INSTRUCTION SET & ADDRESSING MODES

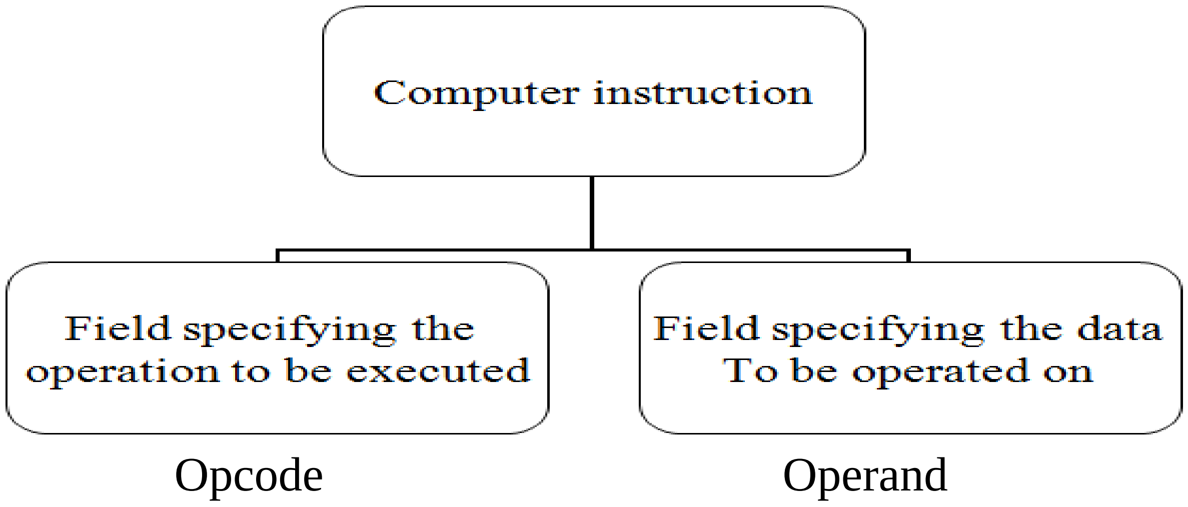
P. K. ROY

Instruction code

A process is controlled by a ***program***

- A program is a set of ***instructions*** that specify the operations, data, and the control sequence.
- An instruction is stored in binary code that specifies a sequence of micro-operations.
- Instruction codes together with data are stored in memory (Stored Program Concept)
- A computer will usually have a variety of instruction codes. It is the function of the control unit to interpret each instruction code and provide the necessary control functions needed to process the instruction.

Instruction Format


Computer instruction
Field specifying the operation to be executed
Field specifying the data To be operated on
Opcode
Operand

More precisely, there is another field that specifies the way the operand or the effective address is determined – ***Mode field.*** Generally, this field is implicitly associated with the ***opcode field.***

Mode	**Opcode**	**Operand**
Operation to be performed		Data/Effective address

Types of Instruction

Are of 3 types –

1. Memory Reference Instructions
   - => Micro-operation performed with certain memory location (primary memory)
2. Register Reference Instructions
   - => Micro-operation within different registers
3. Input/Output Instructions
   - => I/O operations

Depending on these 3, most computer instructions can be classified into 3 categories –

1. Data Transfer Instructions
2. Data Manipulation Instructions
3. Program Control Instructions

Data Transfer Instructions

Transfer of data from one location to another without changing its content.

- Between memory & processor registers
- Between processor registers & input or output
- Between processor registers themselves

Typical data transfer instructions are –

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Manipulation Instructions

Perform operations on data & provide the computational capabilities for the computer.

Usually divided into 3 basic types –

1. Arithmetic Instructions
2. Logical & Bit Manipulation Instructions
3. Shift Instructions

1. Arithmetic Instructions

Perform the basic arithmetic operations –

- Addition
- Subtraction
- Multiplication
- Division
- Increment
- Decrement

Name	Mnemonic
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Increment	INC
Decrement	DEC
Add With Carry	ADDC
Subtract With Borrow	SUBB
Negate(2’s Complement)	NEG

2. Logical & Bit Manipulation Instructions

- Perform binary operations on bit-streams stored in registers.
- Useful for manipulating individual bit or group of bits that represent binary-coded information.

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear Carry	CLRC
Set Carry	SETC
Complement Carry	COMC
Enable Interrupt	EI
Disable Interrupt	DI

3. Shift Instructions

- Shifting of bits of a word to the left or right.

Name	Mnemonic
Logical Shift Right	SHR
Logical Shift Left	SHL
Arithmetic Shift Right	ASHR
Arithmetic Shift Left	ASHL
Circular Shift Right	CIR
Circular Shift Left	CIL
Rotate Right	ROR
Rotate Left	ROL
Rotate Right Through Carry	RORC
Rotate Leftt Through Carry	ROLC

Program Control Instructions

- Specify conditions for altering the content of PC, while data transfer & manipulation instructions specify conditions for data processing operations.
- Cause a break in the sequence of instructions.
- Capability for branching to different program segments.
- Provide control over the flow of program execution.

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

RISC (Reduced Instruction Set Computer)

1. All operations are performed in registers.
2. Simple load & store instructions used for memory access.
3. Instructions can be decoded easily.
4. Incorporates a few addressing modes.
5. Supports simple fixed-length instruction format.
6. Generally employs hardwired control unit.
7. Supports pipelining (one instruction per clock cycle).
8. Computations are performed at relatively faster speed.
9. Comparatively lengthy programs that take up larger memory.
10. Requires a large no. of registers.

   e.g. Power PC, Ultra SPARC, etc.

CISC (Complex Instruction Set Computer)

1. All instructions are present in memory.
2. Complex & large no. of instruction set.
3. Decoding of instructions is a complicated & time consuming task.
4. Large no. of addressing modes.
5. Smaller & faster programs, thus, improving performance.
6. Uses micro-programmed control unit.
7. Doesn’t support pipelining (different instructions take different amount of clock cycle).
8. Supports variable-length instruction formats.

   e.g. Intel 80486, Celeron Computers, Pentium Processors, etc.

