



# SNS COLLEGE OF ENGINEERING

Kurumbapalayam (PO), Coimbatore - 641 107

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

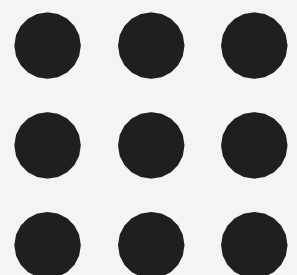
DEPARTMENT OF CSE (IoT)

COURSE NAME: 23ITT201 DIGITAL PRINCIPLES AND  
COMPUTER ORGANIZATION

II YEAR/ III SEM

Unit 3 : COMPUTER FUNDAMENTALS

Addressing Modes

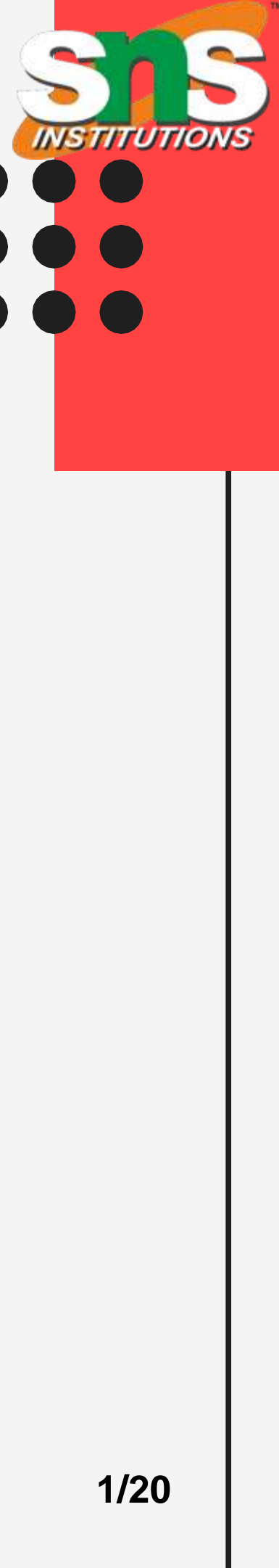


23ITT201 / DP & CO / D.KAVITHA/AP/CSE(IoT) / Unit 3 / COMPUTER FUNDAMENTALS

10/19/2024



# Addressing Modes



The different ways in which the location of an operand is specified in an instruction are referred to as **addressing modes**.

## Different Addressing modes

- Implied
- Immediate
- Register
- Direct/Absolute
- Indirect
- Index
- Relative
- Autoincrement
- Autodecrement



# Effective Address (EA)



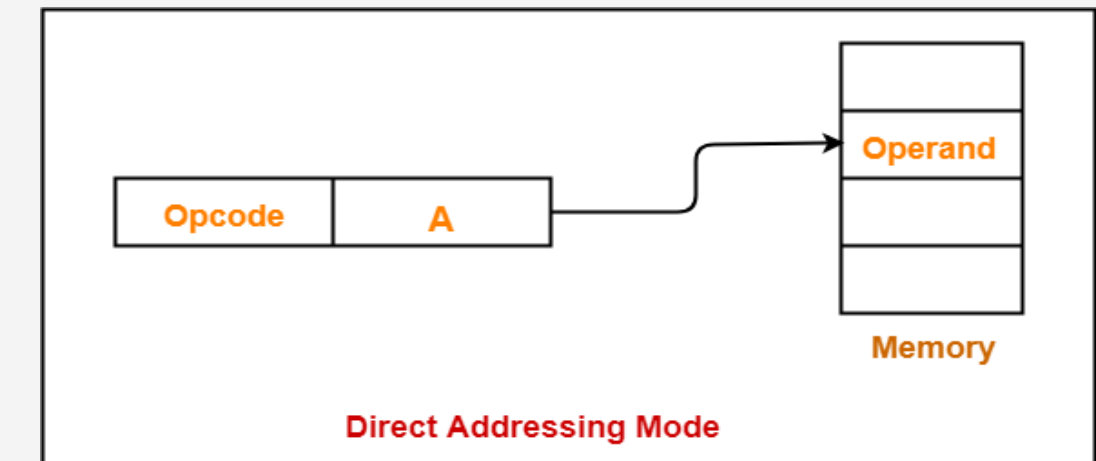
- In the addressing modes that follow, the instruction does not give the operand or its address explicitly.
- Instead, it provides information from which an effective address (EA) can be derived by the processor when the instruction is executed.
- The **effective address** is the location of an operand which is stored in memory.

