Topic V20

Computer Arithmetic:
Addition and Subtraction
Readings: (Section 3.2)

Arithmetic for Computers

Operations on integers

Addition and subtraction

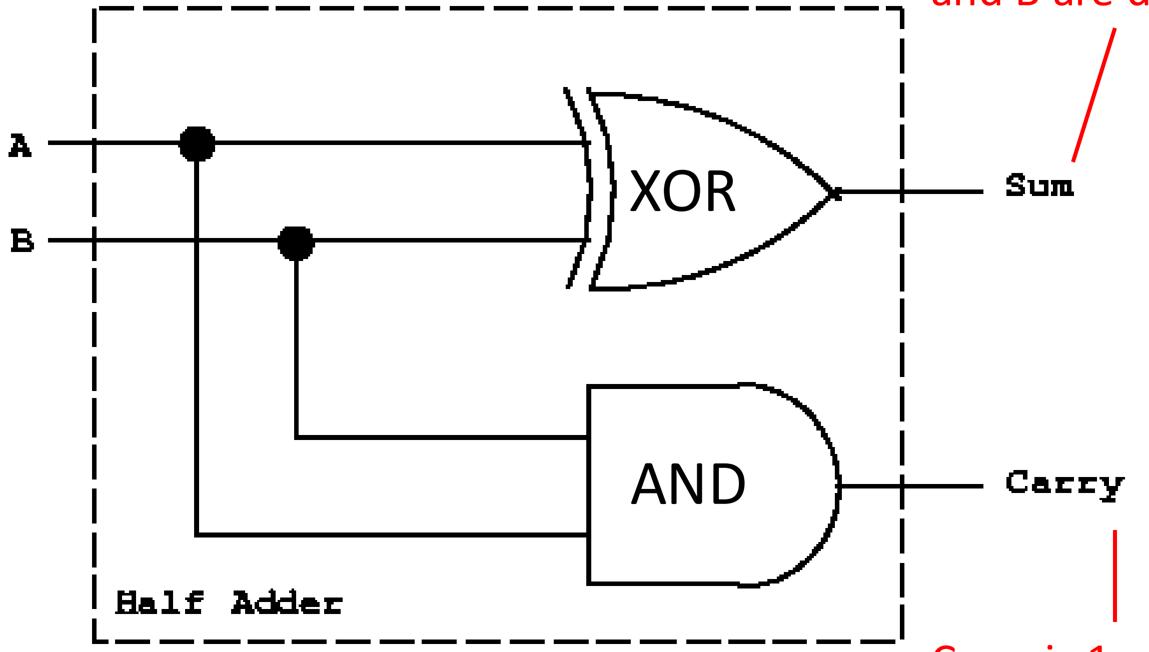
Multiplication and division

Dealing with overflow

Floating-point real numbers

Representation and operations

Bit-wise Addition



Sum is 1 when A and B are different

Carry is 1 when A and B are both 1

$$\begin{array}{lrr} \text{Carry} & 0 & \\ & & 0 \\ & + & 0 \\ \hline \text{Sum} & & 0 \end{array}$$

