RISC V Architecture

Prof. H R Vanamala
Department of Electronics and Communication Engg.

PES UNIVERSITY
CELEBRATING 50 YEARS

RISC V ARCHITECTURE

UNIT 4: Arithmetic for Computers

Prof. H R Vanamala

Department of Electronics and Communication Engineering

Floating Point - Multiplication

Consider a 4-digit decimal example

$1.110 \times 10^{10} \times 9.200 \times 10^{-5}$

1. Add exponents

   For biased exponents, subtract bias from sum

   New exponent = 10 + –5 = 5

2. Multiply significands

   $1.110 \times 9.200 = 10.212 \Rightarrow 10.212 \times 10^{5}$

3. Normalize result & check for over/underflow

   1.0212×10^{6}

4. Round and renormalize if necessary

   1.021×10^{6}

5. Determine sign of result from signs of operands

   $+1.021 \times 10^{6}$

Floating Point – Multiplication - binary example

Consider a 4-digit binary example

$1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$ (0.5 × –0.4375)

1. Add exponents

   Unbiased: –1 + –2 = –3

   Biased: (–1 + 127) + (–2 + 127) = –3 + 254 – 127 = –3 + 127

2. Multiply significands

   $1.000_2 \times 1.110_2 = 1.1102 \Rightarrow 1.110_2 \times 2^{-3}$

3. Normalize result & check for over/underflow

   $1.110_2 \times 2^{-3}$ (no change) with no over/underflow

4. Round and renormalize if necessary

   $1.110_2 \times 2^{-3}$ (no change)

