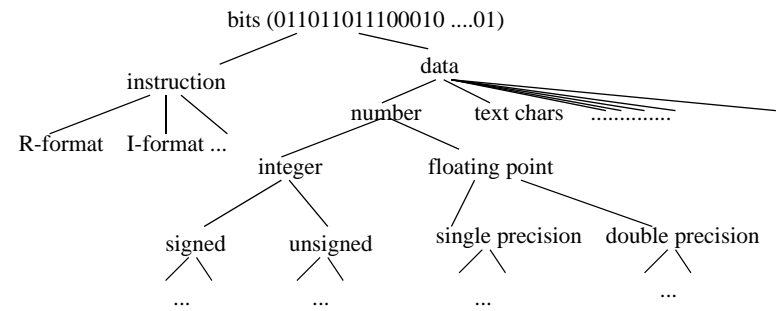


Number Systems and Arithmetic

or

Computers go to elementary school

What do all those bits mean now?



Questions About Numbers

- How do you represent
 -
 -
 - really large numbers?
 - really small numbers?
- How do you
 - do arithmetic?
 - identify (e.g. overflow)?
- What is an ALU and what does it look like?
 - ALU=arithmetic logic unit

Introduction to Binary Numbers

Consider a 4-bit binary number

Decimal	Binary	Decimal	Binary
0	0000	4	0100
1	0001	5	0101
2	0010	6	0110
3	0011	7	0111

Examples of binary arithmetic:

$$\begin{array}{r}
 3 + 2 = 5 \\
 \begin{array}{r}
 0 \ 0 \ 1 \ 1 \\
 + 0 \ 0 \ 1 \ 0 \\
 \hline
 \end{array}
 \end{array}$$

$$\begin{array}{r}
 3 + 3 = 6 \\
 \begin{array}{r}
 0 \ 0 \ 1 \ 1 \\
 + 0 \ 0 \ 1 \ 1 \\
 \hline
 \end{array}
 \end{array}$$

Negative Numbers?

- We would like a _____ that provides
 - obvious representation of 0,1,2...
 - uses adder for addition
 - single value of 0
 - equal coverage of positive and negative numbers
 - easy detection of sign
 - easy negation

Some Alternatives

- Sign Magnitude -- MSB is sign bit, rest the same
 - 1 == 1001
 - 5 == 1101
- One's complement -- flip all bits to negate
 - 1 == 1110
 - 5 == 1010

Two's Complement Representation

- 2's complement representation of negative numbers
 - Take the bitwise inverse and add 1
- Biggest 4-bit Binary Number: 7 Smallest 4-bit Binary Number: -8

<u>Decimal</u>	<u>Two's Complement Binary</u>
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Two's Complement Arithmetic

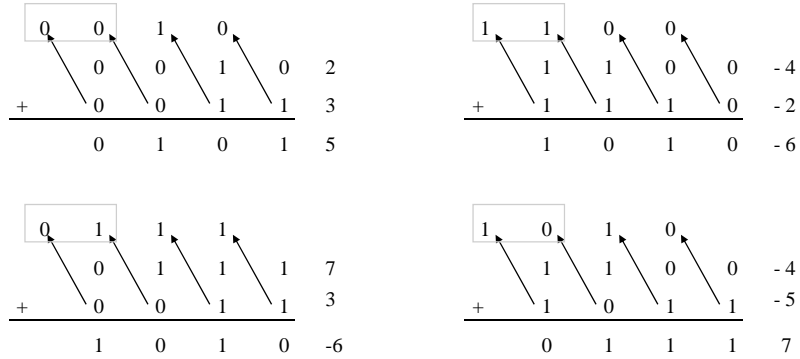
Decimal	2's Complement Binary	Decimal	2's Complement Binary
0	0000	-1	1111
1	0001	-2	1110
2	0010	-3	1101
3	0011	-4	1100
4	0100	-5	1011
5	0101	-6	1010
6	0110	-7	1001
7	0111	-8	1000

