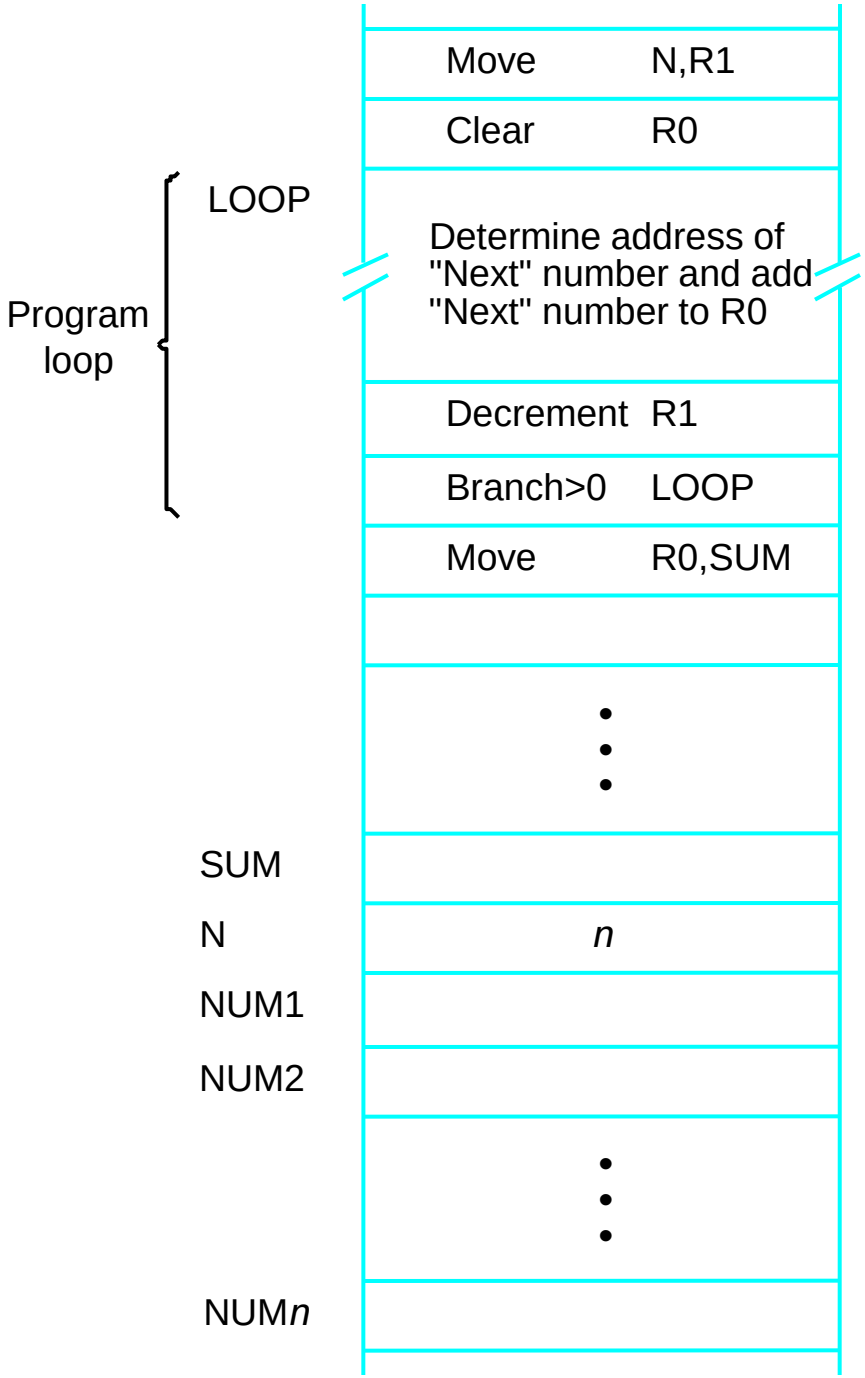
Addressing Modes Utility and Subroutines

