TR D1(L1,B1),D2(B2)

Translate

SS_1

Op Code	LL_1	B_1D_1	D_1D_1	B_2D_2	D_2D_2

TR is used to translate one string of characters to another string of characters based on a translate table which defines the substitutions which should occur. For instance, **TR** could be used to convert a string of characters from lower case to upper case, or from ASCII to EBCDIC.

Operand 1 designates a string contain in memory which is to be translated. Operand 2 designates the translate table. This table is also called a "table of functions". **TR** translates one byte at a time, starting with the leftmost byte of Operand 1, proceding from left to right. Each byte that is translated is used as a displacement into the table. The "function byte" that is found in the table is substituted for the byte in Operand 1. Since a byte can contain any value from X'00' to X'FF', the table of functions is usually 256 bytes long to accommodate the range of addresses from table + X'00' to table + X'FF'.

Let us consider how a particular byte is translated using a table of functions called "TABLE". For example, how would a byte containing X'A2' be translated? Since X'A2' = 162 in base 10, X'A2' would be replaced by the byte at address "TABLE+162". In other words, the byte we are translating acts as a displacement into the table, and is replace by the table byte in that position.

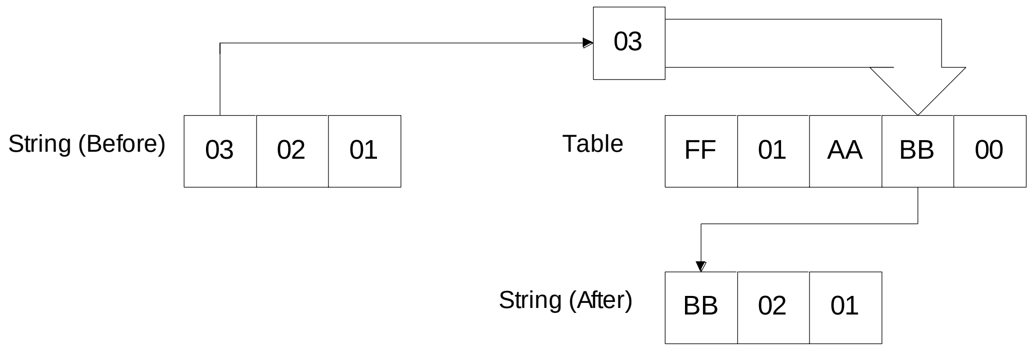
Consider the following example. To keep things simple, we begin with a small (5 bytes) table and assume that the bytes in the string which is being translated generate displacements that stay inside the table. (For safety, most **TR** tables are 256 bytes in length.) We assume the following definitions.

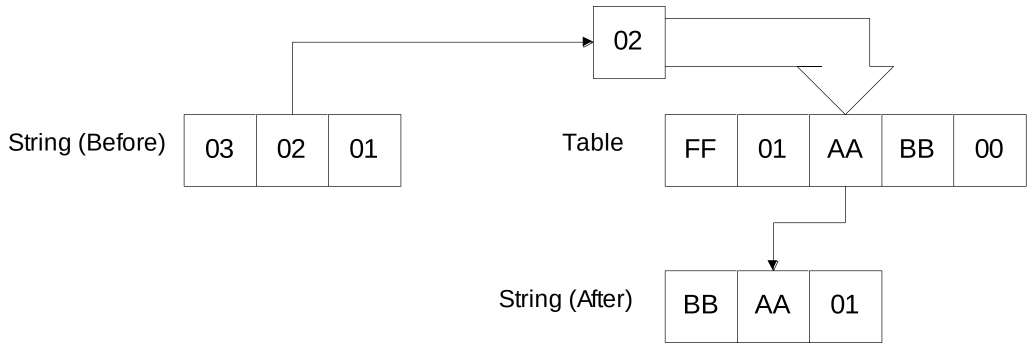
```
STRING    DC   X'030201'
TABLE     DC   X'FF01AABB00'
```

