

Instruction Format

- An instruction consists of two parts:-
 - a) Operation Code or **Opcode**
 - Specifies the operation to be performed by the instruction.
 - Various categories of instructions: data transfer, arithmetic and logical, control, I/O and special machine control.
 - b) **Operand(s)**
 - Specifies the source(s) and destination of the operation.
 - Source operand can be specified by an immediate data, by naming a register, or specifying the address of memory.
 - Destination can be specified by a register or memory address.

ADD R1, R2.

$R1 = R1 + R2.$

ADD LOCA, R1

MOVE R1, R2. / $LOCA \leftarrow [LOCA] \leftarrow R1$

$R1 \leftarrow R2$

BRANCH 0x4A10

ADD R1, LOCA.

Instruction Formats

- Three-Address Instructions
 - ADD R1, R2, R3 $R1 \leftarrow R2 + R3$
- Two-Address Instructions
 - ADD R1, R2 $R1 \leftarrow R1 + R2$
- One-Address Instructions
 - ADD M $AC \leftarrow AC + M[AR]$
- Zero-Address Instructions
 - ADD $TOS \leftarrow TOS + (TOS - 1)$



Addressing Modes

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

Name	Assemble r	Addressin g	function
Immediate	#Value	O	erand = Value
Register	R_i	PE	= R_i
Absolute (Direct)	LOC	AE	= LOC
Indirect	(R_i) (LOC)	AE A	= $[R_i]$ = $[\text{LOC}]$
Index	$X(R_i)$	AE	= $[R_i] + X$
Base with index	(R_i, R_j)	AE	= $[R_i] + [R_j]$
Base with index and offset	$X(R_i, R_j)$	AE A	= $[R_i] + [R_j] + X$
Relative	$X(\text{PC})$	E	= $[\text{PC} + X]$
Autoincrement	(R_i) +	AE A	= $[R_i]$; Increment R_i
Autodecrement	$-(R_i)$	E A	Decrement R_i ; = $[R_i]$

Immediate Addressing

- The operand is part of the instruction itself.
 - No memory reference is required to access the operand.
 - Fast but limited range (because a limited number of bits are provided to specify the immediate data).
- Examples:
 - `ADD #25` `// ACC = ACC + 25`
 - `ADDI R1,R2,42` `// R1 = R2 + 42`

opcode	immediate data
--------	----------------

Direct Addressing

- The instruction contains a field that holds the memory address of the operand.



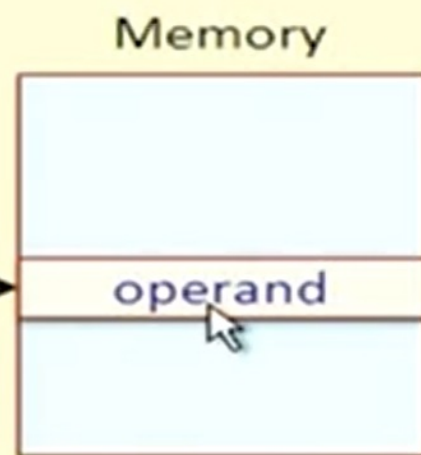
- Examples:
 - `ADD R1,20A6H` `// R1 = R1 + Mem[20A6]`
- Single memory access is required to access the operand.
 - No additional calculations required to determine the operand address.
 - Limited address space (as number of bits is limited, say, 16 bits).

Direct Addressing

- The instruction contains a field that holds the memory address of the operand.



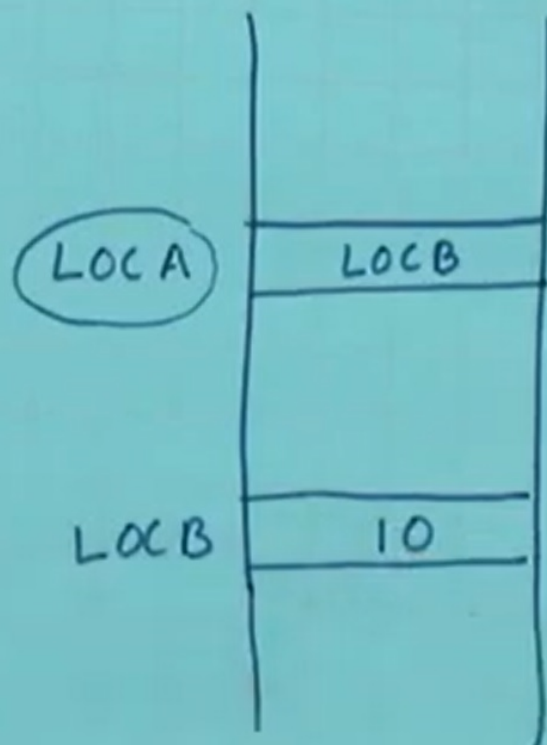
- Examples:
 - `ADD R1,20A6H` `// R1 = R1 + Mem[20A6]`
- Single memory access is required to access the operand.
 - No additional calculations required to determine the operand address.
 - Limited address space (as number of bits is limited, say, 16 bits).