Branching

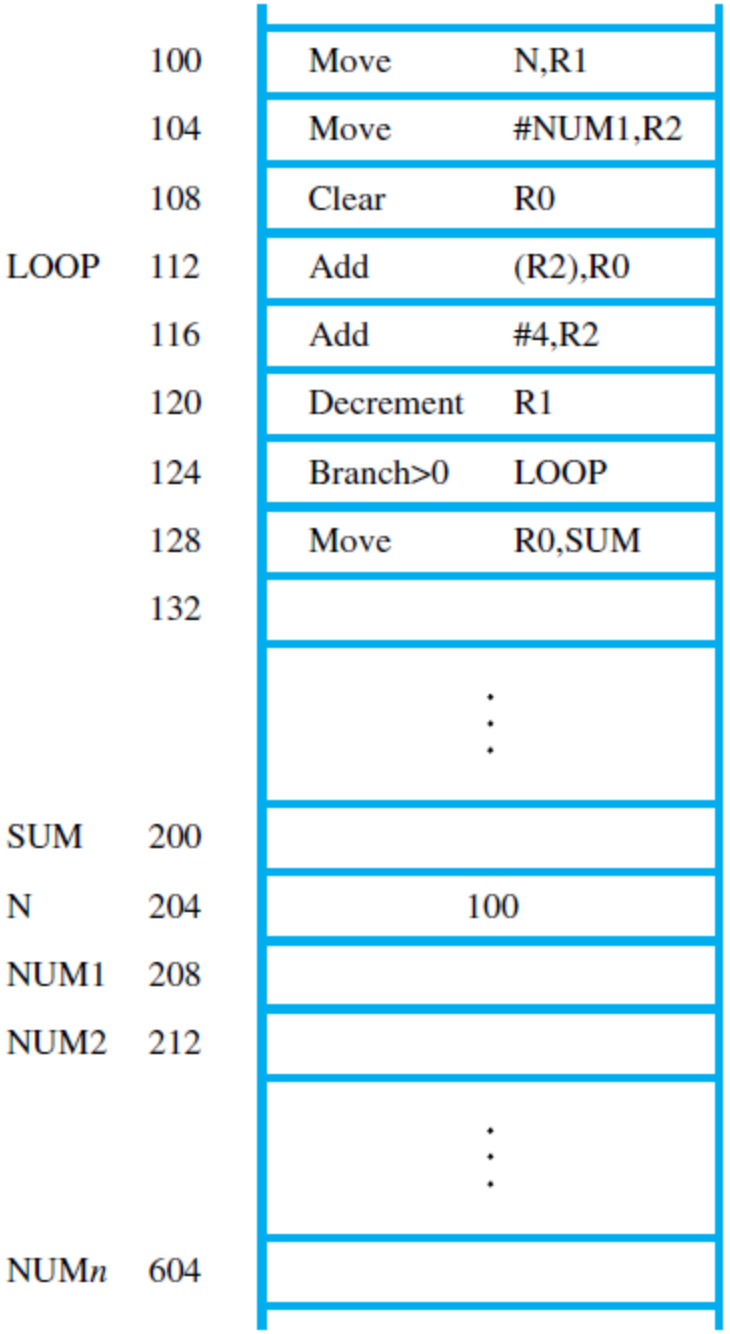
Branch target

Conditional branch

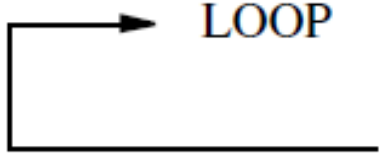
Using a loop to add *n* numbers.



