KETHAVATH MANOHAR
manoharkethavath1230@gmail.com
IIITDM Kancheepuram
Chennai,India
7036569701.

Title: 4 BIT DISCRETE ALU (Arithmetic logic unit)

ALU - Arithmetic Logic Unit

The CPU consists of three main sections: **the storage of** variables (registers), **the** control **circuit (microcode)** and the **ALU.** The ALU (Arithmetic Logic Unit) is the part of **the** CPU that does **the actual computation** and **health** testing.

For example, **when adding** two binary numbers, it is the ALU that **produces** the result. **When a** program needs to execute code if two values are **equal,** the ALU performs **a value** comparison and flags **whether** the condition is **true** or **false.**

Modern CPUs are made up of millions of transistors **(billions of them today!) that are impossible to replicate** at home. But a simple CPU **(such as the Z80)** has only 8500 transistors. Computers **of** the past **(like** many of **IBM's** mainframe computers) were built with discrete 4000 and 7400 series chips.

Project Requirements

Basic Understanding of **Boolean Concepts**
Basic Understanding of **Logic Gates**

Binary Addition

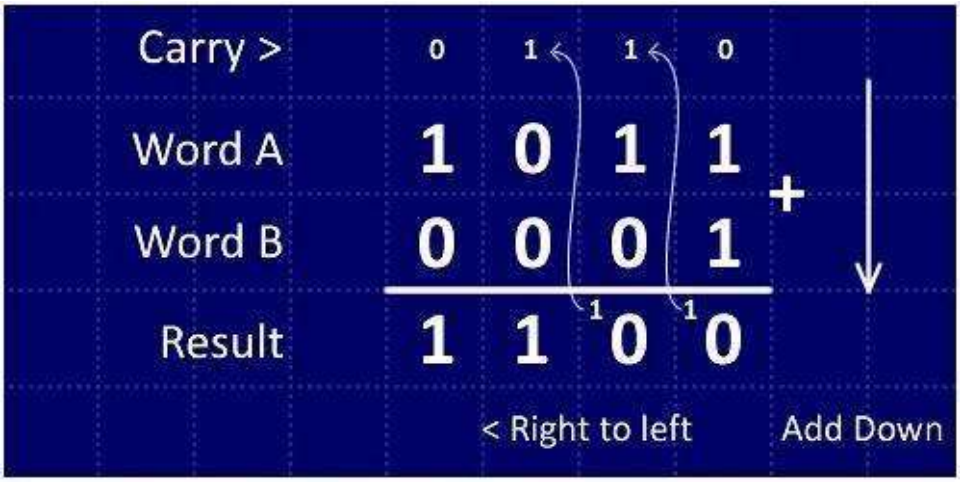
The two basic ALU operations are addition and subtraction.

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 10 (This is also 0 + carry bit)

The carry bit is used as **follows.** where "0(c)" means "no carry bit" and "1(c)" means "carry bit".

- 0 + 0 +0(c) = 0
- 0 + 1 +0(c) = 1
- 1 + 0 +0(c) = 1
- 1 + 1 +0(c) = 10
- 0 + 0 +1(c) = 1
- 0 + 1 +1(c) = 10
- 1 + 0 +1(c) = 10
- 1 + 1 +1(c) = 11

To add 10 and 10 in binary form, start by writing them in **one long addition form. Add bits to** the **column from right to left** using the rules **above. If there is** a carry from **the bit addition, shift it left one column and include it as** a bit in the **addition.**

