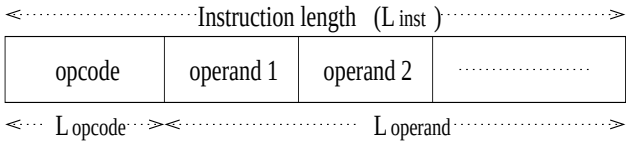
0.3.2 Operand addressing



The technique to refer an operand is known as the operand addressing mode.

0.3.2.1 Immediate addressing

In immediate addressing one or more operands are specified with in the instruction operand fields as constant.

It avoids computation of operand address and accessing of registers or memory.

Figure 2: Immediate addressing mode

If word size is smaller than the number of bits required to address an operand, then this scheme can help in reducing the instruction length. In Intel 8085,

LDI 53

transfers 53 to AC. It is a 2-byte instruction (one for LDI and other for 53).

On the other hand,

LDA X

also transfers 53 to AC if location X contains 53.

LDA X is a 3-byte instruction. 1-byte is for LDA and 2 bytes are to specify the operand's address in main memory.

0.3.2.2 Absolute or Direct addressing

It can be memory direct (Figure 3(a))/ register direct (Figure 3(b)).

No computation is required to find the effective address of operand (A_{eff}).

In memory direct addressing only one memory reference is required to get operand.

Size of operand field depends on the size of memory address space.

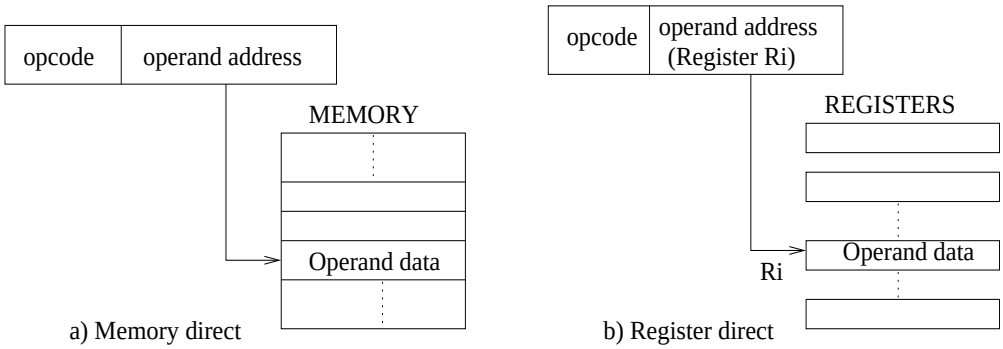


Figure 3: Direct addressing mode

Example of direct addressing -

Add X, Y	$\Rightarrow X \leftarrow X + Y$;	Memory direct addressing;	X, Y memory locations.
Add R_1, R_2	$\Rightarrow R_1 \leftarrow R_1 + R_2$;	Register direct addressing;	R_1, R_2 registers.

0.3.2.3 Indirect addressing

The address specified in the operand field directs the address of operand.

If operand field is X, the effective address of operand is

$$A_{eff} = [X].$$

X can be a memory location or a register.

