Relative Addressing

As we have seen, all symbolic addresses are based on variants of the concept of base address (stored in a base register) and an offset.

Note that the offset, encoded as a 12–bit unsigned integer, is always non–negative.

The possible offset values range from 0 through 4095 inclusive.

We now introduce a way to reference a storage position relative to the symbolic address of another label. This allows direct reference to unlabeled storage.

The form of a relative address is **`LABEL+N`**, where N is the byte offset of the desired storage relative to the symbolic address associated with **`LABEL`**.

Again, note the lack of spaces in the relative address. This is important.

Consider the two data declarations.

```
F1    DC F‘0’   A four–byte full-word.
F2    DC F‘2’   Another full-word at address F1 + 4
```

Consider the following two instructions. They are identical.

```
L  R6, F2
L  R6, F1+4
```

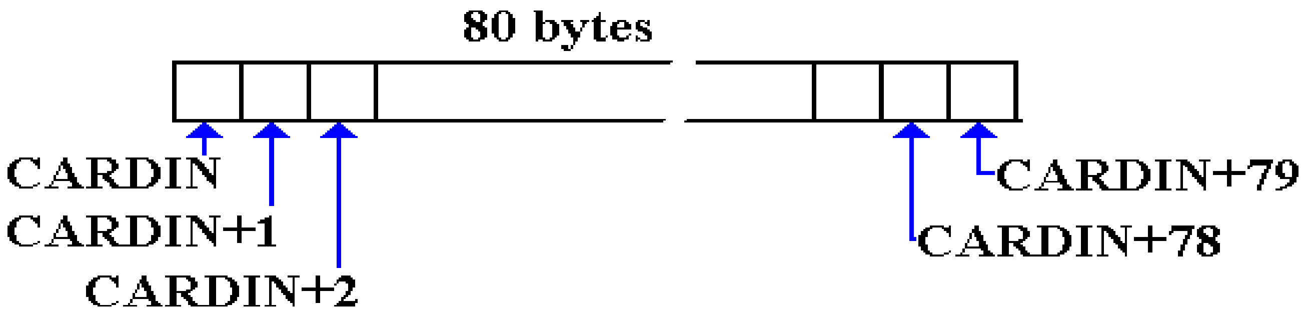
Relative Addressing: A More Common Use

The most common use of relative addressing is to access an unlabeled section of a multi–byte storage area associated with a symbolic address.

Consider the following very common declaration for card data. It sets aside a storage of 80 bytes to receive the 80 characters associated with standard card input.

```
CARDIN  DS CL80
```

While only the first byte (at offset 0 from **`CARDIN`**) is directly named, we may use relative addressing to access any byte directly. Consider this figure.



The second byte of input it is at address **`CARDIN+1`**, the third at **`CARDIN+2`**, etc.

Remember that the byte at address **`CARDIN+N`** is the character in column (N + 1) of the card. Punched cards do not have a column 0.