ECE 251: Floating-Point Architecture

Week 08 Notes

Prof Rob Marano

Spring 2026

Overview

- Transitioning to **Chapter 3** of the textbook.
- Previous focus was exclusively on the **Integer Data Type** (signed/unsigned).
- Real-world engineering requires processing fractional decimal values.
- We explore:
  - The limitations of Fixed-Point logic.
  - The **IEEE 754 Protocol** standard.
  - How MIPS handles numbers using **Coprocessor 1 (FPU)**.
  - Hardware implementation and FPU pipelining.
  - Immediates and literal constraints in ISA.

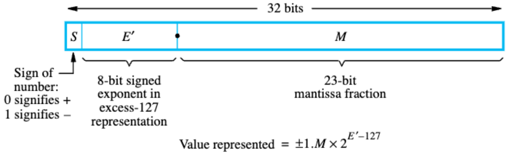
1. The Decimal Dilemma: Fixed vs. Floating Point

- Physical memory and CPU registers have a strict 32-bit limit.
- **The Fixed-Point Approach:** Divide the 32 bits evenly (e.g., 16 bits integer, 16 bits fraction).
  - **Pro:** Hardware simplicity. Traditional integer `add` works natively.
  - **Con:** Extreme range limitation. Maximum computable macro-scale number is $32,767$.
  - Lacks the vast dynamic range needed for physics simulations or graphics rendering.

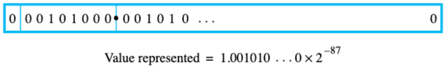
The Floating-Point Protocol (IEEE 754)

- Standardized by IEEE in 1985 to maximize dynamic range and precision.
- Uses scientific notation: hardware dynamically “floats” the decimal point.
- **Binary Floating Point Formula:** $(-1)^s \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-\text{Bias})}$

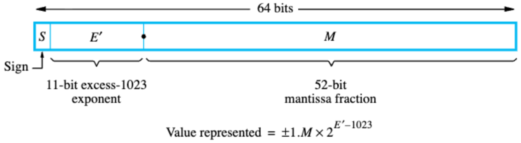
Standard Representation (IEEE 754)


32 bits
S
E′
M
Sign of number: 0 signifies + 1 signifies −
8-bit signed exponent in excess-127 representation
23-bit mantissa fraction
Value represented = ±1.M × 2^(E′−127)

(a) Single precision



(b) Example of a single-precision number


64 bits
S
E′
M
Sign
11-bit excess-1023 exponent
52-bit mantissa fraction
Value represented = ±1.M × 2^(E′−1023)

(c) Double precision

MIPS Single Precision (32-bit) Format

Mathematically packed into three distinct fields:

Sign (1 bit)	**Exponent (8 bits)**	**Fraction / Mantissa (23 bits)**
Bit 31	Bits 30–23	Bits 22–0

- **Sign:** $1 =$ negative, $0 =$ positive.
- **Exponent:** Uses a Bias of $+127$ for Single Precision.
- **Mantissa:** The precise fractional magnitude.

2. Hardware Segregation: Coprocessor 1 (The FPU)

- FPU arithmetic requires significantly different logical gates than integer math.
- Combining logic into one monolithic ALU creates massive processor bottlenecks.
- MIPS formally delegates decimal math to a secondary chip: **Coprocessor 1 (FPU)**.
- **FPU Register File:** Maintains its own separate array of 32 physical registers (`$f0` through `$f31`).
  - **Single Precision:** 32 bits per register natively.
  - **Double Precision:** Pairs adjacent registers (e.g., `$f0`/`$f1`) for 64-bit payloads.