- Examples: $7 - 6 = 7 + (-6) = 1$ $3 - 5 = 3 + (-5) = -2$

$$\begin{array}{r}
 0 \quad 1 \quad 1 \quad 1 \\
 + \quad 1 \quad 0 \quad 1 \quad 0 \\
 \hline
 \end{array}$$

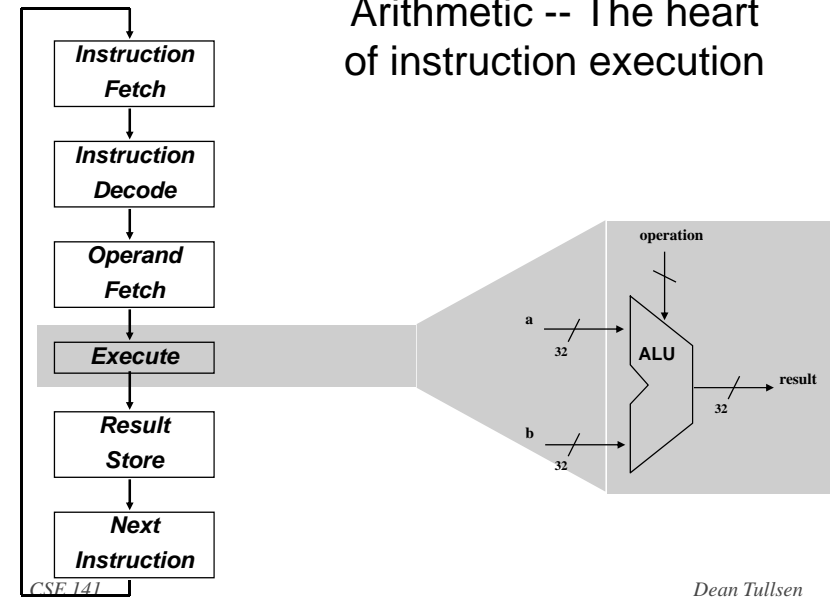
$$\begin{array}{r}
 0 \quad 0 \quad 1 \quad 1 \\
 + \quad 1 \quad 0 \quad 1 \quad 1 \\
 \hline
 \end{array}$$

Overflow Detection

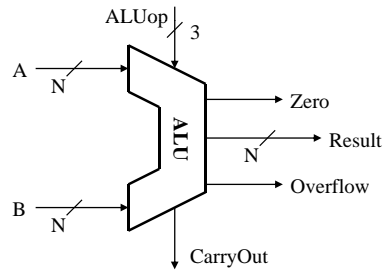


So how do we detect overflow?

Arithmetic -- The heart of instruction execution



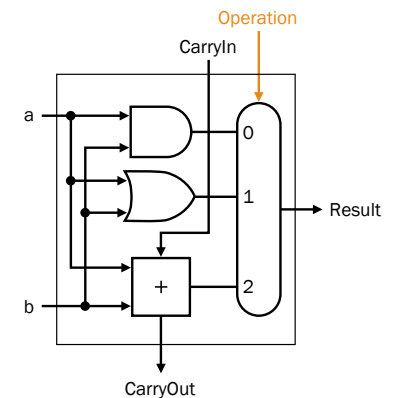
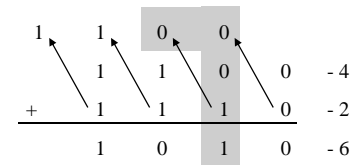
Designing an Arithmetic Logic Unit