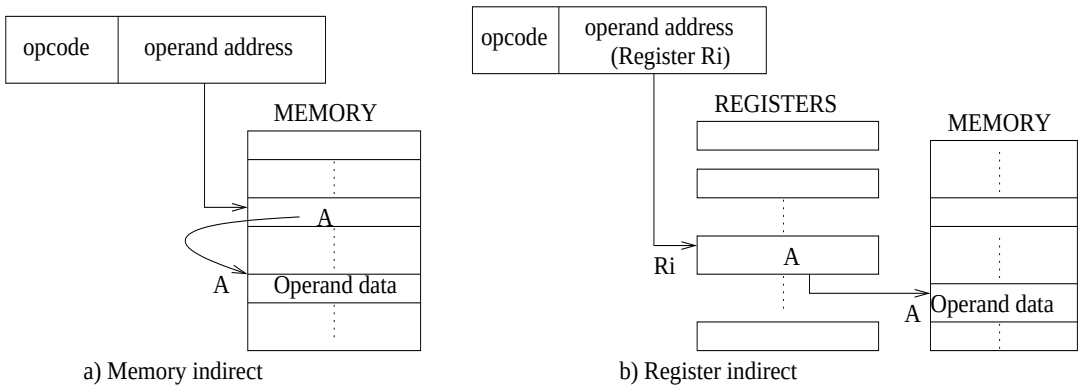


Figure 4: Indirect addressing mode

There can be memory indirect (Figure 4(a)) or register indirect (Figure 4(b)).

Example: memory indirect -

$$\text{ADD I} \quad X \Rightarrow AC \leftarrow AC + [[X]] \Rightarrow \leftarrow AC + [A].$$

For register direct or inderect, the size of address field is very small.

Indirect addressing introduces delay in operand fetching.

0.3.2.4 Relative addressing

In relative addressing, operand field specifies displacement of actual operand from the current instruction in execution (Figure 5) or PC content.

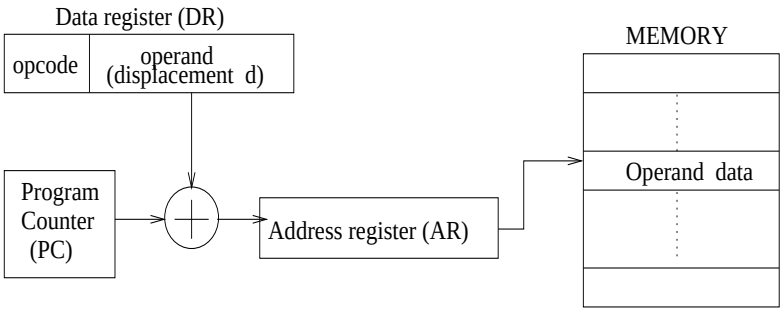


Figure 5: Relative addressing mode

Example:

Store Rel d

Stores content of accumulator (AC) to memory location L = PC + d. That is,

A_{eff} = PC + displacement (d).

It signifies that a part of operand address is implicit in relative addressing.

Relative addressing avoids relocation of operand addresses while loading a program in memory.

In a program if most of the memory references are close to instruction in execution, then relative addressing can be very effective in reducing the operand field size.

Example: Let CPU is with 16 address lines and program size is less than 1K word.

Then in relative addressing, operand field d=10-bit. In direct addressing, it is 16-bit.

0.3.2.5 Page addressing

In this mode, only a part (say YY) of operand address is specified in operand field.

Operand address is computed considering content (XX) of page register and YY.

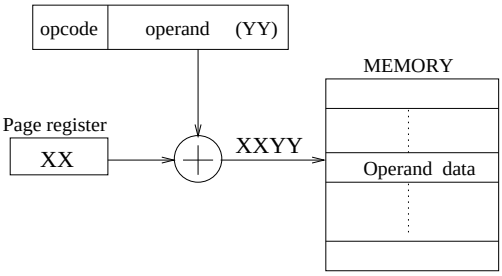


Figure 6: Page addressing

Page register content is modified by OS when switched to different page of memory.

0.3.3 Base addressing

Operand field specifies displacement of operand relative to content of a base register specified in instruction.

Base register reference can also be implicit (not mentioned explicitlt in instruction).

There can be a number of base registers in a computer.

Base register is loaded with a value when program is loaded for execution.

It can be effective one when there is locality of memory references.