Move LOC, R0    EA = LOC

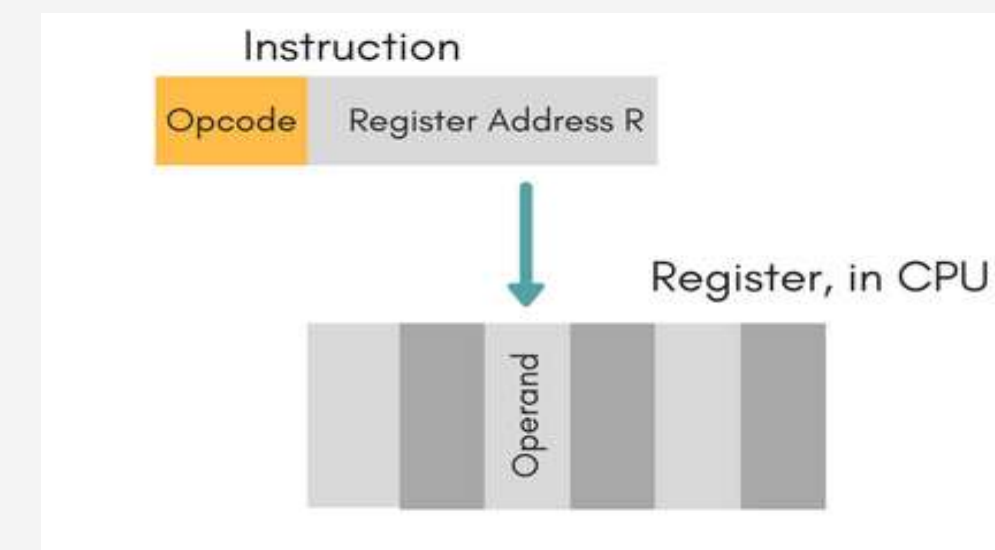
# Addressing Modes

- Implied
  - Instructions that comprise only an opcode without an operand
  - Ex: INCA

- Immediate mode
  - The use of a **constant** in “MOV 5, R1” or “MOV #5, R1” i.e.  $R1 \leftarrow 5$



- Absolute (Direct) Address – Implementation of **variables**
  - Operand is in a memory location
  - E.g. Move LOC, R1
- Register Mode
  - Indicate register holds the operand



