# RISC-V build 32-bit constants with LUI and ADDI

by Micro Admin

> *LUI (load upper immediate) is used to build 32-bit constants and uses the U-type format. LUI places the U-immediate value in the top 20 bits of the destination register rd, filling in the lowest 12 bits with zeros.*

*I found this in manual, but if I want to move 0xffffffff to a register, all the code I need is:*

```
LUI x2, 0xfffff000
ADDI x2, x2, 0xfff
```

*But a problem occurred, ADDI will extend sign to make a immediate data to a signed number, so* `0xfff` *will be extend to* `0xffffffff`.

*It make* `x2` *to* `0xfffefff` *but not* `0xffffffff`

*and what is an good implementation to move a 32bits immediate to register?*

TL;DR: The 32-bit constant you want to load into `x2` is `0xffffffff` which corresponds to *-1*. Since *-1* is in the range *[-2048, 2047]*, this constant can be loaded with a single instruction: `addi x2, zero, -1`. You can also use the `li` pseudoinstruction: `li, x2, -1` which the assembler, in turn, translates to `addi x2, zero, -1`.

## Loading a 32-bit constant with a `lui`+`addi` sequence

In general, we need a `lui+addi` sequence – two instructions – for loading a 32-bit constant into a register. The `lui` instruction encodes a 20-bit immediate, whereas the `addi` instruction encodes a 12-bit immediate. `lui` and `addi` can be used to load the upper 20 bits and the lower 12 bits of a 32-bit constant, respectively.

Let $N$ be a 32-bit constant we want to load into a register: $N \equiv n_{31} \dots n_0$. Then, we can split this constant into its upper 20 bits and lower 12 bits, $N_U$ and $N_L$, respectively: $N_U \equiv n_{31} \dots n_{12}$ ; $N_L \equiv n_{11} \dots n_0$

In principle, we encode $N_U$ in the immediate in `lui` and $N_L$ in the immediate in `addi`. Nevertheless, there is a difficulty to handle **if the most significant bit of the 12-bit immediate in `addi` is *1*** because the immediate value encoded in the `addi` instruction is ***sign extended*** to 32 bits. If this is the case, the `addi` instruction adds to the destination register not $N_L$, but $\boldsymbol{N_L - 4096}$ instead — *-4096* (or $-2^{12}$) is the resulting number when the upper 20 bits are *1*s and the lower 12 bits are *0*s.

To compensate for the unwanted term *-4096*, we can add *1* to `lui`'s immediate – the LSB of the immediate in `lui` corresponds to bit *#12* – so, adding *1* to this immediate results in adding *4096* to the destination register which cancels out the *-4096* term.

## Loading a 32-bit constant with a single `addi` instruction

The issue explained above is due to the sign extension that the immediate in `addi` undergoes. The decision of sign extending `addi`'s immediate was probably to allow the loading of *small integers* – **integers between *-2048* and *2047*, both inclusive – with a single `addi` instruction**. For example, if the immediate in `addi` were *zero extended* instead of sign extended, it wouldn't be possible to load such a frequent constant like *-1* into a register with just a single instruction.

## Loading a 32-bit constant with the `li` *pseudoinstruction*

In any case, you can always use the `li` pseudoinstruction for loading a 32-bit constant without having to care about what the value of the constant to load is. This pseudoinstruction can load any 32-bit number into a register, and it is, therefore, simpler to use and less error-prone than manually writing the `lui+addi` sequence.

If the number fits in `addi`'s immediate field (*[-2048, 2047]*), the assembler will translate the `li` pseudoinstruction into just an `addi` instruction, otherwise, `li` will be translated into a `lui+addi` sequence and the complication explained above is handled automatically by the assembler.



.net-assembly

› How to make sure columns in QTableView are resized to the maximum

## Leave a Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Search

Search

## Latest Post

CodeLobster IDE – Free Code Editor for PHP, JavaScript, HTML and CSS

KVO on Swift's computed properties

Babel error: Class constructor Foo cannot be invoked without 'new'

Dependency convergence error

Is it possible to have a global launch.json file?