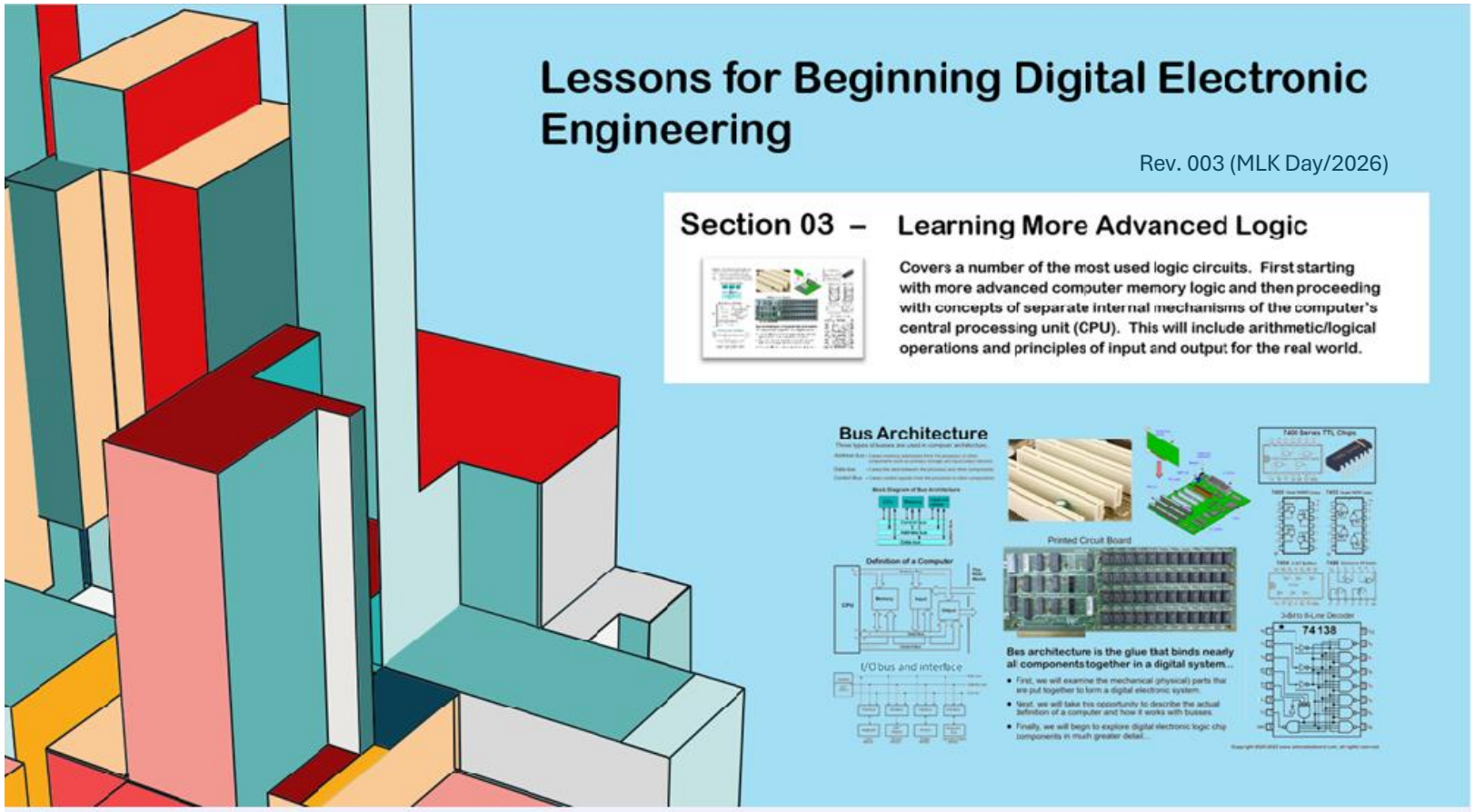
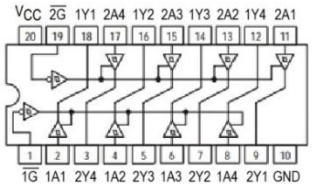


Table of Contents

"Digital parts work together in a circuit like people play their best parts in a community!"

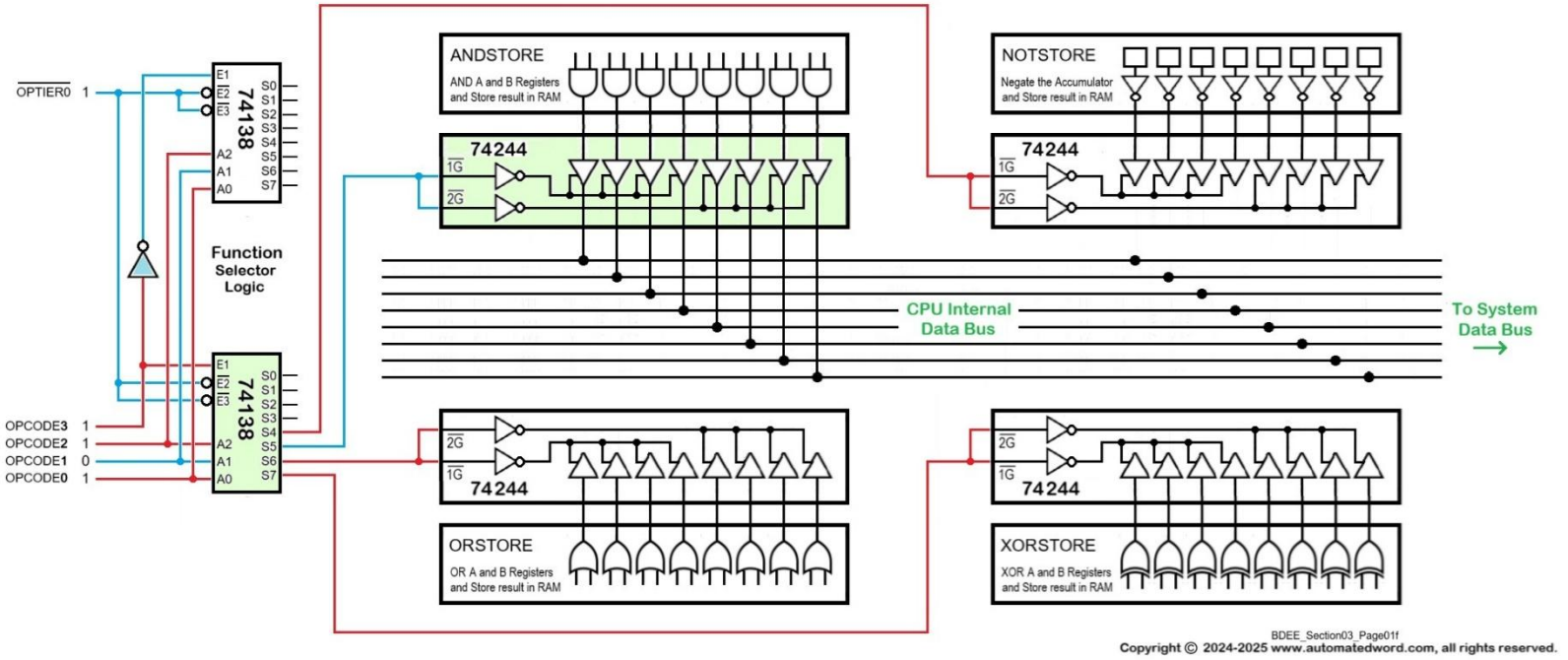
Electronic TRI-State Buffering (74244 8-Bit TRI-State Buffer/Line Driver)



74244 Pinout

A TRI-State buffer is a special electronic gate that has a third state besides 5-volts and 0-volts (1 and 0). The third state, without getting into too much technical specification, is where the gate's output can be to set to what is called the high-impedance state. Essentially this means, for all practical purposes, the gate disconnects itself from the circuit its output leads are connected to. The TRI-State buffer provides the only way to get around the rule of not connecting the outputs of logic gates together - allowing different gates to take turns driving the same inputs.