# Indirection and Pointers

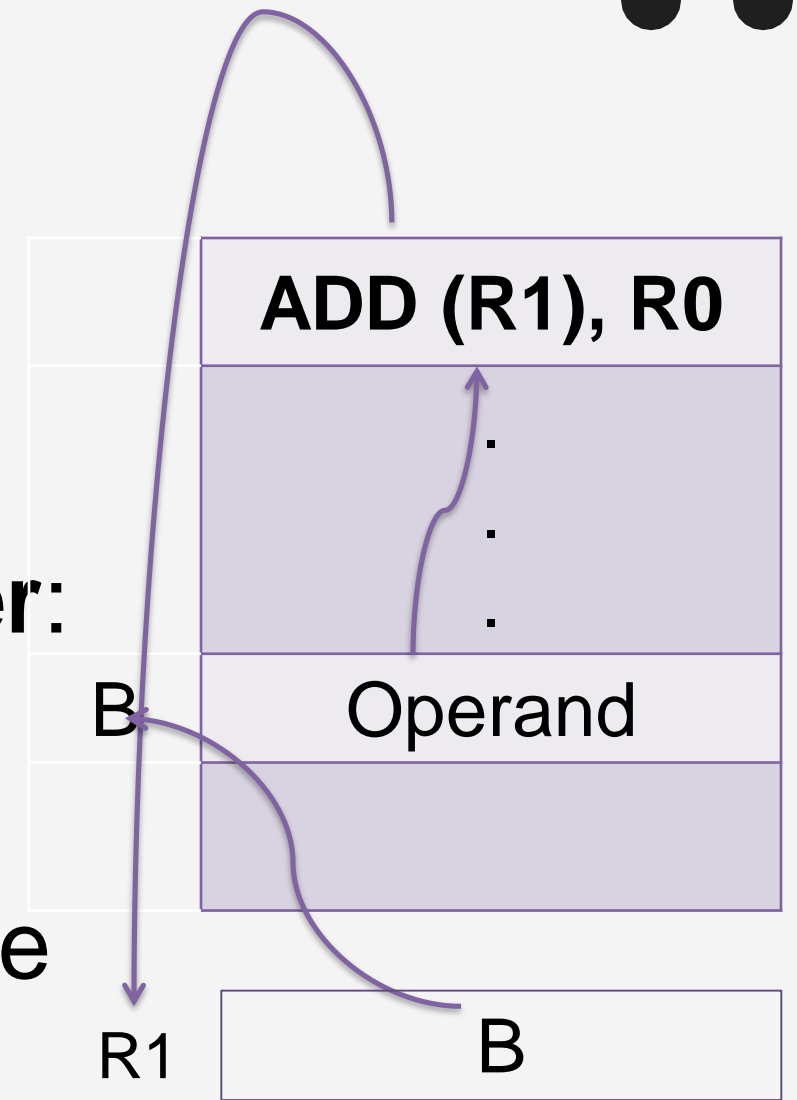
- Indirect Addressing – Instruction provides information from which memory address of operand determined
- EA of the operand is the contents of register or memory location whose address appears in the instruction.

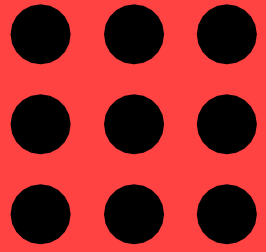
## Indirect addressing through a general purpose register:

- Indicate the register (e.g. R1) that holds the address of the variable (e.g. B) that holds the operand

ADD (R1), R0

- The register or memory location that contain the address of an operand is called a **pointer**



