Machine language instruction components:

In general, machine language instructions consist of

1. **opcode**: the operation to be performed
2. **operand**(s): that to which the op code applies

An operand specifies a "*target address*" to be accessed in performing the operation.

Since the bit patterns that make up the machine language instruction are not easily digestible by humans, an encoding (called **assembly language**) is employed that uses *mnemonics* to represent the opcodes and allows both symbolic and base 10 references to represent operands.

For example, in the 1-address instruction

```
LDA   21
```

"LDA" is the *mnemonic* for the opcode (load accumulator A) and "21" is the operand (an address given in base 10).

The manner of specification of the target address is called the **addressing mode** of the machine language instruction.

A program for translating assembly language programs is called an **assembler**. There is a one-to-one correspondence between assembly language instructions and the machine language instructions generated by the assembler. [warning: there are additional "assembler directives" for directing the manner in which the assembler generates machine language; do not make the mistake of assuming that there is machine language corresponding to these]

Commonly employed addressing modes: (1-address examples)

1. *direct addressing* - the target address is the value of the operand; e.g.,

   LDA 21 [*with actual machine code (in hexadecimal)* 030015]

   The effect of this instruction is to load accumulator A with the "word" stored at memory location 15_{16}.

2. *immediate addressing* - the target address is the address of the operand part of the instruction; i.e., the value being accessed is part of the instruction and so is "*immediately*" present; e.g.,

   LDA #21 [*with actual machine code* 010015]

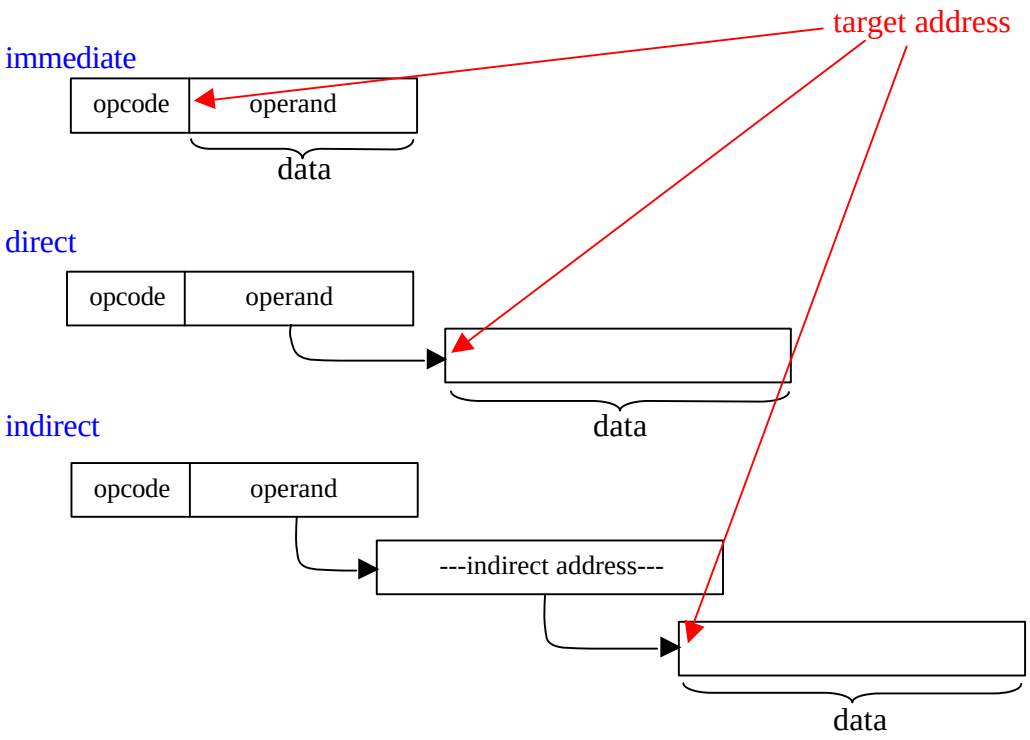
   The effect of this instruction is to load the "*immediate*" value of 21 (hex 15) into register A (no memory fetch required).

3. *indirect addressing* - the target address is the address stored at the memory location addressed by the operand; e.g.,

   LDA @21 [*with actual machine code* 020015]

   which loads into A the word whose address is stored at memory location 21.

