MEF
UNIVERSITY
Your Freedom in Learning

Microprocessors

Tuba Ayhan

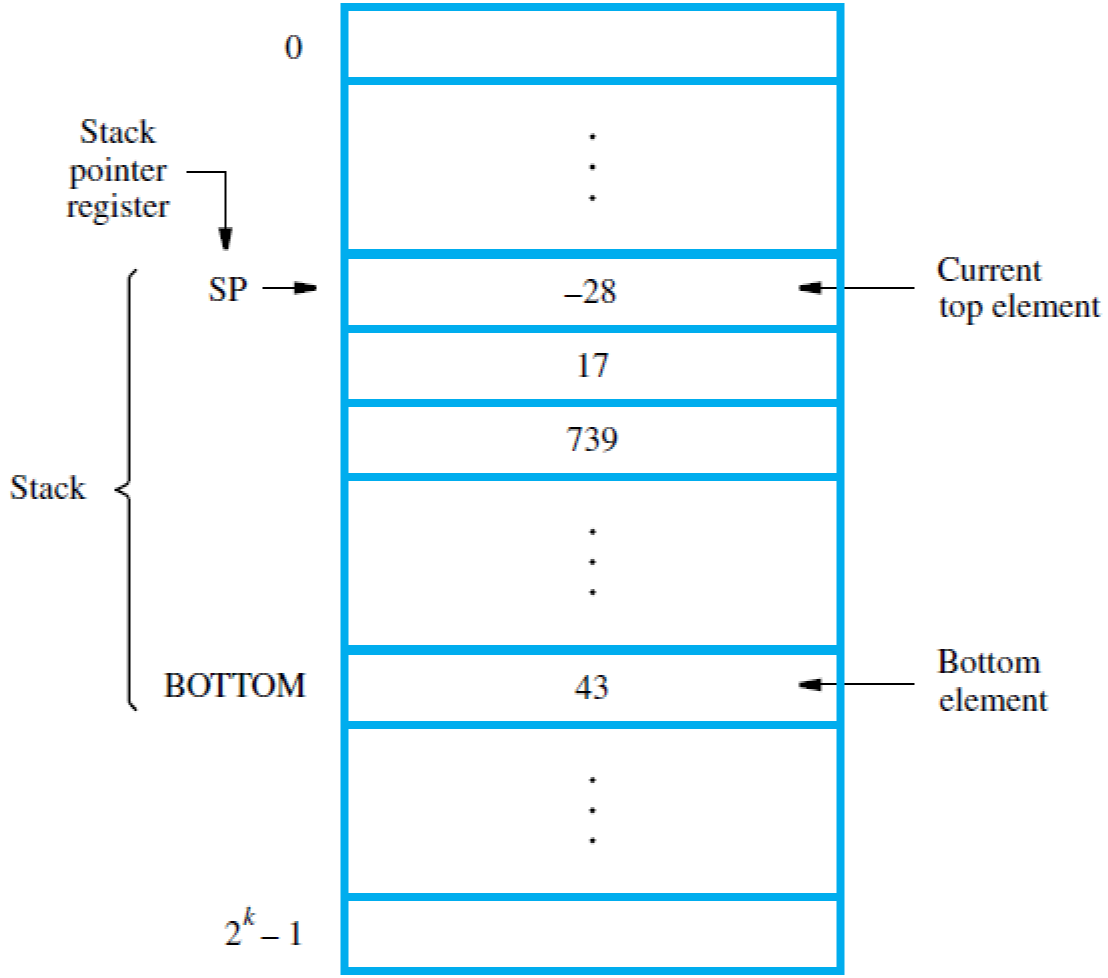
MEF University

Stacks and subroutines

Computer Organization and Embedded Systems, Hamacher et. al

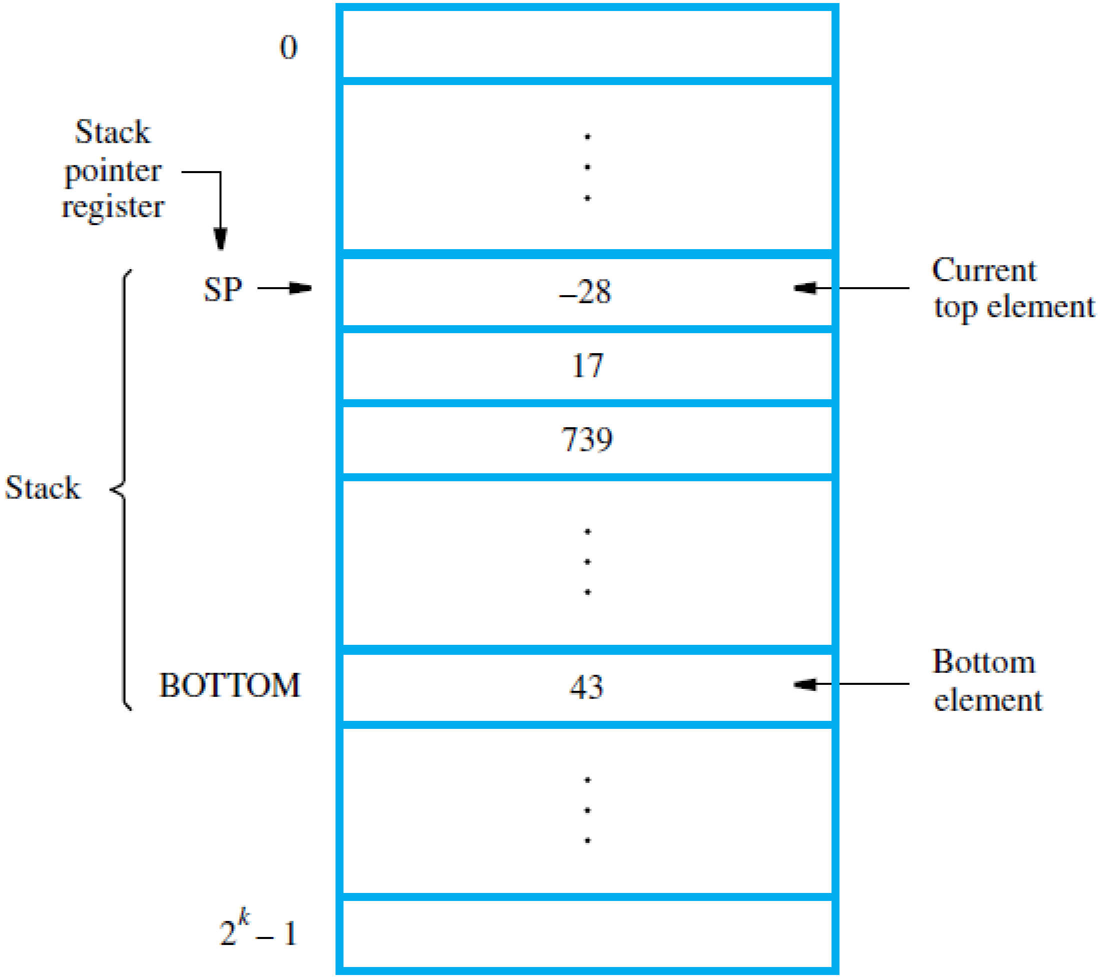
Stacks

- A (pushdown) or a last-in–first-out (LIFO) stack is
  - A list of data elements (usually words)
  - Access restriction: elements can be added or removed at one end of the list only.
- Remove/add data from TOP end, other end is called the BOTTOM.
- Push:placing a new item.
- Pop: removing the top item.
- Stack pointer (SP): points to the *processor stack*.
  - The first element is placed in location BOTTOM, and when new elements are pushed onto the stack, they are placed in successively lower address locations.

