Arithmetic Logic Unit

David Strikaitis

Computer Architecture for game devices

# Introduction (*What is an ALU*)

An ALU also known as an arithmetic Logic Unit is composed of three main instruments, the logic unit, the arithmetic unit, and the decoder. When trying to understand the ALU it is a lot easier to break it down into these three separate parts, as we will further in this document. This will of course involve explanations of the units, and breakdowns to show different examples of how they function.

# History (*a brief overview of ALUs*)

The Early ALU, was originally hypothesized by John von Neumann, who we can recognize as the figurehead of the von Neumann architecture for computing machines. He had originally proposed an early concept of an ALU in 1945 in a report from the EDVAC computer frame. A true ALU would not get easily into the hands of the consumer for many years due to high prices of electricity at the time and the price to even manufacture the chip.

The chip that will be referenced from here on out as a basic ALU is known as the DM74LS1814 bit chip, this chip was first introduced by Texas instruments in the 7400 series of integrated cards, this chip showed a lot of promise at the time and was used in the CPU of many minicomputers for years to come including the NOVA 1200 and the XEROX Alto computer which pioneered the word “*Desktop*” and the word “*GUI*” or “*graphical user Interface*”.

# Breakdown (*What can it do?*)

Of course, to understand the logic of the ALU, we must know what it can do, in this list we will be referring to the DM74LS181 chip, but other chips may have more or less logical operations.

This chip has 2 different types of function, Logical and Arithmetic, the Arithmetic functions can again be broken down into with or without carry.

*The following diagram is taken as a breakdown of all possible expressions that this ALU can do.*

A table with text and symbols

Description automatically generated

Fig.1 Table of ALU patterns

Now that we know what the ALU can do, how does it know what to do?

*The following is a basic Logic unit from a 1-bit ALU:*

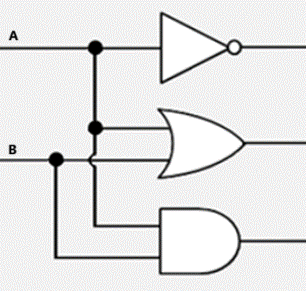


Fig.2 Image of Basic logic diagram

A 4-bit ALU will of course have a more complex Logic Unit, but to break down the ALU this will be easier to understand. The ALU here has 2 inputs (A/B) this will then feed into another set of gates which will inevitably control the output of the ALU.

In this scenario, the NOT gate represents NOT A and feeds the output out into another set of gates that controls what the output is (this is controlled by the decoder).

In a basic sense all outputs are created sequentially and are output out of the Logic Unit, but then the decoder and arithmetic Units decide what the actual output is.

# Breakdown (Decoder)

*What Follows is a diagram of a decoder to assist in visualization:*

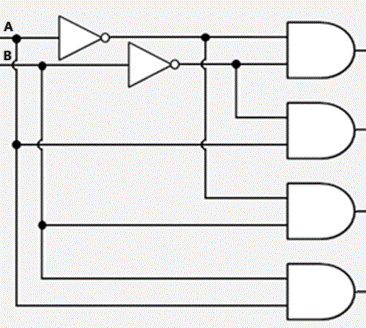
**

Fig.3 Image of basic decoder

In simple terms, the decoder only lets one of these gates work at a time, each gate will then relate to a certain part of the Logic Unit, and that will allow only one input to pass, in this example, the first gate corresponds to the NOT A in Fig.2, this therefore means when both inputs into this gate are 0, we get the desired output of NOT A.

We can break down this logic diagram of each output into a simple Boolean table, each input has only one Boolean output of true/1.

|  |  |  |
| --- | --- | --- |
| A | B | OUTPUT |
| 0 | 0 | First Gate |
| 0 | 1 | Second Gate |
| 1 | 0 | Third Gate |
| 1 | 1 | Fourth Gate |

Fig.4 Boolean expression of which output is activated.

Now that we know which gate is activated, we can discern what we are attempting to get as an output, these two parts of the ALU work very closely, as both require each other to function.

We may however observe that in Fig.2 we only have 3 outputs, whereas in Fig.3 we have 4, the fourth output is used in the Arithmetic Unit.

# Breakdown (*Arithmetic Unit)*

The Arithmetic Unit in the Example I will be explaining is composed of a Full Adder, in other ALUs the composition will be different, as the Full Adder circuit only supports 1 bit addition. The DM74LS181 chip for example uses four Full Adder circuits in a system known as a “Ripple Adder”, this system takes the carry out of the Full Adder as an input into the next Full Adder and will continue until all the Adders have done the calculations and the Full Adders carry out can again be taken as an output.

What is a “*Full Adder*”? The Full Adder circuit is comprised of 2 Half Adder circuits.

A Half Adder circuit is comprised of simply an AND gate and a XOR gate.

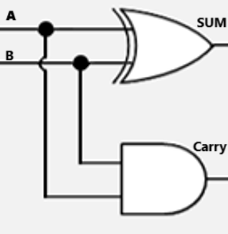


Fig.5 A Half Adder circuit

The Half Adder Circuit Above is Very simple to understand, it takes 2 single Bit inputs and outputs a 1 in the Carry Bus if both are 1, and it displays 1 in the Sum Bus if only one of either A or B is a 1, This is simple binary Addition.