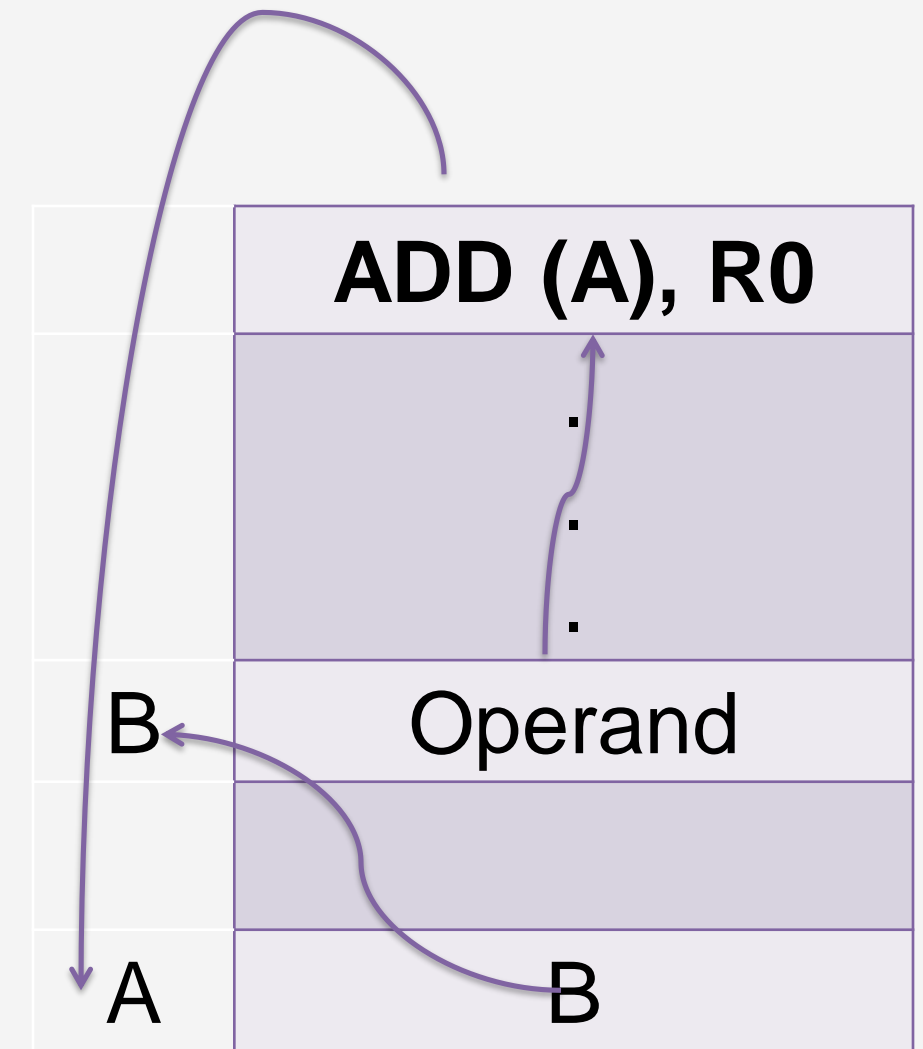


# Indirection and Pointers

## Indirect addressing through a memory location:

- Indicate the memory variable (e.g. A )that holds the address of the variable (e.g. B) that holds the operand

ADD (A), R0





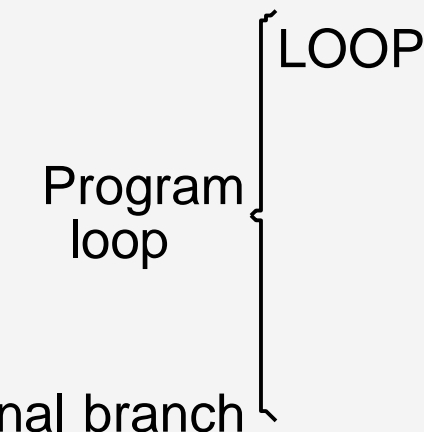
# Indirect Addressing Example

## Addition of N numbers

```

Move    N,R1          ; N = Numbers to add
Move    #NUM1,R2      ; R2= Address of 1st no.
Clear   R0            ; R0 = 00
Loop :  Add    (R2), R0 ; R0 = [NUM1] + [R0]
        Add    #4, R2   ; R2= To point to the next
                        ; number
        Decrement R1    ; R1 = [R1] -1
        Branch>0 Loop; Check if R1>0 or not if
                        ; yes go to Loop
        Move   R0, SUM  ; SUM= Sum of all no.

```



Move	N,R1
Clear	R0
Determine address of "Next" number and add "Next" number to R0	
Decrement R1	
Branch>0 LOOP	
Move	R0,SUM
⋮	
SUM	
N	
NUM1	
NUM2	
⋮	
NUMn	

Using a loop to add *n* numbers.



# Example



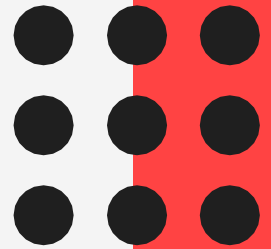
## Addition of N numbers

1. Move N,R1 ; N = 3
2. Move #NUM1,R2 ; R2= 10000H
3. Clear R0 ; R0 = 00
4. Loop : Add (R2), R0 ; R0 = 10 + 00 = 10
5. Add #4, R2 ; R2 = 10004H
6. Decrement R1 ; R1 = 2
7. Branch>0 Loop ; Check if R1>0 if  
; yes go to Loop
8. Move R0, SUM ; SUM=