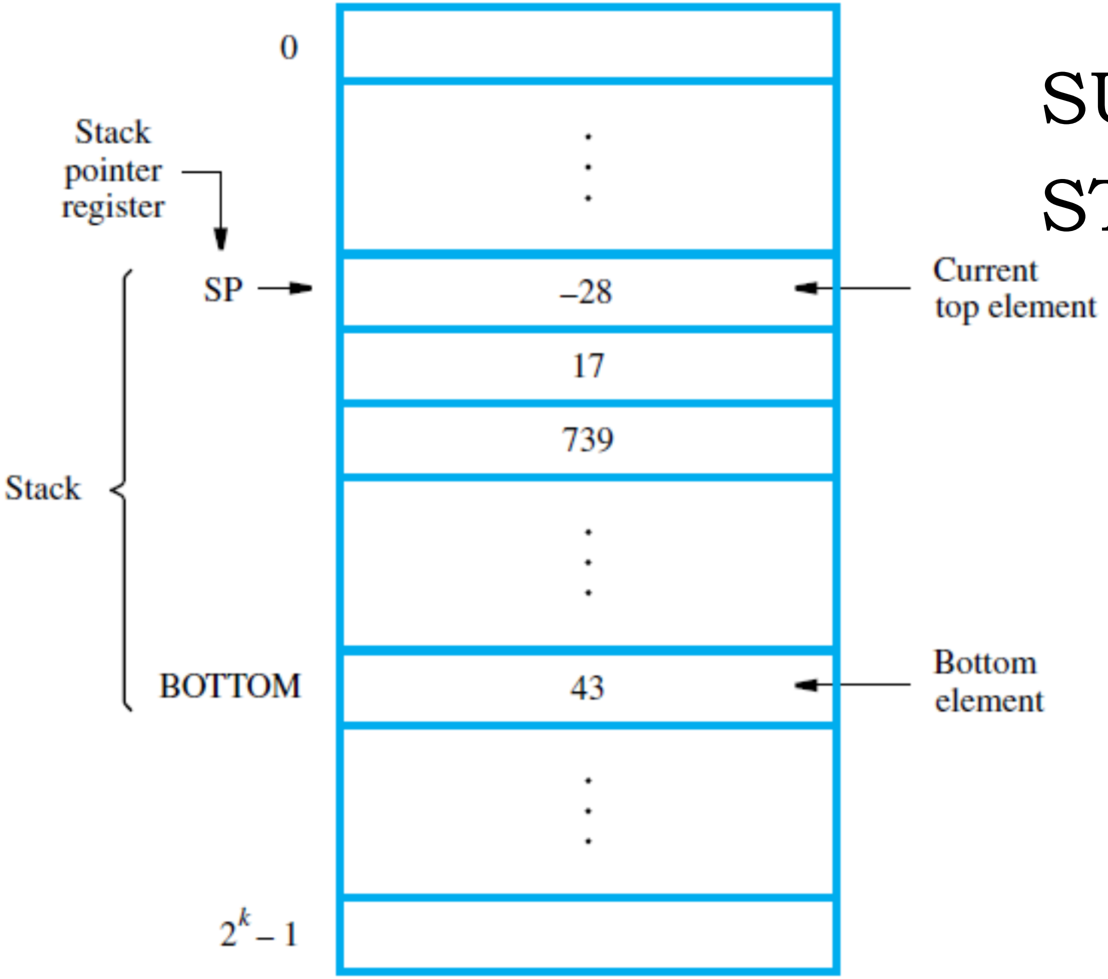


Stacks – Example

- The first element is placed in location BOTTOM, and when new elements are pushed onto the stack, they are placed in successively lower address locations.