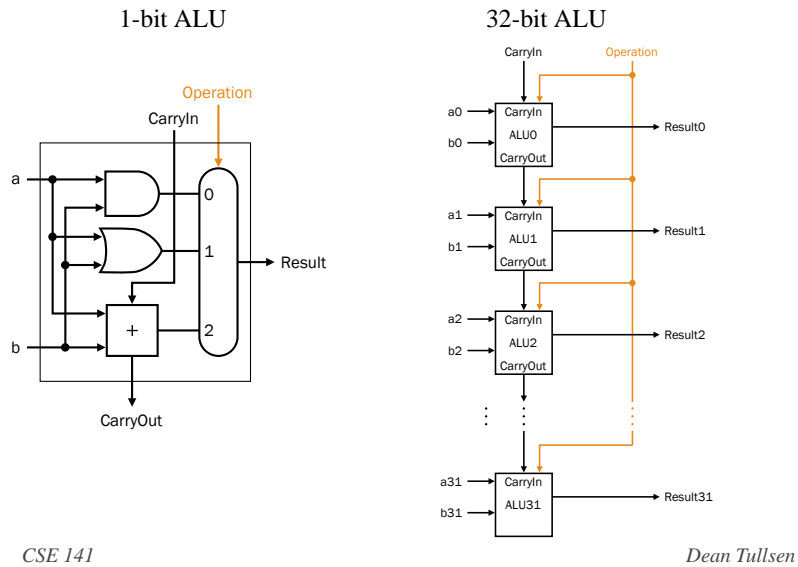
ALU Control Lines (ALUOp)	Function
000	And
001	Or
010	Add
110	Subtract
111	Set-on-less-than

A One Bit ALU

- This 1-bit ALU will perform AND, OR, and ADD

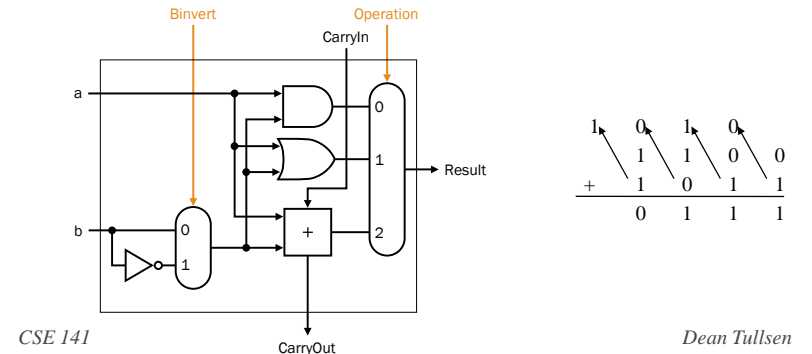


A 32-bit ALU



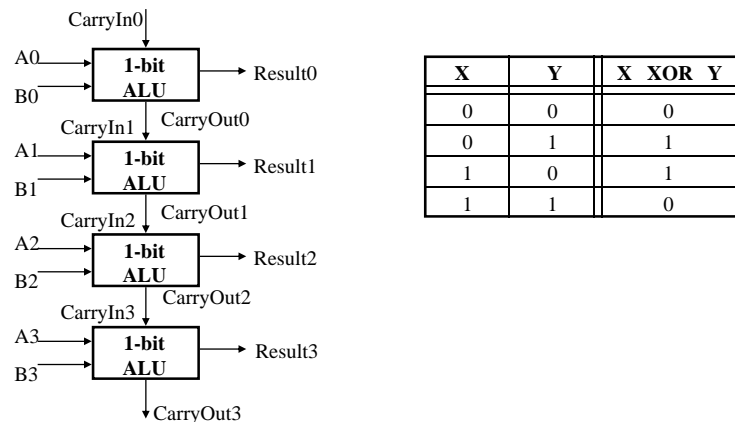
How About Subtraction?

- Keep in mind the following:
 - (A - B) is the same as: 2's Complement negate: Take the bit-wise inverse of every bit and add 1
- Bit-wise inverse of B is !B:
 - $A - B = A + (-B) = A + (!B + 1) = A + !B + 1$



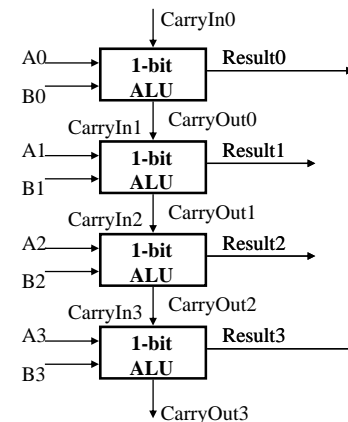
Overflow Detection Logic

- Carry into MSB != Carry out of MSB
 - For a N-bit ALU: $Overflow = CarryIn[N - 1] \text{ XOR } CarryOut[N - 1]$



Zero Detection Logic

- Zero Detection Logic is just one BIG NOR gate
 - Any non-zero input to the NOR gate will cause its output to be zero

