Lecture 10: Floating Point, Digital Design

- Today's topics:
  - FP arithmetic
  - Intro to Boolean functions

Examples

Final representation: $(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- Represent -0.75_{ten} in single and double-precision formats

  Single: (1 + 8 + 23)

  Double: (1 + 11 + 52)

- What decimal number is represented by the following single-precision number?

  1 1000 0001 01000…0000

Examples

Final representation: $(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- Represent -0.75_{ten} in single and double-precision formats

  Single: (1 + 8 + 23)
  1 0111 1110 1000…000

  Double: (1 + 11 + 52)
  1 0111 1111 110 1000…000

- What decimal number is represented by the following single-precision number?
  1 1000 0001 01000…0000
  -5.0

Details

- The number “0” has a special code so that the implicit 1 does not get added: the code is all 0s
  (it may seem that this takes up the representation for 1.0, but given how the exponent is represented, that’s not the case)
  (see discussion of denorms (pg. 222) in the textbook)

- The largest exponent value (with zero fraction) represents +/- infinity

- The largest exponent value (with non-zero fraction) represents NaN (not a number) – for the result of 0/0 or (infinity minus infinity)

- Note that these choices impact the smallest and largest numbers that can be represented

FP Addition

- Consider the following decimal example (can 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

FP Addition

- Consider the following decimal example (can 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}

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

Check for overflow/underflow

Round

1.002×10^{2}

Re-normalize

FP Addition – Binary Example

- Consider the following binary example

$1.010 \times 2^{1} + 1.100 \times 2^{3}$

Convert to the larger exponent:

$0.0101 \times 2^{3} + 1.1000 \times 2^{3}$

Add

1.1101×2^{3}

Normalize

1.1101×2^{3}

Check for overflow/underflow

Round

Re-normalize

IEEE 754 format: 0 10000010 11010000000000000000000

FP Multiplication

- Similar steps:
  - Compute exponent (careful!)
  - Multiply significands (set the binary point correctly)
  - Normalize
  - Round (potentially re-normalize)
  - Assign sign

MIPS Instructions

- The usual add.s, add.d, sub, mul, div
- Comparison instructions: c.eq.s, c.neq.s, c.lt.s….
  These comparisons set an internal bit in hardware that is then inspected by branch instructions: bc1t, bc1f
- Separate register file $f0 - $f31 : a double-precision value is stored in (say) $f4-$f5 and is referred to by $f4
- Load/store instructions (lwc1, swc1) must still use integer registers for address computation

Code Example

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

```
(argument fahr is stored in $f12)
 lwc1   $f16, const5
 lwc1   $f18, const9
 div.s  $f16, $f16, $f18
 lwc1   $f18, const32
 sub.s  $f18, $f12, $f18
 mul.s  $f0, $f16, $f18
 jr     $ra
```

Fixed Point

- FP operations are much slower than integer ops
- Fixed point arithmetic uses integers, but assumes that every number is multiplied by the same factor
- Example: with a factor of 1/1000, the fixed-point representations for 1.46, 1.7198, and 5624 are respectively 1460, 1720, and 5624000
- More programming effort and possibly lower precision for higher performance