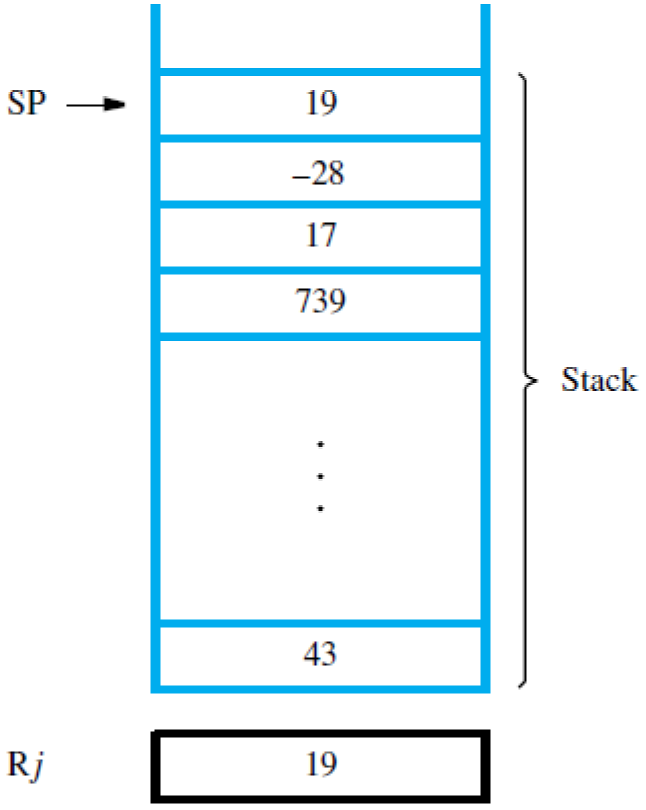


Stacks – Example, push



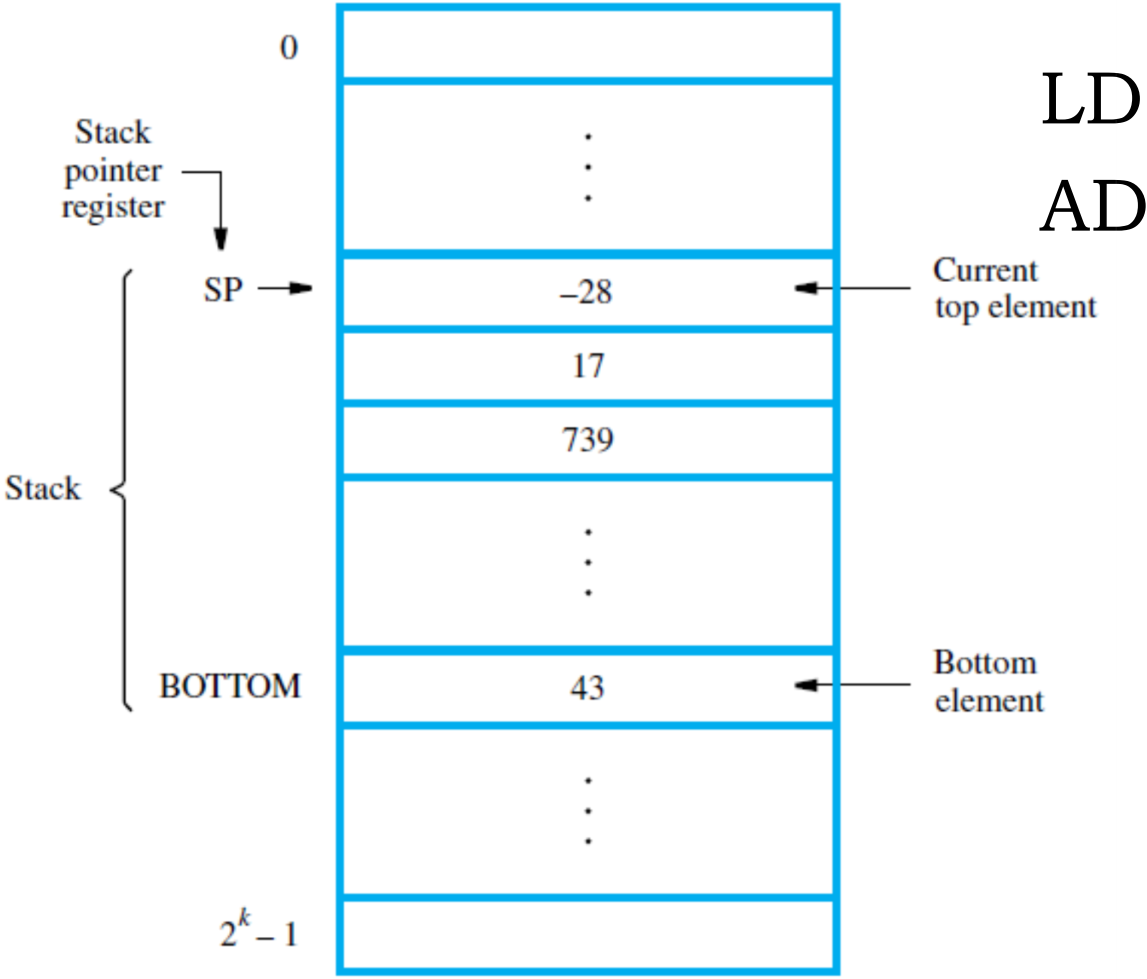
SUB SP, SP, #4

STR Rj, [SP]

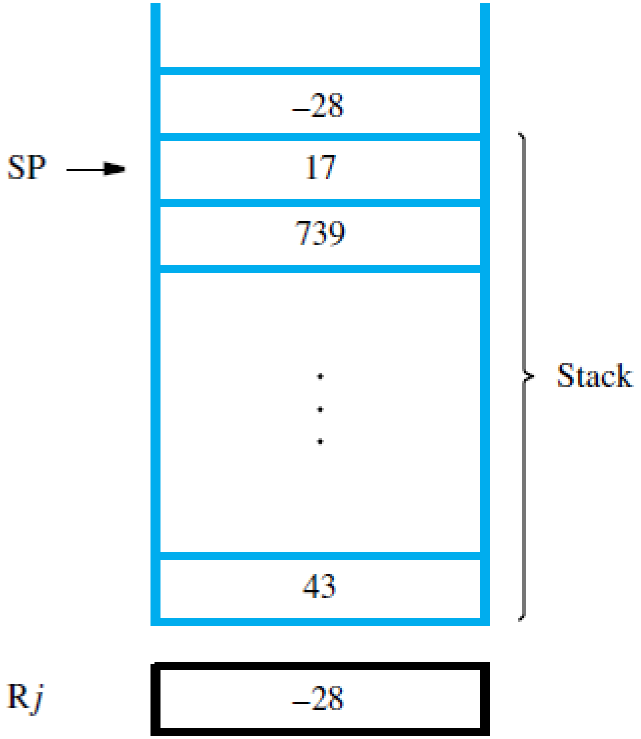


(a) After push from R*j*

Stacks – Example, pop



```
LDR Rj, [SP]
ADD SP, SP, #4
```



(b) After pop into Rj

Subroutine

- Perform a ***particular task*** *many times* on different data values
- This **task** is stored as a block of instructions → **Subroutine**
- When a program branches to a subroutine we say that it is *calling* the subroutine.

Call instruction

- The subroutine is said to *return* to the program, after the subroutine is executed.