Utility of using Indirect Addressing

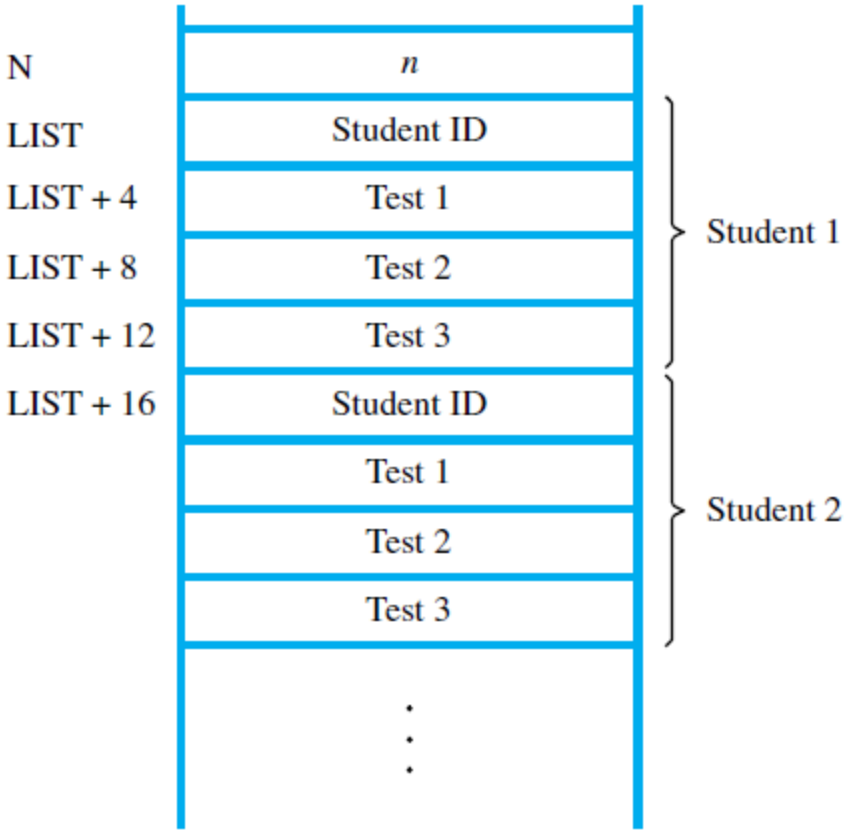


Using Auto-increment Mode



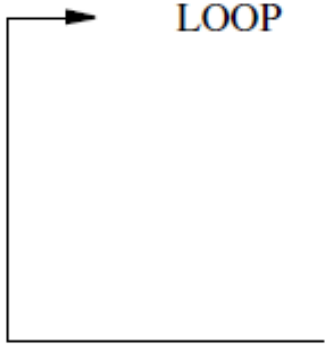
```
        Move    N,R1          }
        Move    #NUM1,R2      } Initialization
        Clear   R0            }
LOOP    Add     (R2)+,R0
        Decrement R1
        Branch>0  LOOP
        Move    R0,SUM
```

Utility of Indexed Addressing



A list of students' marks

Find sum of Test1, Test2 and Test 3 scores



```
        Move      #LIST,R0
        Clear     R1
        Clear     R2
        Clear     R3
        Move      N,R4
LOOP    Add       4(R0),R1
        Add       8(R0),R2
        Add       12(R0),R3
        Add       #16,R0
        Decrement R4
        Branch>0  LOOP
        Move      R1,SUM1
        Move      R2,SUM2
        Move      R3,SUM3
```

