

Today's Discussion



- More on Binary: 2's complement & multiplication
- How to translate to LC2K
- How to write in LC2K

EECS 370

Lab 2: Project 1 - LC2K ISA

Negative Numbers Using 2's Complement



A way to represent integers using binary, both positive and negative

In 2's Complement, the most significant bit (MSB) is negative.

For example, in a 4-bit number,

the MSB typically represents 23, but in 2's Complement, the MSB instead represents -23

So, if the MSB is 0, the number will be positive (or zero)

And, if the MSB is 1, the number will be negative

The other bits remain the same, let's see how with an example:

Converting 3₁₀ into -3₁₀ with 4-bit Integers



With a 4-bit, 2's Complement, binary number, we represent 3 as

$$0011_{2} \rightarrow 0*(-2^{3}) + 0*2^{2} + 1*2^{1} + 1*2^{0} = 3$$

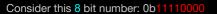
Conversely, we represent -3 with the same constraints as

$$1101_{3} \rightarrow 1*(-2^{3}) + 1*2^{2} + 0*2^{1} + 1*2^{0} = -3$$

To convert quickly and easily, flip the bits and add 1!

Comparison of Representations





Unsigned:
$$2^7 \times 1 + 2^6 \times 1 + 2^5 \times 1 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 0 = 128 + 64 + 32 + 16 = 240$$

Signed:
$$-2^7 \times 1 + 2^6 \times 1 + 2^5 \times 1 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 0 = -128 + 64 + 32 + 16 = -16$$

Notice that $2^8 - 16 = 240$

To sign extend to a larger size, just copy the MSB:

0b111111111110000 is -16 in 16-bit signed representation.

Addition in 2's Complement



Consider 8 + 4 = 12 = 0b1100 (unsigned 4-bit) And, -8 + 4 = -4 = 0b1100 (signed 4-bit) Notice the identical binary values. Also. = 0b1111111111111111 (signed 16-bit)

And, 2¹⁶ - 1 = 65535 = 0b111111111111111 (unsigned 16-bit)

This is why we use 2's complement! Addition is the same as normal binary

Multiplying in Decimal (Project 1m)

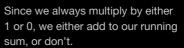


We can perform the same operation

in binary to multiply

n decimal, we multiply by "shifting" each number as we go		104
	Х	26
Note the added 0 on the right when		624
multiplying the 2	+	2086

Multiplying in Binary (Project 1m)



We perform one "shift" by adding a value to itself.

We get the value of each bit by masking with almost all 0's, and then checking if the result equals 0.

(Hint: Shift the mask also)

The Add instruction



The Nor Instruction



The add instruction allows us to do some bit shifting:

By doing this multiple times, we get multiplication:

NOR in LC2K is bitwise: First do OR with A \mid B, then NOT with ~result.

Recall from 270 or 203 that NOR is a universal gate; any gate can be built from it (AND, OR, etc.) https://en.wikipedia.org/wiki/NOR_logic

Since
$$A = A \mid A$$

 $A \mid B = \sim (\sim (A \mid B)) = \sim (NOR(A, B))$ Since $A = A \mid A$
 $A \mid B = \sim (\sim (A \mid B)) = \sim (NOR(A, B))$ Double Negation
 $A \mid B = \sim (\sim A \mid \sim B) = NOR(\sim A, \sim B)$ By DeMorgan's Laws

nor (nor(A,A), nor (B,B))

Combining Add and Nor



The Beq instruction



We can do subtraction by combining them:

$$A - B = A + (-B) = A + (\sim B + 1)$$

We can set or mask bitfields, like in Lab 1:

First shift with multiple add instructions, then do some nor operations.

How can you use this in your multiplication algorithm?

The Beg mandenen

But (if the offset is a label,) it (ust branches to the label.)

Review lec 4/5 & P1 Walkthrough for examples, and be comfortable with:

- If/Else statements
- While Loops
- Do-While Loops
- For Loops

EEUS 3/0 Fall 2024 19 EEUS 3/0

The Beq instruction



Testing Project 1a



Beq instructions implement conditional branches, but how do we do unconditional branches like in ARM?

When the 2 registers in beq are the same register, the branch is unconditional.

TELLE LIT bue,

Example: beq 1 1 offset is unconditional since r1 = r1 always.

There are 2^{10} (add, nor) + $3*2^{22}$ (lw, sw, beq) + 2^6 (jalr) + 2 (noop, halt) = 12,584,002 possible LC2K instructions.

You cannot test all of these!

But, you can test every opcode, regA, regB, and destReg.

You should also test many types of offsets and .fill values:

- Numbers (16-bit 2's complement)
- Absolutely-resolved labels for lw/sw/.fill
- Relatively-resolved labels for beq

Fall 2024 16 EECS 370 Fall 2024 1

Problem 3: LC2K Assembler Test Cases



Take some time to write some test cases for Project 1a

Guidelines:

- Also submit corresponding machine code
- Must not have errors
- Must catch 3 bugs
- Each test must be 5 lines or fewer

Pro tip: Use to debug your Project 1a

LC2K to Machine Code Conversion Example





000000010000001000000000000111

= 0x00810007

EECS 370 Fall 2024 18 EECS 370 Fall 2024 19