$$\begin{array}{rr} 0 & \\ & 0 \\ + & 1 \\ \hline & 1 \end{array}$$

$$\begin{array}{rr} 0 & \\ & 1 \\ + & 0 \\ \hline & 1 \end{array}$$

$$\begin{array}{rr} 1 & \\ & 1 \\ + & 1 \\ \hline & 0 \end{array}$$

Adding three bits

1 Cin

1 A

+ 1 B

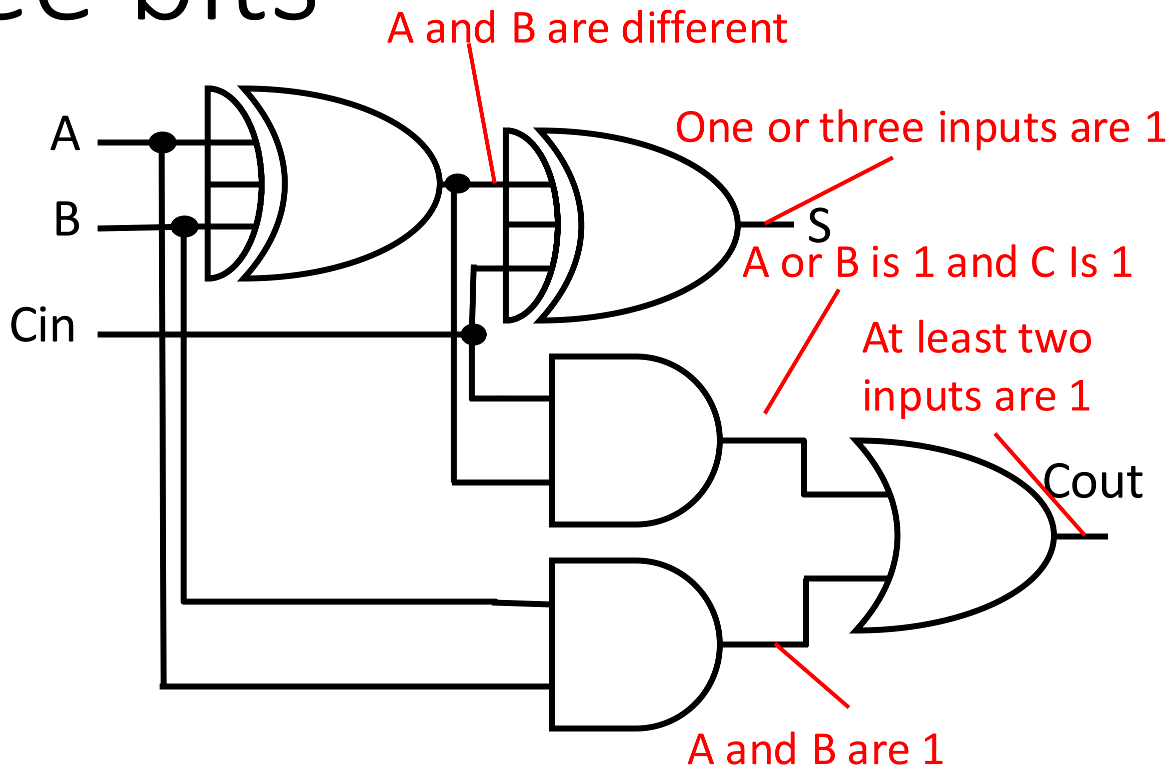
Adding three bits

```
    1         1 1  Cin
    0           1  A
+   1     +     1  B
-----     -------
                1
```

Adding three bits

```
  0 0        0 0        1 1        1 1  Cin
    0          0          0          1  A
+   0      +   1      +   1      +   1  B
-----      -----      -----      -----
    0          1          0          1
```