This is all the potential outputs based on its inputs:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | SUM | CARRY |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Fig.6 Truth Table for A Half Adder Circuit

This Half Adder Circuit feeds into another Half Adder Circuit to create a full adder circuit, the reason we do this in an ALU is so that we can have multiple ALUs in a parallel circuit, this allows us to process longer bits of data so even a set of 1-bit ALUs could process long strings without issue.

# 4 Bit ALU

The DM74LS181 Chip is inherently very similar to the 1-bit ALU that we have been looking at thus far, the difference being it has more functions, can support two 4-bit input strings, and has an extra range of things it can do.

What does it Look like? The DM74LS181 Chip has a decently complex system but it is easier to understand now that we know how it works.

A diagram of a circuit

Description automatically generated

Fig.6 DM74LS181 Logic Diagram

This image shows a table of a total of 24 Pins also known as inputs/outputs, a multitude of AND, NOT, NAND, and XOR gates. The basic idea of what this implement does is the same as a 1-bit ALU but bigger, most of the functions of this ALU we have already covered and can be found in the diagram in Fig.1.

# Upscaling

Often, we want to process more than just 4-bits of data at a time, what does that mean for us?

-Option A: Run it again.

One option that is very popular is to take the same ALU and pass through chunks of data at a time, this means that we continuously go through the data until we hit the end, this is of course slow as it means that for a 32-bit number to be compared to another 32-bit number we have to run through it a total of 8 times, this means that the response time of the ALU is very slow as it takes 8 times the processing cycles to process one chunk of data. This does however mean that the architecture is cheaper for the end user as they need only to pay for 1 ALU rather than Many.

-Option B: More ALUs

Of course, the other option is too upscale the amount of ALUs we have, these ALUs would be placed into a Parallel circuit and would all work on a bit of data simultaneously, this does however mean that the device will require a way of splitting up the data and putting it back together. The issue with this solution is of course that it is expensive as each ALU costs a decent amount of money to purchase and it will also consume more electrical power and therefore also produce more heat, which will again cost more as a better cooling solution will be needed.

-Option C: Make It Bigger

This is the other obvious solution to this problem but also comes with the same draw backs as the previous solution, these include price and heat. So why would you want a bigger ALU as opposed to a Parallel circuit of them?

It runs much faster, the ALU will be able to process all 8/16/32(etc.)-bits of data much faster as it won’t need to be split up, and reformed into the data, this again comes with the major downside of it being very expensive to implement, but it is the most efficient solution to the problem.

# The Future

The future of ALUs is also promising, right now in the computer world we can see that Moore’s Law that the given transistors in a computer will double every eighteen months and the price will be halved is now on a steady decline as it is becoming more and more untrue in the most recent of years.

This could soon change as we begin to introduce a new concept, biological computing. A new concept involving Actin molecules allows the possibility of even smaller transistors than currently allowed by quantum properties, and the issue of current leakage.

What does this mean for us? If more research is carried out on this concept there is a chance that we could soon have actin-based computers, in our hands. This would also potentially further research into other biochemistry-based fields.

# Conclusion

What is an ALU? A computer component, usually found in a central processing unit (CPU) that can carry out different task on different lengths of data, it does this by receiving a certain signal through the control busses and the series of bits that the calculation is meant to be done on.

It does these logical and arithmetic operations all at the same time and can be broken down into 3 main areas, the Logic Unit, which does the logical operations on the ALU such as inverting, the Decoder which transforms the input signals into what needs to be actually done by passing signals to a series of AND gates, and the Arithmetic unit which is comprised of Adder circuits and performs different bit wise operations and also feeds into the series of gates controlled by the Decoder.

A larger ALU although better at processing information is more expensive and larger physically. The solutions to these problems also include having multiple smaller circuits and repeating through the same set of data multiple times.

The future of ALUs too looks very promising as Moore’s Law is currently being more and more difficult to continue with, we are looking at new possibilities for computing completely.

##### References

*ALU references:*

[*https://pdf.datasheetcatalog.com/datasheets/560/493318\_DS.pdf*](https://pdf.datasheetcatalog.com/datasheets/560/493318_DS.pdf)

[*https://www.ti.com/lit/ds/symlink/sn54ls181.pdf*](https://www.ti.com/lit/ds/symlink/sn54ls181.pdf)

[*https://web.archive.org/web/20120131123316/http://www.aeromech.usyd.edu.au/MTRXLAB/ref/components/TTL/DM74LS181.pdf*](https://web.archive.org/web/20120131123316/http:/www.aeromech.usyd.edu.au/MTRXLAB/ref/components/TTL/DM74LS181.pdf)

[*http://www.bitsavers.org/pdf/dec/pdp12/fpp12/DEC-12-HFPPA-A-D\_FPP12A\_Floating-Point\_Processor\_Users\_Manual.pdf*](http://www.bitsavers.org/pdf/dec/pdp12/fpp12/DEC-12-HFPPA-A-D_FPP12A_Floating-Point_Processor_Users_Manual.pdf)

*Actin Based Computing:*

[*https://link.springer.com/article/10.1007/s00542-019-04590-1*](https://link.springer.com/article/10.1007/s00542-019-04590-1)

*1 Bit ALU for diagrams:*

[*https://www.101computing.net/wp/wp-content/uploads/1-bit-ALU.png*](https://www.101computing.net/wp/wp-content/uploads/1-bit-ALU.png)