Operand Addressing & Instruction Representation

- Number Of Operands Per Instruction
- Four basic architectural types

$$\left.\begin{array}{l}\text{– 0-address Instructions}\\ \text{– 1-address Instructions}\\ \text{– 2-address Instructions}\\ \text{– 3-address Instructions}\end{array}\right\}\begin{array}{l}\text{CISC}\\ \text{Instructions}\end{array}$$

– RISC Instructions

0-Address Instructions

- No explicit operands in the instruction
- Operands kept on stack in memory
- Instruction removes top *N items from stack*
- Instruction leaves result on top of stack

e.g. Considering X = (A + B) * (C + D)

TOS => Top of the Stack
M[X] => Memory Location of X
A,B,C,D,X => Data

0 – address

Instructions	Micro-operations
PUSH A	TOS $\leftarrow$ A
PUSH B	TOS $\leftarrow$ B
ADD	TOS $\leftarrow$ (A + B)
PUSH C	TOS $\leftarrow$ C
PUSH D	TOS $\leftarrow$ D
ADD	TOS $\leftarrow$ (C + D)
MUL	TOS $\leftarrow$ (A + B) * (C + D)
ST X	M[X] $\leftarrow$ TOS

1-Address Instructions

- One explicit operand per instruction
- Second operand is implicit
  - Always found in hardware register
  - Known as *accumulator*

e.g. Considering $X = (A + B) * (C + D)$

Instructions	Micro-operations
LD A	$AC \leftarrow M[A]$
ADD B	$AC \leftarrow AC + M[B]$
ST T	$M[T] \leftarrow AC$
LD C	$AC \leftarrow M[C]$
ADD D	$AC \leftarrow AC + M[D]$
MUL T	$AC \leftarrow AC * M[T]$
ST X	$M[X] \leftarrow AC$

AC is the Accumulator

2-Address Instructions

- Two explicit operands per instruction
- Result overwrites one of the operands
- Operands known as *source and destination*
- Works well for instructions such as memory copy

e.g. Considering $X = (A + B) * (C + D)$

Instructions	Micro-operations
MOV R1 , A	R1 $\longleftarrow$ M[A]
ADD R1 , B	R1 $\longleftarrow$ R1 + M[B]
MOV R2 , C	R2 $\longleftarrow$ M[C]
ADD R2 , D	R2 $\longleftarrow$ R2 + M[D]
MUL R1 , R2	R1 $\longleftarrow$ R1 * R2
MOV X , R1	M[X] $\longleftarrow$ R1

R1 , R2 : General purpose registers

3-Address Instructions

- Three explicit operands per instruction
- Operands specify *source*, *destination, and result*

e.g. Considering $X = (A + B) * (C + D)$

Instructions	Micro-operations
ADD R1 , A , B	$R1 \leftarrow M[A] + M[B]$
ADD R2 , C , D	$R2 \leftarrow M[C] + M[D]$
MUL X , R1 , R2	$M[X] \leftarrow R1 * R2$

RISC Instructions

- Processor is restricted to simple ***"Load & Store"*** instructions when communicating between CPU & memory.
- All the other instructions are executed within the processor registers without referring to memory.
- Computational-type instructions have 3 addresses which are for specifying processor registers.

e.g. Considering $X = (A + B) * (C + D)$

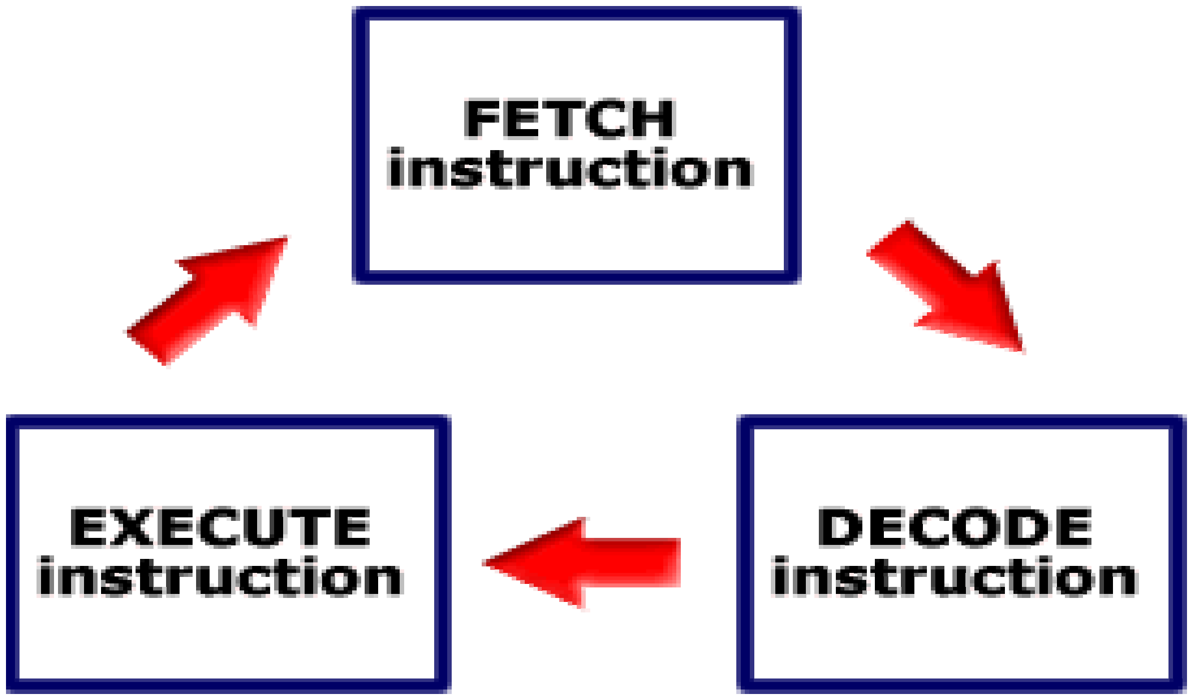
Instructions	Micro-operations
LD R1 , A	$R1 \leftarrow M[A]$
LD R2 , B	$R2 \leftarrow M[B]$
ADD R3 , R1 , R2	$R3 \leftarrow R1 + R2$
LD R4 , C	$R4 \leftarrow M[C]$
LD R5 , D	$R5 \leftarrow M[D]$
ADD R6 , R4 , R5	$R6 \leftarrow R4 + R5$
MUL R7 , R3 , R6	$R7 \leftarrow R3 * R6$
ST X , R7	$M[X] \leftarrow R7$