Cin	ai	bi	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



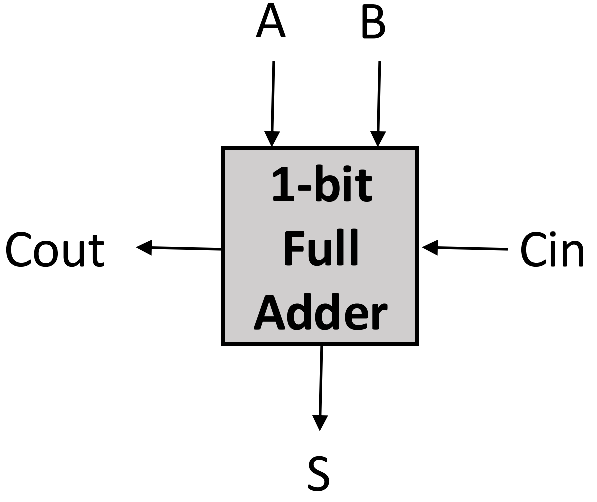
A and B are different

One or three inputs are 1

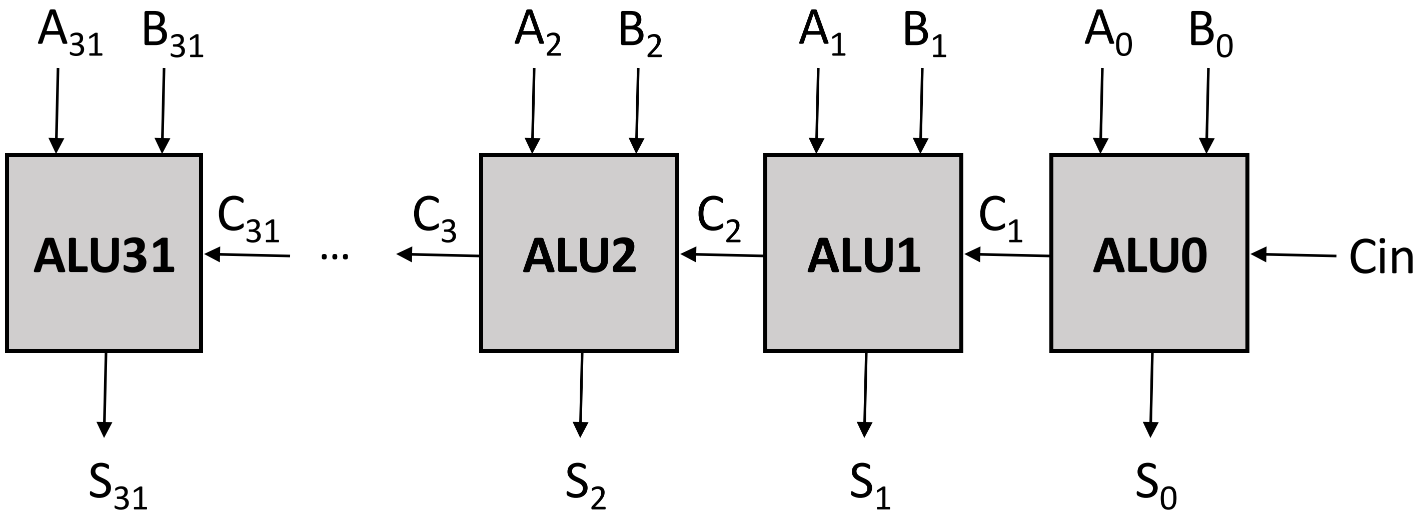
A or B is 1 and C Is 1

At least two inputs are 1

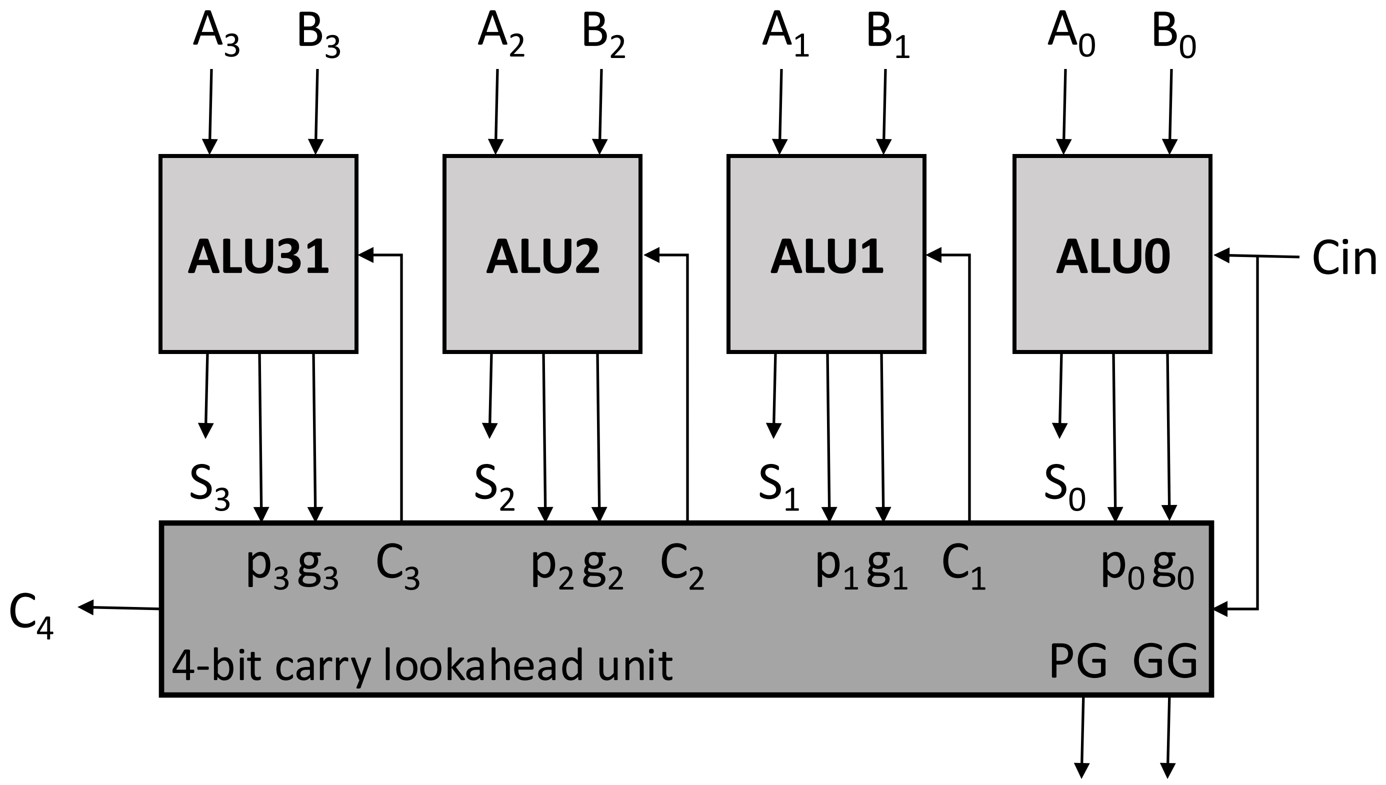
A and B are 1



Ripple-carry Adder



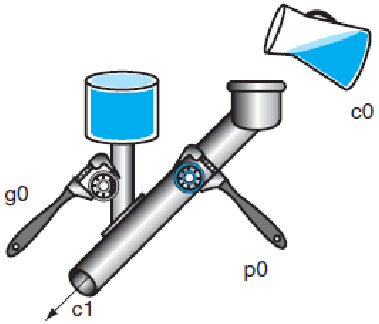
Carry Lookahead Adder

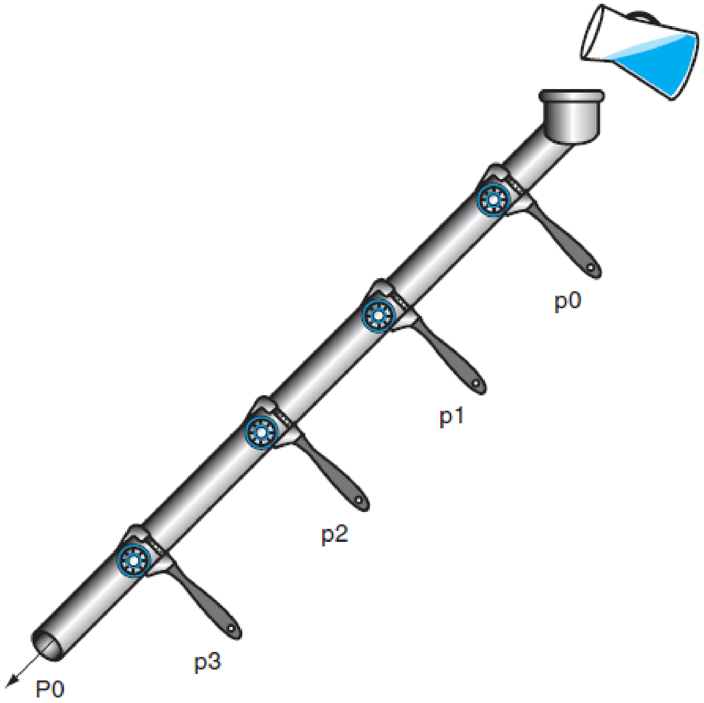


Generate: $g_i = (A_i \cdot B_i)$

Propagate: $p_i = (A_i + B_i)$ $C_{i+1} = g_i + p_i \cdot c_i$

Carry Lookahead





Overflow

```
  0111
+0001
______
```

Overflow happened

Sign of the result is changed

Integer Addition

Example: 7 + 6

```
  …000111
+ …000110
---------
```

Integer Addition

Example: 7 + 6

```
   00110
  …000111
+ …000110
---------
  …001101
```

Overflow if result out of range

A + B		A	
		Positive	Negative
B	Positive		
	Negative		

Integer Subtraction

Add negation of second operand

Example: 7 – 6 = 7 + (-6)

$$\begin{array}{rr}
 & 1\,1\,1\,1\,1\,1\,1\,1\ldots1\,1\,1\,1\,1\,1\,0 \\
+7 & 0\,0\,0\,0\,0\,0\,0\,0\ldots0\,0\,0\,0\,0\,1\,1\,1 \\
-6 & +\,1\,1\,1\,1\,1\,1\,1\,1\ldots1\,1\,1\,1\,1\,0\,1\,0 \\
\hline
+1 & 0\,0\,0\,0\,0\,0\,0\,0\ldots0\,0\,0\,0\,0\,0\,0\,1
\end{array}$$