Subroutines

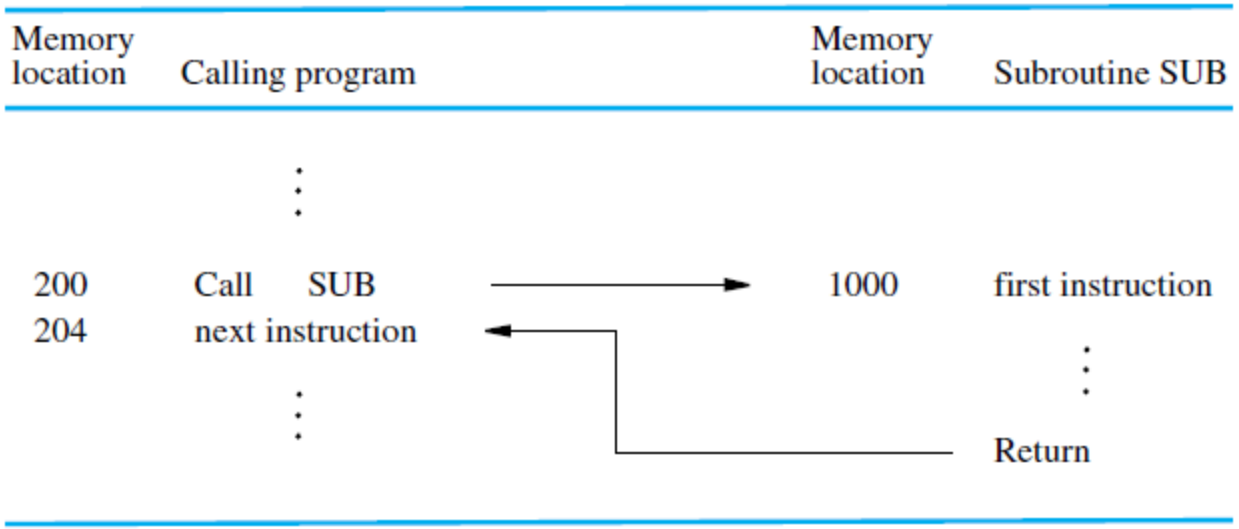
- In a given program, it is often necessary to perform a particular subtask many times on different data values.
- Such a subtask is usually called a *subroutine.*
- Only one copy of the instructions that constitute the subroutine is placed in the memory,.
- Any program that requires the use of the subroutine simply branches to its starting location – **Calling** the subroutine.
- After a subroutine has been executed, the calling program must resume execution, continuing immediately after the instruction that called the subroutine – **Returning** from subroutine.

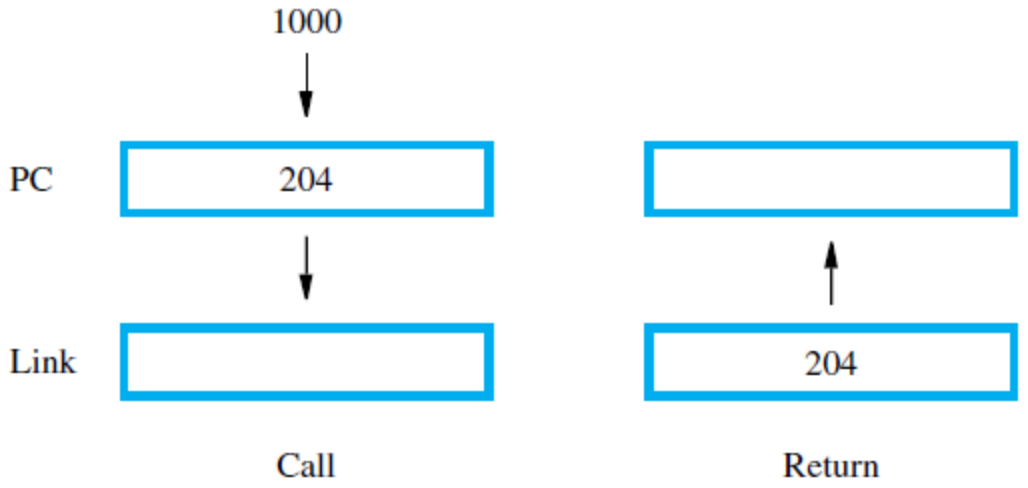
Calling and Returning

- Calling
  - Store the contents of the PC in the **link register.**
  - Branch to the target address specified by the instruction.