Operand Types

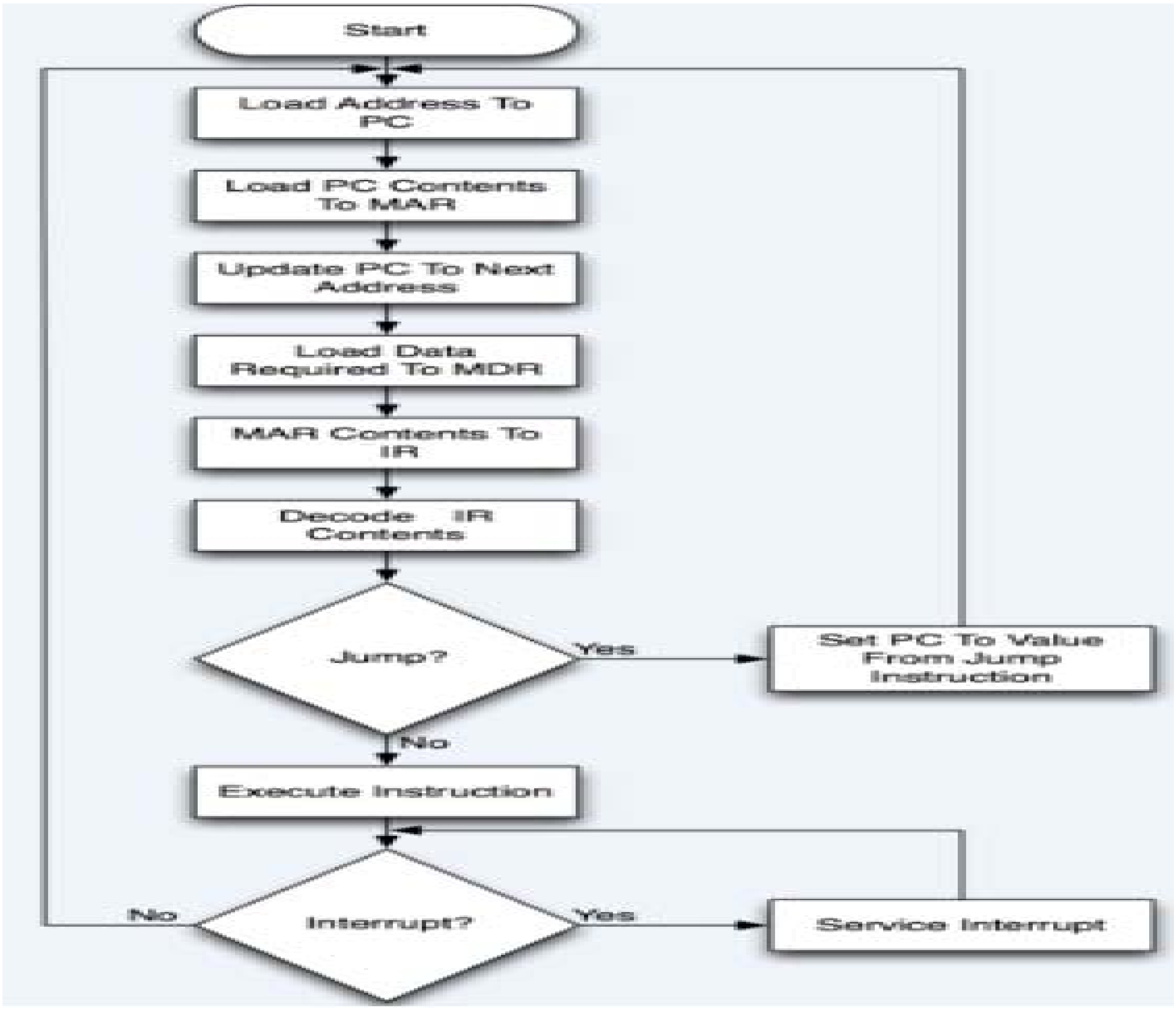
- Operand that specifies a source
  - Signed constant
  - Unsigned constant
  - Contents of a register
  - Value in a memory location
- Operand that specifies a destination
  - Single register
  - Pair of contiguous registers
  - Memory location
- Operand that specifies a constant is known as ***immediate*** value
- Memory references usually much more expensive than immediate or register access

<u>Instruction Cycle</u>

The processing required for a single instruction.