Return instruction

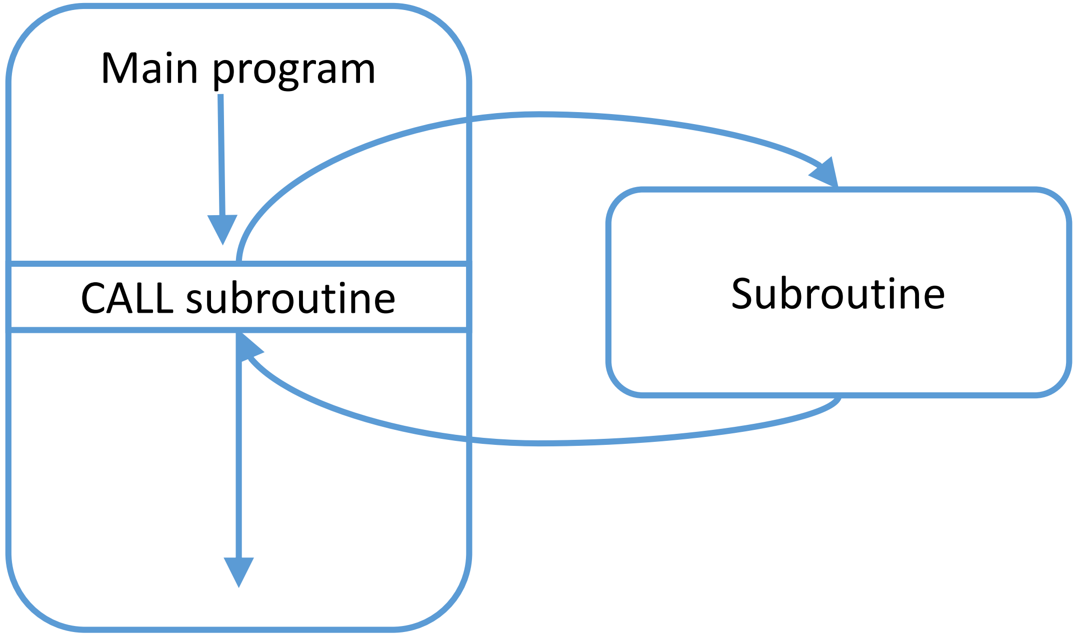
Subroutine linkage method

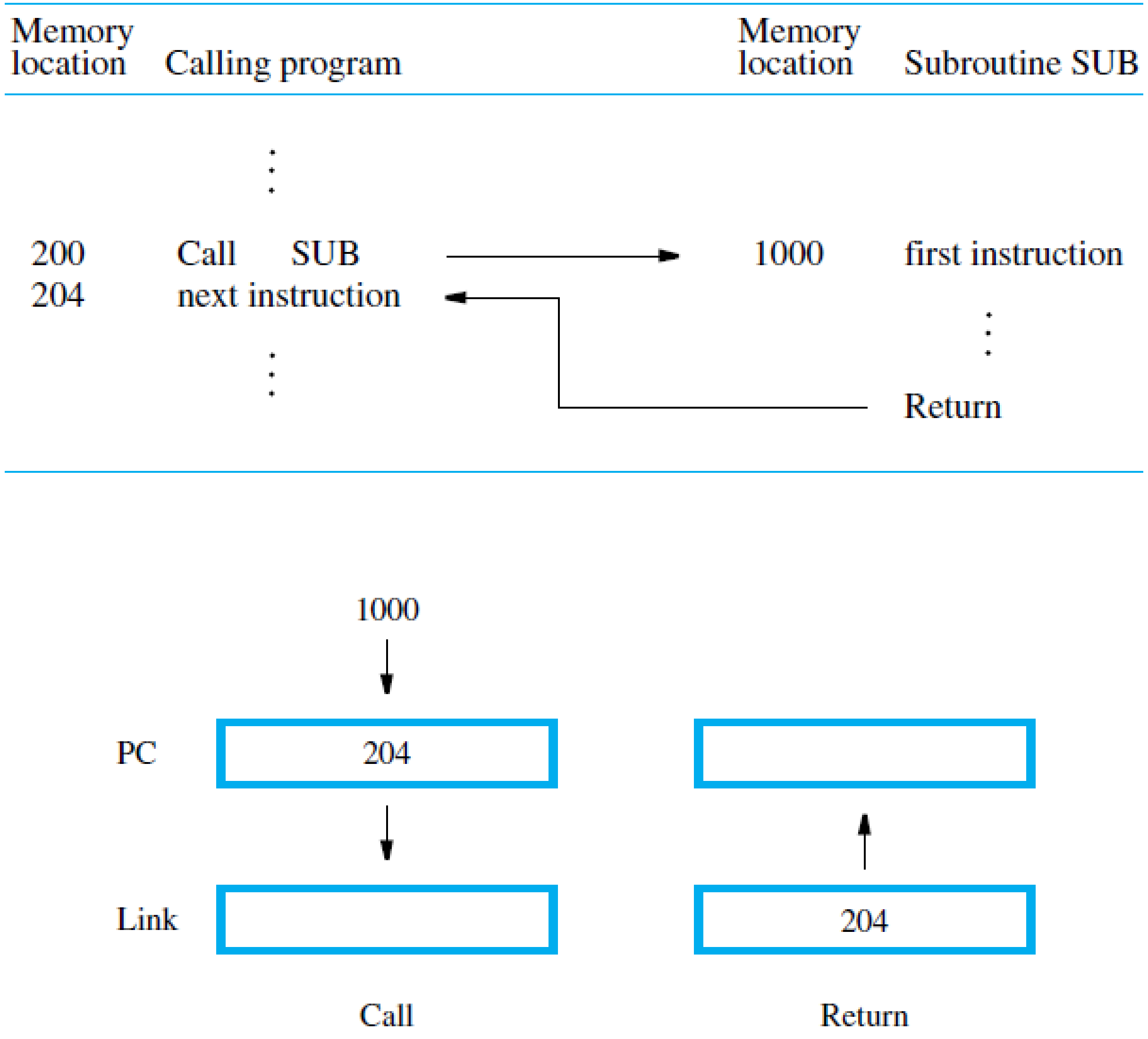
- The contents of the PC must be saved by the Call instruction to enable correct return to the calling program.
- Simplest way→

  Save the return address in a specific location: **link register**

- Call instruction
  - Store the contents of the PC in the link register.
  - Branch to the target address specified by the Call instruction.
- Return instruction
  - Branch to the address contained in the link register.

