

Floating point operations in MIPS

32 separate single precision FP registers in MIPS

f0, f1, f2, ... f31,

Can also be used as 16 double precision registers

f0, f2, f4, f30

These reside in a **coprocessor** in the same package

Operations supported

add.s \$f2, \$f4, \$f6 # f2 = f4 + f6 (single precision)

add.d \$f2, \$f4, \$f6 # f2 = f4 + f6 (double precision)

(Also subtract, multiply, divide format are similar)

lwc1 \$f1, 100(\$s2) # f1 = M [s2 + 100] (32-bit load)

mtc1 \$t0, \$f0 # f0 = t0 (move to coprocessor 1)

mfc1 \$t1, \$f1 # t1 = f1 (move from coprocessor 1)

Sample program

Evaluation of a Polynomial $a.x^2 + b.x + c$

Pseudo-
instruction

```
# $f0 --- x
# $f2 --- sum of terms
.....
# Evaluate the quadratic
l.s      $f2,a      # sum = a
mul.s    $f2,$f2,$f0 # sum = ax

l.s      $f4,b      # get b
add.s    $f2,$f2,$f4 # sum = ax + b
mul.s    $f2,$f2,$f0 # sum = (ax+b)x = ax^2 + bx

l.s      $f4,c      # get c
add.s    $f2,$f2,$f4 # sum = ax^2 + bx + c
.....

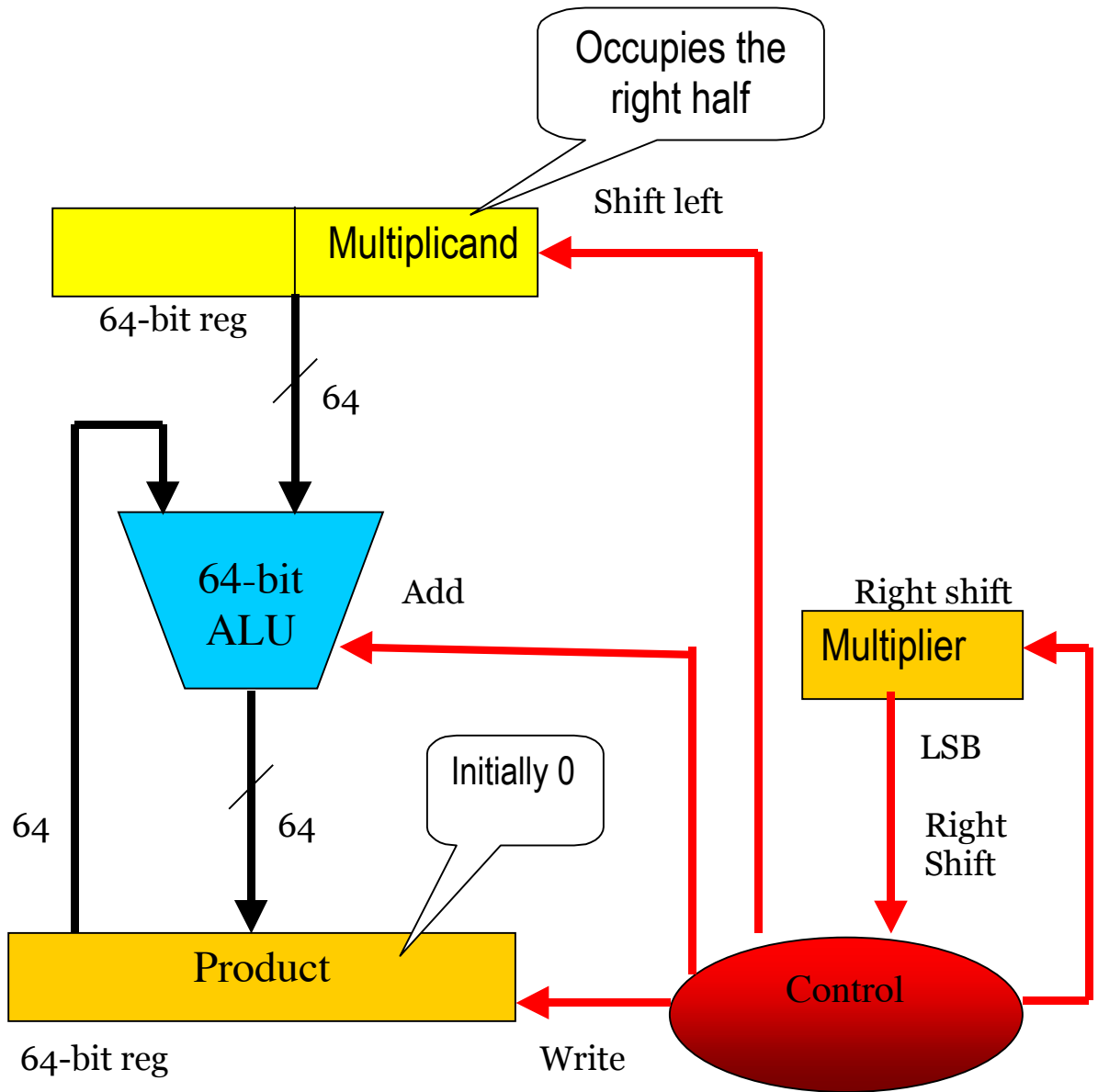
.data
a: .float 1.0
b: .float 1.0
c: .float 1.0
```

Multiplication algorithm

| | | | | | | | |
|--------------|---|---|---|---|---|---|---|
| Multiplicand | | | | 1 | 0 | 0 | 1 |
| Multiplier | | | | 1 | 0 | 1 | 0 |
| | | | | 0 | 0 | 0 | 0 |
| | | | 1 | 0 | 0 | 1 | 0 |
| | | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| Product | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| | | | | | | | |

The basic operations are **ADD** and **SHIFT**. Now note how it is implemented by hardware. By now, you know all the building blocks

A Hardware Multiplier



Division

The restoring division algorithm follows the simple idea from the elementary school days. It involves subtraction and shift. Here is an implementation by hardware