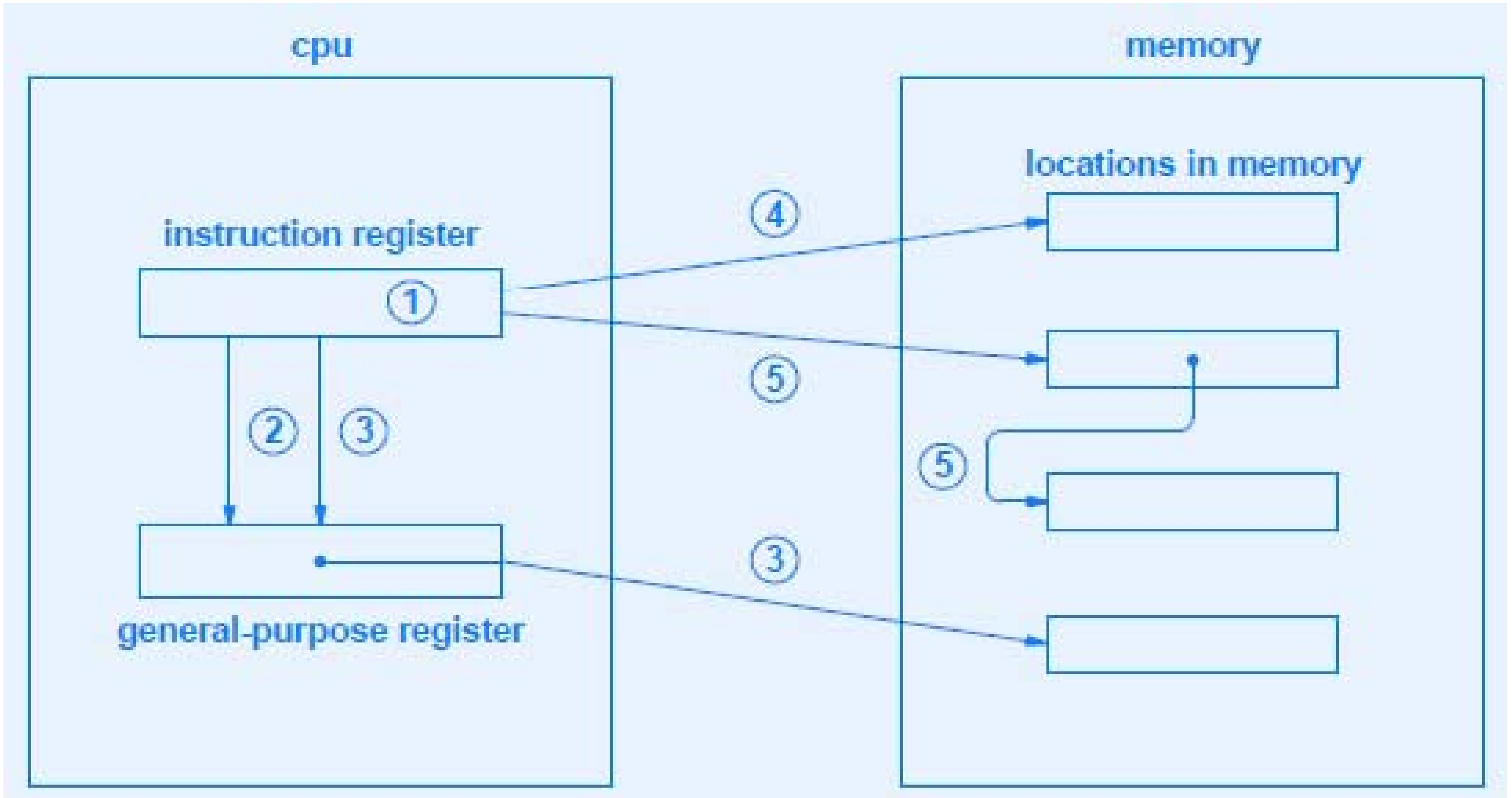
Note: *Execute block implicitly consists of storing of result after execution of certain instruction.*


Start
Load Address To PC
Load PC Contents To MAR
Update PC To Next Address
Load Data Required To MDR
MAR Contents To IR
Decode IR Contents
Jump?
Yes
Set PC To Value From Jump Instruction
No
Execute Instruction
Interrupt?
No
Yes
Service Interrupt

Addressing Modes

- The different ways in which the location of an operand is specified in an instruction are referred as ***addressing modes***.
- An instruction may have more than one address field and each field may have it's own addressing mode.

- ***Goal:***
  1. It offers the programmers various facilities such as counters for loop control, pointers to memory & indexing of data thereby enabling them to write more efficient program.
  2. It tends to decrease the no. of bits in the address fields of the instruction.



1 Immediate value (in the instruction)
2 Direct register reference
3 Indirect through a register
4 Direct memory reference
5 Indirect memory reference

Different Types of Addressing Modes

1. Implied Addressing Mode
2. Immediate Addressing Mode
3. Direct Addressing Mode
4. Indirect Addressing Mode
5. Register Addressing Mode (Direct)
6. Register Indirect Addressing Mode
7. Auto Increment/Decrement Addressing Mode
8. Relative Addressing Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode

Displacement Addressing

1. <u>Implied Addressing Mode</u>

- Operands are implicitly specified in the instruction.
- Also known as ***Implicit or Inherent*** addressing mode.
- All the register reference instructions that use an accumulator are implied mode instructions.
- 0-address instructions for stack-organized computer.

e.g. Rotate Accumulator (ROR/ROL)

ADD, SUB, MUL (for 0- stack organization)

2. **Immediate Addressing Mode**

- The actual operand (rather than address) is specified in the instruction.
- No memory reference to fetch data.
- Useful when the registers are to be initialized with some constant value.
- Execution of instruction is faster because the operand is available as soon as the instruction is fetched.
- However, the value of the operand is restricted by the length of the operand field (limited range).

