**LA** is used to initialize the register specified by operand 1 with the address designated by operand 2. The number of bits that are loaded can be 24, 31, or 64 depending on the addressing mode. Operand 2 may be expressed using explicit notation or symbolic notation, or a combination of both. Remember that each byte in memory is numbered and that the number assigned to a byte is its address. The address of a field is the address of the first byte of the field. Consider the following example,

\[ \text{LA} \ R9, \text{AFIELD} \]

<table>
<thead>
<tr>
<th>Op Code</th>
<th>( R_1 X_2 )</th>
<th>( B_2 D_2 )</th>
<th>( D_2 D_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The address of the fullword “AFIELD”, x’00001000’, is copied to register 9, destroying the previous value in R9. The fullword is unchanged by this operation. We are assuming a 31-bit addressing mode.

Since **LA** is an RX instruction, an index register may be coded as part of operand 2 as in the example below. We assume that register 6 is used as an index register and initially contains x’0000002F’. When the assembler processes the expression AFIELD(R6), it uses the symbol AFIELD to determine a base register and a displacement, leaving R6 as the index register. The address which is loaded into register 9 is the “effective address” computed by adding the base register contents, plus the index register contents, plus the displacement:

\[
\text{Effective address} = C(\text{Base register}) + C(\text{Index register}) + \text{displacement}
\]

We assume that the contents of the base register plus the displacement is x’00001000’. Then the effective address is x’00001000’ + x’0000002F’ = x’0000102F’.

\[ \text{LA} \ R9, \text{AFIELD}(R6) \]
The example above uses a mixture of symbolic and explicit addressing. The instruction could also be coded using only explicit addresses:

```
LA   R9, 30(R7,R8)
```

In the example above assume that R7 contains x'00001000' and that R8 contains x'00000020'. R7 is treated as an index register, R8 is the base register, and 30 is a displacement. The effective address is C(R7) + C(R8) + 30 = x'00001000' + x'00000020' + x'000001E' = x'0000103E' (Remember that a decimal 30 is 1E in hexadecimal.) After the instruction has executed, R9 contains x'0000103E'.

**Examples**

**Some Unrelated Load Addresses**

R4 = X'12121212'
R5 = X'00000008'
R6 = X'00000004'

Assume that AFIELD has address x'00003000'.

```
AFIELD    DC   F'4'       AFIELD = X'00000004'
```

```
LA  R4,AFIELD     R4 = X'00003000'
LA  R4,AFIELD(R6) R4 = X'00003004'
LA  R4,AFIELD(R5) R4 = X'00003008'
LA  R4,20(R5,R6)  R4 = X'00000020' 4 + 8 + 20 = 32 = X'20'
```

Using R0 as an index indicates that no index register is desired:

```
LA  R4,3(R0,R6)   R4 = X'00000007'  4 + 3 = 7
```

Consider the next two consecutively executed instructions.

```
LA  R4,AFIELD            R4 = X'00003000'
LA  R4,L'AFIELD(R0,R4)   R4 = X'00003004'
```

In the example above, the length attribute (L') is used as a displacement.

**Tips**

1. An old assembler joke:

   **Novice:** What’s the difference between a Load instruction and a Load Address instruction?

   **Old Hand:** About a week of debugging.

   Seriously, you should pay attention when coding **L** or **LA**. Both instructions compute the address of operand 2. In the case of **L**, the machine retrieves the contents of the fullword in memory at the specified address and places the four bytes in a register. In the case of **LA**, the address is simply stored in a register.
2. The **LA** instruction is often used to change the location referenced by a DSECT:

```assembly
TEST  DSECT
TESTREC DS 0CL80
X     DS   ...

USING TEST,R5
LA   R5,TABLE   POINT AT TABLE AREA
...
LA   R5,L’TESTREC(R0,R5)  MOVE THE DSECT
```