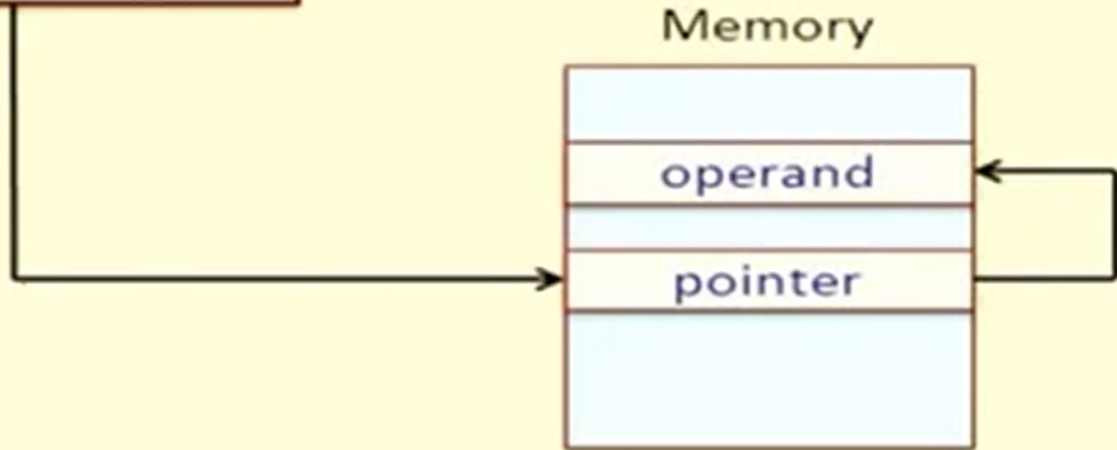


Indirect Addressing

- The instruction contains a field that holds the memory address, which in turn holds the memory address of the operand.
- Two memory accesses are required to get the operand value.
- Slower but can access large address space.
 - Not limited by the number of bits in operand address like direct addressing.
- Examples:
 - `ADD R1,(20A6H)` `// R1 = R1 + (Mem[20A6])`

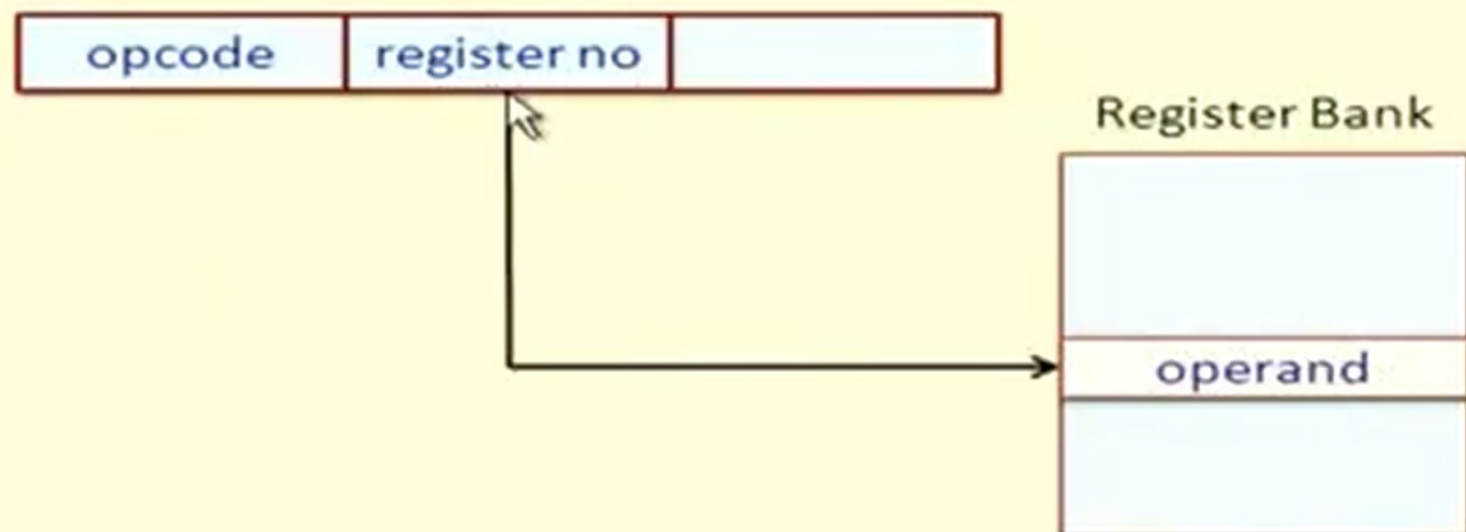
ADD RI, (LOCA)