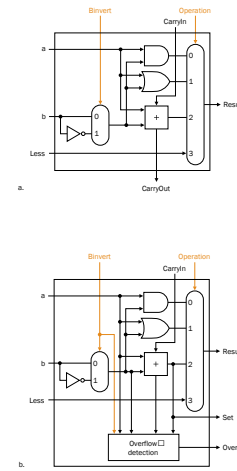


Set-on-less-than

- Do a subtract
- use sign bit
 - route to bit 0 of result
 - all other bits zero

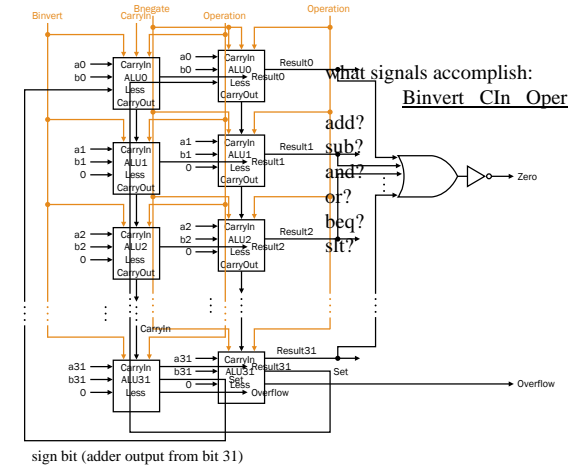
CSE 141

Dean Tullsen



CSE 141

Full ALU

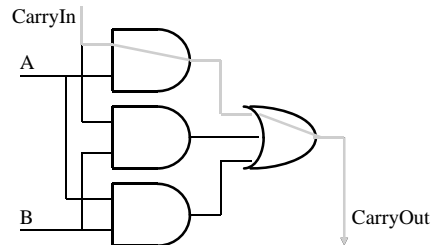
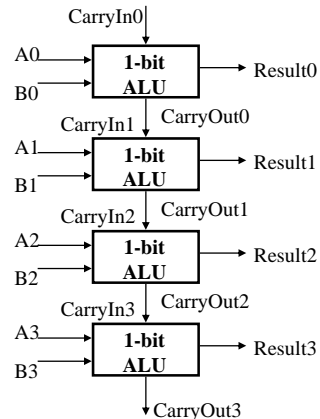


sign bit (adder output from bit 31)

Dean Tullsen

The Disadvantage of Ripple Carry

- The adder we just built is called a “Ripple Carry Adder”
 - The carry bit may have to propagate from LSB to MSB
 - Worst case delay for an N-bit RC adder: 2N-gate delay



The point -> ripple carry adders are slow. Faster addition schemes are possible that accelerate the movement of the carry from one end to the other.

- Paper and pencil example:

```

Multiplicand      1001
Multiplier        x 1011
-----

```

Product = ?

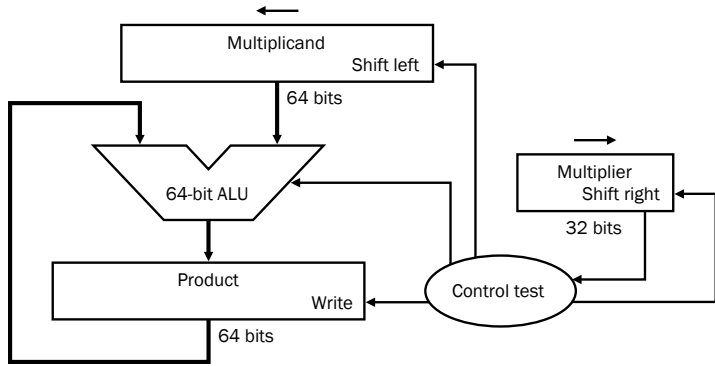
- m bits x n bits = m+n bit product
- Binary makes it easy:
 - 0 => place 0 (0 x multiplicand)
 - 1 => place multiplicand (1 x multiplicand)

CSE 141

Dean Tullsen

MULTIPLY HARDWARE

- 64-bit Multiplicand reg, 64-bit ALU, 64-bit Product reg, 32-bit multiplier reg



CSE 141

Dean Tullsen

Observations on Multiply

- MIPS registers Hi and Lo are left and right half of Product
- Gives us MIPS instruction MultU
- What about signed multiplication?
 - easiest solution is to make both positive & remember whether to complement product when done.