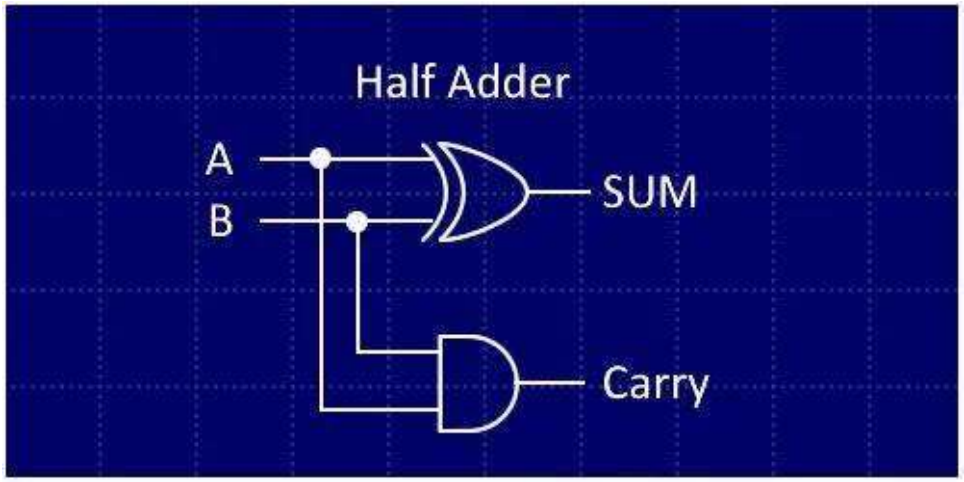


we are adding 1011 and 0001 (11 + 1 = 12). Starting from the far right we add 1 + 1, which gives us 10 (0 and a carry bit). Then we move to the next column (the second from the right) and add all the bits. carry bit is also included in this addition operation. This means we are adding three digits: 1 (the carry bit), 1, and 0.

The circuit

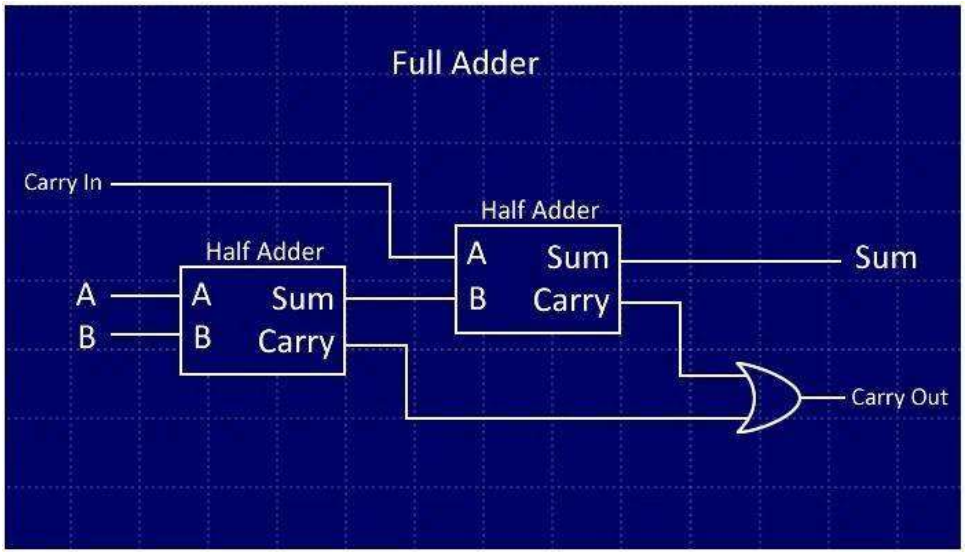
half adder has two inputs and two outputs as shown in the figure below. The two inputs represent two individual bits, the **sum** output represents the sum of the two bits in **1-bit form,** and the **carry out** is the carry bit from the addition.

HALF ADDER

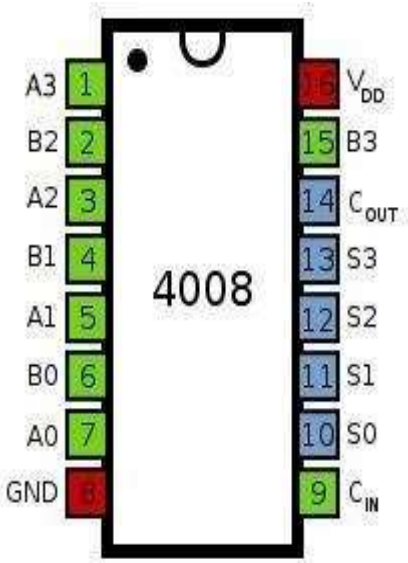


A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

FULL ADDER



4008 4-bit full adder pinout



Binary Subtraction

Implements a binary system called **"two's complement". The system has some** rules:

To negate a number **(e.g.** change 5 **to** -5), **invert** all bits and add 1.
MSB (most significant bit) **1** is

Example:
10010 is negative
00010 is positive
You can determine the value of a negative binary **number by following rule 1:**