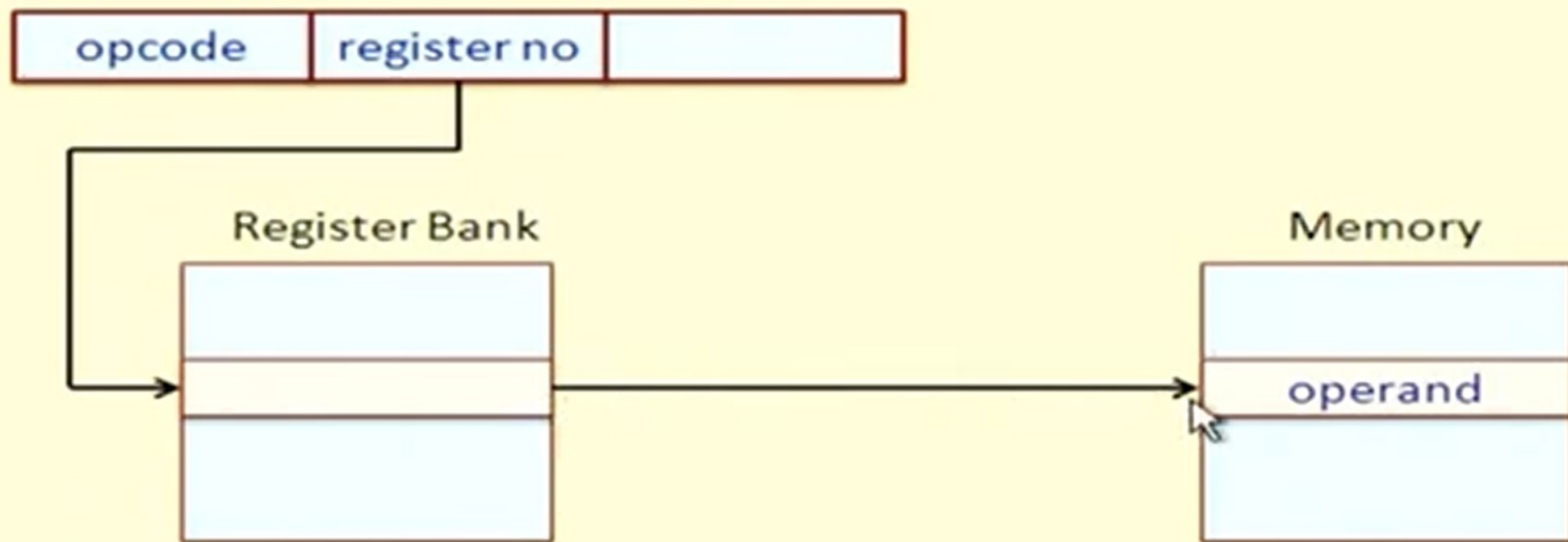
Register Addressing

- The operand is held in a register, and the instruction specifies the register number.
 - Very few number of bits needed, as the number of registers is limited.
 - Faster execution, since no memory access is required for getting the operand.
- Modern load-store architectures support large number of registers.
- Examples:
 - `ADD R1,R2,R3` `// R1 = R2 + R3`
 - `MOV R2,R5` `// R2 = R5`



Register Indirect Addressing

- The instruction specifies a register, and the register holds the memory address where the operand is stored.
 - Can access large address space.
 - One fewer memory access as compared to indirect addressing.
- Example:
 - `ADD R1,(R5)` `// PC = R1 + Mem[R5]` 