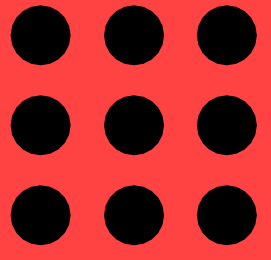


# Example



## Addition of N numbers

1. Move N,R1 ; N = 3
2. Move #NUM1,R2 ; R2= 10000H
3. Clear R0 ; R0 = 00
4. Loop : Add (R2), R0 ; R0 = 20 + 10 = 30
5. Add #4, R2 ; R2 = 10008H
6. Decrement R1 ; R1 = 1
7. Branch>0 Loop ; Check if R1>0 if  
; yes go to Loop
8. Move R0, SUM ; SUM=



# Example



## Addition of N numbers

1. Move N,R1 ; N = 3
2. Move #NUM1,R2 ; R2= 10000H
3. Clear R0 ; R0 = 00
4. Loop : Add (R2), R0 ; R0 = 30 + 30 = 60
5. Add #4, R2 ; R2 = 1000CH
6. Decrement R1 ; R1 = 0
7. Branch>0 Loop ; Check if R1>0 if  
; yes go to Loop
8. Move R0, SUM ; SUM= 60



# Indexing and Arrays



- Useful in lists and arrays
- Index mode: The EA of the operand is generated by adding a constant value to the contents of a register.

- Symbolic representation

$X(R_i)$  ;  $X = \text{constant value}$

$EA = X + (R_i)$

- $X$  defined as offset or displacement

# Two ways of using Index mode

Address	Memory
	Add 20(R1), R2
	.
	.
	.
	.
10000	
	.
	.
	.
	.
10020	Operand

Offset=20

R1	10000
----	-------

Offset is given as a Constant

Address	Memory
	Add 10000(R1), R2
	.
	.
	.
	.
10000	
	.
	.
	.
	.
10020	Operand

Offset=20

R1	20
----	----

Offset is in the index register

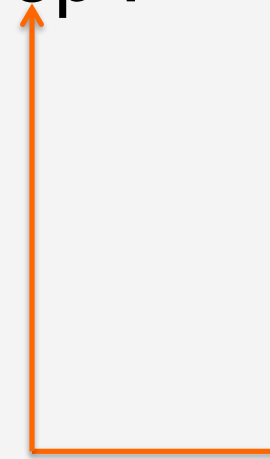
# Example: Indexing and Arrays

- Array

List of students marks

Address	Memory	Comments
N	n	No. of students
LIST	Student ID1	Student 1
LIST+4	Test 1	
LIST+8	Test 2	
LIST+12	Test 3	
LIST+16	Student ID2	Student 2
LIST+20	Test 1	
LIST+24	Test 2	
LIST+28	Test 3	

Loop :



```

Move #LIST, R0
Clear R1
Clear R2
Clear R3
Move N, R4
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
    
```

Program to find the sum of marks of all subjects and store it in memory.

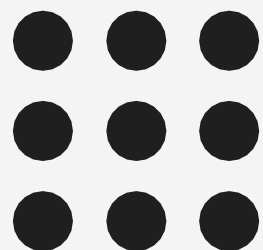
- Indexed addressing used in accessing test marks from the list



# Relative Addressing



- Relative mode – the effective address is determined by the index mode using the program counter in place of the general-purpose register.
- $X(PC)$  – note that  $X$  is a signed number
- Commonly used to specify target address in branch instruction  
**Branch > 0      LOOP**
- This location is computed by specifying it as an offset from the current value of PC.
- Branch target may be either before or after the branch instruction, the offset is given as a signed num.





# Relative addressing mode - Example

## Addition of N numbers

	<i>Move</i>	<i>N, R1</i>	<i>; N = Numbers to add</i>
	<i>Move</i>	<i>#NUM1, R2</i>	<i>; R2= Address of 1<sup>st</sup> no.</i>
	<i>Clear</i>	<i>R0</i>	<i>; R0 = 00</i>
<i>1000</i>	<i>Loop: Add</i>	<i>(R2), R0</i>	<i>; R0 = [NUM1] + [R0]</i>
<i>1004</i>	<i>Add</i>	<i>#4, R2</i>	<i>; R2= To point to the next number</i>
<i>1008</i>	<i>Decrement</i>	<i>R1</i>	<i>; R1 = [R1] - 1</i>
<i>1012</i>	<i>Branch&gt;0</i>	<i>Loop; Check if R1&gt;0 or not if</i>	<i>; yes go to Loop</i>
<i>1016</i>	<i>Move</i>	<i>R0, SUM</i>	<i>; SUM= Sum of all no.</i>