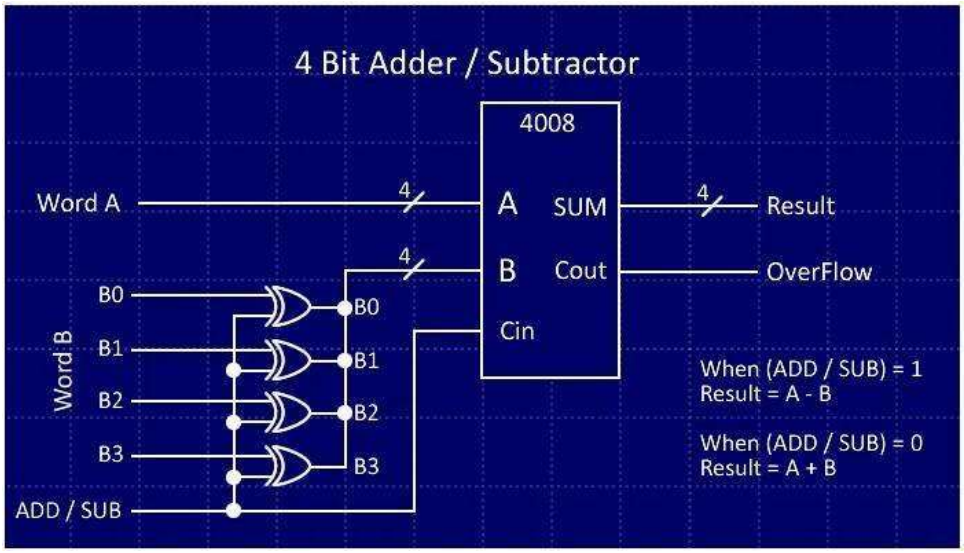
0001 = 1: negate this = **(1110** + 1 = 1111) = -1
1001 = **- 7: Deny** this = (0110 + 1 = 0111) = 7
0110 = 6: **Deny** this = (1001 + 1 = 1010) = -6

Perform addition and subtraction using the same adder circuit. So if **you want** to turn **a** 4-bit adder into a 4-bit **adder/subtracter, you** just need to **install** a single 4070 IC **(quad-XOR).** We feed the binary number inputs into one input of each XOR gate, use the other XOR input as an add/subtract line, and then feed this same line into the carry in of the adder.

If we want to **draw** B from A, **raise** the line **of subtraction.** This does two things:

Inverts all **received** bits on input **B.**

Add 1 to the adder.



The buffer also has an enable line to **select** the output **of** the adder/subtractor.

Logical Functions

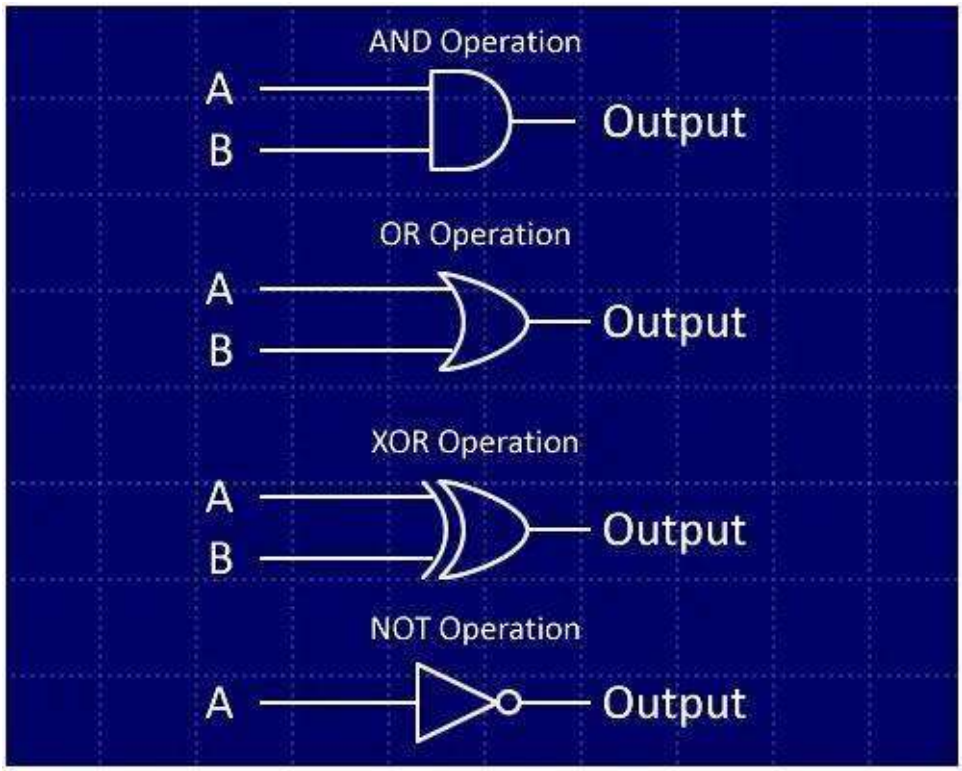
Logical functions are useful when bit manipulation is **required. Suppose you have** a microcontroller **with** an 8-bit **port** and you use the lower 4 bits to read from a 4-bit data bus. When **reading** from the **port,** the upper **4 bits must be removed. This is because these** bits can affect program execution. **Therefore,** to remove these bits, you can **hide** them **using the** logical AND function.