The schematic diagram below demonstrates the usefulness of this component when separate logic processes want to feed a common output (typically, a data bus). The 74244 gives you two 4-bit 3-state noninverting buffers.



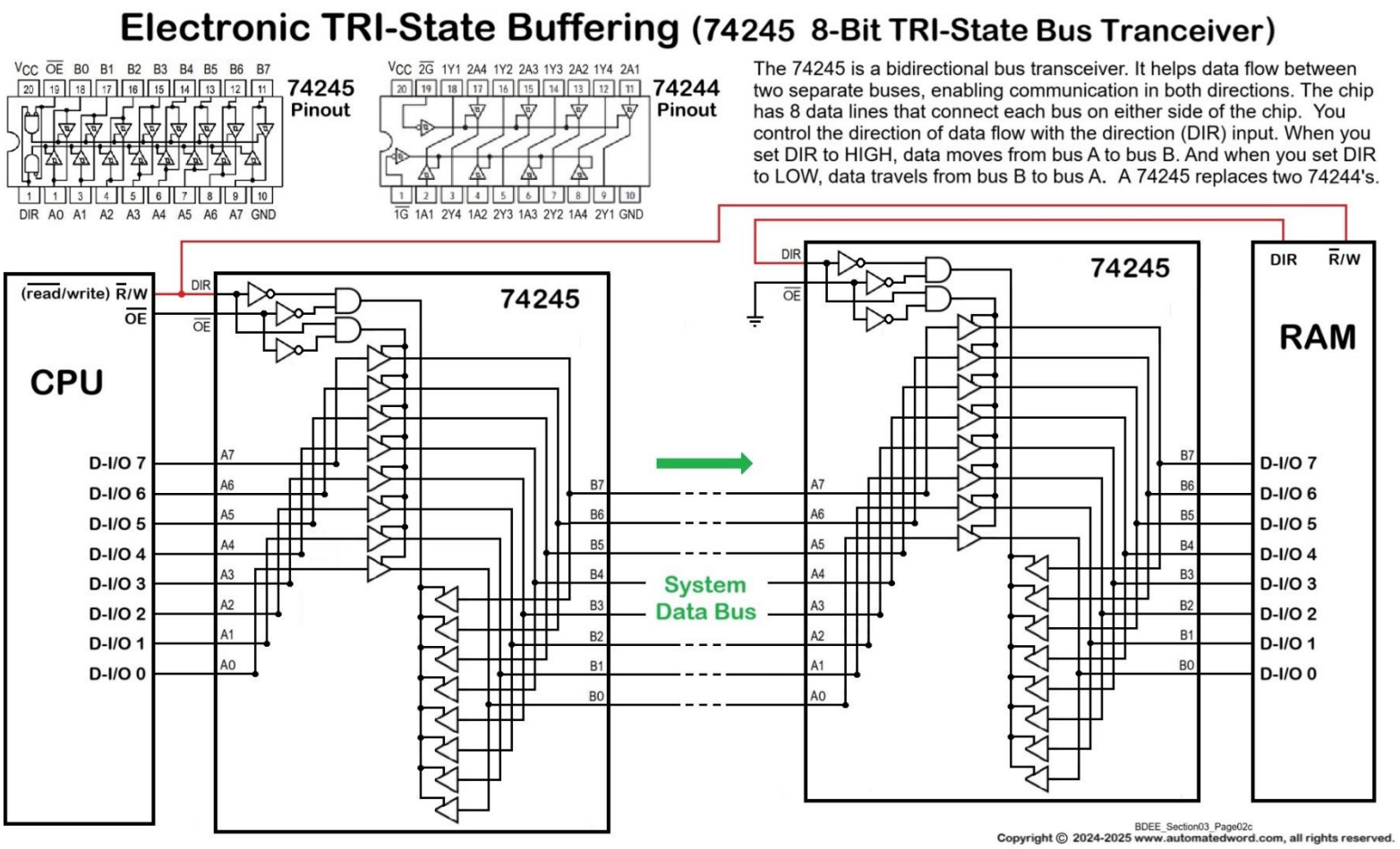
If you remember rule #1 of connecting logic gates – that you can never connect the output of logic gates together. TRI-State buffering provides the one exception to that rule. The above schematic shows that several different functions within the CPU want to place their results on the system data bus – well, this can be done with the help of a "TRI-State Buffer". This buffer is not a logic gate but, it has a specialized output. Its output can be put into what is called the "High Impedance State" which means that, for all intents and purposes, it has disconnected itself from the bus. The connected leads stay in place but no logical signal can be emitted on its output leads. This state is referred to as the "Z" state in documents.

The logical process of sending data over the data bus from any number of sources can be accomplished provided that each source takes its turn to access the bus. The 74138 on the left is reacting to the CPU signals during an instruction cycle and this establishes the marshalling process which allows only one function at a time to place its function results on the data bus. In the above schematic, the CPU signals (OPCODE 0 – 3) have selected the ANDSTORE function to send its result while the others are shut down.

It should be noted that these buffers are also specialized compared to the normal logic gate outputs in that they are able to drive up to 20-24 standard logic inputs and this makes them very useful for driving busses of all types – busses are usually connected to many components at a time (like RAM modules). You can view the datasheet for the 74373 in the BDEE Course Quick Datasheet Viewer: 74LS244

7400 Series Guide: 74LS244 (3-state buffers) What Makes Tristate Buffers Unique in Circuit Design
YouTube - Tristate Buffers Tri-State Buffers - Avoiding Bus Contention
YouTube - Tri State Digital Buffer - Multiple Drive Logic Three-state logic - Wikipedia
How does a TRI-State buffer work - Do It Easy With ScienceProg Buffers for Digital Logic Gates
YouTube - Half vs Full Duplex in Tri-State Buffer Create Tri-State Buffer in VHDL – Nandland

  (Back to TOC)



The 74245 is also a TRI-State Buffer component like the 74244. However, it is designed to fulfill a very specific purpose within digital electronic circuitry and that is to facilitate a bi-directional bus. The quintessential example of a bidirectional bus is the system data bus of a computer. In almost every computer system design there exists a pathway for the CPU to send a byte of data to RAM for storage as well as to read a byte that has already been stored in RAM back to the CPU. It would be inefficient for the design to have two different data busses – one for the CPU to send a byte to RAM and yet another to receive a byte from RAM back to the CPU. So, this is where the 74245 comes in very handy…