- PC = 1016
- To branch to Loop (1000), offset X = -16
- $X(PC) = -16(1016) = -16 + 1016 = 1000$

# Additional Modes

- Autoincrement mode – the effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.
- (Ri)+. The increment is 1 for byte-sized operands, 2 for 16-bit operands, and 4 for 32-bit operands.
- Autodecrement mode: -(Ri) – decrement first and used as an EA

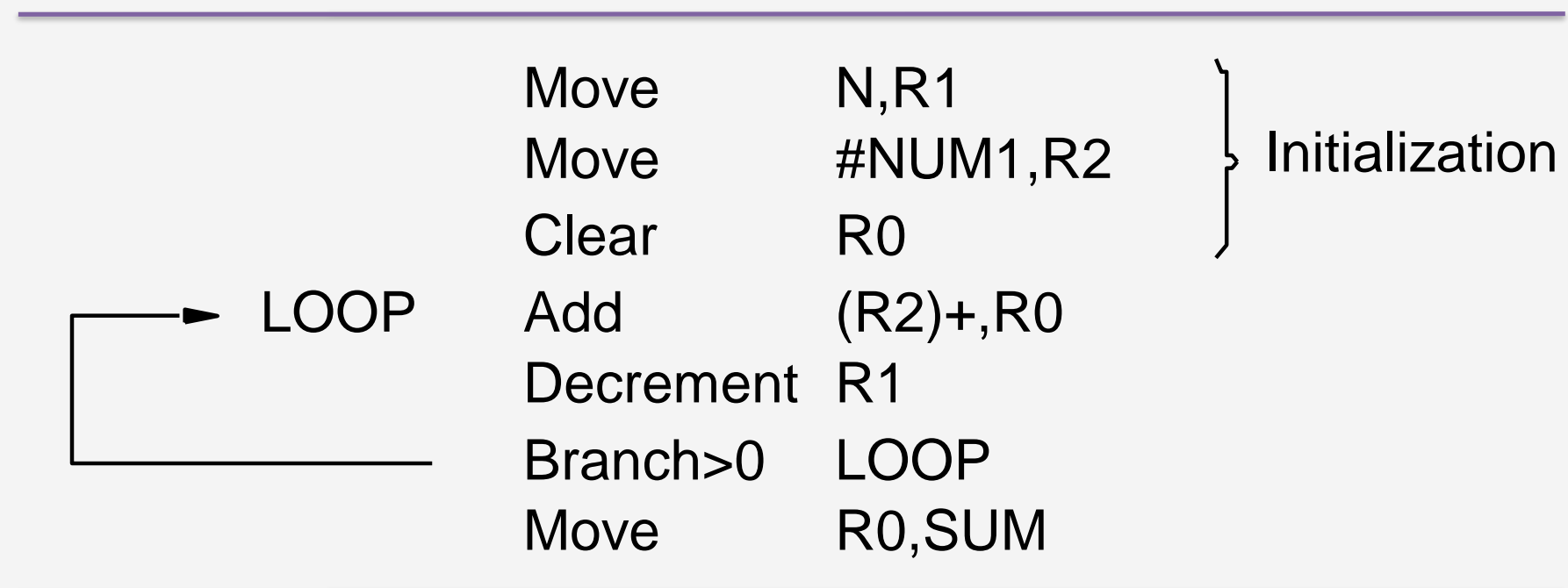


Figure 2.16. The Autoincrement addressing mode used in the program of Figure 2.12.