Specialized FPU Instructions

- Standard MIPS integer instructions (`add`, `lw`) cannot physical access the FPU.
- Requires dedicated Coprocessor commands (`.s` for single, `.d` for double):
    - **Loading/Storing:** `lwc1 $f0, 0($a0)` (Load to Coprocessor 1)
    - **Arithmetic:** `add.s $f0, $f1, $f2` (Floating-Point Addition)
    - **Moving Data:** `mfc1 $t0, $f12` (Move *From* Coprocessor 1)

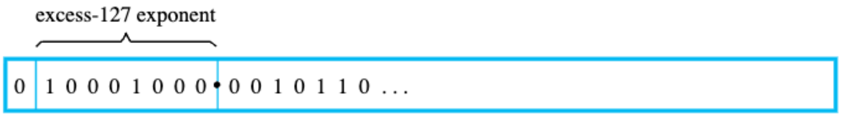
F-Type Instruction Format

opcode	**fmt**	**ft**	**fs**	**fd**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

3. IEEE 754 Standard Deep Dive

- **Biased Notation for Exponents:** FPU applies $+127$ bias instead of Two's Complement. Ensures exponents are strictly positive binary integers for fast comparator logic.
- **Normalized Mantissas:** The leading `1` is implicitly hidden (`1.xxxxx`), providing 24 bits of effective precision in a 23-bit space.
- **Special Values:** `0` (Exp$=$0, Frac$=$0) and **NaN** (Exp$=$255, Frac$\neq$0).

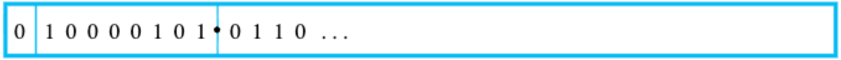
Normalizing a Mantissa (IEEE 754)



(There is no implicit 1 to the left of the binary point.)

$$\text{Value represented} = +0.0010110\ldots \times 2^{9}$$

(a) Unnormalized value



$$\text{Value represented} = +1.0110\ldots \times 2^{6}$$

(b) Normalized version

Example: Decimal to Single Precision

Problem: Convert -0.75_{10} to IEEE 754 Single Precision.

1. **Binary Fraction:** $-0.75_{10} = -3/4 = -0.11_2$.
2. **Normalize:** $-1.1_2 \times 2^{-1}$.
3. **Sign Field:** Negative $\rightarrow$ 1.
4. **Exponent Field:** True is -1. Add bias: $-1 + 127 = 126 \rightarrow$ 01111110.
5. **Mantissa Field:** Drop hidden 1, pad fraction 1 with zeros $\rightarrow$ 100 0000...

Final 32-bit Code: 1 01111110 10000000000000000000000 (0xBF400000)

Example: IEEE 754 to Decimal

Problem: Convert `0xC1480000` back to base-10 decimal.

1. **Hex to Binary:** `1100 0001 0100 1000 0000 0000 0000 0000`
2. **Slice:** `1 | 10000010 | 10010000000000000000000`
3. **Sign:** `1` $\rightarrow$ Negative $(-)$.
4. **Exponent:** $130 - 127(\text{bias}) = 3 \rightarrow 2^3$.
5. **Mantissa:** Append hidden `1.` $\rightarrow 1.1001_2$.
6. **Combine:** $-1.1001_2 \times 2^3 \rightarrow -1100.1_2$.
7. **Convert:** $-(8 + 4 + 0.5) = \mathbf{-12.5_{10}}$.

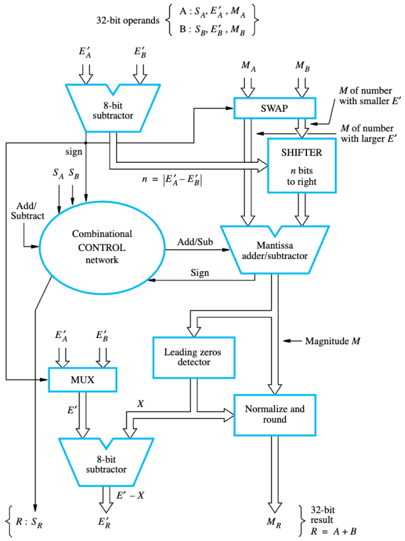
4. Literal Values in ISA Design

- **Literals/Immediates:** Hardcoded numbers inside code (e.g., `addi $t0, $t0, 4`).
- **"Make the Common Case Fast":** Small constants make up over 50% of operands. MIPS limits the immediate field purely to 16 bits via the **I-Type Format** to avoid memory fetches.
- **The 32-bit Alternative:** What if we need to load `0x003D0900`? We must use a two-instruction macro:
  - `lui $s0, 0x003D` (Loads `0x003D` to upper 16-bits).
  - `ori $s0, $s0, 0x0900` (ORs `0x0900` to lower 16-bits).