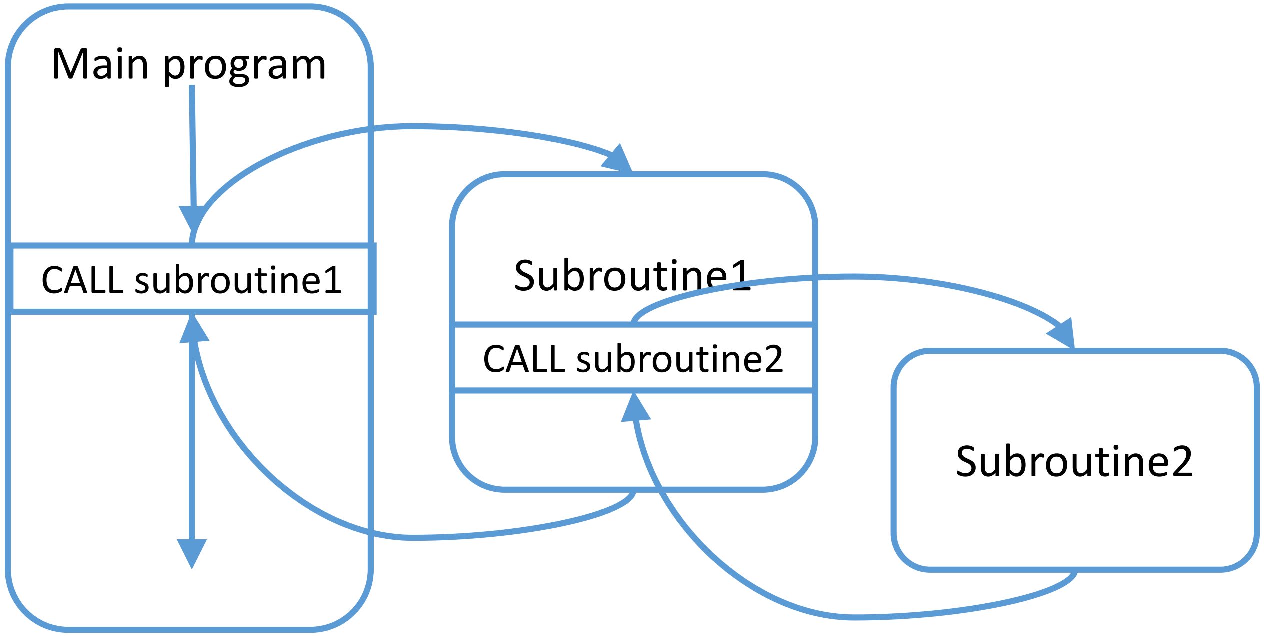
Subroutine





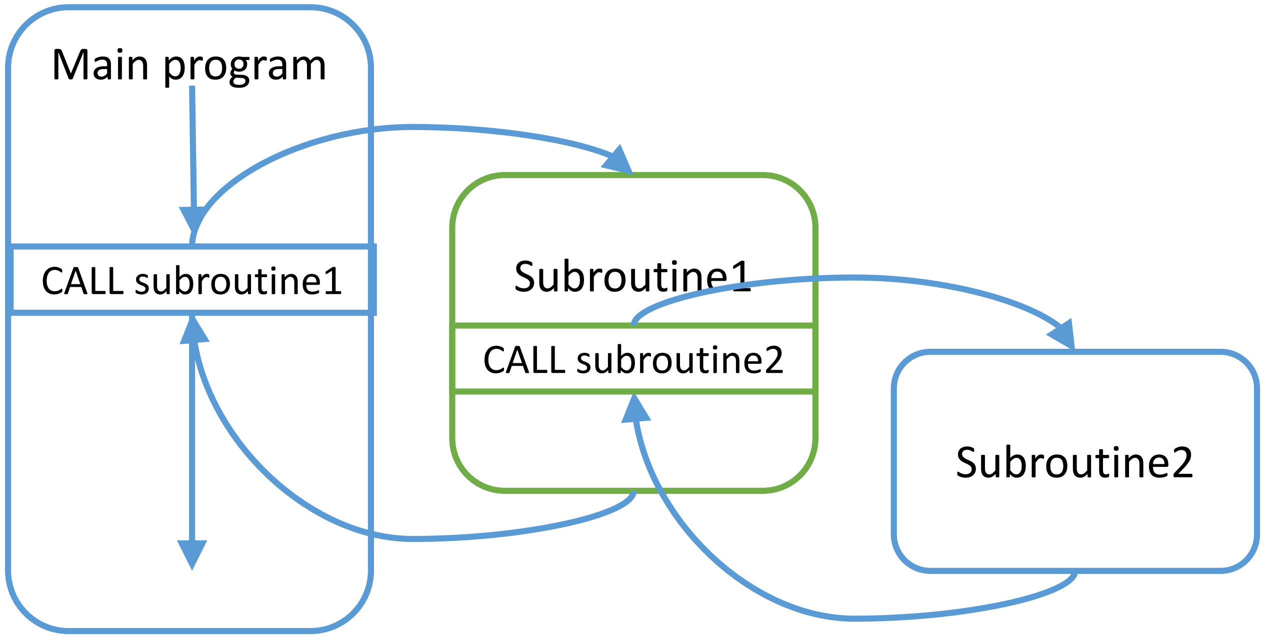
Memory location	Calling program	Memory location	Subroutine SUB
	⋮		
200	Call SUB	1000	first instruction
204	next instruction		⋮
	⋮		Return

Subroutine Nesting and the Processor Stack

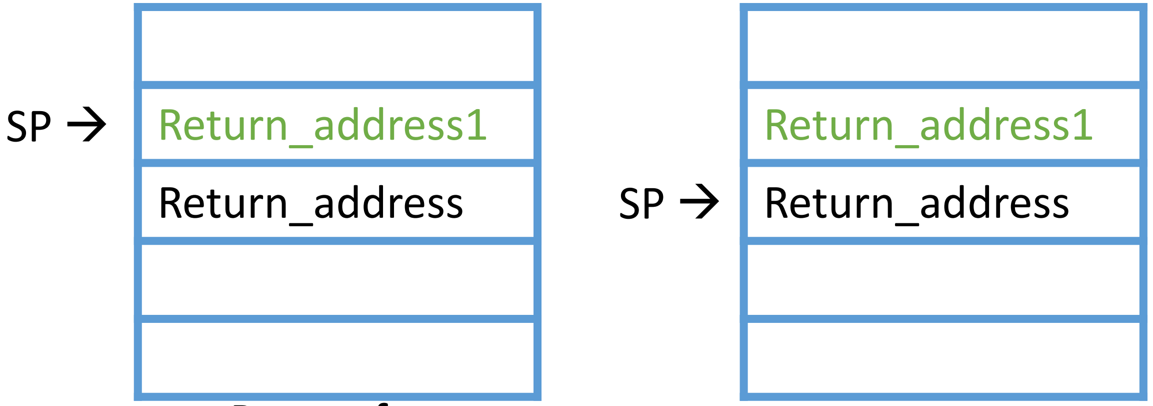


- *subroutine nesting* is to have one subroutine call another.
- The return address of the second call is also stored in the link register, overwriting its previous contents!
- Don’t loose the return address!!
  - Save the contents of the link register in some other location before calling another subroutine.

Subroutine Nesting and the Processor Stack



- return addresses are generated and used in a *last-in–first-out* order.
- return addresses associated with subroutine calls should be *pushed onto the processor stack.*



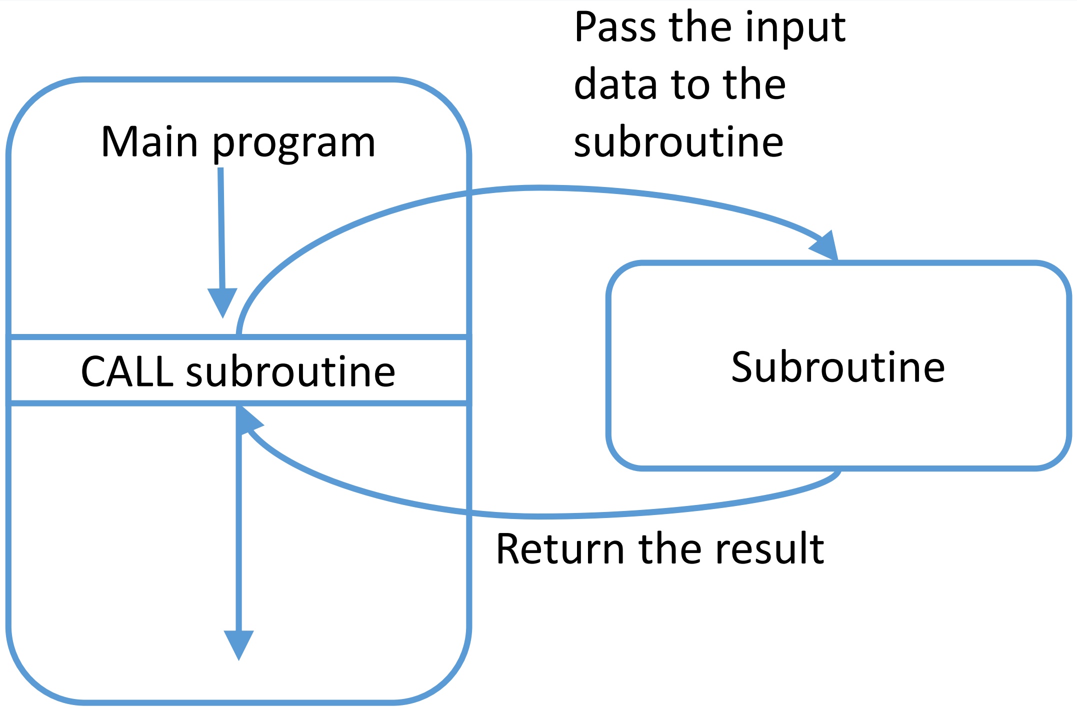
	Return from Subroutine 2
SP →	Return_address1
	Return_address

	Return from Subroutine 1
	Return_address1
SP →	Return_address

Return from Subroutine 2

Return from Subroutine 1

Parameter Passing



- Main program provides to the subroutine the parameters:
  - the operands or their addresses, to be used in the computation
- Subroutine returns other param.s:
  - the results of the computation
- The parameters may be placed
  - in registers or in memory locations, where they can be accessed by the subroutine.
  - Or, on the processor stack.