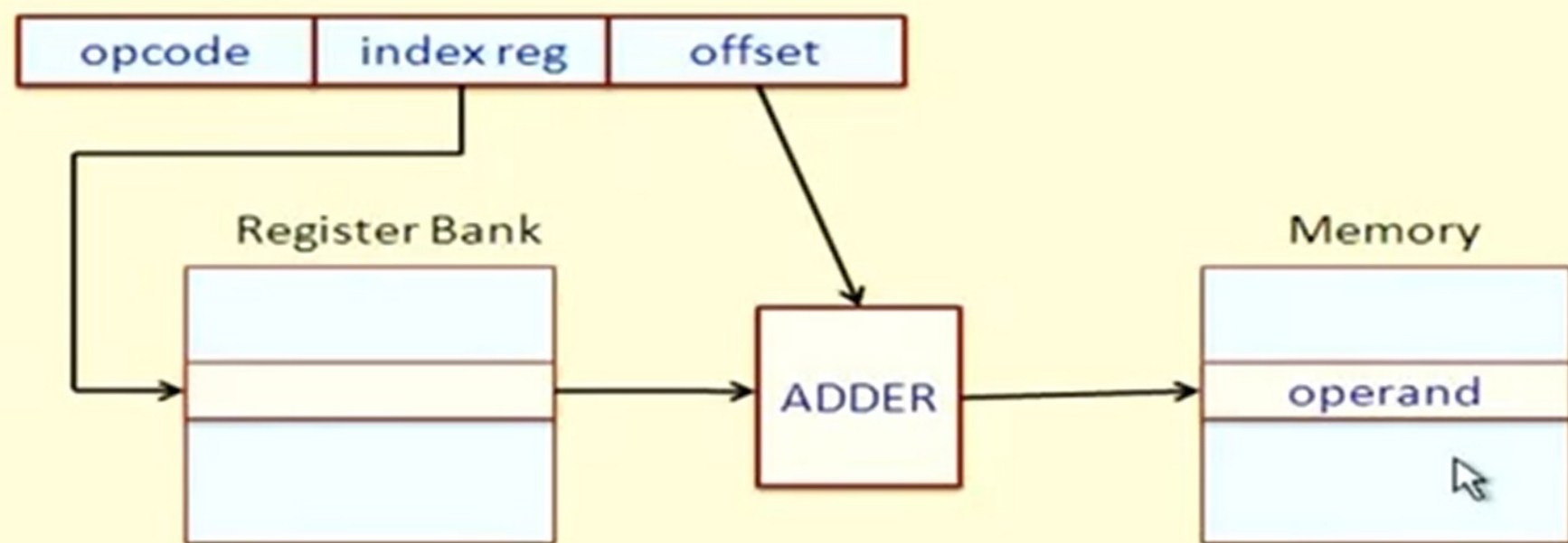
Relative Addressing (PC Relative)

- The instruction specifies an offset of displacement, which is added to the program counter (PC) to get the effective address of the operand.
 - Since the number of bits to specify the offset is limited, the range of relative addressing is also limited.
 - If a 12-bit offset is specified, it can have values ranging from -2048 to +2047.



Indexed Addressing

- Either a special-purpose register, or a general-purpose register, is used as *index register* in this addressing mode.
- The instruction specifies an offset of displacement, which is added to the index register to get the effective address of the operand.
- Example:
 - `LOAD R1,1050(R3) // R1 = Mem[1050+R3]`
- Can be used to sequentially access the elements of an array.
 - Offset gives the starting address of the array, and the index register value specifies the array element to be used.



Stack Addressing

- Operand is implicitly on top of the stack.
- Used in zero-address machines earlier.
- Examples:
 - ADD
 - PUSH X
 - POP X
- Many processors have a special register called the stack pointer (SP) that keeps track of the stack-top in memory.
 - PUSH, POP, CALL, RET instructions automatically modify SP.

Some Other Addressing Modes

- Base addressing
 - The processor has a special register called the *base register* or *segment register*.
 - All operand addresses generated are added to the base register to get the final memory address.
 - Allows easy movement of code and data in memory.
- Autoincrement and Autodecrement
 - First introduced in the PDP-11 computer system.
 - The register holding the operand address is automatically incremented or decremented after accessing the operand (like `a++` and `a--` in C).