ADDRESSING MODES OF COMPUTER - I

In general, a program operates on data that reside in the computer's memory. These data can be organized in a variety of ways. If we want to keep track of students' names, we can write them in a list. Programmers use organizations called data structures to represent the data used in computations. These include lists, linked lists, arrays, queues, and so on.

Programs are normally written in a high-level language, which enables the programmer to use constants, local and global variables, pointers, and arrays. The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	# Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri)	EA = [Ri]
	(LOC)	EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri, Rj)	EA = [Ri] + [Rj]
Base with index and offset	X (Ri, Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
Autodecrement	-(Ri)	Decrement Ri; EA = [Ri]

EA = effective address
Value = a signed number

IMPLEMENTATION OF VARIABLE AND CONSTANTS:-

Variables and constants are the simplest data types and are found in almost every computer program. In assembly language, a variable is represented by allocating a register or memory location to hold its value. Thus, the value can be changed as needed using appropriate instructions.

Register mode - The operand is the contents of a processor register; the name (address) of the register is given in the instruction.

Absolute mode – The operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called Direct).

The instruction

Move LOC, R2

Processor registers are used as temporary storage locations where the data is a register are accessed using the Register mode. The Absolute mode can represent global variables in a program. A declaration such as

Integer A, B;

Immediate mode – The operand is given explicitly in the instruction.
For example, the instruction

Move $200_{immediate}$, R0

Places the value 200 in register R0. Clearly, the Immediate mode is only used to specify the value of a source operand. Using a subscript to denote the Immediate mode is not appropriate in assembly languages. A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand. Hence, we write the instruction above in the form

Move #200, R0

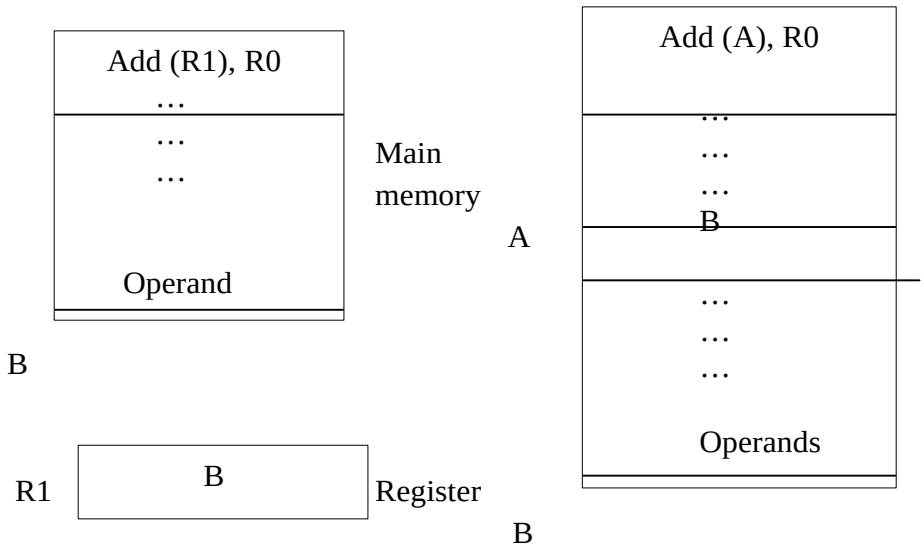
INDIRECTION AND POINTERS:-

In the addressing modes that follow, the instruction does not give the operand or its address explicitly, Instead, it provides information from which the memory address of the operand can be determined. We refer to this address as the effective address (EA) of the operand.

Indirect mode – The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

To execute the Add instruction in fig (a), the processor uses the value which is in register R1, as the effective address of the operand. It requests a read operation from the memory to read the contents of location B. the value read is the desired operand, which the processor adds to the contents of register R0. Indirect addressing through a memory location is also possible as shown in fig (b). In this case, the processor first reads the contents of memory location A, then requests a second read operation using the value B as an address to obtain the operand

Fig (a) Through a general-purpose register (b) Through a memory location



Address	Contents	
	Move	N, R1
	Move	#NUM, R2
	Clear	R0
LOOP	Add	(R2), R0
	Add	#4, R2
	Decrement	R1
	Branch > 0	LOOP
	Move	R0, SUM

The register or memory location that contains the address of an operand is called a pointer. Indirection and the use of pointers are important and powerful concepts in programming.

In the program shown Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2. The initialization section of the program loads the counter value n from memory location N into R1 and uses the immediate addressing mode to place the address value NUM1, which is the address of the first number in the list, into R2. Then it clears R0 to 0. The first two instructions in the loop implement the unspecified instruction block starting at LOOP. The first time through the loop, the instruction **Add (R2), R0** fetches the operand at location NUM1 and adds it to R0. The second Add instruction adds 4 to the contents of the pointer R2, so that it will contain the address value NUM2 when the above instruction is executed in the second pass through the loop.

Where B is a pointer variable. This statement may be compiled into

```
Move    B, R1
Move    (R1), A
```

Using indirect addressing through memory, the same action can be achieved with

```
Move    (B), A
```

Indirect addressing through registers is used extensively. The above program shows the flexibility it provides. Also, when absolute addressing is not available, indirect addressing through registers makes it possible to access global variables by first loading the operand's address in a register.

Source : http://elearningatria.files.wordpress.com/2013/10/cse-iv-computer-organization-10cs46-notes.pdf