5. Determine sign: +ve × –ve ⇒ –ve

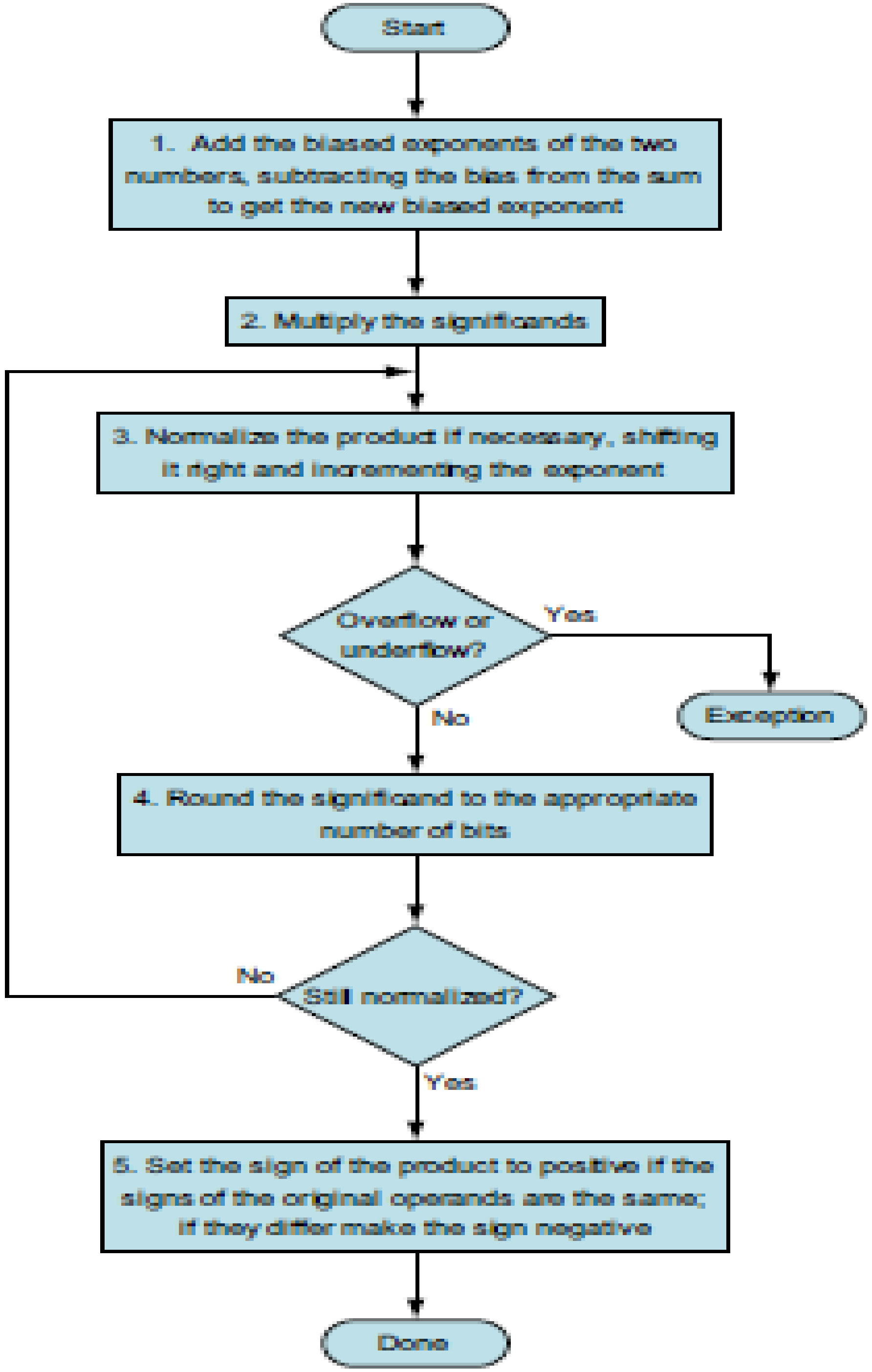
   $-1.110_2 \times 2^{-3} = -0.21875$

Floating Point – Multiplication - Hardware

- FP multiplier is of similar complexity to FP adder
  - But uses a multiplier for significands instead of an adder
- FP arithmetic hardware usually does
  - Addition, subtraction, multiplication, division, reciprocal, square-root
  - FP ↔ integer conversion
- Operations usually takes several cycles
  - Can be pipelined

Floating Point - Multiplication

Floating-point multiplication:
The normal path is to execute steps 3 and 4 once,
but if rounding causes the sum to be un normalized, we must repeat step 3



Floating Point - Instructions in RISC-V

Separate FP registers: f0, …, f31

- double-precision
- single-precision values stored in the lower 32 bits

FP instructions operate only on FP registers

- Programs generally don't do integer ops on FP data, or vice versa
- More registers with minimal code-size impact

FP load and store instructions

```
flw, fld
fsw, fsd
```

Floating Point - Multiplication

Single-precision arithmetic

fadd.s, fsub.s, fmul.s, fdiv.s, fsqrt.s
e.g., fadds.s f2, f4, f6

Double-precision arithmetic

fadd.d, fsub.d, fmul.d, fdiv.d, fsqrt.d
e.g., fadd.d f2, f4, f6

Single- and double-precision comparison

feq.s, flt.s, fle.s
feq.d, flt.d, fle.d
Result is 0 or 1 in integer destination register

Use beq, bne to branch on comparison result
Branch on FP condition code true or false

Floating Point - Multiplication

The RISC-V code to load two single precision numbers from memory, add them, and then store the sum.

```
flw f0, 0(x10) // Load 32-bit F.P. number into f0
flw f1, 4(x10) // Load 32-bit F.P. number into f1
fadd.s f2, f0, f1 // f2 = f0 + f1, single precision
fsw f2, 8(x10) // Store 32-bit F.P. number from f2
```

Floating Point - Compiling a Floating-Point C Program into RISC-V

Assembly Code: convert temperature in Fahrenheit to Celsius:

C code:-

```
        float f2c (float fahr)
 {
  return ((5.0/9.0)*(fahr - 32.0));
}
```

fahr in f10, result in f10, literals in global memory space: RISC-V code

```
f2c:
   flw    f0,const5(x3)  // f0 = 5.0f
   flw    f1,const9(x3)  // f1 = 9.0f
   fdiv.s f0, f0, f1  // f0 = 5.0f / 9.0f
   flw    f1,const32(x3) // f1 = 32.0f
   fsub.s f10,f10,f1  // f10 = fahr – 32.0
   fmul.s f10,f0,f10  // f10 = (5.0f/9.0f) * (fahr–32.0f)
   jalr   x0,0(x1)    // return
```

Floating Point - Compiling Floating-Point C Procedure with Two-Dimensional Matrices into RISC-V

