Chapter 3

Arithmetic for Computers
Introduction

Review: MIPS Design Principles

- Simplicity
  - Fixed size instructions.
  - Small number of instruction formats.
  - Opcode always the first 6 bits.
- Smaller is faster
  - Limited instruction set.
  - Limited number of registers in register file.
  - Limited number of addressing modes.
- Make the common case fast
  - Arithmetic operands from the register file only.
  - Allow instructions to contain immediate operands and branch targets.
Arithmetic for Computers

- Operations on integers
  - Addition and subtraction.
  - Multiplication and division.
  - Dealing with overflow.
- Floating-point numbers
  - Representation and operations.
  - Dealing with overflow and underflow.

Binary Representation

- This binary number
  01011000 00010101 00101110 11100111
  represents the decimal quantity:
  \[ 0 \times 2^{31} + 1 \times 2^{30} + 0 \times 2^{29} + \ldots + 1 \times 2^0 \]
- An unsigned 32-bit word can represent \(2^{32}\) numbers between 0 and \(2^{32} - 1\)
- If we wish to also represent negative numbers, we can represent \(2^{31}\) positive numbers (including zero) and \(2^{31}\) negative numbers.
Chapter 3 — Arithmetic for Computers, Intro

Positive and Negative Numbers

0000 0000 0000 0000 0000 0000 0000 0000\textsubscript{two} = 0\textsubscript{ten}
0000 0000 0000 0000 0000 0000 0000 0001\textsubscript{two} = 1\textsubscript{ten}

... 0111 1111 1111 1111 1111 1111 1111 1111\textsubscript{two} = 2^{31}-1

1000 0000 0000 0000 0000 0000 0000 0000\textsubscript{two} = -2^{31}
1000 0000 0000 0000 0000 0000 0000 0001\textsubscript{two} = -(2^{31} - 1)
1000 0000 0000 0000 0000 0000 0000 0010\textsubscript{two} = -(2^{31} - 2)

... 1111 1111 1111 1111 1111 1111 1111 1111\textsubscript{two} = -2
1111 1111 1111 1111 1111 1111 1111 1111\textsubscript{two} = -1

2’s Complement Form

The same hardware can be used for 2’s complement addition and subtraction without any conversions.

0000 0000 0000 0000 0000 0000 0000 0000\textsubscript{two} = 0\textsubscript{ten}
0000 0000 0000 0000 0000 0000 0000 0001\textsubscript{two} = 1\textsubscript{ten}

... 0111 1111 1111 1111 1111 1111 1111 1111\textsubscript{two} = 2^{31}-1

1000 0000 0000 0000 0000 0000 0000 0000\textsubscript{two} = -2^{31}
1000 0000 0000 0000 0000 0000 0000 0001\textsubscript{two} = -(2^{31} - 1)
1000 0000 0000 0000 0000 0000 0000 0010\textsubscript{two} = -(2^{31} - 2)

... 1111 1111 1111 1111 1111 1111 1111 1111\textsubscript{two} = -2
1111 1111 1111 1111 1111 1111 1111 1111\textsubscript{two} = -1
**Example**

- Compute the 32-bit 2's complement representations for the following decimal numbers:
  - 5, -5, -6

  5:   0000 0000 0000 0000 0000 0000 0000 0101
  -5: 1111 1111 1111 1111 1111 1111 1111 1011
  -6: 1111 1111 1111 1111 1111 1111 1111 1010

**Signed / Unsigned**

- The hardware recognizes two formats:
  - **Unsigned** (corresponding to the C declaration `unsigned int`)
    - All numbers are positive, a 1 in the most significant bit represents magnitude, not sign.
  - **Signed** (C declaration is `signed` or just `int`)
    - Numbers can be +/-, a 1 in the MSB means the number is negative.

- This distinction enables us to represent twice as many numbers when we’re sure that we don’t need negatives.
MIPS Instructions

- Consider a comparison instruction:
  \[
  \text{slt } \$t0, \$t1, \$zero
  \]
  where \$t1 contains the 32-bit number: 1111 01...01
  What gets stored in \$t0?

- The result depends on whether \$t1 is a signed or unsigned number – the compiler/programmer must track this and accordingly use either \text{slt} or \text{sltu}

\[
\begin{align*}
\text{slt } \$t0, \$t1, \$zero & \text{ stores 1 in } \$t0 \\
\text{sltu } \$t0, \$t1, \$zero & \text{ stores 0 in } \$t0
\end{align*}
\]

Sign Extension

- Occasionally, 16-bit signed numbers must be converted into 32-bit signed numbers – for example, when doing an add with an immediate operand.
- The conversion is simple: take the most significant bit and use it to fill up the additional bits on the left – known as sign extension.

So \(2^{10}\) goes from 0000 0000 0000 0010 to 0000 0000 0000 0000 0000 0000 0000 0010

And \(-2^{10}\) goes from 1111 1111 1111 1110 to 1111 1111 1111 1111 1111 1111 1111 1110
Alternative Representations

- The following two intuitive number representations were discarded because they required additional conversion steps before arithmetic could be performed on the numbers.
  - Sign-and-magnitude: The most significant bit represents +/- and the remaining bits express the magnitude.
  - One’s complement: -x is represented by inverting all the bits of x.
- Both representations above suffer from two zeroes.

Recap

- 2’s complement representation.
- Signed vs. Unsigned number representation.