This function **concatenates the** bits **of a** word **(port)** with another number **(a** number **without** the upper **4** bits). If **you** AND the port with 0x0F (0b00001111), **keep** the lower **4** bits (because x AND 1 = x) and remove the upper **4** bits (because x AND 0 = 0).

AND is used to remove bits
NOT is used when **all bits** need to **be inverted** (0000 **becomes** 1111)
OR is used to **combine** bits (0110 OR 0001 is 0111)
XOR is used to **select the** selected **bit to invert (0101)** XOR 0100 is 0001)

ALU **logic unit is** AND, OR, XOR, and NOT **gates. Each** bus buffer for each logical unit **has an enable line** so that each unit can be selected individually.

4081 - Quad AND **Gate**
4070 - Quad XOR Gate
4071 - Quad OR Gate
4049 - Hex NOT Gate
74HC125 - Output Buffer (for bus isolation)



Logic gates showing the four logical functions of our ALU

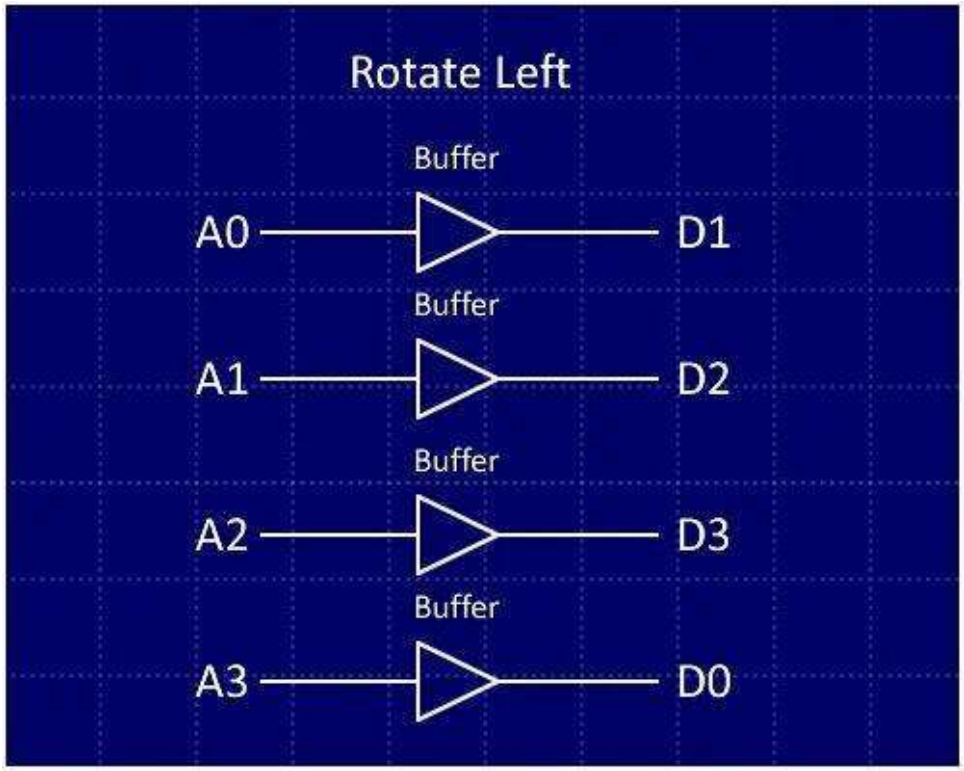
- A and B represent bit 0, 1, 2, or 3 from word A and the corresponding bit from word B.
- NOT operations involve only one word; each bit in the word is complemented, regardless of the state of the other word.

