Pseudoinstructions

Introduction

Pseudoinstructions means "fake instruction". You might wonder why they exist. When designing a modern ISA, one criteria is to decide whether an instruction should be part of the ISA or not. In the past, if you could see a use for the instruction, it was often added to ther instruction set. This lead to bloated and slow implementations.

In the 1980s, researcher began questioning the assumptions of how to design an ISA. The original thinking was to write ISAs that would be compact and useful to compiler writers. Compactness was important because memory was very expensive up until the 1980's.

Thus, instructions were often variable-sized, i.e., some instructions had as few as one byte, to as many as, say, 8 or more bytes. The thinking was "why use any more bytes than necessary".

Once memory became cheaper, it was reasonable to think that perhaps instruction sizes could be larger, and perhaps fixed in size. This would allow hardware designers to use advanced techniques like pipelining. By having equal sized instructions, it was easier to fetch the instructions, because no decoding was required.

Initially, RISC ISA designers thought that minimizing the total number of instructions was the best way to go. Fewer instructions meant simpler hardware, and simpler hardware meant easier optimization.

Fewer instructions also meant larger volume of code. What used to be executed in one CISC instruction might require ten or more RISC instructions. The hope was that faster hardware (and caches) would offset the increase in code size.

This turned out not to be the case. It was decided that the best way to decide whether an instruction should or should not be included was benchmarking. Use typical C or FORTRAN code, and compile it to assembly. Run the code on a CPU, and see if the time to execute runs significantly faster with the instruction as opposed to without. If it performs better, leave the instruction in. If not, leave it out.

This idea was fairly revolutionary. Instead of designing for compiler writers and memory, the idea was to design ISAs for performance.

However, by taking out some instructions, assembly language programmers would find it a little harder to write code. To make it easier for them, pseudoinstructions were added. Pseudoinstructions do not correspond to real MIPS instructions.

Instead, the assembler, a program that converts assembly language programs to machine code, would then translate pseudoinstructions to real instructions, usually requiring at least one on more instructions.

Pseudoinstructions not only make it easier to program, it can also add clarity to the program, by making the intention of the programmer more clear.

Pseudo instructions

Here's a list of useful pseudo-instructions.

- **mov $rt, $rs**

  Copy contents of register **s** to register **t**, i.e. **R[t] = R[s]**.

- **li $rs, immed**

  Load immediate into to register **s**, i.e. **R[s] = immed**. The way this is translated depends on whether **immed** is 16 bits or 32 bits.

- **la $rs, addr**

  Load address into to register **s**, i.e. **R[s] = addr**.

- **lw $rt, big($rs)**

  Load a word into memory with a 32-bit offset (called **big**). Notice that this is normally not allowed, because only 16-bit offsets are permitted.

  Similar pseudo-instructions exist for **sw**, etc.

There are a series of branch instructions that are also useful. Other ISAs often have pseudo-instructions for the convenience of the programmer.

Translating Some Pseudoinstructions

Pseudoinstruction	Translation
mov $rt, $rs	addi $rt, $rs, 0
li $rs, small	addi $rt, $rs, small
li $rs, big	lui $rs, upper(big) ori $rs, $rs, lower(big)
la $rs, big	lui $rs, upper(big) ori $rs, $rs, lower(big)
lw $rt, big($rs)	lui $t0, upper(big) ori $t0, $t0, lower(big) add $t0, $rs, $t0 lw $rt, 0($t0)

where **small** means a quantity that can be represented using 16 bits, and **big** means a 32 bit quantity. **upper(big)** is the upper 16 bits of a 32 bit quantity. The assembler must figure out how to get the upper 16 bits of a 32-bit value (that takes a little work). **lower(big)** is the lower 16 bits of the 32 bit quantity. Again, the assembler must figure out. Thus, **upper(big)** and **lower(big)** are not real instructions.

If you were to do the translation, you'd have to break it up yourself to figure out those quantities.