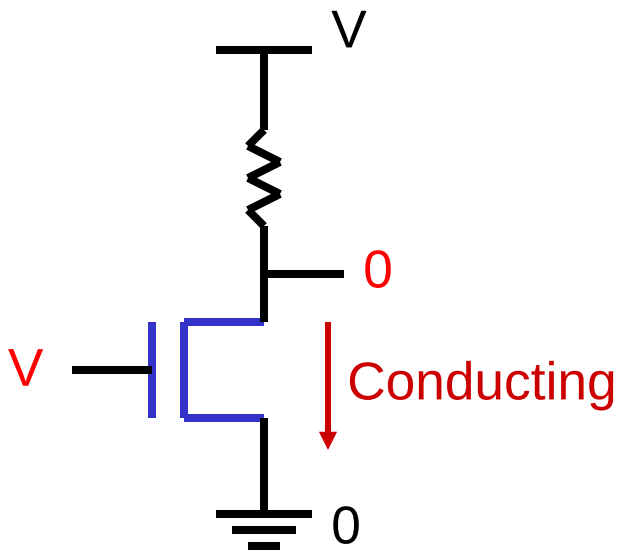
Subword Parallelism

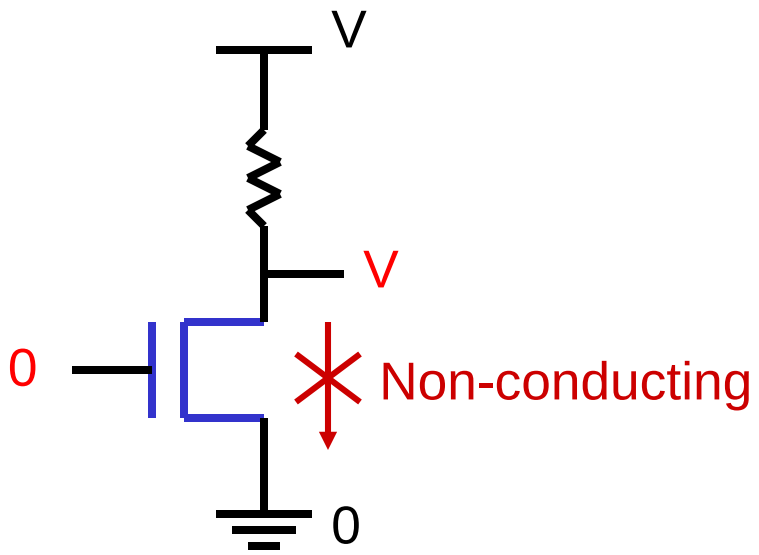
- ALUs are typically designed to perform 64-bit or 128-bit arithmetic
- Some data types are much smaller, e.g., bytes for pixel RGB values, half-words for audio samples
- Partitioning the carry-chains within the ALU can convert the 64-bit adder into 4 16-bit adders or 8 8-bit adders
- A single load can fetch multiple values, and a single add instruction can perform multiple parallel additions, referred to as subword parallelism

Digital Design Basics

- Two voltage levels – high and low (1 and 0, true and false)
  Hence, the use of binary arithmetic/logic in all computers

- A transistor is a 3-terminal device that acts as a switch





Logic Blocks

- A logic block has a number of binary inputs and produces a number of binary outputs – the simplest logic block is composed of a few transistors
- A logic block is termed *combinational* if the output is only a function of the inputs
- A logic block is termed *sequential* if the block has some internal memory (state) that also influences the output
- A basic logic block is termed a *gate* (AND, OR, NOT, etc.)

We will only deal with combinational circuits today

Truth Table

- A truth table defines the outputs of a logic block for each set of inputs

- Consider a block with 3 inputs A, B, C and an output E that is true only if *exactly* 2 inputs are true

A	B	C	E

Truth Table

- A truth table defines the outputs of a logic block for each set of inputs

- Consider a block with 3 inputs A, B, C and an output E that is true only if *exactly* 2 inputs are true

A	B	C	E
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Can be compressed by only representing cases that have an output of 1

Boolean Algebra

- Equations involving two values and three primary operators:
  - OR : symbol + , X = A + B ➔ X is true if at least one of A or B is true
  - AND : symbol . , X = A . B ➔ X is true if both A and B are true
  - NOT : symbol $\bar{}$, X = $\bar{A}$ ➔ X is the inverted value of A

Boolean Algebra Rules

- Identity law : $A + 0 = A \quad ; \quad A \cdot 1 = A$
- Zero and One laws : $A + 1 = 1 \quad ; \quad A \cdot 0 = 0$
- Inverse laws : $A \cdot \overline{A} = 0 \quad ; \quad A + \overline{A} = 1$
- Commutative laws : $A + B = B + A \quad ; \quad A \cdot B = B \cdot A$
- Associative laws : $A + (B + C) = (A + B) + C$
  $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- Distributive laws : $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
  $A + (B \cdot C) = (A + B) \cdot (A + C)$

DeMorgan's Laws

- $\overline{A + B} = \overline{A} \cdot \overline{B}$

- $\overline{A \cdot B} = \overline{A} + \overline{B}$

- Confirm that these are indeed true

Title

- Bullet