e.g. ADD 5

Add 5 to contents of accumulator

5 is operand

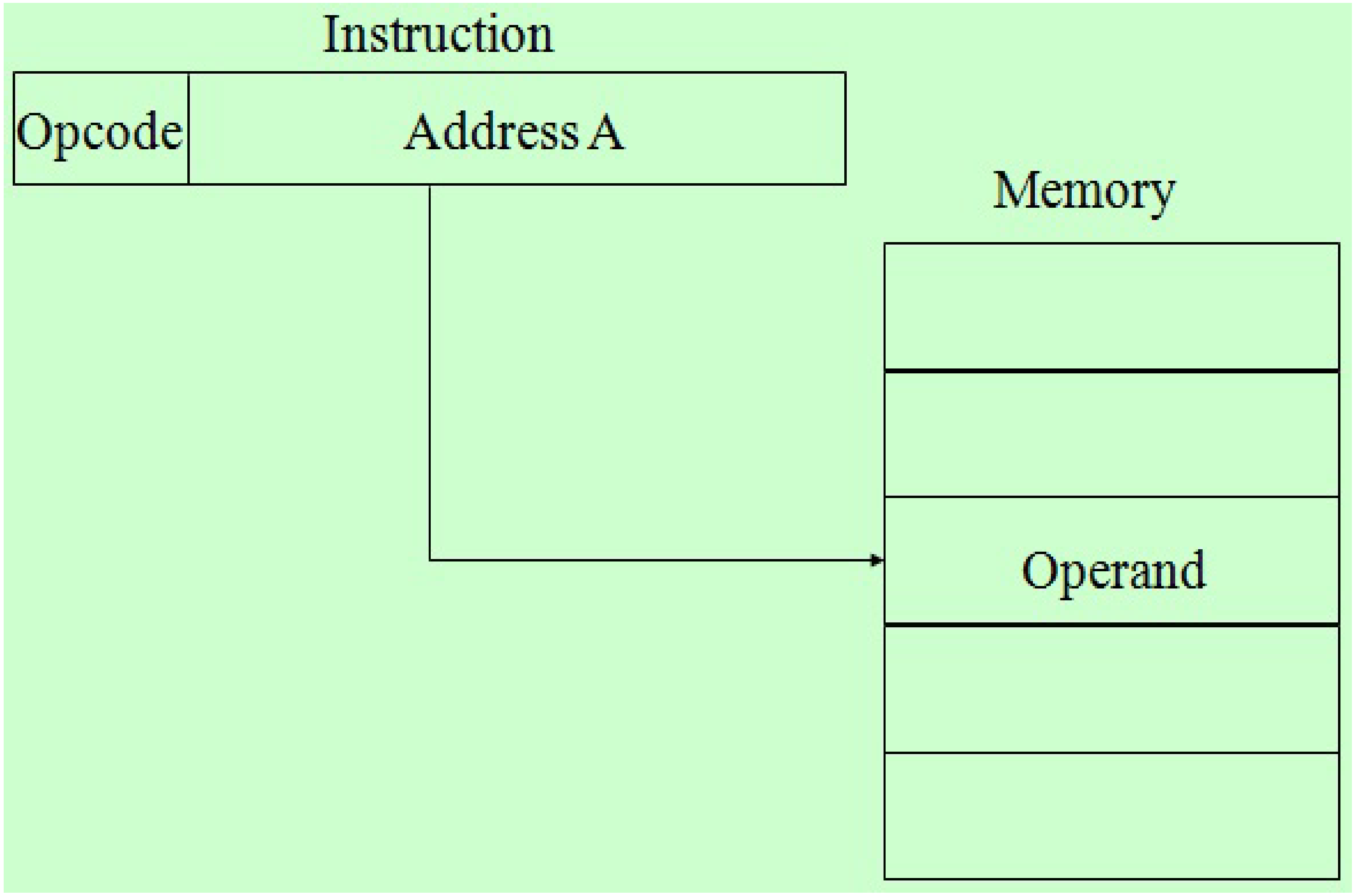
3. <u>Direct Addressing Mode</u>

- Address field contains address of operand
- Effective address (EA) = address field (A)

  e.g. ADD A

  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space
- Also known as ***Absolute addressing*** mode.

Cntd…

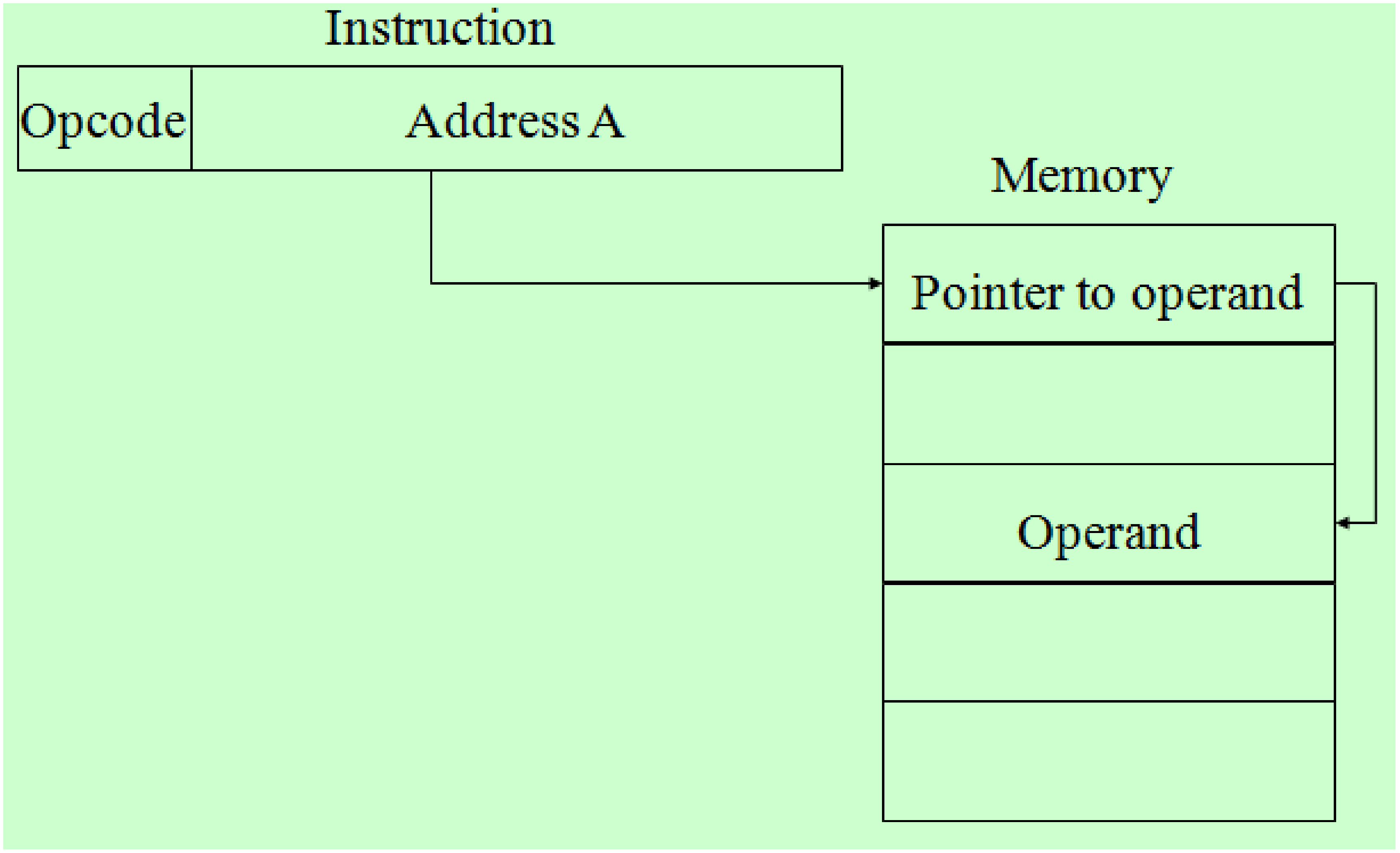


4. **Indirect Addressing Mode**

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA = (A)
  - Look in A, find address (A) and look there for operand

  e.g. ADD (A) => AC $\longleftarrow$ AC + M[M[A]]
  - Add contents of cell pointed to by contents of A to accumulator
- Large address space => 2^n where n = word length
- Helpful to implement the concept of pointers.
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
- Multiple memory accesses to find operand => slower

