## Addressing Modes

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer register as memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction between the operand is activity referenced. Computer use addressing mode technique for the purpose of accommodating one or both of the following provisions.

1. To give programming versatility to the uses by providing such facilities as pointer to memory, counters for top control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing fields of the instruction.

Addressing Modes: The most common addressing techniques are

* + Immediate
  + Direct
  + Indirect
  + Register
  + Register Indirect
  + Displacement
  + Stack

All computer architectures provide more than one of these addressing modes. The question arises as to how the control unit can determine which addressing mode is being used in a particular instruction. Several approaches are used. Often, different opcodes will use different addressing modes. Also, one or more bits in the instruction

format can be used as a mode field. The value of the mode field determines which addressing mode is to be used.

What is the interpretation of effective address. In a system without virtual memory, the effective address will be either a main memory address or a register. In a virtual memory system, the effective address is a virtual address or a register. The actual mapping to a physical address is a function of the paging mechanism and is invisible to the programmer.

|  |  |  |
| --- | --- | --- |
| Opcode | Mode | Address |

## Immediate Addressing:

The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:

OPERAND = A

This mode can be used to define and use constants or set initial values of variables. The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand. The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the world length.

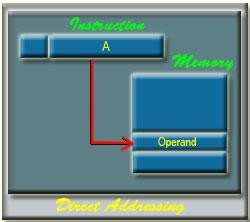


## Direct Addressing:

A very simple form of addressing is direct addressing, in which the address field contains the effective address of the operand:

EA = A

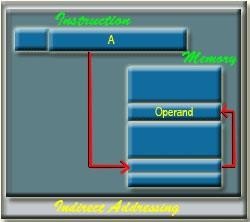
It requires only one memory reference and no special calculation.



## Indirect Addressing:

With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is known as indirect addressing:

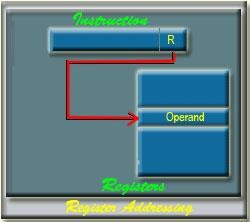
EA = (A)



## Register Addressing:

Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address: EA = R

The advantages of register addressing are that only a small address field is needed in the instruction and no memory reference is required. The disadvantage of register addressing is that the address space is very limited.



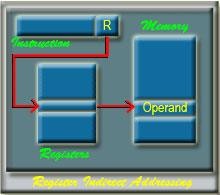
The exact register location of the operand in case of Register Addressing Mode is shown in the Figure 34.4. Here, 'R' indicates a register where the operand is present.

## Register Indirect Addressing:

Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location. It requires only one memory reference and no special calculation.

EA = (R)

Register indirect addressing uses one less memory reference than indirect addressing. Because, the first information is available in a register which is nothing but a memory address. From that memory location, we use to get the data or information. In general, register access is much more faster than the memory access.



## Displacement Addressing:

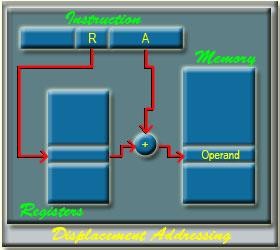
A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing, which is broadly categorized as displacement addressing:

EA = A + (R)

Displacement addressing requires that the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

The general format of Displacement Addressing is shown in the Figure 4.6. Three of the most common use of displacement addressing are:

* Relative addressing
* Base-register addressing
* Indexing



## Relative Addressing:

For relative addressing, the implicitly referenced register is the program counter (PC). That is, the current instruction address is added to the address field to produce the EA. Thus, the effective address is a displacement relative to the address of the instruction.

## Base-Register Addressing:

The reference register contains a memory address, and the address field contains a displacement from that address. The register reference may be explicit or implicit. In some implementation, a single segment/base register is employed and is used implicitly. In others, the programmer may choose a register to hold the base address of a segment, and the instruction must reference it explicitly.

## Indexing:

The address field references a main memory address, and the reference register contains a positive displacement from that address. In this case also the register reference is sometimes explicit and sometimes implicit. Generally index register are used for iterative tasks, it is typical that there is a need to increment or decrement the index register after each reference to it. Because this is such a common operation, some system will automatically do this as part of the same instruction cycle.

This is known as auto-indexing. We may get two types of auto-indexing: -one is auto-incrementing and the other one is -auto-decrementing. If certain registers are devoted exclusively to indexing, then auto-indexing can be invoked implicitly and automatically. If general purpose register are used, the auto index operation may need to be signaled by a bit in the instruction.

Auto-indexing using increment can be depicted as follows:

EA = A + (R) R = (R) + 1

Auto-indexing using decrement can be depicted as follows:

EA = A + (R) R = (R) - 1

In some machines, both indirect addressing and indexing are provided, and it is possible to employ both in the same instruction. There are two possibilities: The indexing is performed either before or after the indirection. If indexing is performed after the indirection, it is termed post indexing

EA = (A) + (R)

First, the contents of the address field are used to access a memory location containing an address. This address is then indexed by the register value. With pre indexing, the indexing is performed before the indirection: EA = ( A + (R)

An address is calculated, the calculated address contains not the operand, but the address of the operand.

## Stack Addressing:

A stack is a linear array or list of locations. It is sometimes referred to as a pushdown list or last-in-first-out queue. A stack is a reserved block of locations. Items are appended to the top of the stack so that, at any given time, the block is partially filled. Associated with the stack is a pointer whose value is the address of the top of the stack. The stack pointer is maintained in a register. Thus, references to stack locations in memory are in fact register indirect addresses. The stack mode of addressing is a form of implied addressing. The machine instructions need not include a memory reference but implicitly operate on the top of the stack.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Value addition: A Quick View** | | | | | |
| **Various Addressing Modes with Examples** | | | | | |
|  | | | | | |
|  | The most common names for addressing modes (names may differ among architectures) | | | |  |
| **Addressing modes** | **Example Instruction** | **Meaning** | **When used** |
| Register | Add R4,R3 | R4 <- R4 + R3 | When a value is in a register |
| Immediate | Add R4, #3 | R4 <- R4 + 3 | For constants |
| Displacement | Add R4, 100(R1) | R4 <- R4 + Mem[100+R1] | Accessing local variables |
| Register deffered | Add R4,(R1) | R4 <- R4 + M[R1] | Accessing using a pointer or a computed  address |
| Indexed | Add R3, (R1  + R2) | R3 <- R3 + Mem[R1+R2] | Useful in array addressing: R1 - base of array  R2 - index amount |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Direct | Add R1, (1001) | R1 <- R1 + Mem[1001] | Useful in  accessing static data |  |
| Memory deferred | Add R1, @(R3) | R1 <- R1 +  Mem[Mem[R3]] | If R3 is the address of a  pointer *p*, then mode yields *\*p* |
| Auto- increment | Add R1, (R2)+ | R1 <- R1 +Mem[R2] R2 <- R2 + *d* | Useful for stepping through arrays in a loop. R2 - start of array  *d* - size of an element |
| Auto- decrement | Add R1,- (R2) | R2 <-R2-*d*  R1 <- R1 + Mem[R2] | Same as  autoincrement. Both can also be used to  implement a stack as push and pop |
| Scaled | Add R1, 100(R2)[R3] | R1<- R1+Mem[100+R2+R3\**d*] | Used to index arrays. May be applied to any base addressing  mode in some machines. |
| Notation:  <- - assignment  Mem - the name for memory:  Mem[R1] refers to contents of memory location whose address is given by the contents of R1 | | | | | |
| **Source: Self** | | | | | |