Note:



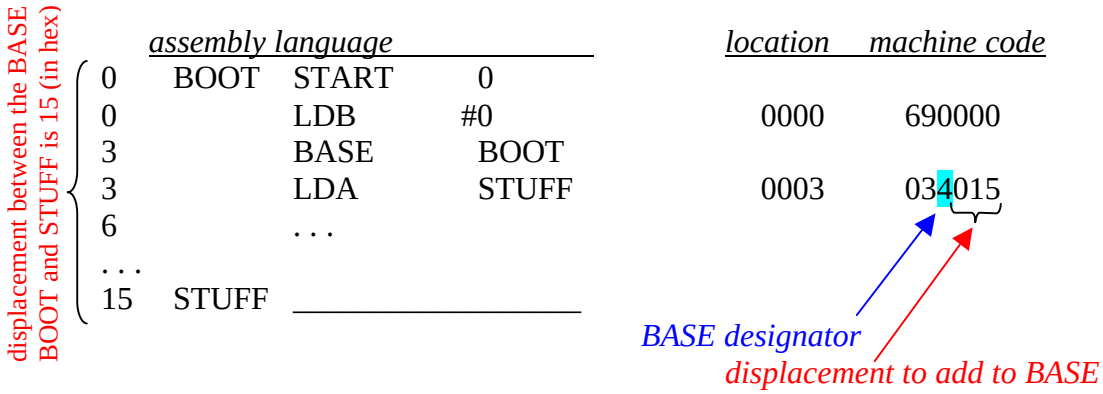
4. *base/displacement* (or base/relative) *addressing* - given that there is a designated "**base register**", the target address is derived from the computation

operand-value + (base-register)

the parentheses indicate the "contents of" the register

i.e., the operand is treated as a "*forward displacement*" off of a designated base register. The resulting value can be used as an immediate value, a direct address, or an indirect address.

For the assembler an assembler directive called a "BASE directive" is used to specify the value the assembler is to use for the base register; e.g.,



START is an assembler directive that specifies to the assembler what to use as the starting address for the module (WARNING: it is given in hex).

BASE BOOT specifies that the base register has the address of BOOT.

It is the responsibility of the programmer to see that the base register is actually loaded with the address of BOOT - this is what LDB #0 does.

The assembler may then resolve an address such as STUFF by using base-displacement addressing, so long as the STUFF is a forward reference and the value of the displacement does not exceed FFF in hex (the number of bits set aside in the instruction for the displacement). This means that

$0 \le \text{disp} \le \text{FFF}$ (in hex) or equivalently that

$0 \le \text{disp} \le 4095$ in decimal.

In the translation of LDA STUFF to 034015, note that bits have been given up to indicate in the machine language instruction that base-displacement addressing is to be used. For immediate and indirect addressing

LDA	#STUFF	014015
LDA	@STUFF	024015

only the op code is different, since the computed address can be use as either the immediate, direct, or indirect address.

5. *program counter/relative addressing* - the target address is derived from the computation

    operand-value + (PC)

    In contrast to base/displacement addressing, the operand is treated as a forward or backward displacement from the program counter. The displacement is the same 3 hex digits as for base displacement, except that it is treated as a *12-bit twos complement integer*. This means the displacement range is

    800 ≤ disp ≤ 7FF (in hex) or equivalently that
    -2048 ≤ disp ≤ 2047 in decimal.

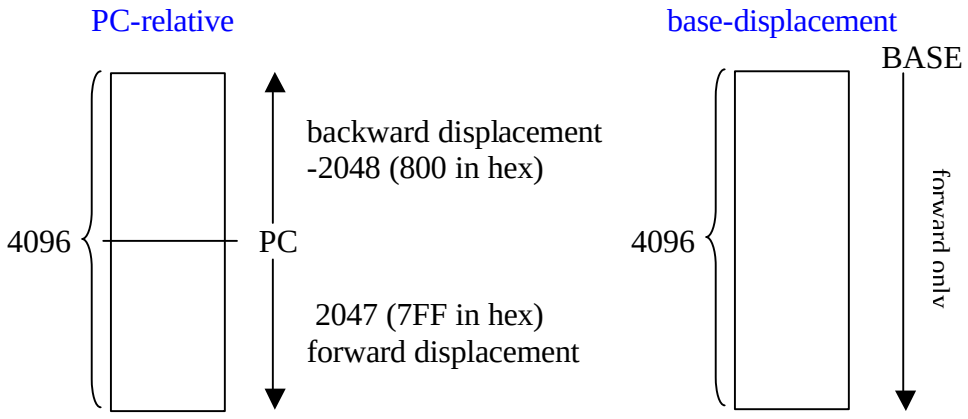
    The computed value can be used as an immediate address, a direct address, or an indirect address. In particular, LDA STUFF in the following example

displacement

	assembly language			*location*	*machine code*
0	BOOT	START	0		
3		LDB	#0	0000	690000
3		LDA	STUFF	0003	03200F
6		. . .			
. . .					
15	STUFF				

    has a different displacement than with base/displacement. When LDA STUFF is loaded, the PC advances to 6 and the displacement between the address of STUFF and the PC is 15 - 6 = F (hex).

Remark: the displacement represents a "window" of 4096 either forward from the BASE register, or centered around the PC.



6. *relocation/direct addressing* - the target address is computed as

   operand-value + hidden-base-value

   set by operating system

7. *indexed/direct addressing* - the target address is derived by adding on the contents of the "index" register in address computation. It can be used in conjunction with either base/displacement or PC/relative addressing, *but only in direct mode*; e.g.,

   LDA TABLE,X with translation 03803F

   address of TABLE

   indexed addressing designator

   A is loaded with the at the address obtained by adding the address of TABLE and the contents of the index register (X).