Cntd…

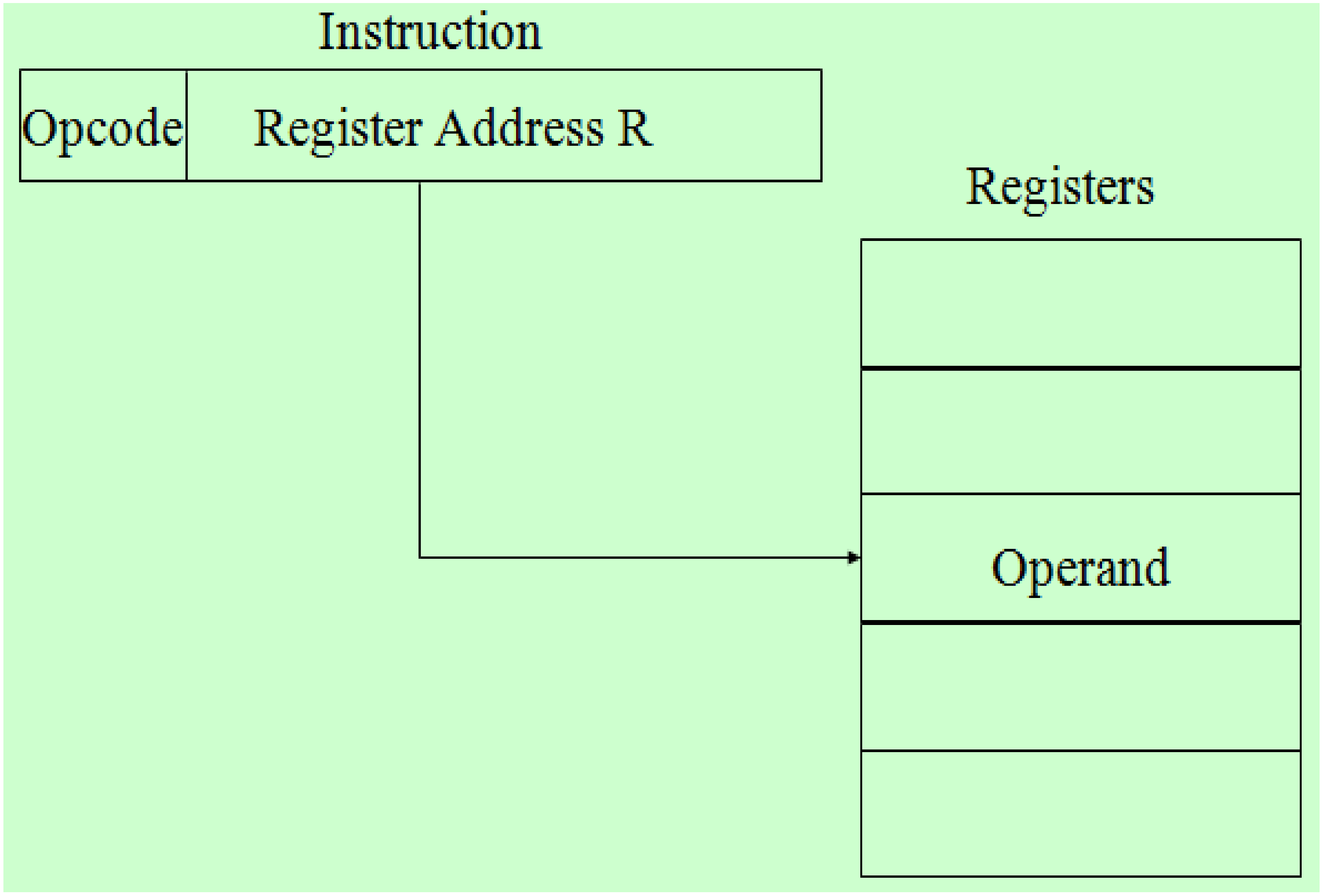

Instruction
Opcode
Address A
Memory
Pointer to operand
Operand

5. Register Addressing Mode (Direct)

- Address field refers to a processor register that contains the operand itself.
- EA = R e.g   ADD   R1,R2
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing
  - ***N.B. :*** C programming

    register int a;