5. Microarchitectural Hardware Implementation

- **Integer Datapath:** ~ 1 cycle.
- **FPU Datapath:** Must pipe sequentially:
  1. **Alignment** (Shifting)
  2. **Execution** (Fraction Add)
  3. **Normalization**
  4. **Rounding** (IEEE limit)
- **Bottleneck:** FPUs natively **pipeline** across 4 to 6 cycles constraint.

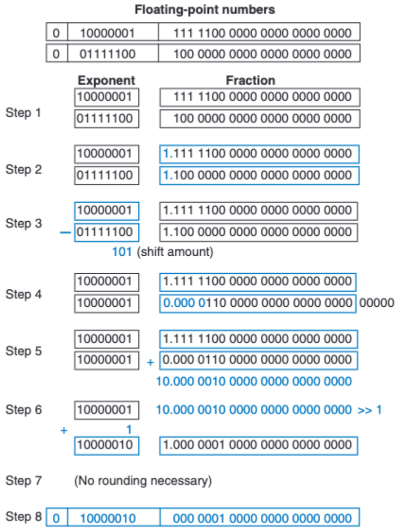
FPU Datapath Architecture



Example: FPU Pipeline Addition Trace

- **Trace:** $7.875_{10} + 0.1875_{10}$
- **Step 1-2:** Unpack Exponent/Fraction.
- **Step 3-4:** Compare Exponents and **Align** the fractional mantissas.
- **Step 5:** Perform fractional **Execution** (Addition).
- **Step 6:** **Normalize** the shifted result.
- **Step 7-8:** **Round** and pack back into 32-bit register format.

FPU Pipeline Addition Timeline



```
                     Floating-point numbers
        | 0 | 10000001 | 111 1100 0000 0000 0000 0000 |
        | 0 | 01111100 | 100 0000 0000 0000 0000 0000 |

               Exponent          Fraction
Step 1         10000001     111 1100 0000 0000 0000 0000
               01111100     100 0000 0000 0000 0000 0000

Step 2         10000001     1.111 1100 0000 0000 0000 0000
               01111100     1.100 0000 0000 0000 0000 0000

Step 3         10000001     1.111 1100 0000 0000 0000 0000
             – 01111100     1.100 0000 0000 0000 0000 0000
                    101 (shift amount)

Step 4         10000001     1.111 1100 0000 0000 0000 0000
               10000001     0.000 0110 0000 0000 0000 0000 00000

Step 5         10000001     1.111 1100 0000 0000 0000 0000
               10000001   + 0.000 0110 0000 0000 0000 0000
                           10.000 0010 0000 0000 0000 0000

Step 6         10000001    10.000 0010 0000 0000 0000 0000 >> 1
             +        1
               10000010     1.000 0001 0000 0000 0000 0000

Step 7         (No rounding necessary)

Step 8  | 0 | 10000010 | 000 0001 0000 0000 0000 0000 |
```

6. The Quake III Inverse Square Root Hack

- How do we approximate $\frac{1}{\sqrt{x}}$ without halting the CPU for 20+ cycles?
- **The MIPS Trick:** Move float to integer register, bit-shift right, subtract from magic constant!

```
# Move float bits from FPU ($f12) into CPU Integer Register ($t0)
mfc1  $t0, $f12

# i = 0x5f3759df - ( i >> 1 );
srl   $t1, $t0, 1          # Bit-shift right by 1
lw    $t2, magic           # Load magic number (0x5f3759df)
sub   $t0, $t2, $t1        # Integer subtraction
```

- **Why it works:** IEEE 754 structure is inherently logarithmic.
- Shifting bits right (`i >> 1`) mathematically scales exponent by 0.5 (calculating $x^{-0.5}$).
- Subtracting from `0x5f3759df` negates target exponent, approximates curve, and re-aligns +127 bias!

Conclusion

- Decimal values break classical fixed-point architectures.
- The global IEEE 754 standard introduces computational limits and brilliant bias formatting to squeeze massive range out of 32 bits.
- Physical processor frequency dictates that these complex derivations be outsourced to a segregated, pipelined Coprocessor unit.

Questions and Hardware Review