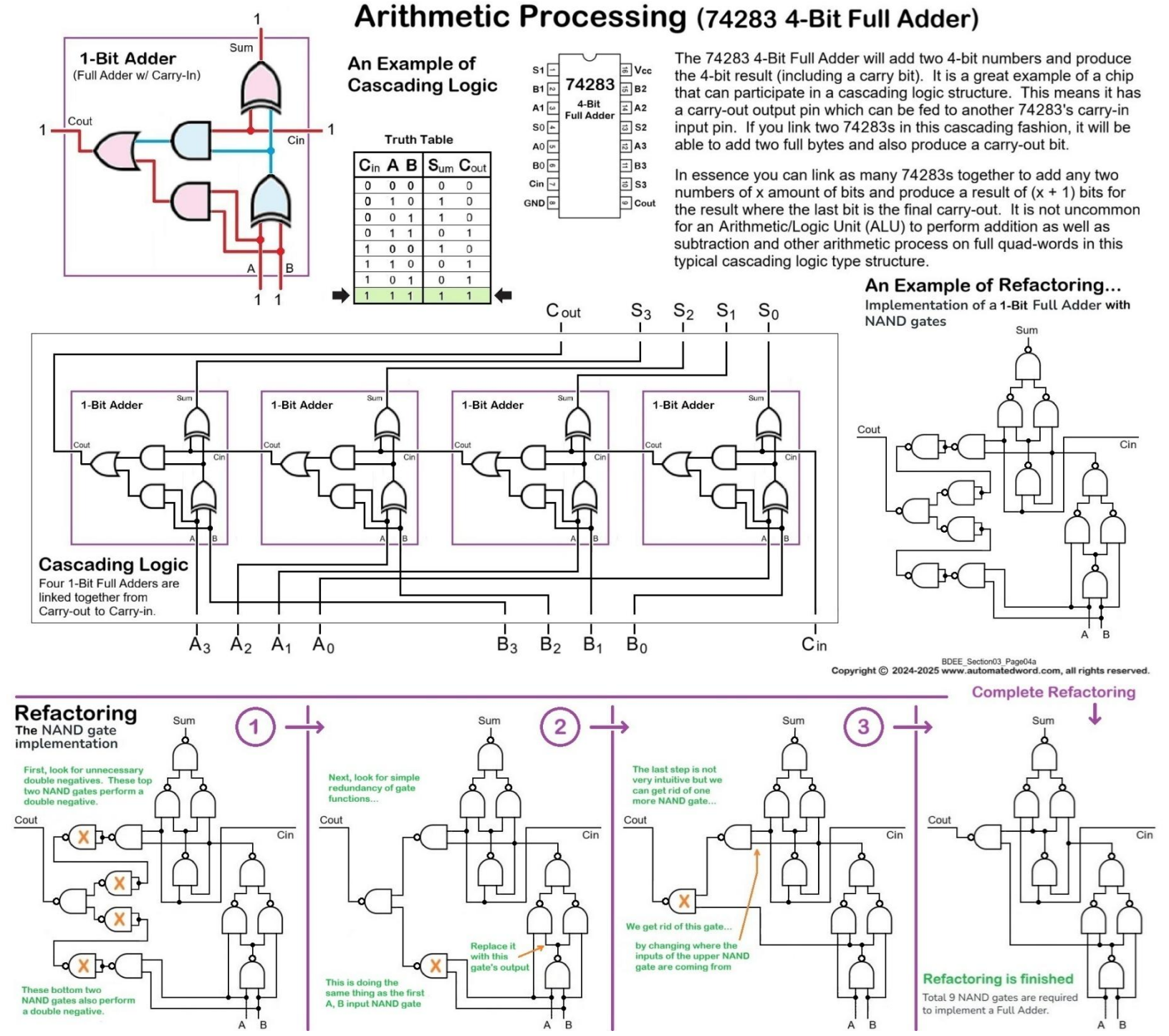
You can think of it this way: The 74245 has a set of 8 pins on one side (side A) and another set of 8 on the other side (side B). Each set of pins can be either inputs or outputs depending upon the signal level of the "DIR" input pin. When DIR is high (1), the A-side pins become inputs and the B-side pins become outputs thus allowing voltage level signals to travel through the chip from A to B. Conversely, when DIR is low (0), the B-side pins become inputs and the A-side pins become outputs thus allowing voltage level signals to travel through the chip in the other direction, from B to A. The Output Enable pin (OE) will turn the outputs to the high-impedance (high-Z) state if OE is high (1) else, if OE is low (0) then high-Z is off.

If you think about it, you can accomplish the exact same Tri-State functionality by using two 74244's each facing the opposite direction and being wired to the bus as is depicted in the above diagram. But why use twice as many chips to accomplish the same task – it is a lot more wiring and consumption of power.

The above schematic shows a 74245 at either end of a system data bus – one at the CPU and the other at a RAM module thus facilitating an efficient bi-directional data bus. This displays among other buffering circuitry a very common specific operation which the 74245 is perfectly designed for. Datasheet: 74LS245

7400 Series Guide: 74LS245 (Octal Bus Transceiver) YouTube - How To Test 74LS245 Transceiver

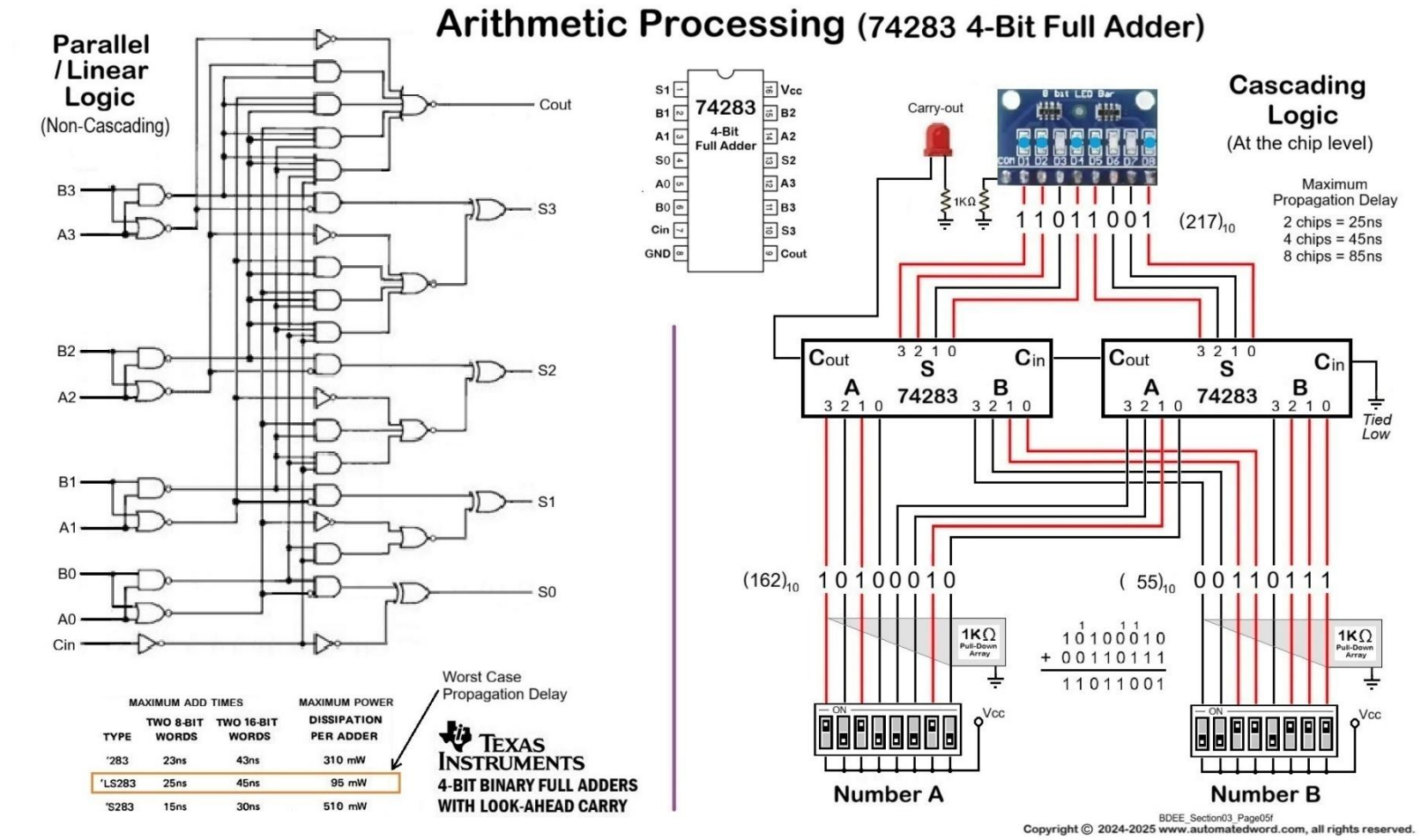
  (Back to TOC)