Subtraction overflows if result out of range

A - B		A	
		Positive	Negative
B	Positive	No overflow	If (sign == 0) overflow
	Negative	If (sign == 1) overflow	No overflow

Dealing with Overflow

Some languages (e.g., C) ignore overflow

Use RISC-V add, addi, sub instructions with no checks

Other languages (e.g., Ada, Fortran) require raising an exception

RISC-V has no special instructions support for overflow checks on integer arithmetic operations

Use RISC-V add, addi, sub instructions with a check

On overflow, invoke exception handler

Save PC in user exception program counter (uepc)

Jump to handler address which is stored in user trap handler base address (utvec)

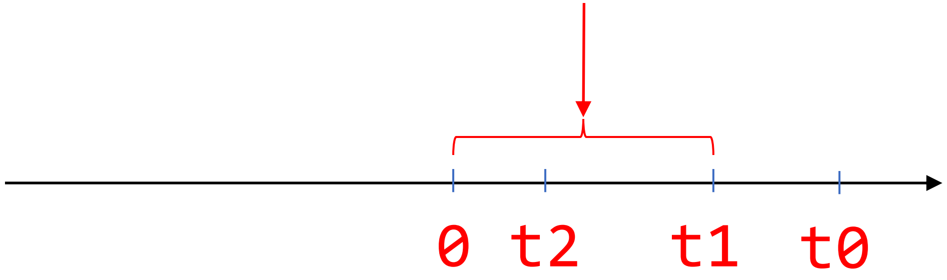
Dealing with Overflow (examples)

Unsigned addition:

```
add      t0, t1, t2
bltu     t0, t1, overflow
```

RISC-V does not have an `addu` instruction (there is no automatic overflow detection)

If `t0` is here, there was overflow



Example:

t1 = 0x80000001

t2 = 0x80000001

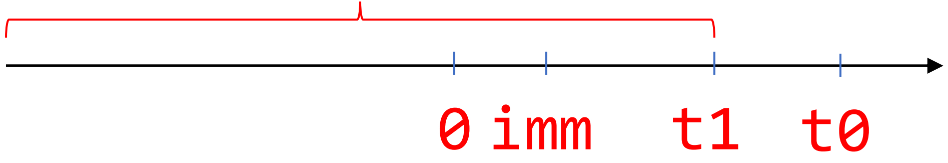
…

t0 = 0x00000002

Dealing with Overflow (examples)

Signed addition
(with one operand sign known):

```
addi      t0, t1, +imm
blt       t0, t1, overflow
```



If t0 is here, there was overflow

Example:

t1 = 0x7FFFFFF1

imm = 0x0000000F

…

t0 = 0x80000000

know that imm is positive

t0 is negative

Dealing with Overflow (examples)

Signed addition
(general case):

```
add      t0, t1, t2   # c = a + b
slti     t3, t2, 0    # if b < 0
slt      t4, t0, t1   # if c < a
bne      t3, t4, overflow
```

Example 1:

t1 = 0x7FFFFFF1

t2 = 0x0000000F   b < 0 is false

…

t0 = 0x80000000   c < a is true

Example 2:

