The MIPS R4000, part 4: Constants

devblogs.microsoft.com/oldnewthing/20180405-00

April 5, 2018

Raymond Chen

Since the MIPS R4000 has a fixed 32-bit instruction size, it cannot have a generalized "load 32-bit immediate constant" instruction. (There would be no room in the instruction for the opcode!)

If you look at the integer calculations available, you see that there are some ways of generating constants in a single instruction.

Constants in the range `0x00000000` to `0x0000FFFF` can be generated in one instruction by using `ORI`, which treats its 16-bit immediate as an unsigned value.

```
ORI     rd, zero, imm16
```

Constants in the range `0xFFFF8000` to `0xFFFFFFFF` can be generated with the `ADDIU` instruction, which treats its 16-bit immediate as a signed value.

```
ADDIU   rd, zero, imm16
```

If we had a `NORI` instruction, then we could have used it to generate constants in the range `0xFFFF0000` to `0xFFFFFFFF`:

```
NORI    rd, zero, imm16
```

But alas that instruction doesn't exist.

To build 32-bit values that cannot be created with these one-instruction tricks, you can use the `LUI` instruction, which means "load upper immediate".

```
LUI     rd, imm16           ; rd = imm16 << 16
```

It loads the 16-bit immediate value into the upper 16 bits of the destination register and zeroes out the bottom 16 bits. You can then follow this up with an `ORI` to finish the job:

```
LUI     rd, XXXX            ; rd = XXXX0000
ORI     rd, rd, YYYY        ; rd = XXXXYYYY
```

There is a data dependency here, and you might expect a pipeline bubble because the `ORI` depends on the result of the previous instruction, which won't be available until the write-back stage four cycles later. However, the processor supports integer arithmetic forwarding: The result of an arithmetic operation produced in the execute stage can be fed directly to the execute stage of the next instruction, thereby avoiding a stall.

Since the constant is loaded up 16 bits at a time, when a module needs to be relocated, moving it by a multiple of 64KB permits the fixup to be applied only to the `XXXX` part, leaving the `YYYY` part alone. (Previous discussion.) This is a very useful property, because in practice, these two instructions may not be adjacent to each other. The compiler might choose to interleave other calculations to avoid the data dependency stall.

There are a few pseudo-instructions provided by the assembler for loading 32-bit constants.

```
LI      rd, imm32           ; rd = imm32 (by whatever means)
LA      rd, global_variable ; rd = address_of global_variable
```

The `LI` pseudo-instruction loads a 32-bit immediate into *rd* using a single-instruction trick if available; otherwise, it uses the two-instruction sequence.

The `LA` pseudo-instruction does the same thing, but the 32-bit value comes from the address of a global variable and is consequently subject to a relocation fixup.

Next time, we'll look at aligned memory access.

Raymond Chen

Follow