Microprocessors

Tuba Ayhan

MEF University – Spring 2017

ARM Processor – Instructions

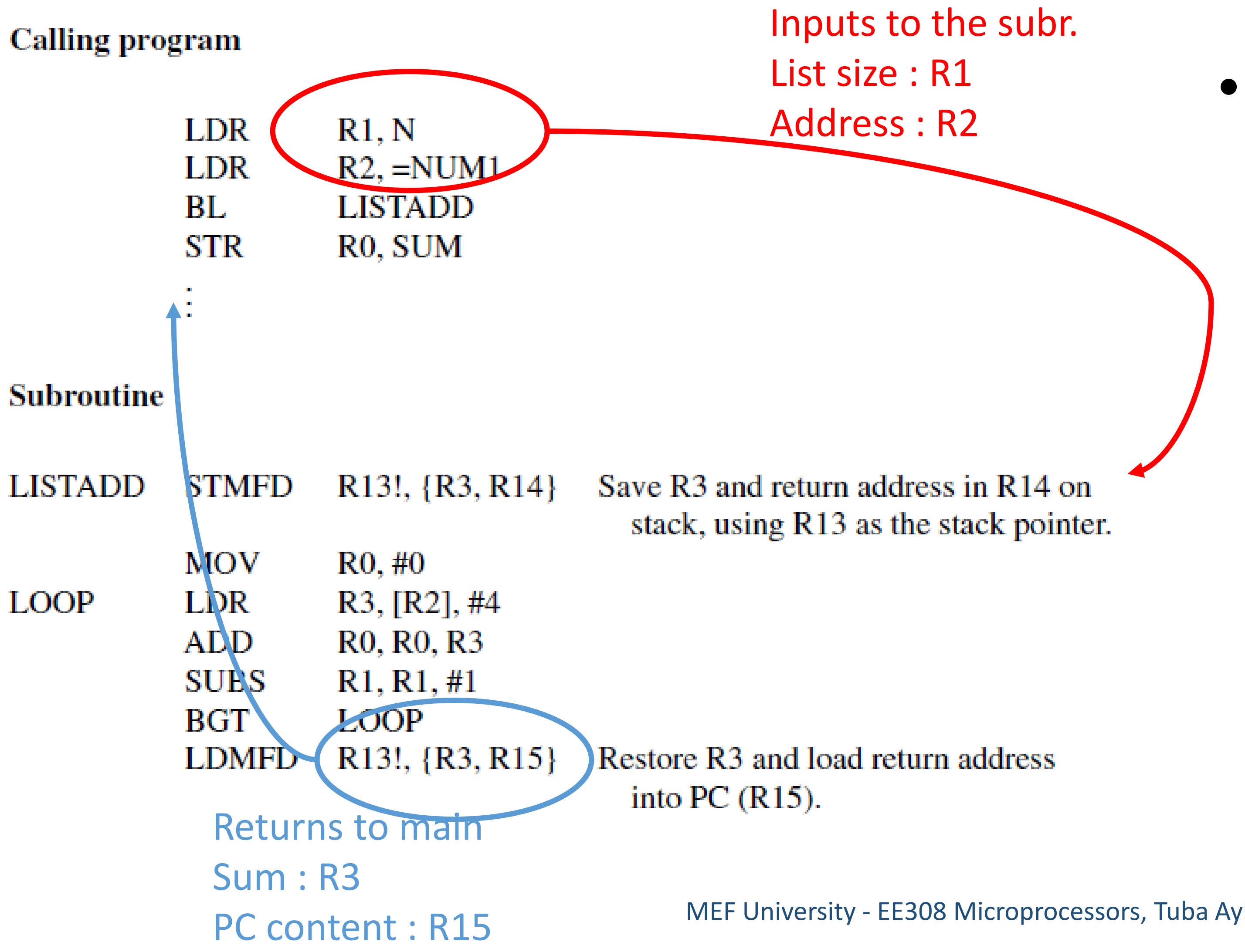
Subroutine Linkage Instructions

Computer Organization and Embedded Systems, Hamacher et. al

Subroutine Linkage Instructions

- Branch and Link (BL) instruction is used to call a subroutine.
- The return address, which is the address of the next instruction after the BL instruction, is loaded into register R14 (link register)

Example – Adding numbers in a list



Calling program

```
            LDR       R1, N
            LDR       R2, =NUM1
            BL        LISTADD
            STR       R0, SUM
            :
```