CSE 141

Dean Tullsen

Divide: Paper & Pencil

Divisor 1000 $\overline{) 1101010}$

Quotient
Dividend

Remainder

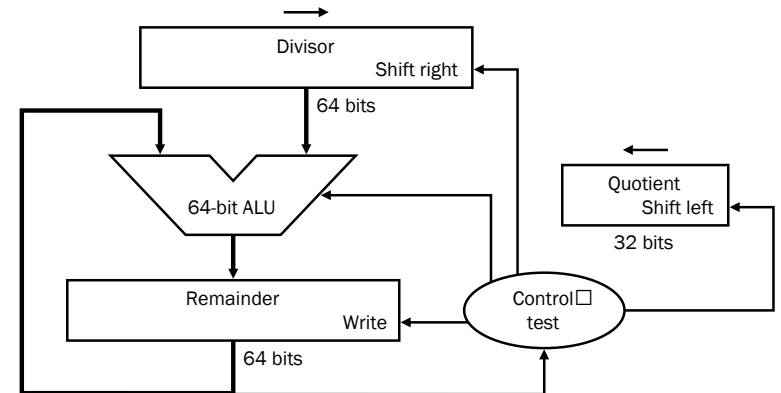
- See how big a number can be subtracted, creating quotient bit on each step
 - Binary $\Rightarrow 1 * \text{divisor}$ or $0 * \text{divisor}$
- Dividend = Quotient x Divisor + Remainder

CSE 141

Dean Tullsen

DIVIDE HARDWARE

- 64-bit Divisor reg, 64-bit ALU, 64-bit Remainder reg, 32-bit Quotient reg



CSE 141

Dean Tullsen

Divide Hardware

- Hi and Lo registers in MIPS combine to act as 64-bit register for multiply and divide
- Signed Divides: Simplest is to remember signs, make positive, and complement quotient and remainder if necessary
 - Note: Dividend and Remainder must have same sign
 - Note: Quotient negated if Divisor sign & Dividend sign disagree

Key Points

- Instruction Set drives the ALU design
- ALU performance, CPU clock speed driven by adder delay
- Multiplication and division take much longer than addition, requiring multiple addition steps.

So Far

- Can do logical, add, subtract, multiply, divide, ...
- But.....
 - what about fractions?
 - what about really large numbers?

Binary Fractions

$$1011_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$$

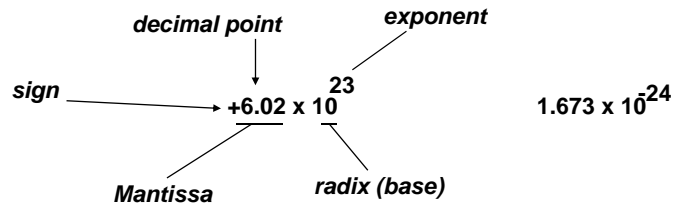
so...

$$101.011_2 = 1x2^2 + 0x2^1 + 1x2^0 + 0x2^{-1} + 1x2^{-2} + 1x2^{-3}$$

e.g.,

$$.75 = 3/4 = 3/2^2 = 1/2 + 1/4 = .11$$

Recall Scientific Notation

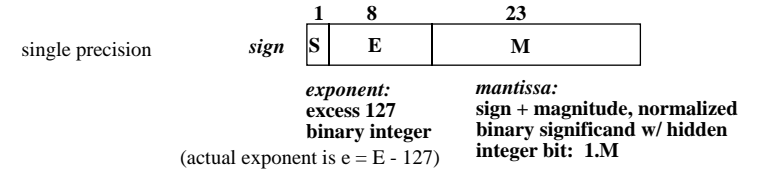


Issues:

- Arithmetic (+, -, *, /)
- Representation, Normal form
- Range and Precision
- Rounding
- Exceptions (e.g., divide by zero, overflow, underflow)
- Errors
- Properties (negation, inversion, if A = B then A - B = 0)

Floating-Point Numbers

Representation of floating point numbers in IEEE 754 standard:



$$N = (-1)^S 2^{E-127} (1.M) \quad 0 < E < 255$$

$$0 = 0\ 00000000\ 0\ \dots\ 0 \quad -1.5 = 1\ 01111111\ 10\ \dots\ 0$$

$$325 = 101000101\ X\ 2^0 = 1.01000101\ X\ 2^8$$

$$= 0\ 10000111\ 010001010000000000000000$$

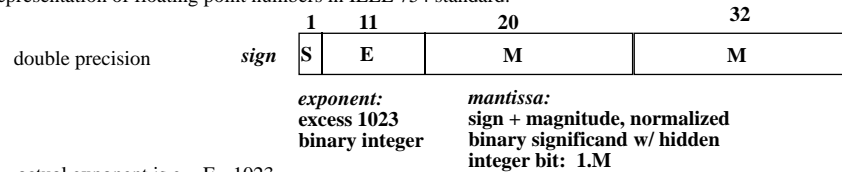
$$.02 = .0011001101100\dots\ X\ 2^0 = 1.1001101100\dots\ X\ 2^{-3}$$

$$= 0\ 01111100\ 1001101100\dots$$

- range of about 2×10^{-38} to 2×10^{38}
- always normalized (so always leading 1, thus never shown)
- special representation of 0 (E = 00000000) (why?)
- can do integer compare for greater-than, sign

Double Precision Floating Point

Representation of floating point numbers in IEEE 754 standard:



actual exponent is $e = E - 1023$

$$N = (-1)^S 2^{E-1023} (1.M) \quad 0 < E < 2048$$

- 52 (+1) bit mantissa
- range of about 2×10^{-308} to 2×10^{308}

Floating Point Addition

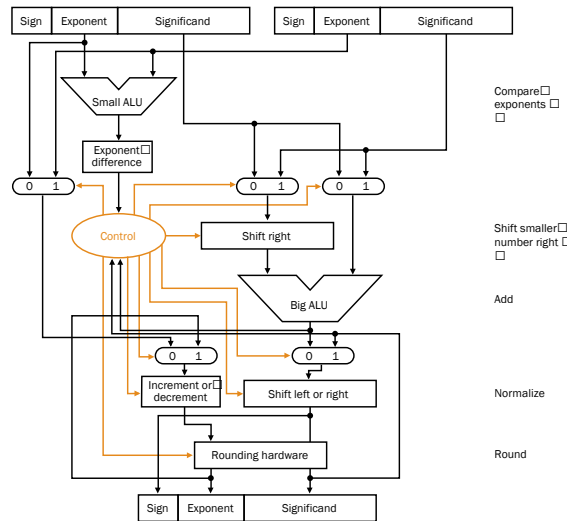
- How do you add in scientific notation?

$$9.962 \times 10^4 + 5.231 \times 10^2$$

- Basic Algorithm

1. Align
2. Add
3. Normalize
4. Round

FP Addition Hardware



CSE 141

Dean Tullsen

Floating Point Multiplication

- How do you multiply in scientific notation?

$$(9.9 \times 10^4)(5.2 \times 10^2) = 5.148 \times 10^7$$

- Basic Algorithm

1. Add exponents
2. Multiply
3. Normalize
4. Round
5. Set Sign

CSE 141

Dean Tullsen

FP Accuracy

- Extremely important in scientific calculations
- Very tiny errors can accumulate over time
- IEEE 754 FP standard has four rounding modes
 - always round up (toward $+\infty$)
 - always round down (toward $-\infty$)
 - truncate
 - round to nearest
 - => in case of tie, round to nearest even
- Requires extra bits in intermediate representations

CSE 141

Dean Tullsen

Key Points

- Floating Point extends the range of numbers that can be represented, at the expense of precision (accuracy).
- FP operations are very similar to integer, but with pre- and post-processing.
- Rounding implementation is critical to accuracy over time.

CSE 141

Dean Tullsen