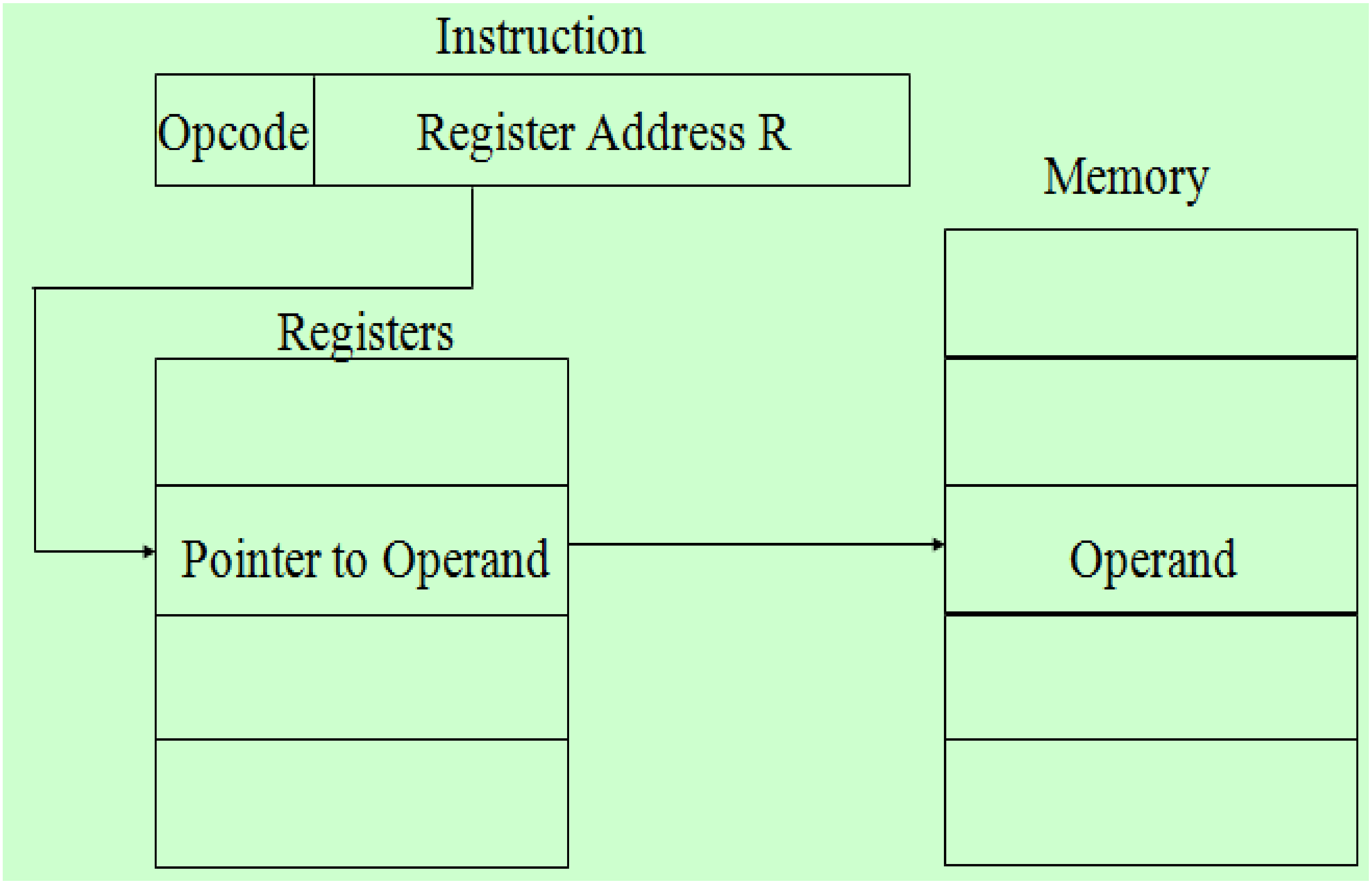
Cntd…



6. Register Indirect Addressing Mode

- The address field refers to a processor register that contains the memory location of the operand instead of the actual operand.
- EA = (R)   e.g. LD   (R1) => AC $\longleftarrow$ M[R1]
- Large address space (2^n)
- Register acts as MAR
- Before using this mode, it must be ensured that the memory address of the operand has already been stored in the register.
- Less no. of bits are to be used in the address field for specifying a register as compared to that of specifying a memory address.

Cntd…


Instruction
Opcode
Register Address R
Memory
Registers
Pointer to Operand
Operand

7. Auto Increment/Decrement Addressing Mode

- Similar to the register indirect mode with the addition that the content of register is automatically incremented after or decremented before its value is used to access the memory.
- Used to point to the next or previous item in a list.
- Auto-increment => (R)+

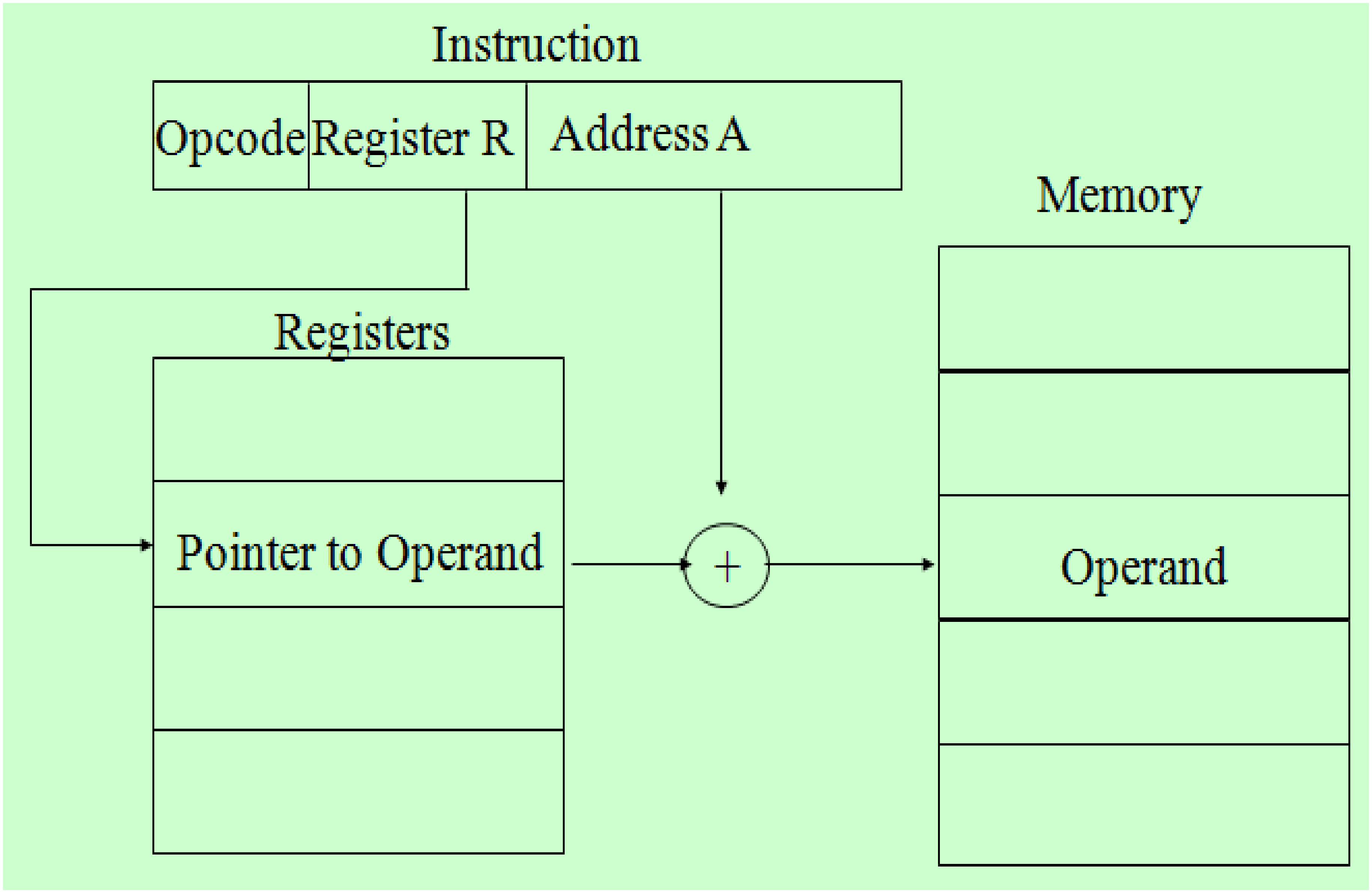
  Auto-decrement => -(R)

```
e.g.     LOOP        ADD     (R2)+ , R0
                     SUB     -(R1) , R0
                     BR>0     LOOP
```

