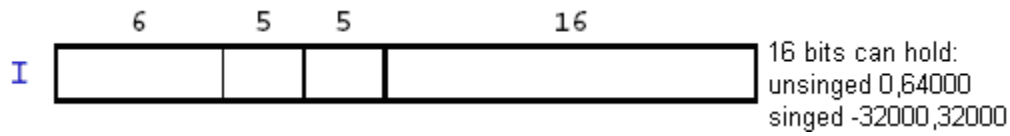
Dealing with Large Values

1) Branching

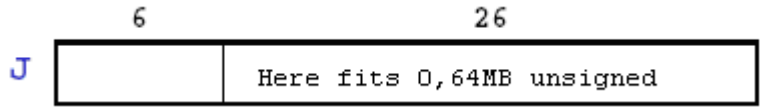


But 64000 is not enough to hold memory addresses.

So the following solutions are used:

- PC-relative rather than starting at `.text`
- Even better: Store the word address (i.e. real or byte address divided by four)
  This gives us 128000 bytes in each direction.
  Note that every MIPS instruction is exactly 4 bytes.
- worst-case: reverse condition → Then use jump

2) Jumping



- Unlike branching jump is not justifiable → must be unsigned starting at `.text`
- Store word address: 0,64KB → 0, 256KB
  Note that 256KB is the MAXIMUM PROGRAM SIZE in MIPS (32-bit processors).

3) I-type with immediate larger than 16 bits

Assume we want to store 90000 in `$t0`:

```
addi  $t0, $0, 90000
```

This is not a legal MIPS instruction because the immediate is out of range.

But 90000 = 0x15F90

And it can be separated into low and high portions: 0x 0001 5F90

We can use these three instructions to store the low and high portions of 90000:

```
lui   $s5, 0x1    # lui = load upper immidiate
addi  $t0, $0, 0x5F90
or    $t0, $s5, $t0
```

`lui $s0, #` (load upper immidiate) is like `addi $s0, $0, #` except it stores the immediate in the upper half. (We could achieve this by using `addi` and shifting 16 bits to the left.)

```
lui $s0, #       --> stores # in upper half :     $s0 [    #    |  0  ]
addi $s0, $0, # --> stores # in lower half :     $s0 [ 0 or 1  |  #   ]
```