Here we will begin to examine the circuitry that allows computers and other electronic devices to analyze and manipulate data to make calculations and otherwise, process data. What could be more fundamental than being able to add two numbers together. Although, in this slide, we begin to discuss a particular chip known as the 74283 – a 4-Bit Full Adder, we will first cover types of circuits that are used to accomplish complex tasks in small bites and how they can be scaled up to handle large scale processes. For example, this slide covers the simplest circuit that can add one column of binary numbers – this means adding one number that is only one column (1 bit) wide with another number which is also only one bit. By replicating this same simple circuit many times over and then chaining them together from one carry-out to the next carry-in makes it possible to add two numbers of as many columns as needed for very large numbers.

As a side note, this is also a perfect circuit to demonstrate the notion of “refactoring”. When implemented as NAND gates we can get a glimpse of a process that I will leave you to learn in more advanced courses. Ultimately, this is called “Cascading” type logic.

  (Back to TOC)



As it turns out this slide depicts the real internal workings of the 74283. While the previous slide explained a logic circuit by "cascading" a repetition of a minimal circuit in order to scale up to a much larger task and as elegant a notion as it might seem, it is very seldom, if ever, the most efficient design of a circuit. By that I mean it is usually terribly slow in its operation as the circuit requires the electrons to flow through too many logic gates in succession (consecutive or serial fashion) in order to arrive at the final summing outputs – especially to the carry-out.

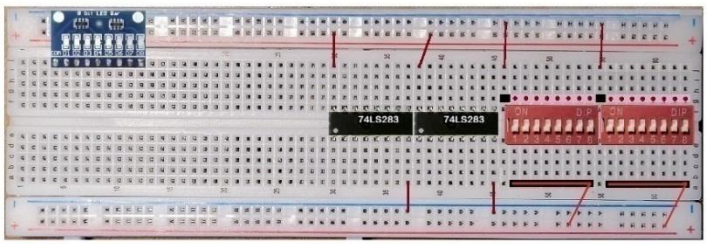
Here is a more detailed explanation… If we say that the propagation time for a TTL gate embedded in a chip is about 5ns nanoseconds then, looking back on the last slide of the "cascading" 1-bit adders starting from the rightmost part of the circuit, the worst case path through all four bits of addition starts with A0, B0 and Cin (carry in). From there the worst case scenario is through three gates to arrive at that first bit's Cout (carry out) – 15ns. That means the next bit will not receive the necessary Cout signal from the first bit until after 15ns meaning, under the worst case scenario, the second bit cannot finish its calculation to its Cout for another 10ns as that must go through at least two more gates. Likewise, for the same reason, the final two 1-bit adders must wait for the previous Cout which adds another 10ns each despite the fact that all four A and B inputs will begin to process the addition in parallel (at the same time), each column must wait for the previous carry-out before it can complete its column's addition. A total worst case scenario of 45ns.

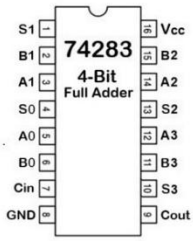
The schematic on the left depicts how the real 74283 circuit is designed to accomplish the same addition of two 4-Bit numbers in only the worst case of 25ns as shown on its datasheet from Texas Instruments. Even though it has more gates than the cascading design, there are only five stages of consecutive flow. In other words, as shown in the TI schematic you will see 5 groups of gates – all the gates in a group are processing signals at the same time so if you examine it carefully you will notice that the longest path for this circuit is no more than five consecutive gates. The schematic is typically drawn to illustrate this fact. On the right is our circuit design for cascading two 74283's.

The 74283 chip has Cin and Cout pins so they can be connected from one chip's Cout to the next chip's Cin in a circuit so they can be cascaded to add two very large numbers together keeping in mind that each cascaded chip adds 20ns to the final summing process as stated in the depicted datasheet numbers. So, all technologies used being equal, the 74283, due to its Parallel/Linear logic design, is capable of adding two 16-bit words in the same amount of time that it takes a cascade design to add a half of a byte (two nibbles) together.

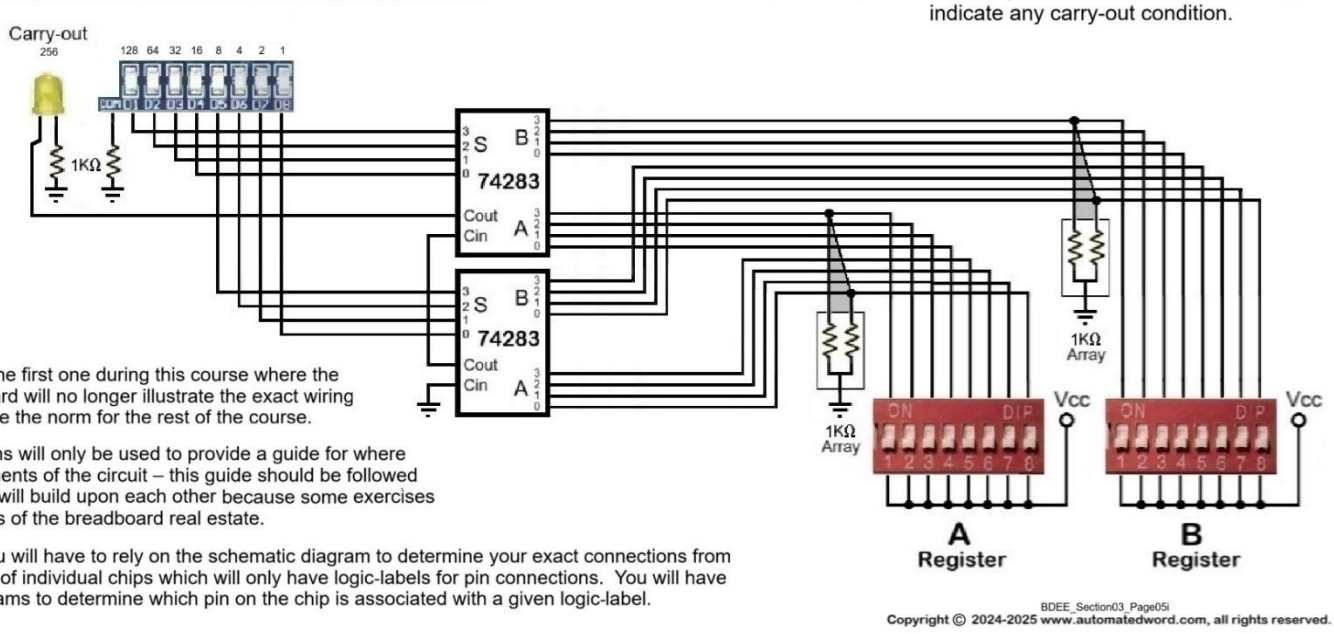
  (Back to TOC)

Exercise: 8 — Arithmetic Processing (74283 4-Bit Full Adder)





The purpose of this exercise is to test a circuit which cascades two 74283's and simulate the A an B 1-byte registers of an ALU (Arithmetic/Logic Unit) for adding two 1-byte numbers. The two 8-line DIP switches are used to simulate the bit values retained in the A and B registers. The 8-line LED panel should indicate the sum of any given setting of the switches and the yellow discrete LED should also indicate any carry-out condition.