Displacement Addressing

- EA = A + (R)
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa

Cntd…


Instruction
Opcode
Register R
Address A
Memory
Registers
Pointer to Operand
+
Operand

8. Relative Addressing Mode

- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- The address part of the instruction is usually a signed number.

9. Indexed Addressing Mode

- A = Index register
- R = displacement
- EA = A + R
- Good for accessing arrays
  - EA = A + R
  - R++

10. Base-Register Addressing Mode

- A holds displacement
- R holds pointer to base address
- R may be explicit or implicit
- EA = A+R
- e.g. segment registers in 80x86

Q+A

1. The operation executed on data stored in registers is called ……

   a) Macro-operation b) Micro-operation

   c) Bit operation d) Byte operation

2. A common bus system can be constructed using……….

   a) Multiplexers b) De-multiplexers

   c) Encoders d) Decoder

3. The ………… part of instruction code specifies the operation to be performed.

   a) Mode b) op-code c) operand d) none of these

4. Which register holds the instruction read from memory ?

   a) AC b) PC c) IR d) TR

5. Which kind of instructions are used to change the sequence of the currently executing program ?

   a) Arithmetic b) logic c) shift d) program control

6. The read phase of the instruction cycle is not required to be performed in case of..

   a) Memory-reference b) Register-reference

   c) Input/output-reference d) both b & c

7. The Load instruction is mostly used to designate a transfer from memory to …..
   a) AC b) PC c) IR d) MAR
8. In a stack-organized computer, there is a support for……….
   a) PUSH & POP instructions only b) 0-address instructions only
   c) PUSH,POP & 0-address instructions d) none of these
9. ***LD A*** is a …….. Address instruction.
   a) 0 b) 1 c) 2 d) 3
10. ***BR & JMP*** are ……….. Instructions.
   a) Arithmetic b) logic c) shift d) program control
11. An operand may be …………
   a) address b) data c) both a & b d) none of these
12. Data transfer is done between ……
   a) Between memory & processor registers
   b) Between processor registers & input or output
   c) Between processor registers themselves
   d) all of these
13. PUSH & POP are ………… instructions
   a) data transfer b) data manipulation
   c) Program control d) Input/output

14. ***CMP & CLR*** are ………… instructions

a) data transfer b) data manipulation

c) Program control d) Input/output

15. ……… only uses simple load/store instructions for memory access.

a) RISC b) CISC c) both a & b d) none of these

16. ………. supports simple fixed-length instruction format.

a) RISC b) CISC c) both a & b d) none of these

17. ………. provides smaller & faster programs.

a) RISC b) CISC c) both a & b d) none of these

18. All instructions are present in memory for ………..

a) RISC b) CISC c) both a & b d) none of these

19. All operations are performed in registers for …………..

a) RISC b) CISC c) both a & b d) none of these

20. ……….. Requires a large no. of registers.

a) RISC b) CISC c) both a & b d) none of these

21. External interrupts are ………. With the program.

a) synchronous b) asynchronous

c) can’t be determined d) none of these

22. Internal interrupts are ……….. of the program.

a) Dependent b) Independent

c) can’t be determined d) none of these

23. Request from I/O device for the transfer of data can be considered as…..

a) External Interrupt b) Internal Interrupt

c) can’t be determined d) none of these

24. Stack overflow can be considered as …….

a) External Interrupt b) Internal Interrupt

c) can’t be determined d) none of these

25. Which of the following is not a characteristic of RISC ?

a) one instruction per clock cycle b) large instruction set

c) Simple addressing modes d) Register-to-register operations

26. Addressing modes are used for ………
   a) For giving versatility to the programmer
   b) For reducing the no. of bits in an instruction
   c) For specifying the rules for interpreting & altering the address field of the instruction.
   d) All of these

27. ***PUSH B*** implies ………. Addressing mode
   a) Immediate b) Direct
   c) Indirect d) Implied

28. ***ADD, SUB, MUL*** (for 0- stack organization) implies ………. Addressing mode .
   a) Immediate b) Direct
   c) Indirect d) Implied

29. ***ADD 5*** implies ………. Addressing mode
   a) Immediate b) Direct
   c) Indirect d) Implied

30. $\boldsymbol{AC \longleftarrow AC + M[M[A]]}$ implies ……. addressing mode.

a) Immediate b) Direct

c) Indirect d) Implied

31. ***ADD R1,R2*** implies ……. addressing mode.

a) Register Direct b) Direct

c) Register Indirect d) Indirect

32. ***LD (R1)*** implies ……. addressing mode.

a) Register Direct b) Direct

c) Register Indirect d) Indirect

33. $\boldsymbol{EA = A + (PC)}$ implies ……. addressing mode.

a) Relative b) Indexed

c) Base-register d) none of these

34. For accessing array, ………. Addressing mode is useful.

a) Relative b) Indexed

c) Base-register d) none of these

References

1. Computer Organization – Carl Hamacher
2. Computer System Architecture – Morris Mano
3. Computer Organization & Architecture – T. K. Ghosh
4. Wikipedia

Thank You