Subroutine

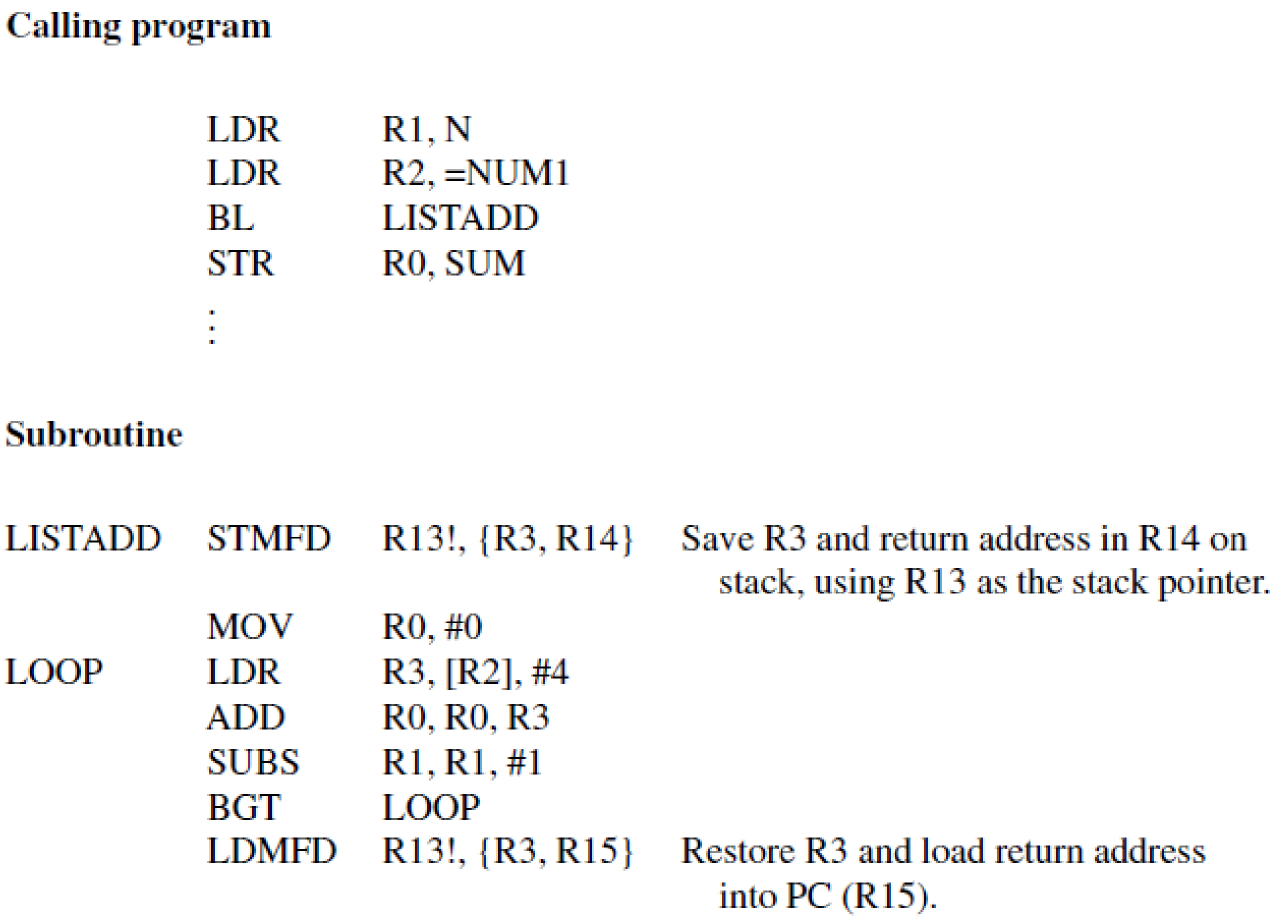
```
LISTADD     STMFD     R13!, {R3, R14}    Save R3 and return address in R14 on
                                             stack, using R13 as the stack pointer.
            MOV       R0, #0
LOOP        LDR       R3, [R2], #4
            ADD       R0, R0, R3
            SUBS      R1, R1, #1
            BGT       LOOP
            LDMFD     R13!, {R3, R15}    Restore R3 and load return address
                                             into PC (R15).
```

Inputs to the subr.
List size : R1
Address : R2

Returns to main
Sum : R3
PC content : R15

- Parameters are passed through registers:
  - The calling program passes the size of the number list and the address of the first number to the subroutine in registers R1 and R2.
  - The subroutine passes the sum back to the calling program in register R0.

Example



Calling program

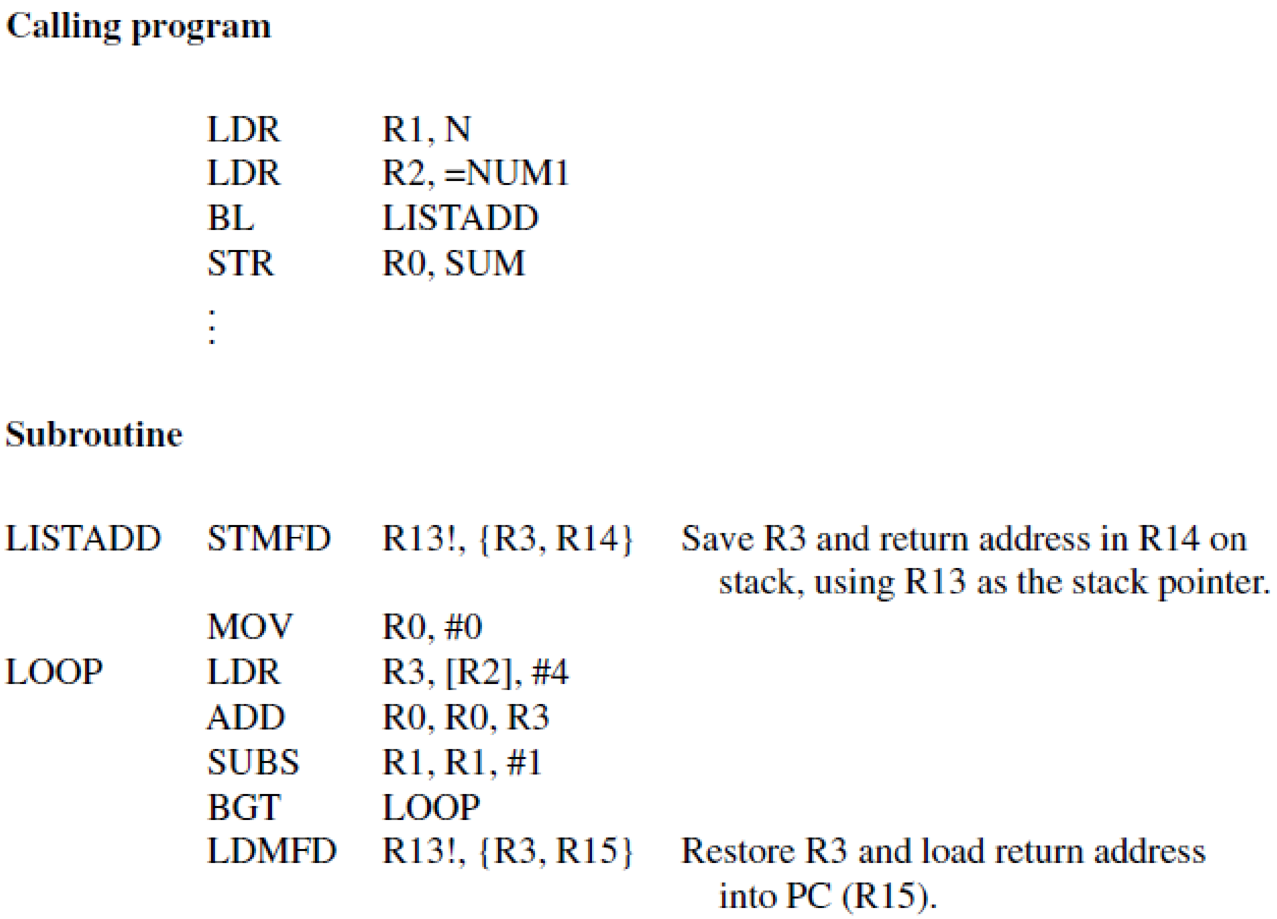
```
            LDR         R1, N
            LDR         R2, =NUM1
            BL          LISTADD
            STR         R0, SUM
            ⋮
```

Subroutine

```
LISTADD     STMFD       R13!, {R3, R14}     Save R3 and return address in R14 on
                                               stack, using R13 as the stack pointer.
            MOV         R0, #0
LOOP        LDR         R3, [R2], #4
            ADD         R0, R0, R3
            SUBS        R1, R1, #1
            BGT         LOOP
            LDMFD       R13!, {R3, R15}     Restore R3 and load return address
                                               into PC (R15).
```

- **STMFD**
  - The contents of the link register R14 and
  - R3 (sum) are saved on the stack by the instruction.
- **FD**
  - The stack grows toward lower addresses.
  - the stack pointer R13 is to be predecremented before pushing words onto the stack.