C = C + A × B

All 32 × 32 matrices, 64-bit double-precision elements

C code:

```
void mm (double c[][], double a[][], double b[][])
{
        size_t i, j, k;
                for (i = 0; i < 32; i = i + 1)
                        for (j = 0; j < 32; j = j + 1)
                                for (k = 0; k < 32; k = k + 1)
                                        c[i][j] = c[i][j] + a[i][k] *b[k][j];
}
```

Addresses of c, a, b in x10, x11, x12, and i, j, k in x5, x6, x7

```
mm:...
        li   x28,32      // x28 = 32 (row size/loop end)
        li   x5,0        // i = 0; initialize 1st for loop
L1:     li   x6,0        // j = 0; initialize 2nd for loop
L2:     li   x7,0        // k = 0; initialize 3rd for loop
          slli x30,x5,5     // x30 = i * 2**5 (size of row of c)
          add  x30,x30,x6   // x30 = i * size(row) + j
          slli x30,x30,3    // x30 = byte offset of [i][j]
          add  x30,x10,x30  // x30 = byte address of c[i][j]
          fld  f0,0(x30)    // f0 = c[i][j]
L3:     slli x29,x7,5     // x29 = k * 2**5 (size of row of b)
        add  x29,x29,x6   // x29 = k * size(row) + j
        slli x29,x29,3    // x29 = byte offset of [k][j]
        add  x29,x12,x29  // x29 = byte address of b[k][j]
       fld  f1,0(x29)    // f1 = b[k][j]
```

Floating Point - Multiplication

```
slli   x29,x5,5      // x29 = i * 2**5 (size of row of a)
add    x29,x29,x7    // x29 = i * size(row) + k
slli   x29,x29,3     // x29 = byte offset of [i][k]
add    x29,x11,x29   // x29 = byte address of a[i][k]
fld    f2,0(x29)     // f2 = a[i][k]
fmul.d f1, f2, f1    // f1 = a[i][k] * b[k][j]
fadd.d f0, f0, f1    // f0 = c[i][j] + a[i][k] * b[k][j]
addi   x7,x7,1       // k = k + 1
bltu   x7,x28,L3     // if (k < 32) go to L3
fsd    f0,0(x30)     // c[i][j] = f0
addi   x6,x6,1       // j = j + 1
bltu   x6,x28,L2     // if (j < 32) go to L2
addi   x5,x5,1       // i = i + 1
bltu   x5,x28,L1     // if (i < 32) go to L1
```

Floating Point - Accurate Arithmetic

---Approximations for a number

IEEE Std 754 specifies additional rounding control
Extra bits of precision (guard, round, sticky)

Choice of rounding modes - Allows programmer to fine-tune numerical behavior of a computation

Not all FP units implement all options
Most programming languages and FP libraries just use defaults

Trade-off between hardware complexity, performance, and market requirements

Floating Point - Accurate Arithmetic

Two extra bits on the right during intervening additions, called **guard and round, respectively.**

guard the first of two: extra bits kept on the right during intermediate calculations of floating point numbers

- used to improve rounding accuracy

round Method to: make the intermediate floating-point result fit the floating-point format

The goal is typically to find the nearest number that can be represented in the format.

It is also the name of the second of two extra bits kept on the right during intermediate floating point calculations, which improves rounding accuracy.

Floating Point - Accurate Arithmetic -Decimal Example

Rounding with Guard Digits: Add $2.56_{ten} \times 10^0$ to $2.34_{ten} \times 10^2$,

The guard digit holds 5 and the round digit holds 6. The sum is

$$\begin{array}{r} 2.3400_{ten} \\ +0.0256_{ten} \\ \hline 2.3656_{ten} \end{array}$$

Doing this without guard and round digits drops two digits from the calculation. The new sum is then

$$\begin{array}{r} 2.34_{ten} \\ +0.02_{ten} \\ \hline 2.36_{ten} \end{array}$$

The answer is $2.36_{ten} \times 10^2$, off by 1 in the last digit from the sum above.

Floating Point - Accurate Arithmetic -Decimal example

Accuracy in floating point is normally measured in terms of the number of bits in error in the least significant bits of the significand

Measure: Units in the last place, or ulp.

sticky bit used in rounding in addition to guard and round that is set whenever there are nonzero bits to the right of the round bit
This sticky bit allows the computer to see the difference between 0.50…00ten and 0.50 … 01ten when rounding.
Ex: Add 5.01ten × 10^{-1} to 2.34ten × 10^{2}, with guarding and rounding
= 0.0050 to 2.34=2.3450.
The sticky bit would be set, since there are nonzero bits to the right.
Without sticky bit , 2.345000 … 00 rounded to the nearest even of 2.34 and with sticky bit to to 2.35.

fused multiply add A : floating-point instruction that performs both a multiply and an add, but rounds only once after the add:a = a + (b × c).