# Summary



Name	Assemblersyntax	Addressingfunction
Immediate	#Value	Op erand = Value
Register	$R_i$	$EA = R_i$
Absolute (Direct)	LOC	$EA = LOC$
Indirect	$(R_i)$ $(LOC)$	$EA = [R_i]$ $EA = [LOC]$
Index	$X(R_i)$	$EA = [R_i] + X$
Relative	$X(PC)$	$EA = [PC] + X$
Autoincrement	$(R_i) +$	$EA = [R_i]$
Autodecrement	$-(R_i)$	Increment $R_i$ Decrement $R_i$ $EA = [R_i]$

Generic Addressing Modes



# Assessment



1. The instruction, Add #45,R1 does \_\_\_\_\_
  - a) Adds the value of 45 to the address of R1 and stores 45 in that address
  - b) Adds 45 to the value of R1 and stores it in R1
  - c) Finds the memory location 45 and adds that content to that of R1
  - d) None of the mentioned
  
2. Which addressing mode execute its instructions within CPU without the necessity of reference memory for operands?
  - a. Implied Mode    b. Immediate Mode
  - c. Direct Mode    d. Register Mode
  
3. The addressing mode/s, which uses the PC instead of a general purpose register is \_\_\_\_\_
  - a) Indexed with offset    b) Relative    c) Direct
  - d) Both Indexed with offset and direct



# Assessment



4. The addressing mode, where you directly specify the operand value is \_\_\_\_\_

- a) Immediate      b) Direct      c) Definite      d) Relative

5. \_\_\_\_\_ addressing mode is most suitable to change the normal sequence of execution of instructions.

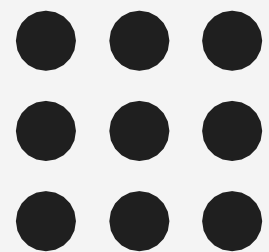
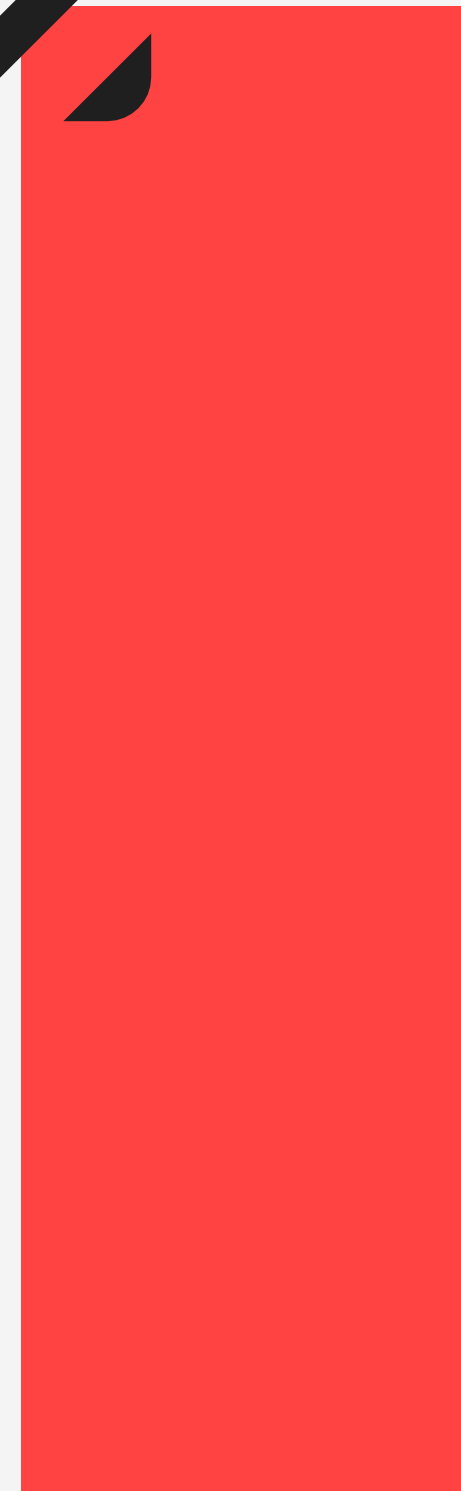
- a) Relative      b) Indirect      c) Index with Offset      d) Immediate



# Answers



1. B
2. D
3. B
4. A
5. A



**23ITT201 / DP & CO / D.KAVITHA/AP/CSE(IoT) / Unit 3 / COMPUTER FUNDAMENTALS**



Thank You