

FLOATING POINT OPERATIONS

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

- Notes

- ▣ Homework 6 will be posted tonight

- Deadline: Mar. 5th

- This lecture

- ▣ Floating point operations

- ▣ Basics of logic design

Recall: Floating Point Addition

- Numbers maintain only 4 decimal digits and 2 exponent digits
 - $9.999 \times 10^1 + 1.610 \times 10^{-1}$
 - Convert to the larger exponent
 - $9.999 \times 10^1 + 0.016 \times 10^1$
 - Add
 - 10.015×10^1
 - Normalize
 - 1.0015×10^2
 - Check for overflow/underflow
 - Round
 - 1.002×10^2
 - Re-normalize

Recall: Floating Point Addition

- Numbers maintain only 4 decimal digits and 2 exponent digits

- $9.999 \times 10^1 + 1.610 \times 10^{-1}$

- Convert to the larger exponent

- $9.999 \times 10^1 + 0.016 \times 10^1$

- Add

- 10.015×10^1

- Normalize

- 1.0015×10^2

- Check for overflow/underflow

- Round

- 1.002×10^2

- Re-normalize

If we had more fraction bits,
these errors would be minimized



Floating Point Addition

- Numbers maintain only 4 binary digits and 2 exponent digits
 - $1.010 \times 2^1 + 1.100 \times 2^3$
 - Convert to the larger exponent
 - $0.0101 \times 2^3 + 1.100 \times 2^3$
 - Add
 - 1.1101×2^3
 - Normalize
 - 1.1101×2^3
 - Check for overflow/underflow

Floating Point Addition

- Numbers maintain only 4 binary digits and 2 exponent digits

- $1.010 \times 2^1 + 1.100 \times 2^3$

- Convert to the larger exponent

- $0.0101 \times 2^3 + 1.100 \times 2^3$

- Add

- 1.1101×2^3

- Normalize

- 1.1101×2^3

- Check for overflow/underflow

- IEEE 754 format (32-bit)

0 10000010 110100000000000000000000

Floating Point Addition

- ❑ **Example:** add the following two single-precision floating point numbers.

[illegible][illegible]

Steps:

1. Convert to larger exponent
2. Add
3. Normalize
4. Round

Floating Point Addition

- **Example:** add the following two single-precision floating point numbers.

A: 01000000011000000000000000000000

B: 011000011010011000000000000000

$E_A = 128$ $M_A = 1.11_{\text{two}}$

$$E_B = 131 \quad M_B = 1.010011_{\text{two}}$$

$E_A = 131$ $M_A = 0.00111_{\text{two}}$

$$E_B = 131 \quad M_B = 1.010011_{\text{two}}$$

$$E_A = E_B = 131$$

$$M_A + M_B = 0.00111_{\text{two}} + 1.010011_{\text{two}} = 1.100001_{\text{two}}$$

[illegible]

Floating Point Multiplication

- Similar steps are required for multiplication
 - ▣ Compute exponent
 - Need to remove bias
 - ▣ Multiply significands
 - May end up unnormalized
 - ▣ Normalize
 - Shift the point
 - ▣ Round
 - Fit in the number of bits
 - ▣ Assign sign
 - Compute sign

Floating Point Multiplication

- **Example:** multiply the following two single-precision floating point numbers.

[illegible][illegible]

Steps:

1. Compute exponent
2. Multiply significands
3. Normalize
4. Round
5. Compute sign

Floating Point Multiplication

- **Example:** multiply the following two single-precision floating point numbers.

[illegible][illegible]

$E_A = 128$ $M_A = 1.11_{\text{two}}$

$$E_B = 131 \quad M_B = 1.010011_{\text{two}}$$

$$E_{A \times B} = 128 + 131 - \mathbf{127} = 132 \quad M_{A \times B} = 10.01000101_{\text{two}}$$

$$E_{A \times B} = 133$$

$$M_{A \times B} = 1.001000101_{\text{two}}$$

A x B: 11000010100100010100000000000000

Floating Point Instructions

- MIPS employs separate registers for floating point
 - ▣ 32-bit registers: \$f0, \$f1, ..., \$f31.
 - ▣ Each register represents a single-precision number
 - ▣ Register pairs are used for double-precision
 - Example: \$f0 refers to {\$f0, \$f1}

