31 25	24 20	19 15	14 12	11 7	6 0	
0000001	rs2	rs1	000	rd	0110011	R mul
0000001	rs2	rs1	001	rd	0110011	R mulh
0000001	rs2	rs1	010	rd	0110011	R mulhsu
0000001	rs2	rs1	011	rd	0110011	R mulhu
0000001	rs2	rs1	100	rd	0110011	R div
0000001	rs2	rs1	101	rd	0110011	R divu
0000001	rs2	rs1	110	rd	0110011	R rem
0000001	rs2	rs1	111	rd	0110011	R remu

Figure 4.2: RV32M opcode map has instruction layout, opcodes, format type, and names. (Table 19.2 of [Waterman and Asanović 2017] is the basis of this figure.)

```
# Compute unsigned division of a0 by 3 using multiplication.
   0: aaaab2b7    lui   t0,0xaaaab  # t0 = 0xaaaaaaab
   4: aab28293    addi  t0,t0,-1365 #    = ~ 2^32 / 1.5
   8: 025535b3    mulhu a1,a0,t0    # a1 = ~ (a0 / 1.5)
   c: 0015d593    srli  a1,a1,0x1   # a1 = (a0 / 3)
```

Figure 4.3: RV32M code to divide by a constant by multiplying. It takes careful numerical analysis to show that this algorithm works for any dividend, and for some other divisors, the correction step is more complicated. The proof of correctness, and the algorithm for generating the reciprocals and correction steps, is in [Granlund and Montgomery 1994].

The multiply equation is simply:

$$Product = Multiplicand \times Multiplier$$

`sll` **can do signed and unsigned multiplication by** 2^i**.** For example, if a2 = 16 (2^4) then `slli t2,a1,4` produces the same value as `mul t2,a1,a2`.

It's more complicated than divide because the size of the product is the sum of the sizes of the multiplier and the multiplicand; multiplying two 32-bit numbers yields a 64-bit product. To produce a properly signed or unsigned 64-bit product, RISC-V has four multiply instructions. To get the integer 32-bit product—the lower 32-bits of the full product—use `mul`. To get the upper 32 bits of the 64-bit product, use `mulh` if both operands are signed, `mulhu` if both operands are unsigned, and `mulhsu` is one is signed, and the other is unsigned. Since it would complicate the hardware to write the 64-bit product into two 32-bit registers in one instruction, RV32M requires two multiply instructions to produce the 64-bit product.

For almost all processors, multiplies are slower than shifts or adds and divides are much slower than multiplies.

For many microprocessors, integer division is a relatively slow operation. As mentioned above, right shifts can replace unsigned division by powers of 2. It turns out that division by other constants can be optimized, too, by multiplying by the approximate reciprocal then applying a correction to the upper half of the product. For example, Figure 4.3 shows the code for unsigned division by 3.

What's Different? ARM-32 long had multiply but no divide instruction. Divide didn't become mandatory until 2005, almost 20 years after the first ARM processor. MIPS-32 uses special registers (HI and LO) as the sole destination registers for multiply and divide instructions. While this design reduced the complexity of early MIPS implementations, it takes an extra move instruction to use the result of the multiply or divide, potentially reducing performance. The HI and LO registers also increase the architectural state, making it slightly slower to switch between tasks.

■ ***Elaboration: `mulh` and `mulhu` can check for overflow in multiplication.***

There is no overflow when using `mul` for unsigned multiplication if the result of `mulhu` is zero. Similarly, there is no overflow when using `mul` for signed multiplication if all bits in the result of `mulh` match the sign bit of the result of `mul`, i.e., equals 0 if positive or ffff ffff$_{hex}$ if negative.

■ ***Elaboration: It's also easy to check for divide by zero.***

Just add a `beqz` test of the dividend before the divide. RV32I doesn't trap on divide by zero because few programs want that behavior, and the ones that do can easily check for zero in software. Of course, divides by constants never need checks.

■ ***Elaboration: `mulhsu` is useful for multi-word signed multiplication.***

`mulhsu` generates the upper half of the product when the multiplier is signed and the multiplicand is unsigned. It is as a substep of multi-word signed multiplication when multiplying the most-significant word of the multiplier (which contains the sign bit) with the less-significant words of the multiplicand (which are unsigned). This instruction improves performance of multi-word multiplication by about 15%.

4.2 Concluding Remarks

The cheapest, fastest, and most reliable components are those that aren't there.

—C. Gordon Bell, architect of prominent minicomputers

To offer the smallest RISC-V processor for embedded applications, multiply and divide are part of the first optional standard extension of RISC-V. Nevertheless, many RISC-V processors will include RV32M.

4.3 To Learn More

T. Granlund and P. L. Montgomery. Division by invariant integers using multiplication. In *ACM SIGPLAN Notices*, volume 29, pages 61–72. ACM, 1994.

A. Waterman and K. Asanović, editors. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2*. May 2017. URL `https://riscv.org/specifications/`.

Notes

[1] `http://parlab.eecs.berkeley.edu`

5 RV32F and RV32D: Single- and Double-Precision Floating Point

Antoine de Saint Exup'ery, L'Avion (1900-1944) was a French writer and aviator best known for the book *The Little Prince.*

Perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away.

—Antoine de Saint Exup'ery, L'Avion, 1940

5.1 Introduction

Although RV32F and RV32D are separate, optional instruction set extensions, they are often included together. Given single- and double-precision (32- and 64-bit) versions of nearly all floating-point instructions, for brevity we present them in one chapter. Figure 5.1 is a graphical representation of the RV32F and RV32D extension instruction sets. Figure 5.2 lists the opcodes of RV32F and Figure 5.3 lists the opcodes of RV32D. Like virtually all other modern ISAs, RISC-V obeys the IEEE 754-2008 floating-point standard [IEEE Standards Committee 2008].

5.2 Floating-Point Registers

RV32F and RV32D use 32 separate `f` registers instead of the `x` registers. The main reason for the two sets of registers is that processors can improve performance by doubling the register capacity and bandwidth by having two sets of registers without increasing the space for the register specifier in the cramped RISC-V instruction format. The major impact on the instruction set is to have new instructions to load and store the `f` registers and to transfer data between the `x` and `f` registers. Figure 5.4 lists the RV32D and RV32F registers and their names as determined by the RISC-V ABI.

If a processor has both RV32F and RV32D, the single-precision data uses only the lower 32 bits of the `f` registers. Unlike x0 in RV32I, register `f0` is *not* hardwired to 0 but is an alterable register like all the other 31 `f` registers.

The IEEE 754-2008 standard provides several ways to round floating-point arithmetic, which are helpful to determine error bounds and in writing numerical libraries. The most accurate and most common is round to nearest even (RNE). The rounding mode is set in the floating-point control and status register `fcsr`. Figure 5.5 shows `fcsr` and lists the rounding options. It also holds the accrued exception flags that the standard requires.

What's Different? Both ARM-32 and MIPS-32 have 32 single-precision floating-point registers but only 16 double-precision registers. They both map two single-precision registers