- Returning
  - Branch to the address contained in the link register

Calling and Returning

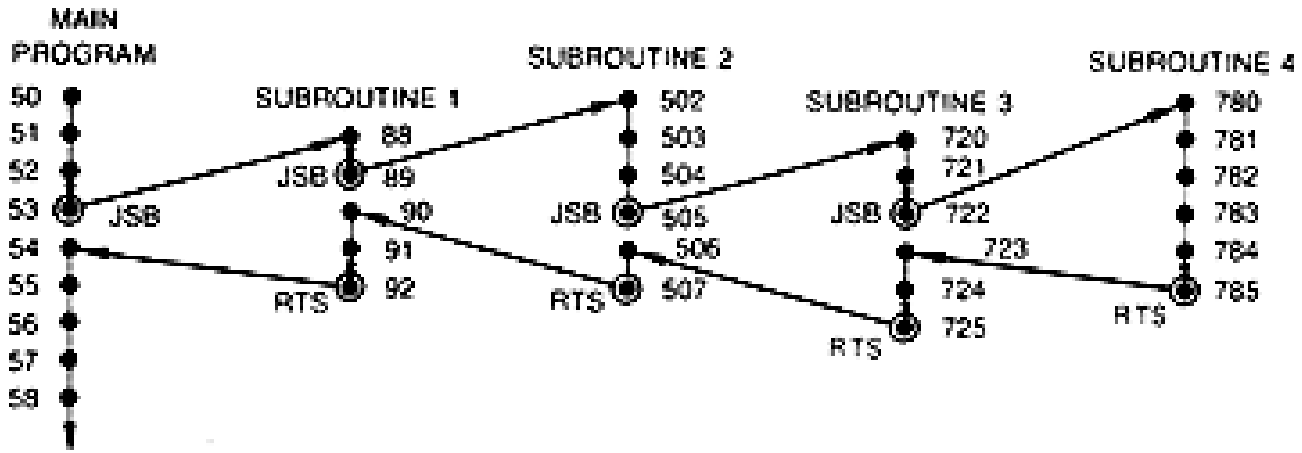




Nested subroutine call

- One subroutine calls another, which calls another and so on.
- In this case, the return address of the second call is also stored in the link register, destroying its previous contents.
- **Solution**
  - The last subroutine returns first.
  - LIFO: Use Processor stack for storing return addresses.
- SP points the top of stack.
- The **Call** instruction pushes the contents of the PC onto the processor stack and loads the subroutine address into the PC.
- The **Return** instruction pops the return address from the processor stack into the PC.

Nested subroutine calls - example



JSB : JUMP TO SUBROUTINE
RTS: RETURN FROM SUBROUTINE

LIFO STACK CONTENTS

Marks addition program using subroutine – passing parameters through registers

Calling program

```
            Move        N,R1          R1 serves as a counter.
            Move        #NUM1,R2      R2 points to the list.
            Call        LISTADD       Call subroutine.
            Move        R0,SUM        Save result.
            ⋮
```

Subroutine

```
LISTADD     Clear       R0            Initialize sum to 0.
LOOP        Add         (R2)+,R0      Add entry from list.
            Decrement   R1
            Branch>0    LOOP
            Return                    Return to calling program.
```

Passing parameters using stack

Assume top of stack is at level 1 below.

```
            Move          #NUM1,−(SP)    Push parameters onto stack.
            Move          N,−(SP)
            Call          LISTADD        Call subroutine
                                           (top of stack at level 2).
            Move          4(SP),SUM      Save result.
            Add           #8,SP          Restore top of stack
                                           (top of stack at level 1).
            ⋮

LISTADD     MoveMultiple  R0−R2,−(SP)    Save registers
                                           (top of stack at level 3).
            Move          16(SP),R1      Initialize counter to n.
            Move          20(SP),R2      Initialize pointer to the list.
            Clear         R0             Initialize sum to 0.
LOOP        Add           (R2)+,R0       Add entry from list.
            Decrement     R1
            Branch>0      LOOP
            Move          R0,20(SP)      Put result on the stack.
            MoveMultiple  (SP)+,R0−R2    Restore registers.
            Return                       Return to calling program.
```

Stack contents