Example	Meaning	Comments
add.s \$f2,\$f4,\$f6	$\$f2 = \$f4 + \$f6$	FP add (single precision)
sub.s \$f2,\$f4,\$f6	$\$f2 = \$f4 - \$f6$	FP sub (single precision)
mul.s \$f2,\$f4,\$f6	$\$f2 = \$f4 \times \$f6$	FP multiply (single precision)
div.s \$f2,\$f4,\$f6	$\$f2 = \$f4 / \$f6$	FP divide (single precision)
add.d \$f2,\$f4,\$f6	$\$f2 = \$f4 + \$f6$	FP add (double precision)
sub.d \$f2,\$f4,\$f6	$\$f2 = \$f4 - \$f6$	FP sub (double precision)
mul.d \$f2,\$f4,\$f6	$\$f2 = \$f4 \times \$f6$	FP multiply (double precision)
div.d \$f2,\$f4,\$f6	$\$f2 = \$f4 / \$f6$	FP divide (double precision)

Floating Point Instructions

- Load/Store instructions by coprocessor 1 (c1)
 - ▣ Still use integer registers for address computation
- Comparison instructions
 - ▣ Set an internal bit (**cond**) to be inspected by branch instructions

Example	Meaning	Comments
lwc1 \$f1,100(\$s2)	\$f1 = Memory[\$s2 + 100]	32-bit data to FP register
swc1 \$f1,100(\$s2)	Memory[\$s2 + 100] = \$f1	32-bit data to memory
bc1t 25	if (cond == 1) go to PC + 4 + 100	PC-relative branch if FP cond.
bc1f 25	if (cond == 0) go to PC + 4 + 100	PC-relative branch if not cond.
c.lt.s \$f2,\$f4	if (\$f2 < \$f4) cond = 1; else cond = 0	FP compare less than single precision
c.lt.d \$f2,\$f4	if (\$f2 < \$f4) cond = 1; else cond = 0	FP compare less than double precision

Code Example

- Convert a temperature in Fahrenheit to Celsius

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

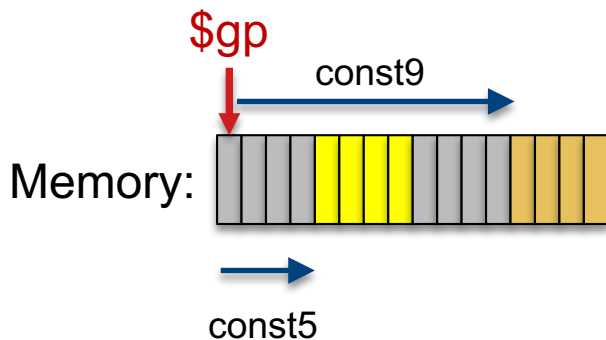
- Assume that constants are stored in global memory

Code Example

- Convert a temperature in Fahrenheit to Celsius

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

- Assume that constants are stored in global memory

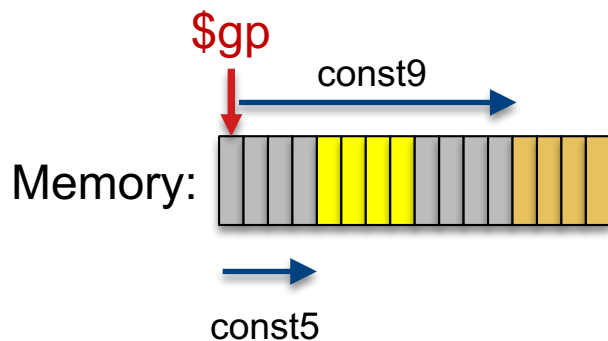


Code Example

- Convert a temperature in Fahrenheit to Celsius

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

- Assume that constants are stored in global memory



```
f2c: mtc1    $a0, $f12  
      lwc1    $f16, const5($gp)  
      lwc1    $f18, const9($gp)  
      div.s   $f16, $f16, $f18  
      lwc1    $f18, const32($gp)  
      sub.s   $f18, $f12, $f18  
      mul.s   $f0, $f16, $f18  
      jr      $ra
```