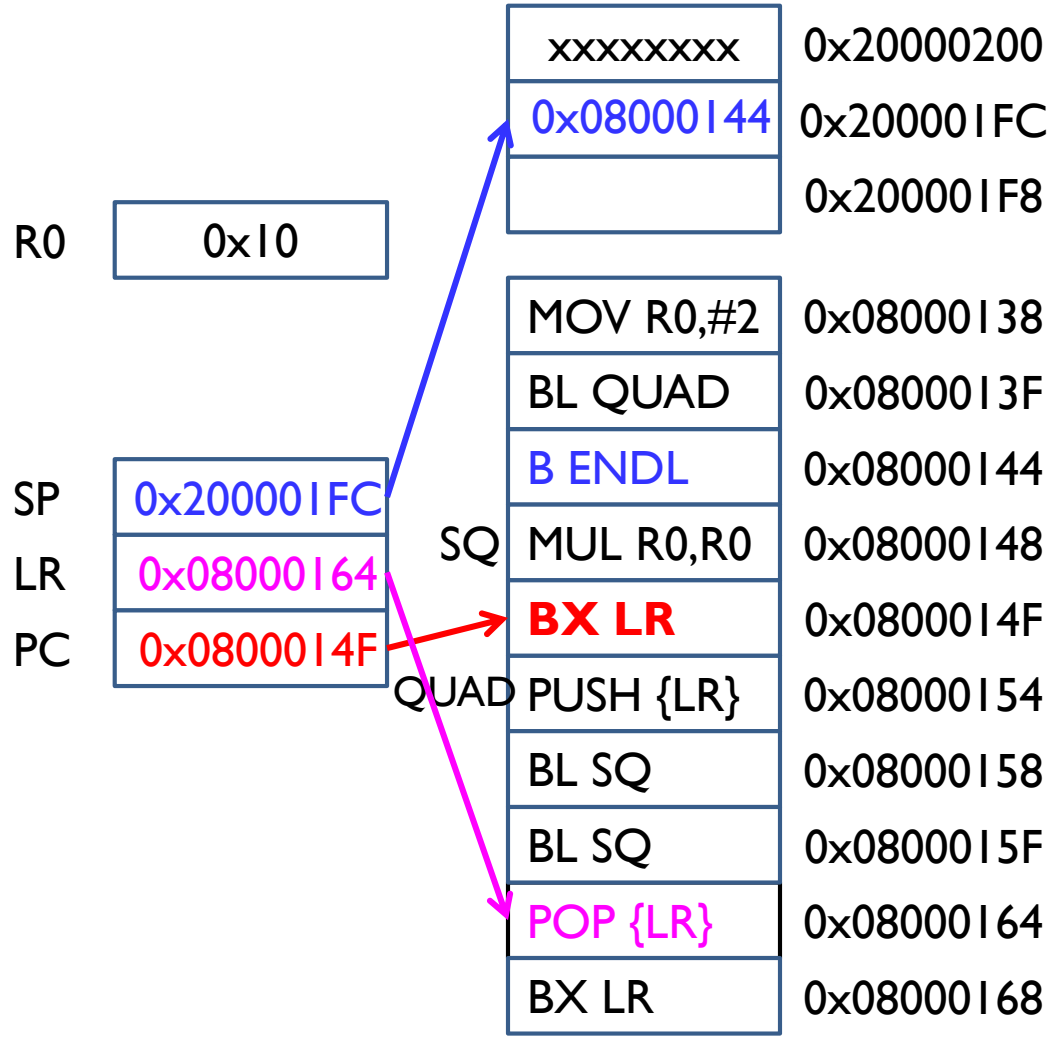


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

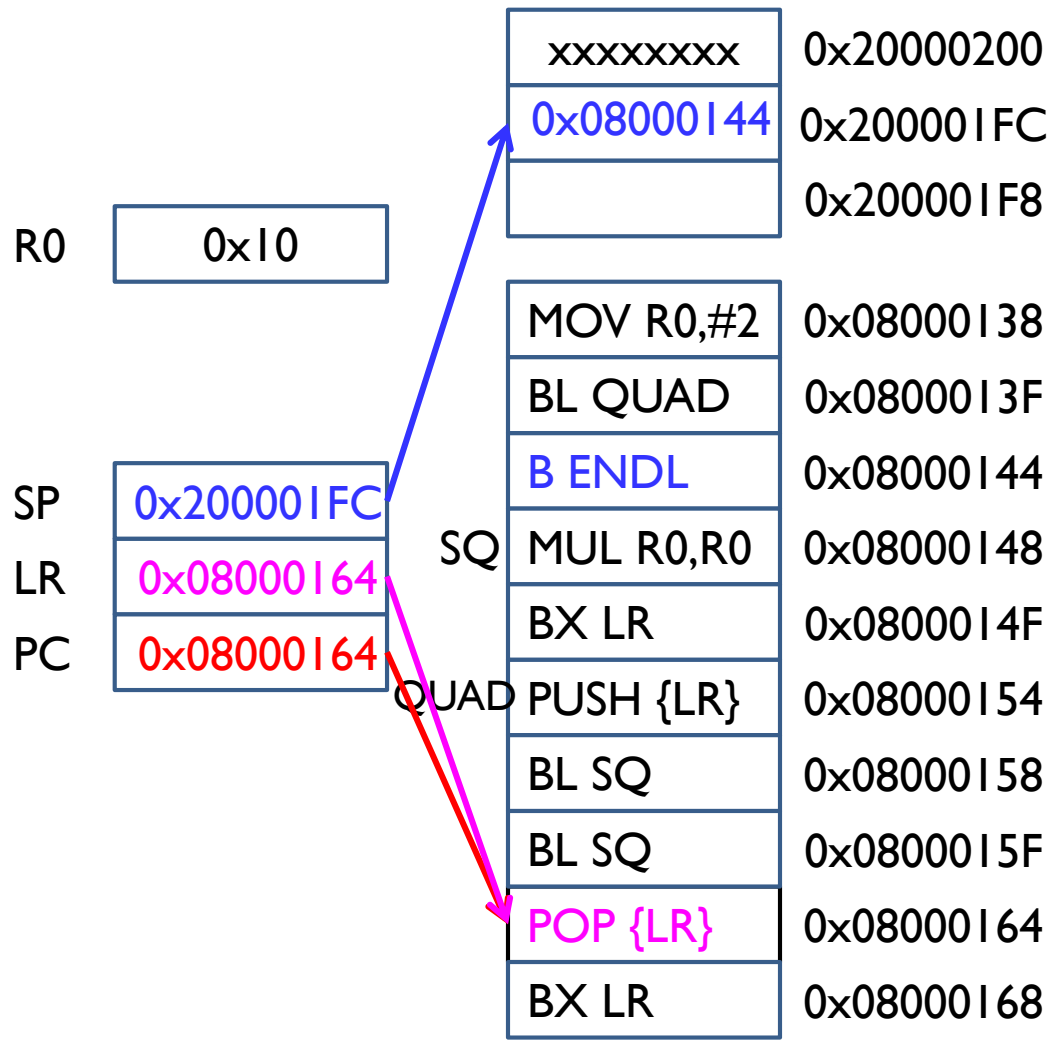


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```

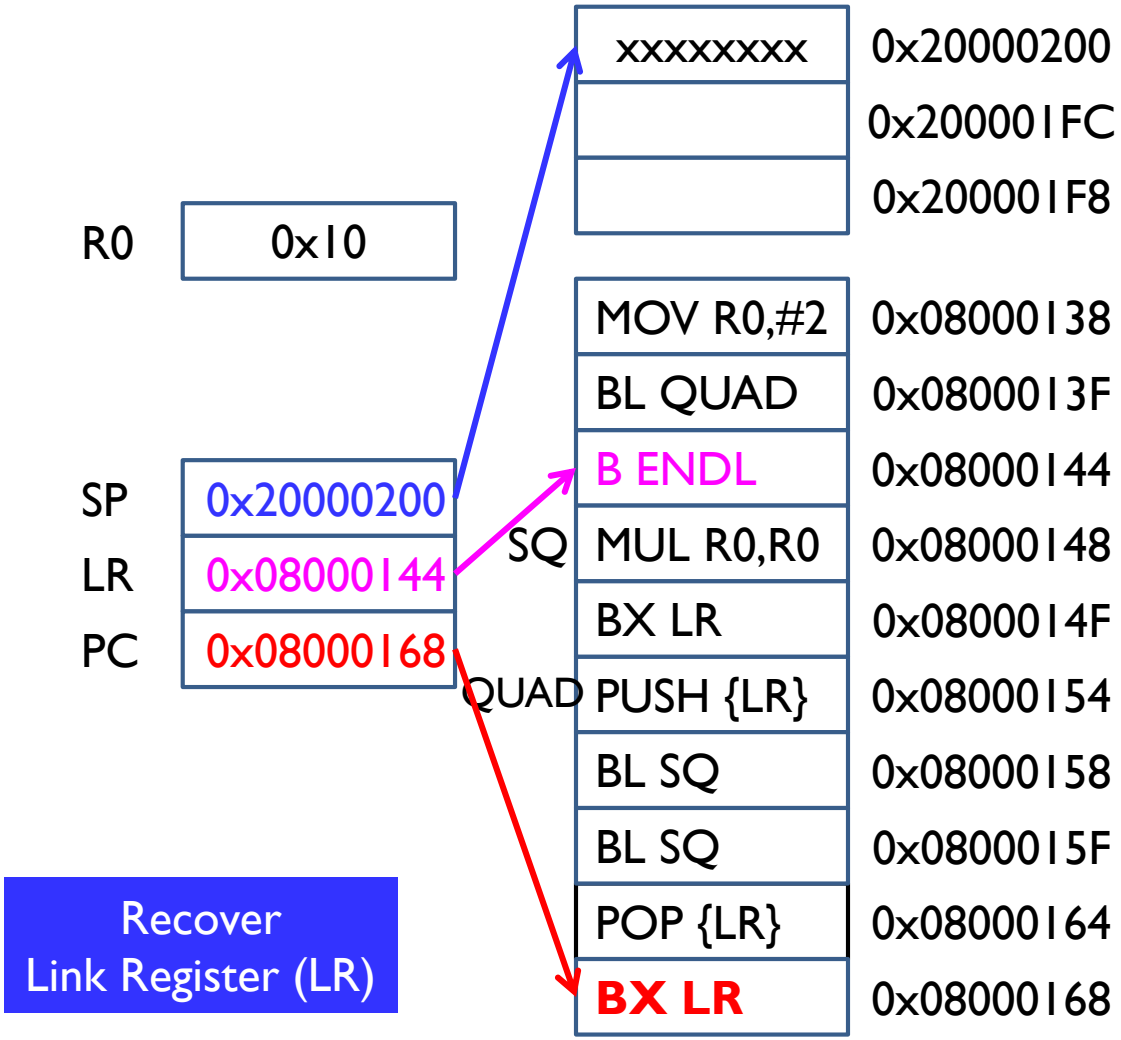


Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```



Recover Link Register (LR)

Example: R0 = R0^4

```
        MOV R0,#2
        BL QUAD
        B ENDL

SQ      MUL R0,R0
        BX LR

QUAD    PUSH {LR}
        BL SQ
        BL SQ
        POP {LR}
        BX LR
ENDL    ...
```