NOTE:

This exercise represents the first one during this course where the illustration of the breadboard will no longer illustrate the exact wiring of the circuit and this will be the norm for the rest of the course.

The breadboard illustrations will only be used to provide a guide for where to place the major components of the circuit – this guide should be followed exactly ensuing exercises will build upon each other because some exercises will require critical amounts of the breadboard real estate.

Also, from this point on you will have to rely on the schematic diagram to determine your exact connections from the logical block diagrams of individual chips which will only have logic-labels for pin connections. You will have to refer to the pinout diagrams to determine which pin on the chip is associated with a given logic-label.

BDEE_Section03_Page05i

When you have constructed this exercise make sure you have turned all eight switches to the OFF position on both 8-line DIP switches before testing... When power is turned on, all LEDs should be off.

The best live test for this circuit is to start with the A-Register switches. First, make sure that all the B-Register switches are OFF (this, of course, will represent a value of 0 in the B-Register). Next, start turning the switches of the A-Register ON from right to left. Since you are adding the value of A to the value 0, you should see the LED panel light turn on corresponding to each switch of the A-Register that you switch ON (from right to left).

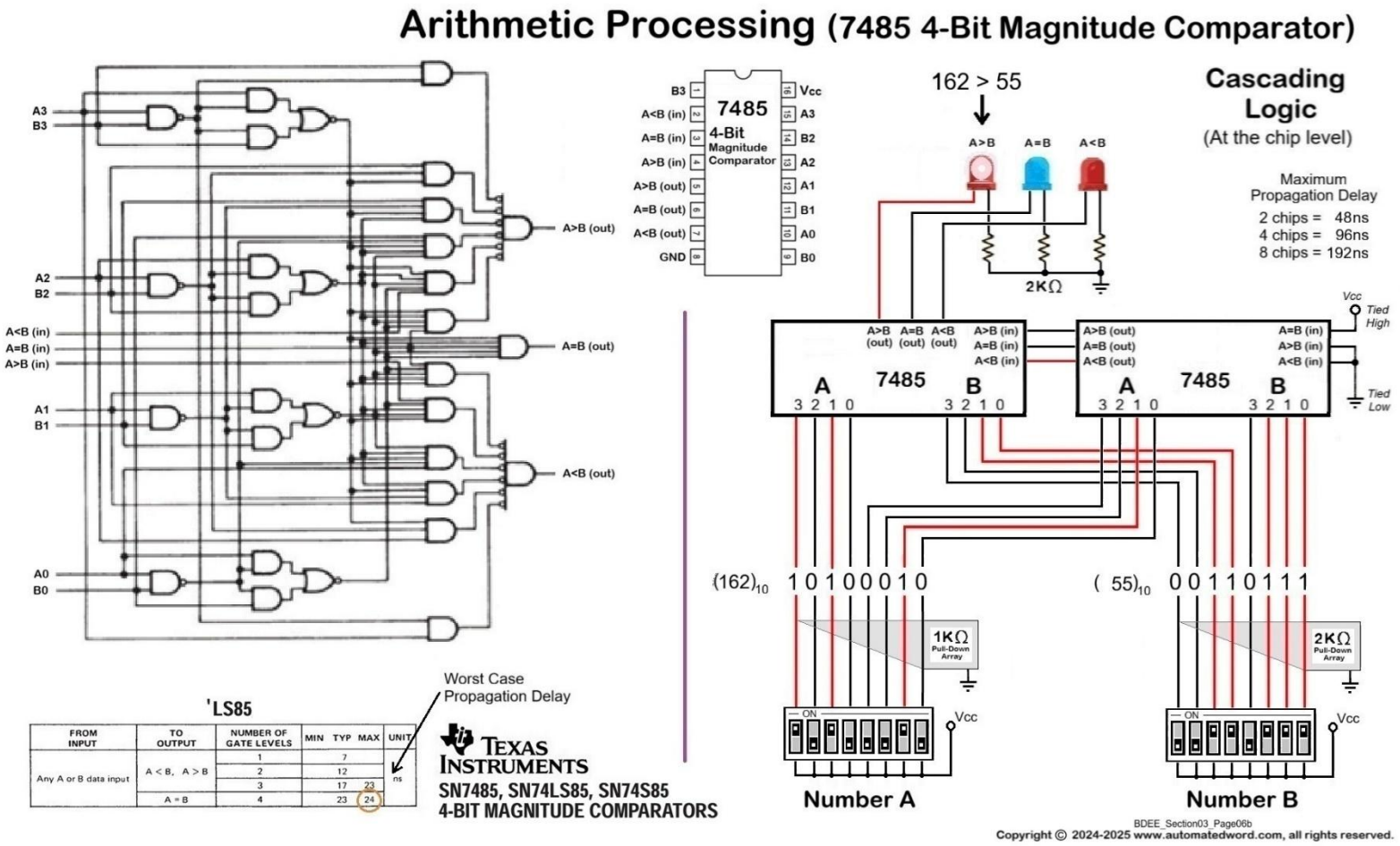
Next, turn all of the A-Register switches OFF so that A represents 0 and perform the same right to left function on the B-Register switches. You should see the LED panel lights turn on corresponding to the B-Register switches as they are switched ON of course, because you are adding B + 0.

Finally, with all the B-Register switches left ON, add 1 (the rightmost switch) from the A-Register and you should see all the lights of the LED panel go off and the Carry-out LED should turn on. This will be the correct result of adding 255 + 1 – the Carry-out represents 2 to the eighth power column (or, 256). When this test is complete, you can be assured that all of your wiring has made the correct connections and any number you enter in the A or B registers will be properly added together (including Carry-out).

Did you know that the 74LS283 is also used to subtract numbers?... Datasheet Viewer: 74LS283

Electronics Hub - Binary Adder and Subtractor Circuits CircuitVerse - 4-bit_adder_and_subtractor
YouTube - 4-Bit Full Adder (with 7-segment display) YouTube - Designing a 7-segment hex decoder
Electronics Lab - Binary Subtractor - Binary Adder YouTube - 7483 WITH PROTEUS SOFTWARE
Carry Look-Ahead Adder - Working, Circuit and Truth Table YouTube - The 74283 4 BIT ADDER
Binary vs. Binary Coded Decimal: YouTube – BCD (Binary Coded Decimal) Adder – Neso Academy

  (Back to TOC)



A computer needs to be able to sense an outcome of a process in order to make decisions about how to proceed during its operation. The magnitude comparator is a good example for how a computer can test the condition of the result of arithmetic operations. Since the magnitude comparator senses the fundamental relationship between two numbers – that is, to decipher if the numbers are less than or greater than each other or, if they are in fact equal. By testing the Boolean value of the three output lines (A<B, A>B and A=B) for "true" or "false" (1 or 0) this gives the CPU the essential tool for deciding one of three possible paths to take for proceeding in its operation regarding numerical processing.

