



Computer System Design Lecture 12: Computer Arithmetic

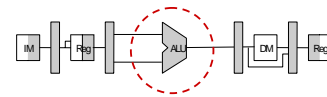
Prof. R. Iris Bahar
EN164
February 23, 2007

Reading: Appendix I, I.1, I.2, I.8



Arithmetic

- Where we've been:
 - Performance (seconds, cycles, instructions)
 - Instruction Set Architecture
 - Basic pipelining
- What's up ahead:
 - Implementing the Architecture

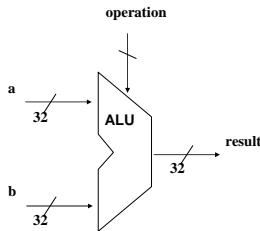


EN164
Lecture 11-2



Arithmetic

- We start with the Arithmetic and Logic Unit



EN164
Lecture 11-3



Numbers

- Bits are just bits (no inherent meaning)
 - conventions define relationship between bits and numbers
- Binary numbers (base 2)
 - 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
 - decimal: $0 \dots 2^n - 1$
- Of course it gets more complicated:
 - numbers are finite (overflow)
 - fractions and real numbers
 - negative numbers
- How do we represent negative (or signed) numbers?*
- Which bit patterns will represent which numbers?*
 - Octal and hexadecimal numbers
 - Floating-point numbers

EN164
Lecture 11-4



Representing Signed Numbers

Sign Magnitude:	1's Complement	2's Complement
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = -0	100 = -3	100 = -4
101 = -1	101 = -2	101 = -3
110 = -2	110 = -1	110 = -2
111 = -3	111 = -0	111 = -1

- Issues: balance, number of zeros, ease of operations.
- 2's complement is best.

EN164
Lecture 11-5



2's Complement Operations

- How do you negate a 2's complement number?*
 - Invert all bits and add 1
 - Remember:** "Negate" and "invert" are different operations.
 - Numbers are negated, bits are inverted.
- How do you convert an n bit number to a 2n bit number?*
 - MIPS 16 bit immediate gets converted to 32 bits for arithmetic
 - copy the most significant bit (the sign bit) into the other bits

"sign extension"

EN164
Lecture 11-7

Addition and Subtraction

- Unsigned numbers add/subtract just like in grade school

0111	0111	0110
+ 0110	- 0110	- 0101
1101	0001	0001
- 2's complement operations are also easy
 - subtraction is done using addition of negated number

0111	
+ 1010	
10001	(result is 0001, carry bit is set)
- 2's complement overflow (result is out of number range)
 - adding two 4-bit numbers does not yield an 4-bit number

0111	
+ 0011	
1010	(result is -6, overflow bit is set)

EN164
Lecture 11-8

Detecting Overflow

- Adding a positive and a negative number can never cause an overflow
- Similarly, no overflow is possible when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
 - overflow when adding two positives yields a negative
 - or, adding two negatives gives a positive
 - or, subtract a negative from a positive and get a negative
 - or, subtract a positive from a negative and get a positive
- Consider the operations $A + B$, and $A - B$
 - Can overflow occur if B is 0?
 - Can overflow occur if A is 0?

EN164
Lecture 11-9

Effects of Overflow

- An exception (interrupt) occurs
 - Control jumps to predefined address for exception handler routine
 - Interrupted address is saved for possible resumption
- Details based on software system / language
 - example: flight control vs. homework assignment
- Some instructions may allow you to ignore overflow
 - e.g., MIPS instructions: addu, addiu, subu
 - Some multimedia/DSP operations do not care about overflow

EN164
Lecture 11-10

Different Implementations

- No clear best way to build something
 - Don't want too many inputs to a single gate
 - Don't want to have to go through too many gates
 - For our purposes, ease of comprehension is important
- Let's look at a 1-bit ALU for addition:

a	b	c_{in}	c_{out}	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$c_{out} = a b + a c_{in} + b c_{in}$$

$$sum = a \oplus b \oplus c_{in}$$

How can we build a 32-bit ALU for AND, OR and ADD?

EN164
Lecture 11-14

32-bit ALU for AND, OR & ADD

1-bit ALU:

EN164
Lecture 11-15

What about subtraction ($a - b$)?

- Two's complement approach: Just negate b and add.
- Use carry-in for LSB for part of the negation

What's another way of implementing this function?

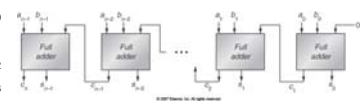
EN164
Lecture 11-16

BROWN

Chained 1-bit adders are slow

- A 32-bit ALU is much slower than a 1-bit ALU.
 - Carry must ripple through all 32 bits to determine result

- $c_1 = b_0c_0 + a_0c_0 + a_0b_0$
- $c_2 = b_1c_1 + a_1c_1 + a_1b_1$
- $c_3 = b_2c_2 + a_2c_2 + a_2b_2$
- $c_4 = b_3c_3 + a_3c_3 + a_3b_3$



- How do you get rid of the ripple effect?*
 - Generate the "carries" a different way

EN164
Lecture 11-19

BROWN

Carry-lookahead adder

- A carry-out signal is "1" when it is propagated to generated by a single bit cell:
 - When would we always generate a carry?* $g_i = a_i b_i$
 - When would we propagate the carry?* $p_i = a_i + b_i$

$$c_1 = g_0 + p_0c_0$$

$$c_2 = g_1 + p_1c_1$$

$$c_3 = g_2 + p_2c_2$$

$$c_4 = g_3 + p_3c_3$$

$$c_2 = g_1 + p_1g_0 + p_1p_0c_0$$

$$c_3 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$$

$$c_4 = \dots$$

- Note that c_i only depends on a_i , b_i , and c_0
- But there is a practical limit to the number of inputs used to generate c_i .

EN164
Lecture 11-20