# CS3432 Computer Organization
# Spring 2025. Homework Assignment 2
**Due: 04/08/2025 7:59PM MDT**
**Individual Assignment**

For this assignment, you have to turn in, on physical paper:

- your circuit diagrams for your basic bricks, such as AND, OR, XOR gates and 1 bit multiplexers,
- your circuit diagrams for your extended full adder, designed in Section 1 and
- your circuit diagrams for your 8-bit arithmetical-logical unit, designed in Section 2.

## 1 An Extended Full Adder

In this Section, we are going to design an extended full adder circuit (EFA). That EFA takes 6 one bit inputs: $a_j$, $b_j$, $c_{in}$, $r_{in}$, $t_1$ and $t_0$. Depending on the four possible combinations of values on $t_1$ and $t_0$, the EFA produces 3 one bit outputs: $s_j$, $c_{out}$ and $r_{out}$.

The EFA can be specified in principle by a truth table with $2^6 = 64$ entries and 3 outputs. However, as the EFA ignores certain inputs in certain cases, it is easier to work with the following overview specification, depending only on $t_1$ and $t_0$ in the first place:

| $t_1$ | $t_0$ | Description | Output Relationship | Ignored Inputs |
|---|---|---|---|---|
| 0 | 0 | Addition Mode | $2\,c_{out} + s_j = a_j + b_j + c_{in}, \quad r_{out} = 0$ | $r_{in}$ |
| 0 | 1 | Shift Left Mode | $s_j = c_{in}, \quad c_{out} = b_j, \quad r_{out} = 0$ | $r_{in}, a_j$ |
| 1 | 0 | Shift Right Mode | $s_j = r_{in}, \quad r_{out} = b_j, \quad c_{out} = 0$ | $c_{in}, a_j$ |
| 1 | 1 | Logical Mode | $s_j = (\overline{c_{in}} \wedge (a_j \otimes b_j)) \vee (c_{in} \wedge (a_j \vee b_j)), \quad c_{out} = c_{in}, \quad r_{out} = 0$ | $r_{in}$ |

In this table above, $\overline{c_{in}}$ stands for the inverted $c_{in}$ signal[1], $\wedge$ stands for the logical AND of two bits, $\vee$ stands for the logical OR of two bits and $\otimes$ stands for the logical XOR[2] of two bits.

**Design a circuit for EFA based on this specification with nothing but two bit input-one bit output NAND gates.** If you wish to use other types of gates, such as AND gates, OR gates, XOR gates, you can design them first and then instantiate them in your EFA diagram. If you wish to use Disjunctive Normal Form, you can do so, but you need to give the full 64 entry truth table first in this case.

**Test your circuit on at least 8 (eight) different input patterns.** You can do the testing by propagating the signals (zeros and ones) across the different wires it contains. You can use eight different colors in the same drawing, unless this becomes too messy.

## 2 An 8-bit Arithmetical-Logical Unit

Based on circuitry for sign- resp. zero-extension (signed resp. unsigned numbers), XOR units to flip input bits, AND units to set inputs to zero, nine EFA circuits, XOR units in output and units for signed comparison with zero, we will now design a full 8-bit Arithmetical-Logical Unit (ALU).

This ALU unit will take to 8-bit inputs $a$ and $b$ (with bits $a_7$ through $a_0$ and $b_7$ through $b_0$, where the bit with index 0 is the least significant bit), as well as a 4-bit steering input $p$ (with bits $p_3$ through $p_0$) and it will produce an 8-bit output $c$ (with bits $c_7$ through $c_0$).

The input/output relationship is specified by the following table:

[1] $\overline{c_{in}} = 1$ if $c_{in} = 0$ and $\overline{c_{in}} = 0$ if $c_{in} = 1$
[2] XOR = exclusive OR

| $p_3$ | $p_2$ | $p_1$ | $p_0$ | Description | Corresponding C code | Type | Ignored inputs |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Addition | `c = a + b` | signed & unsigned | – |
| 0 | 0 | 0 | 1 | Subtraction | `c = a - b` | signed & unsigned | – |
| 0 | 0 | 1 | 0 | Shift Left | `c = b << 1` | signed & unsigned | $a$ |
| 0 | 0 | 1 | 1 | Compare LTU | `c = (a < b)` | unsigned | – |
| 0 | 1 | 0 | 0 | Shift Right Logical | `c = b >> 1` | unsigned | $a$ |
| 0 | 1 | 0 | 1 | Compare LEU | `c = (a <= b)` | unsigned | – |
| 0 | 1 | 1 | 0 | Logical XOR | `c = a ^ b` | signed & unsigned | – |
| 0 | 1 | 1 | 1 | Logical OR | `c = a \| b` | signed & unsigned | – |
| 1 | 0 | 0 | 0 | Compare EQ | `c = (a == b)` | signed & unsigned | – |
| 1 | 0 | 0 | 1 | Unary Minus | `c = -b` | signed & unsigned | $a$ |
| 1 | 0 | 1 | 0 | Compare NEQ | `c = (a != b)` | signed & unsigned | – |
| 1 | 0 | 1 | 1 | Compare LT | `c = (a < b)` | signed | – |
| 1 | 1 | 0 | 0 | Shift Right Arithmetical | `c = b >> 1` | signed | $a$ |
| 1 | 1 | 0 | 1 | Compare LE | `c = (a <= b)` | signed | – |
| 1 | 1 | 1 | 0 | Logical Negation | `c = ~ b` | signed & unsigned | $a$ |
| 1 | 1 | 1 | 1 | Logical AND | `c = a & b` | signed & unsigned | – |

**Design such an ALU, based on your previous basic bricks, in particular based on nine instances of your EFA brick.** If you need to generate intermediate steering signals, give truth tables for these signals and circuits to generate these steering signals. You are not supposed to design circuits for the different operations on 8 bits and then use a big multiplexer; you are rather supposed to use the EFA brick 8 or 9 times.

**Test your ALU for each of of the 16 possible combinations of $p$ and for at least one interesting input combination of $a$ and $b$.** Use several colors and several drawings.