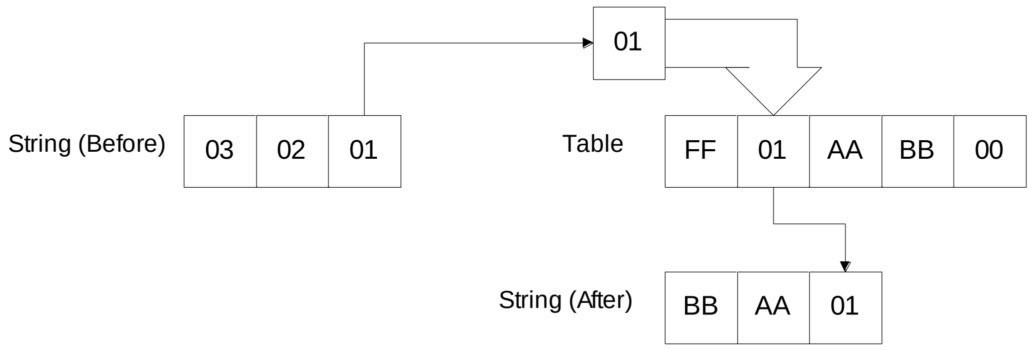
We issue the following command.

```
          TR   STRING,TABLE
```

The execution of this instruction is graphically illustrated on the next page. Let us consider a byte by byte description of the execution of **TR**. The first byte of STRING, X'03', is used as a displacement into TABLE. The byte at address TABLE+3, which is X'BB', replaces the first byte in STRING. This means the string temporarily looks like X'BB0200'. Next, the second byte of STRING, X'02', is used as a displacement into the table. As a result the byte at TABLE+2, which is X'AA', replaces X'02' in STRING. Finally, the third byte in STRING, X'01', is used as a displacement into the table and the byte at TABLE+1, X'01', replaces X'01' in STRING. After the TR has executed, STRING contains X'BBAA01'.







To be safe when using **TR** you should always code a translate table that is 256 bytes long so that you can handle any possible displacement that might be used in the translation. The only exception to this rule occurs when you are sure that the bytes you are translating fall within a specified range.

How could you define a translate table that translates every character to itself? Since each character would be used as a displacement to itself, a translate table of this type would look like X'000102030405060708090A0B0C0D0E0F10...'. There is an easy way to construct such a table:

```
TABLE    DC    256AL1(*-TABLE)
```

This technique works by defining 256 byte address constants of length 1. When defining the first byte in the table the asterisk refers to the same address as TABLE, and so *-TABLE is 0. After defining the first byte the location counter advances and *-TABLE is 1 when defining the second byte. Continuing in this fashion, the expression "*-TABLE" takes on all the values from 0 to 256. As a result, we create a table that translates every character to itself. This is important because in many applications, most of the characters we encounter are left unchanged by **TR**.

For example, suppose we want to translate lower case letters to upper case. To do this we use the following table definition.

```
TABLE     DC    256AL1(*-TABLE)
          ORG   TABLE+C'a'
          DC    C'ABCDEFGHI'
          ORG   TABLE+C'j'
          DC    C'JKLMNOPQR'
          ORG   TABLE+C's'
          DC    C'STUVWXYZ'
          ORG
```

Notice that most bytes translate to themselves. An originate directive is used to position to the table position that contains a lower case “a”. This is followed by a DC that defines the upper case letters A - I. This technique is repeated for “j” and “s”. Since the lower case letters occur in 3 blocks (A - I, J - R, and S - Z) the DC’s above provide for the conversion from lower to upper case.

Examples

Some Unrelated TR’s:

```
TABLE     DC    X'0302010300'    VERY SMALL TRANSLATE TABLE
A         DC    X'040404'
B         DC    X'000003020201'
C         DC    X'00000000'
D         DC    X'01010101'
E         DC    X'020202'
F         DC    X'010203'
                ...               Result:
          TR    A,TABLE          A = X'000000'
          TR    B,TABLE          B = X'03030301010102'
          TR    C,TABLE          C = X'03030303'
          TR    D,TABLE          D = X'02020202'
          TR    E,TABLE          E = X'010101'
          TR    F,TABLE          F = X'020103'
```

Tips

1. Start with a table that translates each byte to itself (TABLE DC 256AL1(*-TABLE)). Use “ORG” to position yourself to an appropriate character (ORG TABLE+C’X’ positions you at ‘X’.) Redefine the area with a ‘DC’.

2. Avoid translation errors by defining all translate tables with 256 bytes.