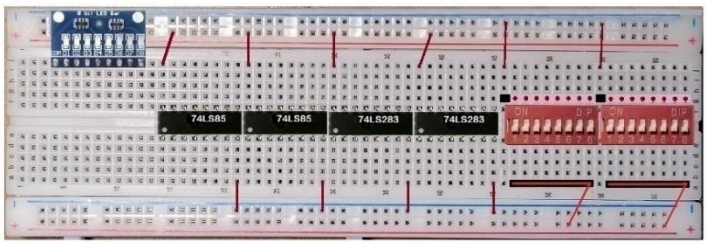
If a computer is useful at all, it is certainly made more useful if it can make decisions in an automatic fashion and the magnitude comparator is part of that type of mechanism. The 7485 4-bit magnitude comparator takes two 4-bit numbers for comparison and issues that relationship at its output pins. The three output pins are always in one of the three possible states and that is, one of the pins is emitting a high signal (1 for "true") and the other two pins are emitting a low signal (0 for "false"). The logic circuitry of a CPU can be wired to incorporate these output signals as input for circuitry that will determine the procedure of operation. This will become apparent as we design our CPU in Section 04.

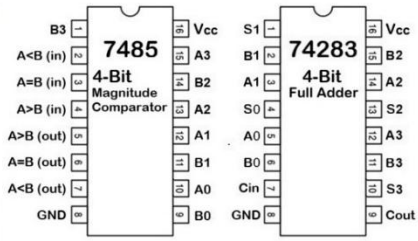
One important note about this slide is the specification information given by Fairchild Semiconductor's datasheet for the 74LS85. You should note the table in the lower left of the slide above. This type of table appears on every datasheet from the manufacturers of digital electronic chips. When you look at any of these datasheets, among a slew of information about the electronic properties of the inputs and outputs of the chip and, usually at the end of it, you will notice this type of a table where the propagation delay of its well tested operation is listed – just look for the table where the "Unit" is "ns" which stands for nanoseconds (billionths of a second). You should also see this table for the 74LS283 adder on page 5. We will be incorporating this type of information in our engineering analysis as we begin to design our BDEE Computer on schematic diagram sheets in the slides to come.

  (Back to TOC)

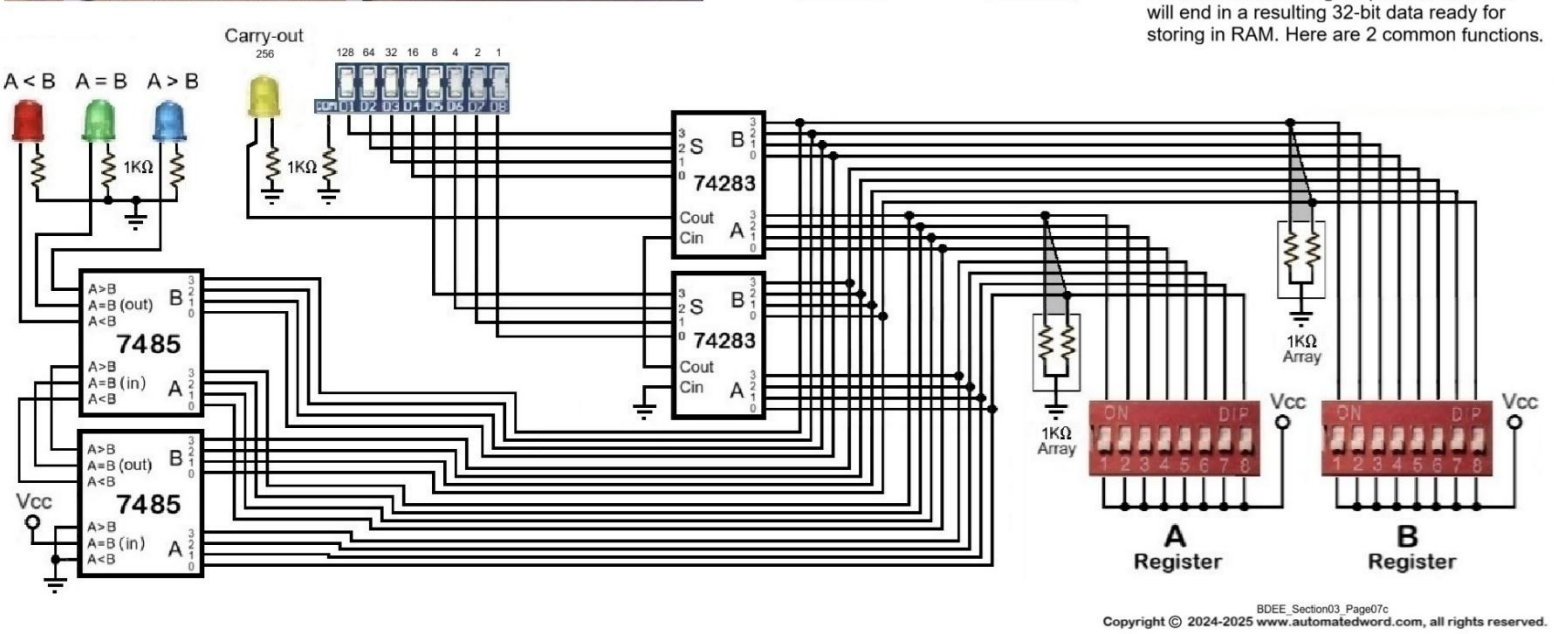
Exercise: 9

Arithmetic Processing (7485 4-Bit Magnitude Comparator)





The purpose of this exercise is to build a circuit that will simulate internal functions of the ALU. The Arithmetic/Logic Unit which works in conjunction with the CPU is largely a collection of functions, each of which, have an internal bus input from two 32-bit CPU registers typically labeled the "A" and "B" registers. These registers are chunks of data that will be combined in functions like adding, subtracting and comparing as well as other arithmetic or logical processes which will end in a resulting 32-bit data ready for storing in RAM. Here are 2 common functions.



When you have constructed this exercise make sure you have turned all eight switches to the off position on both 8-line DIP switches before testing... Now this may seem a bit tedious compared to testing the 74283 Adder, but you will know that your circuit is 100% correct shortly without testing every possible pair of numbers. First, when power is turned on, all LEDs should reflect the A=B state. Since all switches are OFF, you are of course, comparing 0 to 0 and so, the A<B and the A>B LEDs should be off while the A=B LED should be on.

