Loaders
Loader

Diagram showing the process of linking and loading programs A and B into the main memory. The loader is responsible for linking and loading both programs.
Loader

1. memory allocation
2. linking: object module symbolic reference
3. relocation: adjust address dependent locations (address constant)
4. loading: physically place the program
Compile and Go Loaders

Source program → Compile-and-Go translator →

{Compile-and-Go, Assemble-and-Go}
Advantages & Disadvantages

Simple to implement

Disadvantages

1. A portion of the memory is wasted
2. It is necessary to retranslate the program every time
3. It is very difficult to handle multiple segments, specifically when they are in different languages, so it is very difficult to produce modular programs.
General Loader Scheme

In this the loader accepts the assembled machine instructions, data & other information present in the object format & places machine instructions & data in core in an executable format.

The loader is smaller then assembler so more memory is available to the user. Now we don’t need to retranslate the program.

If all the source program translators produce compatible object programs & use compatible linkage conventions, it is possible to write subroutines in different languages.
Absolute Loaders

In this scheme the object program is placed in secondary devices. The loader only accepts the machine language text & places into core at the location prescribed by the assembler. Disadvantage is that the programmer must specify the load address in the program & also if there are multiple subroutines, the programmer must remember the address of each & use that in other subroutines to perform subroutine linkages.
Absolute Loader
Relocating Loader

In this the assembler assembles each procedure segment independently & passes onto the loader the text & information as to relocation & intersegment references. For each source program the assembler outputs a text prefixed by the transfer vector that consists of addresses containing the names of subroutines referenced by the source program. The assembler would also provide to the loader the length of the program & length of the transfer vector. The loader will load each subroutine identified in the Transfer vector. It will then place a transfer instruction to the corresponding subroutine in each entry in the TV.
A2 : 16 bits absolute addr. \(2^{16} = 64 \text{ KB}\)

- no base register
- use of "relocation bits"
  or addr. Field. bit \(\{\begin{array}{c}
= 1 : \text{relocation} \\
= 0 : \text{no relocation}
\end{array}\)

Program length = 48 bytes
Transfer vector = 8 bytes

Source program

<table>
<thead>
<tr>
<th>MAIN</th>
<th>START</th>
<th>EXTRN</th>
<th>SQRT</th>
<th>EXTRN</th>
<th>ERR</th>
<th>ST</th>
<th>14, SAVE</th>
<th>L</th>
<th>1, =F'9'</th>
<th>BAL</th>
<th>14, SQRT</th>
<th>C</th>
<th>1, =F'3'</th>
<th>BNE</th>
<th>ERR</th>
<th>L</th>
<th>14, SAVE</th>
<th>BR</th>
<th>14</th>
<th>SAVE</th>
<th>DS</th>
<th>F</th>
<th>END</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
<td>R1</td>
<td>X2</td>
<td>A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

360 RX inst.

\(\text{addr.} \quad \text{Relocation} \quad \text{Object code} \quad \text{Second halfword} \quad \text{relocation} \quad \text{addr.} \quad \text{T.V.} \quad \text{T.V.} \quad \text{T.V.} \quad \text{condition code} \quad \text{Skip for alignment} \quad \text{L.T.}\)
<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>BC</td>
<td>15,448</td>
</tr>
<tr>
<td>404</td>
<td>BC</td>
<td>15,526</td>
</tr>
<tr>
<td>408</td>
<td>ST</td>
<td>14,436</td>
</tr>
<tr>
<td>412</td>
<td>L</td>
<td>1,440</td>
</tr>
<tr>
<td>416</td>
<td>BAL</td>
<td>14,400</td>
</tr>
<tr>
<td>420</td>
<td>C</td>
<td>1,444</td>
</tr>
<tr>
<td>424</td>
<td>BC</td>
<td>7,404</td>
</tr>
<tr>
<td>428</td>
<td>L</td>
<td>4,436</td>
</tr>
<tr>
<td>432</td>
<td>BCR</td>
<td>15,14</td>
</tr>
<tr>
<td>436</td>
<td>TEMP LOC</td>
<td>1</td>
</tr>
<tr>
<td>440</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>444</td>
<td>3</td>
<td>LENGTH 48 BYTES</td>
</tr>
</tbody>
</table>

: SQRT LENGTH 78 BYTES

: 526 ERR
Pros & cons

- Relocation bits are used to solve the problem of relocation, transfer vector is used to solve the problem of linking & program length info to solve allocation.

Disadvantages
- not suited for loading external data.
- Transfer vector increases the size
- Does not facilitate access to data segments that can be shared