Loading a 32 bit Immediate

Introduction

Is it possible to load a 32 bit immediate to a register? *Immediate* implies the constant is part of the instruction. Let's forget, for a moment, that none of the MIPS instruction formats support 32 bit constants. Is it possible for *any* ISA that uses 32-bit instructions to load an arbitrary 32 bit immediate value to a register?

Of course, by phrasing the question in that way, I'm giving away the answer. If the instruction must be 32-bits, there's no way to do it. First of all, if all 32 bits are used for the immediate, where are the bits for the opcode? Where are the bits to indicate which register to place the immediate value? This information also has to be there, but there's no space.

Notice it's important that we say *arbitrary* 32 bits. Clearly, you can load a 32 bit quantity using **addi**, but since **addi** uses a 16 bit quantity (which is then sign-extended to 32 bits), there's only $\mathbf{2^{16}}$ different 32-bit quantities you can store, not the full $\mathbf{2^{32}}$ you'd normally expect with a 32-bit immediate value.

Even if you left everything implicit, for example, you assumed you were always using register 1, and you assumed that the 32-bit immediate value was always loaded in that register, then, you'd be saying no other instructions are possible.

Since there's no way to do it one instruction, is there a way to do it in two instructions?

Loading 32 bit Immediates in Two Instructions

There's a MIPS instruction called **lui** which stands for "load upper immediate". The instruction looks like:

```
lui  $rt, immed
```

This is an **I-type** instruction. **$rs** is unused in this instruction.

The semantics are:

$$R[t] = IR_{15\text{-}0}\ 0^{16}$$

It loads the upper 16 bits of **R[t]** with the 16 bit immediate, and the lower 16 bits with all 0's.

One possiblility for loading a 32 bit constant, say, 0x0123abcd is:

```
lui  $r1, 0x0123
addi $r1, $r1, 0xabcd
```

However, this has problems. In particular, recall that **addi** sign-extends. If the immediate value is negative, then the upper 16 bits will be all 1's, and adding this will ruin the upper 16 bits.

One solution is to use **ori**

```
lui  $r1, 0x0123
ori  $r1, $r1, 0xabcd
```

ori zero-extends the immediate value. It also takes advantage of the fact that the low 16 bits of the register is all 0's. Thus, using bitwise OR is like adding, if there is no carry. Since you are adding 0x01230000 (which is what **$r1** contains after the **lui** instruction) to 0x0000abdc (which is the zero-extended immediate of the **ori** instruction), there's no carries, thus **ori** behaves like unsigned addition.

The other possibility is to use **addiu** which adds, but does so by zero-extending the immediate instead of sign-extending it.

Using $at

The pseudo instruction, **li $rt, immed** loads an immediate value to a register. This can be a 32 bit value. Such an instruction is translated to:

```
lui  $at, 0x0123
ori  $at, $at, 0xabcd
```

where **$at** is actually register **$r1**. This register is used by the assembler for translating pseudoinstructions to real instructions. After all, when the assembler does this translation, it wants to avoid clobbering other registers. **$at** is reserved specifically for this purpose.

If the immediate value is written in base 10, then the assembler must represent it as a 32 bit 2C binary number, then split the high 16 bits for the **lui** instruction and the low 16 bits for the **ori** instruction.

Machine Code Representation

lui is an I-type instruction. **$rs** is ignored. It can have any value.

Instruction	B_{31-26}	B_{25-21}	B_{20-16}	B_{15-0}

	opcode	register s	register t	immediate
lui $rt, immed	**001 111**	**ignored**	**-**	**immed**

The dashes are 5-bit encoding of the register number in UB. For example, **$r7** is encoded as 00111. The offset is represented in 16-bit 2C.