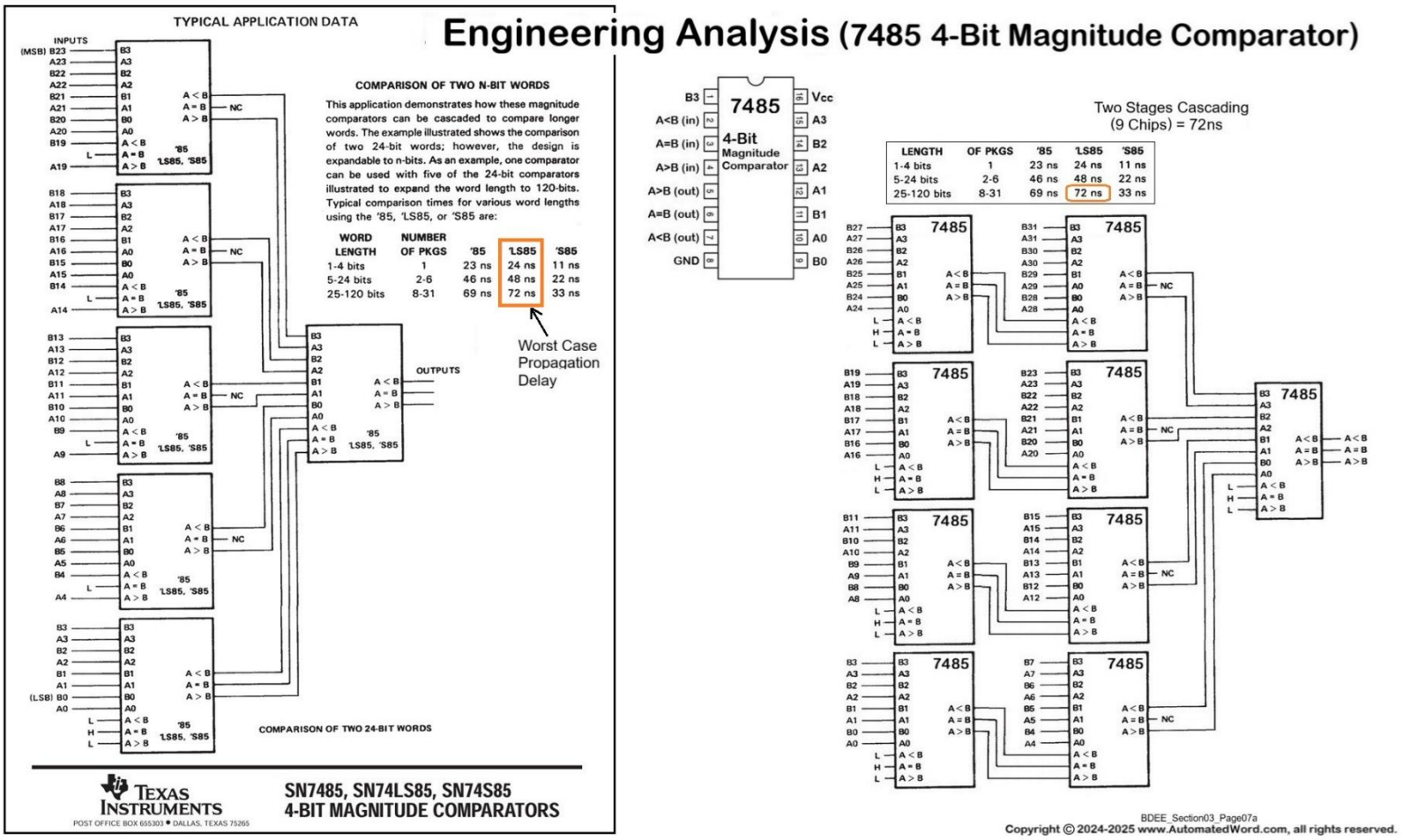
Next, switch the 1's column bit ON (the rightmost switch) of the A-Register. On the B-Register, turn the 2's column bit ON (left of rightmost switch) and you should see that the output LEDs reflect the A<B state. Now, turn the B-Register switch OFF and, on the B-Register, turn the 1's column bit on (rightmost switch) – you should see the output LEDs reflect the A=B state. Now, turn all switches off on both the A and B.

Next, turn the 2's column bit on (left of the rightmost switch) of the A-Register. On the B-Register, turn the 4's column bit on (left of the 2's column) and you should see that the output LEDs reflect the A<B state. Now, turn the B-Register switch off and, on the B-Register, turn the 2's column bit on – you should see the output LEDs reflect the A=B state. Now, turn off the B-Register switch and turn the 1's column switch on and notice that the output LED's reflect the A>B state. Now, turn all switches off on both the A and B.

Finally, proceed in this same manner setting the next higher order bit to the left on the A-Register and test it against the B-Register where you systematically turn on one bit higher than what is on the A-Register the, the same bit and finally, the one bit lower all the while check that the output LEDs reflect the proper state.

If all the possible states appear on the output LEDs, you can now be assured that your wiring is correct and that the cascaded comparators will accurately compare any two 1-byte numbers (0 - 255).

  (Back to TOC)



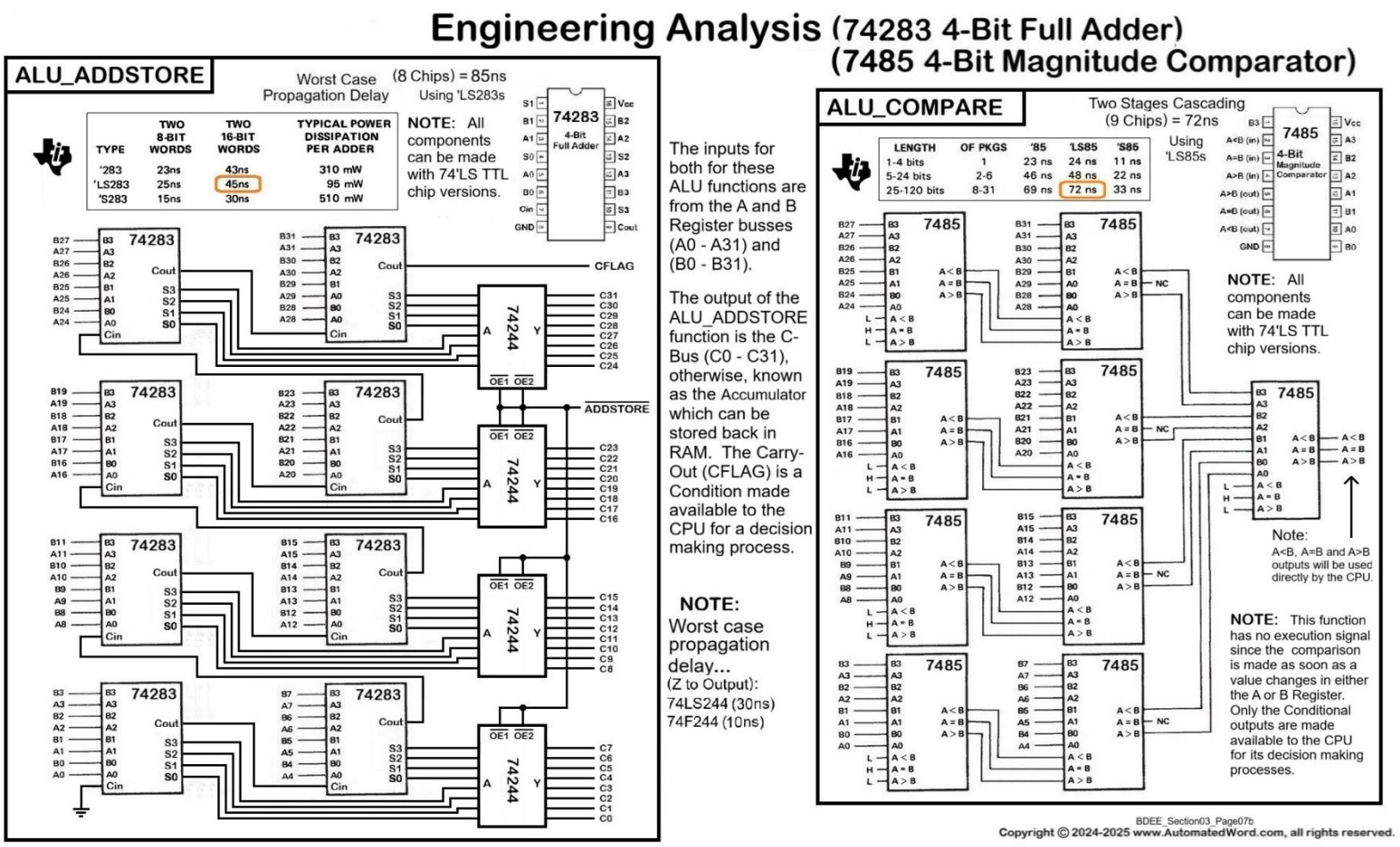
On the left side of this slide is a page taken from the datasheet of the Texas Instruments 74LS85. Very often, the chip manufacturer will insert pages in their datasheets that show “Typical Application Data” (as they have titled this page) or something to that effect. This is to demonstrate how the chip can be used in its most effective and most efficient way and also it may reveal technologies or methodologies used in the underlying electronics. In this case, the page reveals something very important about the 74LS85 regarding its speed and logical performance which we are about to discuss. Note that this would be the same regardless of who the manufacturer is because all of them comply to industry wide standards.

