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

\textbf{add.s} \quad $f2, f4, f6 \quad \# f2 = f4 + f6 \text{ (single precision)}$
\textbf{add.d} \quad $f2, f4, f6 \quad \# f2 = f4 + f6 \text{ (double precision)}$

(Also subtract, multiply, divide format are similar)

\textbf{lwc1} \quad f1, 100($s2) \quad \# f1 = M [s2 + 100] \text{ (32-bit load)}$
\textbf{mtc1} \quad t0, f0 \quad \# f0 = t0 \text{ (move to coprocessor 1)}$
\textbf{mfc1} \quad t1, f1 \quad \# t1 = f1 \text{ (move from coprocessor 1)}
Sample program

Evaluation of a Polynomial \( a.x^2 + b.x + c \)

```
# $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

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Product</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

- **Multiplicand**
  - Occupies the right half
  - Shift left

- **64-bit ALU**
  - Add
  - Initially 0

- **Multiplier**
  - Right shift
  - LSB
  - Right Shift

- **Product**
  - Write

- **Control**
  - 64-bit reg
  - 64
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.