t1 = 0x80000002

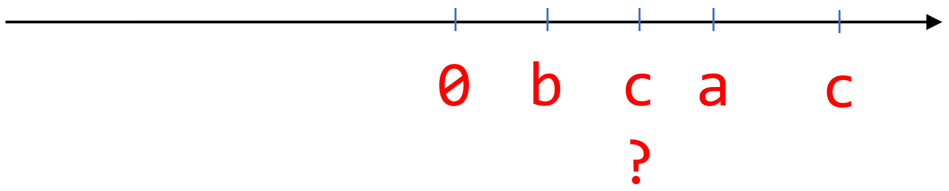
t2 = 0x80000004   b < 0 is true

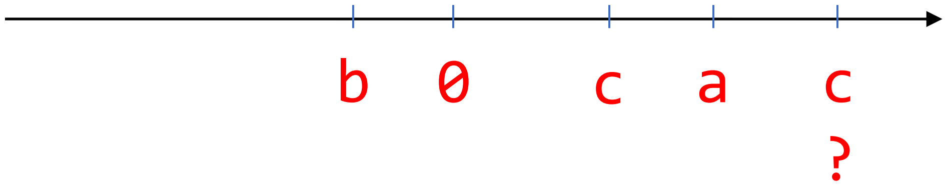
…

t0 = 0x00000006   c < a is false

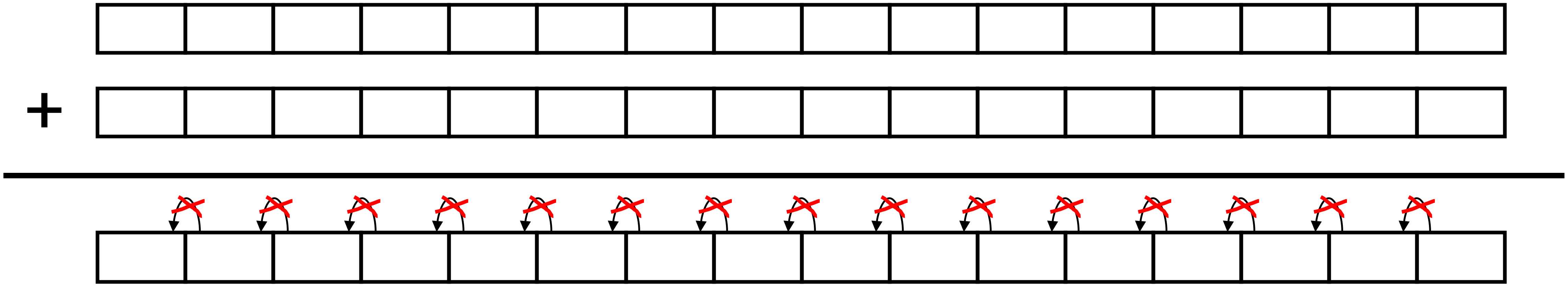
```
add     t0, t1, t2  # c = a+b
slti    t3, t2, 0   # if b<0
slt     t4, t0, t1  # if c<a
bne     t3, t4, overflow
```

c = a+b			
b			





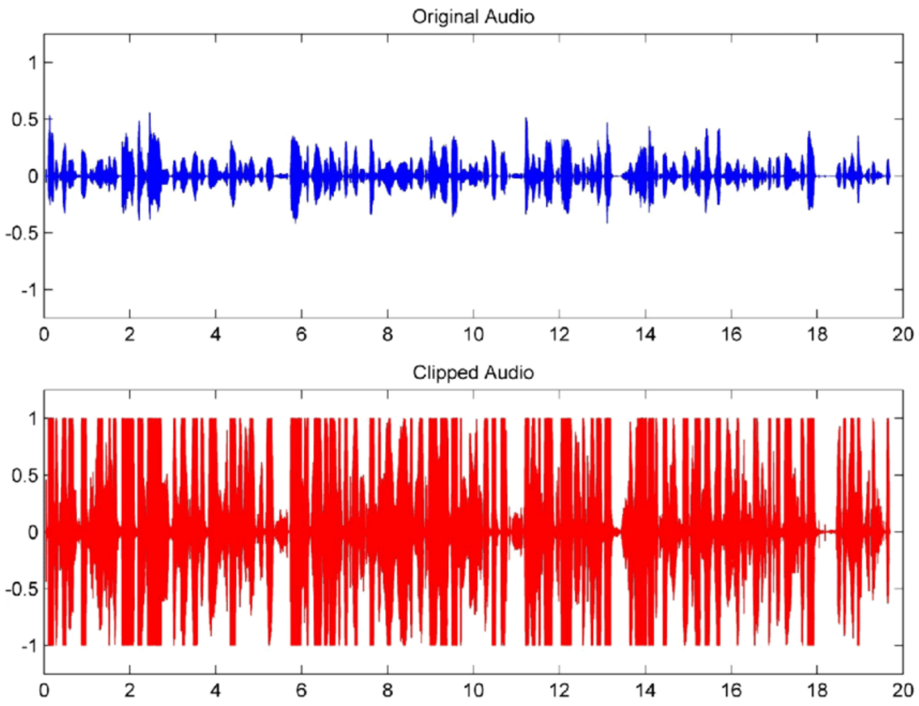
Arithmetic for Multimedia



Graphics and media processing operates on vectors of 8-bit and 16-bit data

- Use 64-bit adder, with partitioned carry chain
  - Operate on 8 x 8-bit, 4 x 16-bit, or 2 x 32-bit vectors
- SIMD (single-instruction, multiple-data)

Arithmetic for Multimedia



Saturating operations

On overflow, result is largest representable value

c.f. 2s-complement modulo arithmetic

E.g., clipping in audio, saturation in video