Unit 4: Arithmetic for Computers

Floating Point - Multiplication

RISC-V floating-point operands

32 floating-point registers	f0 - f31	An f-register can hold either a single-precision floating-point number or a double-precision floating-point number.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4,294,967,292]	Accessed only by data transfer instructions. RISC-V uses byte addresses, so sequential word accesses differ by 4. Memory holds data structures, arrays, and spilled registers.

Refere: ce : Computer Architecture with RISC V - The Hardware/Software Interface: RISC-V Edition by David A. Patterson and John L. Hennessy

RISC-V floating-point assembly language

Arithmetic	FP add single	fadd.s f0, f1, f2	f0 = f1 + f2	FP add (single precision)
	FP subtract single	fsub.s f0, f1, f2	f0 = f1 - f2	FP subtract (single precision)
	FP multiply single	fmul.s f0, f1, f2	f0 = f1 * f2	FP multiply (single precision)
	FP divide single	fdiv.s f0, f1, f2	f0 = f1 / f2	FP divide (single precision)
	FP square root single	fsqrt.s f0, f1	f0 = √f1	FP square root (single precision)
	FP add double	fadd.d f0, f1, f2	f0 = f1 + f2	FP add (double precision)
	FP subtract double	fsub.d f0, f1, f2	f0 = f1 - f2	FP subtract (double precision)
	FP multiply double	fmul.d f0, f1, f2	f0 = f1 * f2	FP multiply (double precision)
	FP divide double	fdiv.d f0, f1, f2	f0 = f1 / f2	FP divide (double precision)
	FP square root double	fsqrt.d f0, f1	f0 = √f1	FP square root (double precision)
Comparison	FP equality single	feq.s x5, f0, f1	x5 = 1 if f0 == f1, else 0	FP comparison (single precision)
	FP less than single	flt.s x5, f0, f1	x5 = 1 if f0 < f1, else 0	FP comparison (single precision)
	FP less than or equals single	fle.s x5, f0, f1	x5 = 1 if f0 <= f1, else 0	FP comparison (single precision)
	FP equality double	feq.d x5, f0, f1	x5 = 1 if f0 == f1, else 0	FP comparison (double precision)
	FP less than double	flt.d x5, f0, f1	x5 = 1 if f0 < f1, else 0	FP comparison (double precision)
	FP less than or equals double	fle.d x5, f0, f1	x5 = 1 if f0 <= f1, else 0	FP comparison (double precision)
Data transfer	FP load word	flw f0, 4(x5)	f0 = Memory[x5 + 4]	Load single-precision from memory
	FP load doubleword	fld f0, 8(x5)	f0 = Memory[x5 + 8]	Load double-precision from memory
	FP store word	fsw f0, 4(x5)	Memory[x5 + 4] = f0	Store single-precision from memory
	FP store doubleword	fsd f0, 8(x5)	Memory[x5 + 8] = f0	Store double-precision from memory

PES UNIVERSITY
CELEBRATING 50 YEARS

Floating Point – Data Type and Instructions to be used:

C type	Java type	Data transfers	Operations
int	int	lw, sw	add, sub, addi, mul, mulh, mulhu, mulhsu, div, divu, rem, remu, and, andi, or, ori, xor, xori
unsigned int	—	lw, sw	add, sub, addi, mul, mulh, mulhu, mulhsu, div, divu, rem, remu, and, andi, or, ori, xor, xori
char	—	lb, sb	add, sub, addi, mul, div, divu, rem, remu, and, andi, or, ori, xor, xori
short	char	lh, sh	add, sub, addi, mul, div, divu, rem, remu, and, andi, or, ori, xor, xori
float	float	flw, fsw	fadd.s, fsub.s, fmul.s, fdiv.s, feq.s, flt.s, fle.s
double	double	fld, fsd	fadd.d, fsub.d, fmul.d, fdiv.d, feq.d, flt.d, fle.d

THANK YOU

Vanamala H R
Department of Electronics and Communication