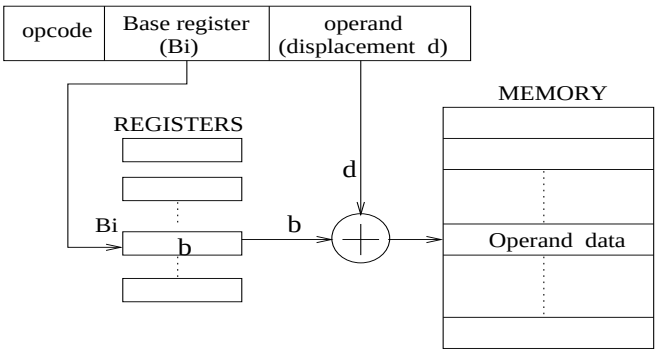


Figure 7: Base addressing

0.3.4 Indexed addressing

In this addressing, operand field of an instruction specifies a memory address A and an index register IX that contains displacement of operand from A. That is,

$$A_{eff} = A + [IX].$$

Example: Load R_d, 200(I_x) $\Rightarrow$ A_{eff} = 200 + 20 = 220 if content of I_x is 20.

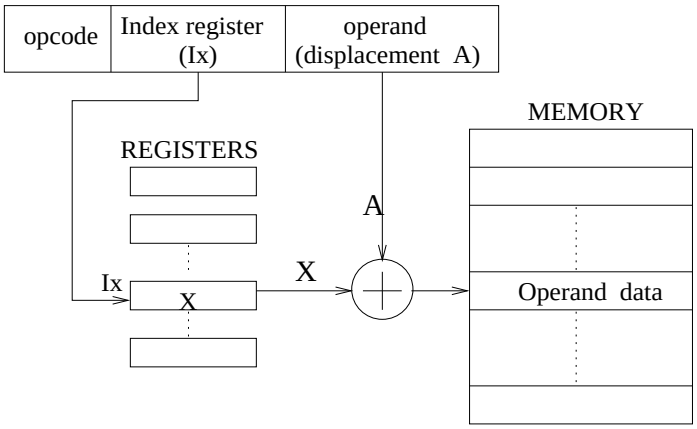


Figure 8: Indexed addressing

Number of bits required for the operand field is comparatively more.

It is effective when we need to perform iterative operations.

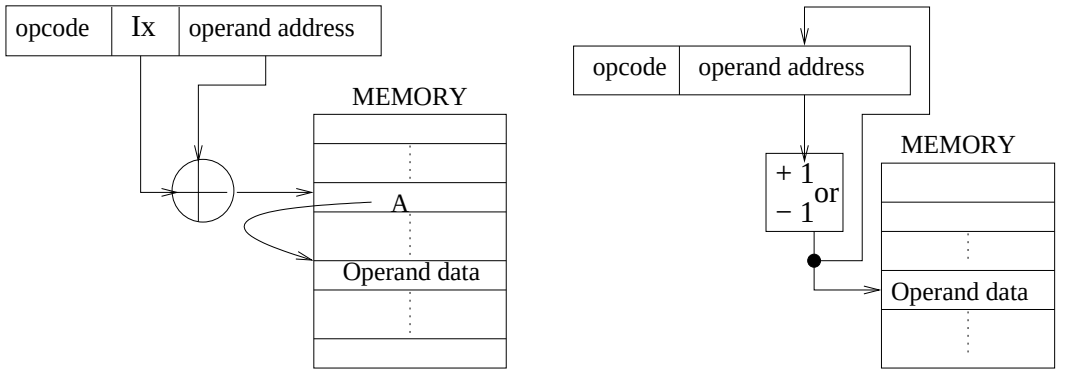
Example: In a program if we need to execute the same operation I iteratively on different operands at A, A+1, A+2, $\cdots$, then with the help of indexing this can be realized in a simpler way. In indexing,

a) The I's operand field is set to A,

b) The index register (IX) chosen is initialized to 0,

c) After each I operation, the IX is incremented by 1.

Some systems realize *autoindexing* to increment/decrement the index register.



a) Indexing with memory indirect b) Auto−increment/decrement memory direct

Figure 9: Compound addressing

0.3.5 Compound addressing

There are systems where one or more addressing modes are mixed.

Two examples are in Figure 9.

The x86 allows a variety of addressing modes such as

Immediate addressing, direct addressing, relative addressing, base addressing, a more sophisticated form of index addressing, base with index and displacement etc.

RISCs allow simple/straightforward addressing modes in comparison to CISC.