Example



Calling program

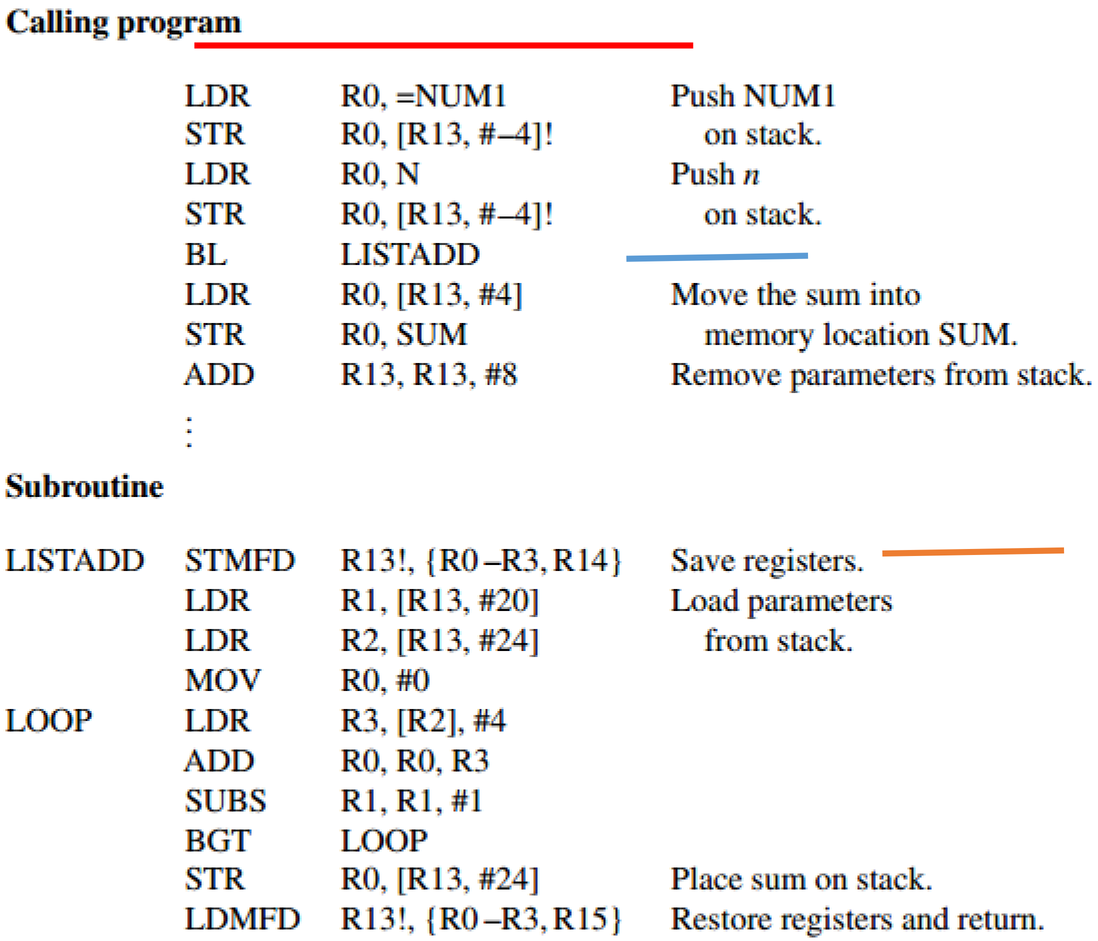
```
            LDR      R1, N
            LDR      R2, =NUM1
            BL       LISTADD
            STR      R0, SUM
            ⋮
```

Subroutine

```
LISTADD     STMFD    R13!, {R3, R14}   Save R3 and return address in R14 on
                                          stack, using R13 as the stack pointer.
            MOV      R0, #0
LOOP        LDR      R3, [R2], #4
            ADD      R0, R0, R3
            SUBS     R1, R1, #1
            BGT      LOOP
            LDMFD    R13!, {R3, R15}   Restore R3 and load return address
                                          into PC (R15).
```

- **LDMFD**
  - restores the contents of register R3
  - pops the saved return address into the PC (R15), (performs return operation automatically.

Example – with stack

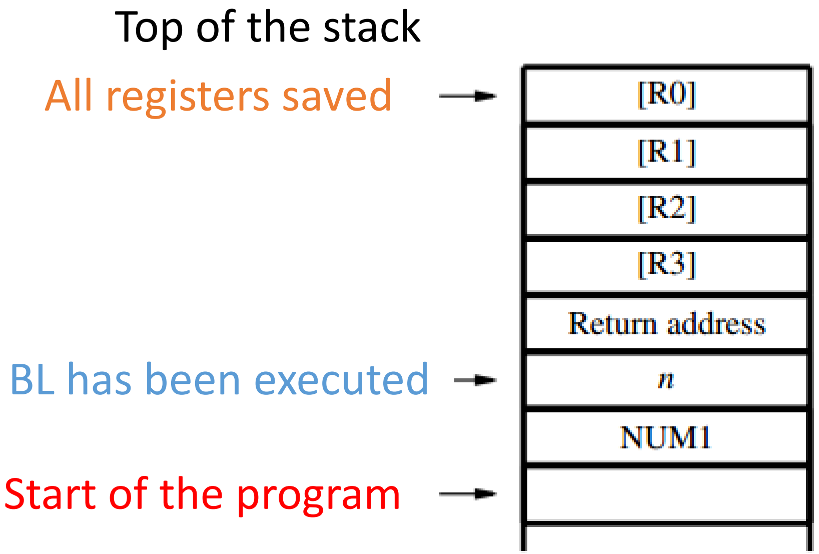


Calling program

```
        LDR     R0, =NUM1            Push NUM1
        STR     R0, [R13, #–4]!        on stack.
        LDR     R0, N                Push n
        STR     R0, [R13, #–4]!        on stack.
        BL      LISTADD
        LDR     R0, [R13, #4]        Move the sum into
        STR     R0, SUM                memory location SUM.
        ADD     R13, R13, #8         Remove parameters from stack.
        ⋮
```

Subroutine

```
LISTADD STMFD   R13!, {R0–R3, R14}   Save registers.
        LDR     R1, [R13, #20]       Load parameters
        LDR     R2, [R13, #24]         from stack.
        MOV     R0, #0
LOOP    LDR     R3, [R2], #4
        ADD     R0, R0, R3
        SUBS    R1, R1, #1
        BGT     LOOP
        STR     R0, [R13, #24]       Place sum on stack.
        LDMFD   R13!, {R0–R3, R15}   Restore registers and return.
```



Top of the stack

All registers saved

BL has been executed

Start of the program

(b) Top of stack at various times