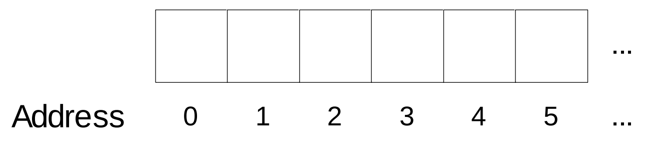
BASE DISPLACEMENT ADDRESSING

Internal memory on an IBM mainframe is organized as a sequential collection of bytes. The bytes are numbered starting with 0 as pictured below,

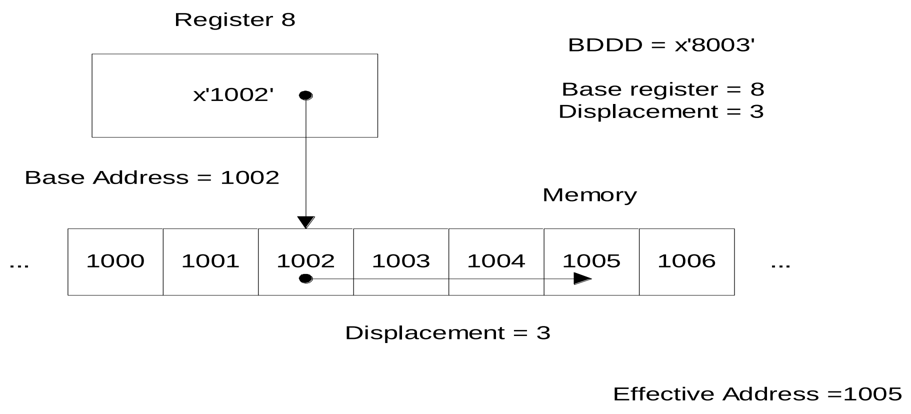


The IBM System 370 architecture allows an address to be expressed as a collection of 31 consecutive bits. The smallest address would be represented by 31 consecutive 0’s which denotes address 0. The largest address would be represented by 31 consecutive 1’s whose value is 2^{31} - 1 = 2,147,483,647. If we work with addresses in this form (lets call these direct addresses), each address occupies a fullword - 4 bytes.

Why do we need an address in the first place? Consider the following instruction,

```
MVC   COSTOUT,COSTIN
```

In order for the machine to move the contents of COSTIN to COSTOUT, it must know the locations of these two fields. The computer identifies a field by the address of the first byte in the field. When an instruction is assembled, the assembler converts any symbols like “COSTOUT” to their corresponding addresses. The assembler does not, however, generate direct addresses, but produces an address in **base** / **displacement** format. Addresses in this form consist of 4 hexadecimal digits or two bytes, BDDD, where B represents a base register and DDD represents a displacement. The smallest displacement is x’000’ = 0 and the largest displacement is x’FFF’ = 4095. Ultimately, the base / displacement address must be converted to a direct address. How does this occur? The diagram below indicates how this process takes place starting with the base/displacement address x’8003’.



Notice that the effective address, which is the direct address equivalent to the base/displacement address, is computed by adding the contents of the base register with the specified displacement.

Effective address = Contents(Base Register) + Displacement

There are two advantages of using base/displacement addresses instead of direct addresses in the object code that the assembler produces:

1) Every address is shorter. Instead of being 4 bytes long, the addresses are only 2 bytes.

2) The base/displacement addresses are correct no matter where the program is loaded in memory. Each symbol is represented by a displacement from a fixed point inside the program. If the program is relocated in memory, the displacement to a given variable does not change. The base register remains fixed as well. As a result, the base/displacement address is correct. On the other hand, if we had used direct addresses, every symbol would have a new address if the program were relocated.