6

Arithmetic in MIPS

Objectives

After completing this lab you will:

- know how to do integer arithmetic in MIPS
- know how to do floating point arithmetic in MIPS
- know about conversion from integer to floating point and from floating point to integer.

Introduction

The definition of the R2000 architecture includes all integer arithmetic within the actual CPU. Floating point arithmetic is done by one of the four possible coprocessors, namely coprocessor number 1.

Integer arithmetic

Addition and subtraction are performed on 32 bit numbers held in the general purpose registers (32 bit each). The result is a 32 bit number itself. Two kinds of instructions are included in the instruction set to do integer addition and subtraction:

- instructions for signed arithmetic: the 32 bit numbers are considered to be represented in 2's complement. The execution of the instruction (addition or subtraction) may generate an overflow.
- instructions for unsigned arithmetic: the 32 bit numbers are considered to be in standard binary representation. Executing the instruction will never generate an overflow error even if there is an actual overflow (the result cannot be represented with 32 bits).

Multiplication and division may generate results that are larger than 32 bits. The architecture provides two special 32 bit registers that are the destination for multiplication and division instructions. These registers are called **`hi`** and **`lo`** as to indicate that they hold the higher 32 bits of the result and the lower 32 bits respectively. Special instructions are also provided to move data from these registers into the general purpose ones (**`$0`** to **`$31`**).

Instruction	Effect
`mfhi Rdest`	`Rdest` ← `hi`
`mflo Rdest`	`Rdest` ← `lo`

Instruction	Effect
`mthi Rsrc`	`hi` ← `Rsrc`
`mtlo Rsrc`	`lo` ← `Rsrc`

Two kinds of instructions are included in the instruction set to do integer multiplication and division:

- instructions for signed arithmetic: the 32 bit numbers are considered to be represented in 2's complement. An integer multiplication will never generate an overflow. Division may overflow. Note however that the instruction will not signal the overflow: this must be done in software.
- instructions for unsigned arithmetic: the 32 bit numbers are considered to be in standard binary representation. Executing the instruction will never generate an overflow error even if there is an actual overflow (this is the case for division).

Floating point arithmetic

Floating point arithmetic is performed by the MIPS' coprocessor 1. The coprocessor has 32 registers, numbered from 0 to 31 (their names are **`$f0`** to **`$f31`**). Each register is 32 bit wide. To accommodate doubles, registers are grouped together (0 with 1, 2 with 3,..., 30 with 31). To simplify things, floating-point operations use only even numbered registers. Arithmetic is performed on single-precision floating-point numbers (32 bit representation), and on double-precision floating-point numbers (64 bit representation). The odd numbered registers are used to hold the least significant 32 bits of a double precision number.

Floating point arithmetic follows the IEEE-754 floating-point standard. A very brief description of number formats in this standard is given in Appendix A.

Floating-point addition, subtraction, multiplication and division may overflow. An overflow means that the exponent is too large to be represented in the exponent field.

The MIPS instruction set provides instructions that, beside floating-point operations, do floating-point comparisons, branching, load and store from/to memory and conversions between floating point formats and between integer and floating-point numbers. Unlike the integer unit where comparisons explicitly set some destination register, in the floating point unit, a comparison will *implicitly* set a condition flag. The condition flag can then be tested by branch instructions.

A summary of native instructions is presented in the table below. The `x` suffix for some instructions should be replaced by a `s` (e.g. `add.s`) to indicate the instruction should operate on single precision floating-point numbers, or by a (e.g. `add.d`) to indicate double precision operation.

Instruction	Comment
`mfc1 Rdest, FPsrc`	Move the content of floating-point register `FPsrc` to `Rdest`
`mtc1 Rsrc, FPdest`	Integer register `Rsrc` is moved to floating-point register `FPdest`
`mov.x FPdest, FPsrc`	Move floating-point register `FPsrc` to `FPdest`
`lwc1 FPdest, address`	Load word from `address` in register `FPdest`[a]
`swc1 FPsrc, address`	Store the content of register `FPdest` at `address`[b]
`add.x FPdest, FPsrc1, FPsrc2`	Add single precision

Instruction	Comment
`sub.x FPdest, FPsrc1, FPsrc2`	Subtract `FPsrc2` from `FPsrc1`
`mul.x FPdest, FPsrc1, FPsrc2`	Multiply
`div.x FPdest, FPsrc1, FPsrc2`	Divide `FPsrc1` by `FPsrc2`
`abs.s FPdest, FPsrc`	Store the absolute value of `FPsrc` in `FPdest`
`neg.x FPdest, FPsrc`	Negate number in `FPsrc` and store result in `FPdest`
`c.eq.x FPsrc1, FPscr2`	Set the floating-point condition flag to true if the two registers are equal
`c.le.x FPsrc1, FPsrc2`	Set the floating-point condition flag to true if `FPsrc1` is less than or equal to `FPsrc2`
`c.lt.x FPsrc1, FPsrc2`	Set the floating-point condition flag to true if `FPsrc1` is less than `FPsrc2`
`bc1t label`	Branch if the floating-point condition flag is true
`bc1f label`	Branch if the floating-point condition flag is false
`cvt.x.w FPdest, FPsrc`	Convert the integer in `FPsrc` to floating-point
`cvt.w.x FPdest, FPsrc`	Convert the floating-point number in `FPsrc` to integer
`cvt.d.s FPdest, FPsrc`	Convert the single precision number in `FPsrc` to double precision and put the result in `FPdest`
`cvt.s.d FPdest, FPsrc`	Convert the double precision number in `FPsrc` to single precision and put the result in `FPdest`

a. A single-precision floating-point number has the same size as a word (32 bits). Inside coprocessor 1 a word will be treated as a single-precision number. A synthetic instruction is available to load a double from memory.
b. Stores in memory a single precision number. A synthetic instruction is available to store a double.