Bit shifting and comparisons

Bitshift is also a very important **feature.** These functions are **common** in microcontrollers with serial ports where data is **injected** bit by bit. This means **that when a bit comes in,** you **have** to take **it, put** it in the **first** (or **last) byte,** then shift **left or right** all the bits in the byte **(Depending** on where you put the incoming **bits).). a** bit).

Interestingly, a **right** bit shift **(eg** 0010 becomes 0001) **corresponds** to division by **2** and a **left** bit shift **(eg** 0010 becomes 0100) **corresponds** to multiplication by **2.**

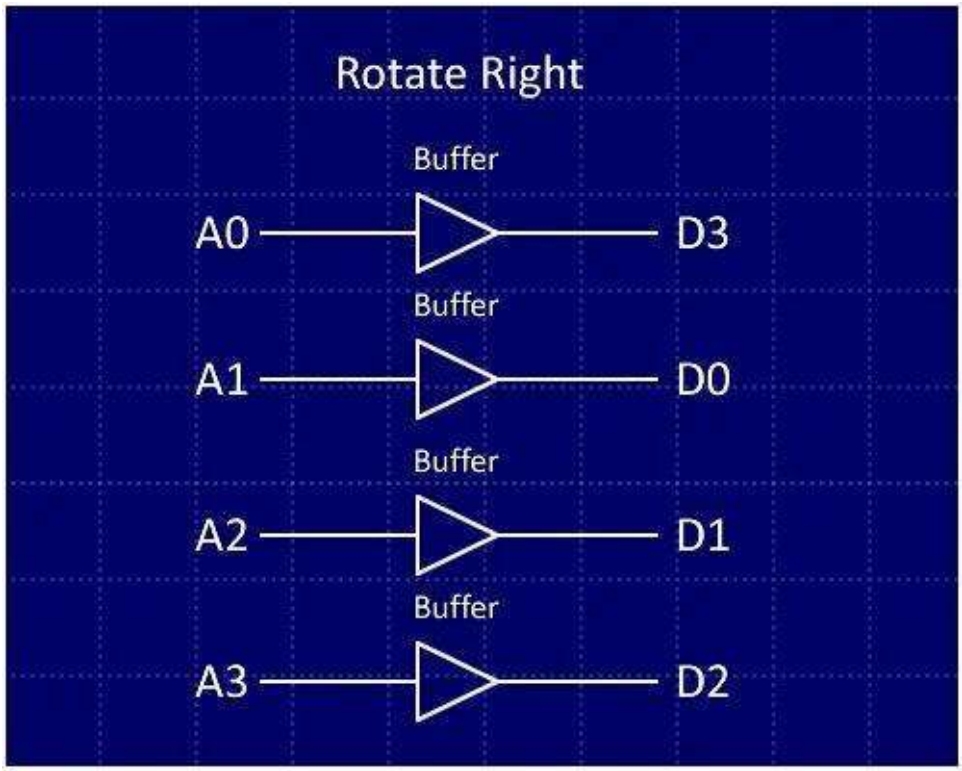
bit shifting **is very easy to do.** The only **chips** needed **are buffers as** the physical data lines are simply rearranged.



LEFT ROTATE

For a rotate left:

- Bit 0 is connected to bit 1 on the output
- Bit 1 is connected to bit 2 on the output
- Bit 2 is connected to bit 3 on the output
- Bit 3 is connected to bit 0 on the output



RIGHT ROTATE

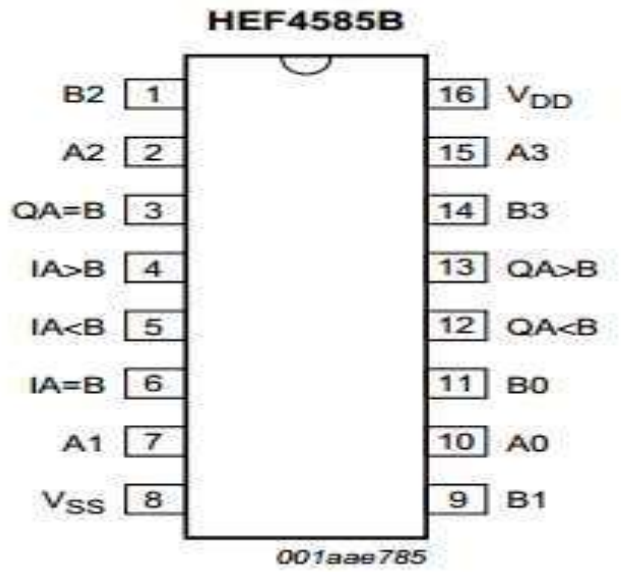
For a rotate right:

- Bit 0 is connected to bit 3 on the output
- Bit 1 is connected to bit 0 on the output
- Bit 2 is connected to bit 1 on the output
- Bit 3 is connected to bit 2 on the output

The **final** operation to **implement** in the ALU is **comparison. This** is very useful (and easy to add). **Use** the **size comparator 4585. This** chip **takes** two 4-bit numbers and **tells** us **if** A > B, **if** A < B, **if** A = B. **Many older**

processors do not have **a size** comparison **feature. In other words,** trying to determine if **one** number **is** greater or less than another was **not so easy. You could do the** subtraction **and then check for** the sign bit, but that **would make** the code less **readable. With** this **ALU,** you **only** need to **put** two numbers on the data bus and the magnitude comparator will **immediately** tell you if the two numbers are equal, less **than** or **greater than.**

4585 The **drawback** of the comparator is that it is a magnitude comparator and does not **use 2's** complement.



A 4585-magnitude comparator.

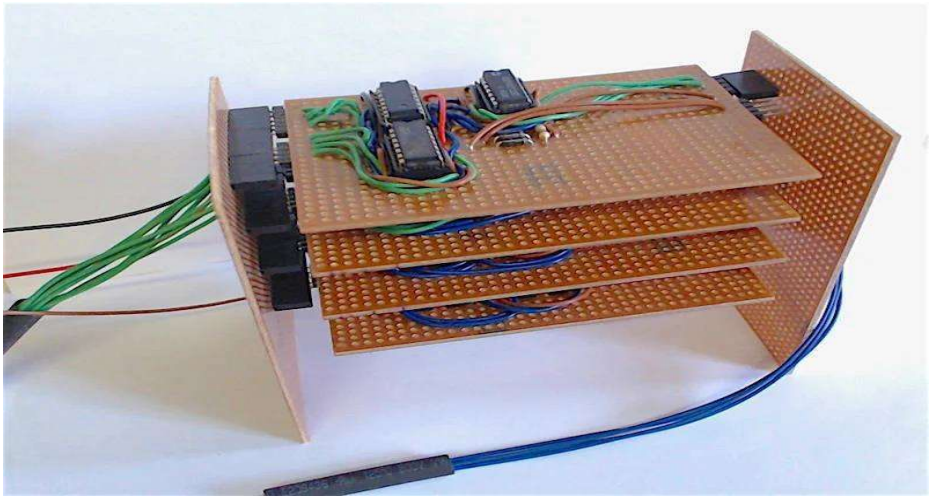
BOM - Bill of Materials

Component	Quantity

4008 (16 DIP)	1
4081 (14 DIP)	1
4070 (14 DIP)	2
4071 (14 DIP)	1
4049 (16 DIP)	1
74HC125	7
100nF Capacitors	14
14 DIP Socket (for soldering the project)	11
16 DIP Socket (for soldering the project)	3

ALU was built on four separate stripboards which all fit on a custom mini shelf (also made of stripboard).

- The front-side buses (left panel in the image) are the inputs A and B
- The back-side buses (right panel in the image) are the data output and the function selection



To control the ALU for testing purposes, a fifth stripboard was used to hold buttons for inputs, LEDs for outputs, and a 4515-decoder chip to decode the outputs.

The
ALU module has an active low enable line. Therefore, the enable **line must be set** to **0V instead of 5V to activate the** module on the data **bus.** 5V **will prevent** the module from sending data.

Key point

The ALU uses a common data bus **with each function** isolated via quad **buffers.** However, it is imperative that **he** only **has** one module **working** at a time **(e.g.,** only ADD is enabled and all others **are** disabled). Enabling more than one module at a time will damage the 74HC125 **chip.**

Summary

ALU, we can mathematically and logically **manipulate** two 4-bit numbers. But this ALU is **just** a small piece of the puzzle! Then **you can** create a memory controller that **can** stream numbers from **the** chip and perform operations on those numbers.