You should understand that one could design a circuit to compare numbers of virtually any size by simply cascading the logic (at the chip level) as we did in the previous exercise. However, if you want to compare two 32-bit numbers with a simple cascaded design, it will take 8 chips and this would take 192ns to complete a 32-bit comparison as is implied elsewhere in the datasheet. So, on this “Typical Application Data” page, the manufacturer is showing us that if we configure a circuit in a way that utilizes a Parallel/ Linear approach while minimizing Cascading we can achieve significantly faster propagation delay as we do intend to compare double-words (4-byte words) or, 32-bit numbers in our BDEE Computer…

First, notice that the leftmost bank of 7485s feed their A<B and A>B outputs to the final 7485’s B and A inputs, respectively. If you think about it, this is a perfectly valid approach of interpreting those output signal as being actual data. This is an inherent property of the internal circuitry which can be traced.

Finally, in order to compare 24-bit numbers (3-byte words) with 6 chips but yet only 1 stage of cascading, the page shows us something else – that another inherent property of the internal circuitry will allow the comparison of the lowest order bits from A and B of the upper (higher order) chips on the right to be connected to the A>B and A<B inputs respectively as long as A=B input is tied low (0). So, with all this knowledge, I designed our 32-bit comparator on the right side of the slide for 72ns instead of 192ns.

  (Back to TOC)



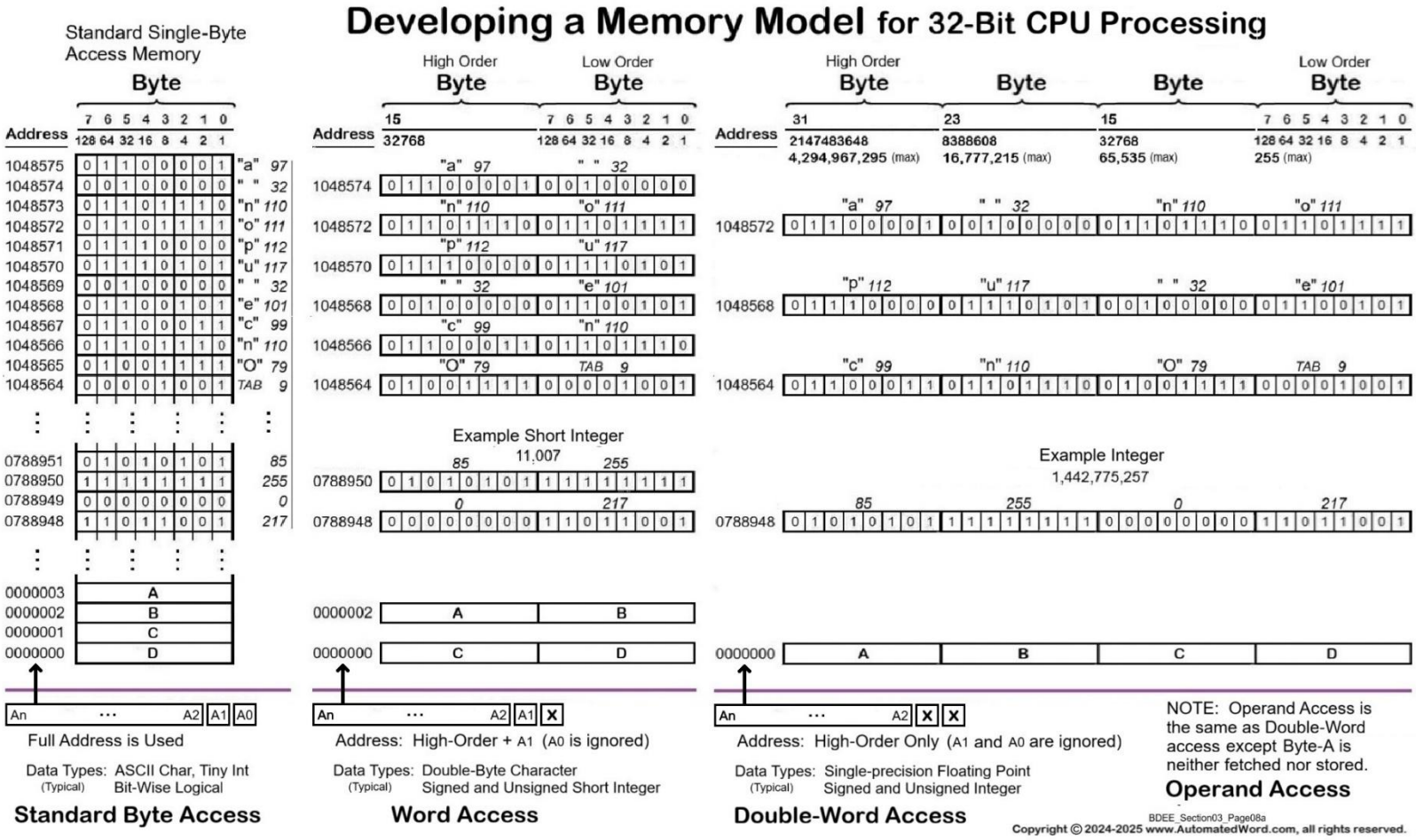
To begin with, a computer's operation is dependent upon a notion that we call "Procedural Logic". This means it is a machine, or mechanism, that moves from state to state over a period of time – and by this, we mean that a computer should move through these states very quickly. For reasons that will be explained in detail later, our BDEE Computer will be designed to perform 5,000,000 instructions per second. This is extremely slow when compared to computers and cell phones today. They operate at billions of instructions per second – and super computers at trillions/sec. The technology of very large complex systems like today's computers and mobile phones has evolved to such a degree that the number of logic gates that can fit on a single chip is numbered in the billions and as it turns out, this process of scaling down in size has led to a large scaling up in processing power – huge leaps in speed.

The point is that our BDEE Computer, in order to perform 5,000 instructions/sec., will have an instruction cycle time of 200ns - 1 second divided by (1/200,000,000,000 "200 billionths") = 5,000,000. 200ns is 0.0000002 seconds. Without further explanation here, this means that any function performed by the CPU while processing data including testing outcomes of those processes must be accomplished in less than 100ns. So, while the ALU_ADDSTORE module above accomplishes its 7 stages of cascading logic in 85ns, the ALU_COMPARE operation needed a little more engineering analysis to come up with the circuitry that will conform to the 100ns restriction. The 74LS85 populated circuit shown above is 72ns.

The two schematic sheets in the above slide represent our first two official design sheets for modules that will make up the overall design of our BDEE Computer. They will be referred to by their Tag Names. Later, in Section 04, we will discuss how the computer becomes a state-machine driven by a clock pulse.

Module Datasheets ('LS Version): 74LS283 – 74LS85 – 74LS244 / ('F, or Fast Version) 74F244
74LS85 ICs | Online Electronics Simulator 4 Bit Magnitude Comparator Logic Circuit Diagrams
YouTube - Design a Magnitude Comparator in Digital Logic Design? 1-bit, 2-bit, 3-bit, n-bit Comparator

  (Back to TOC)



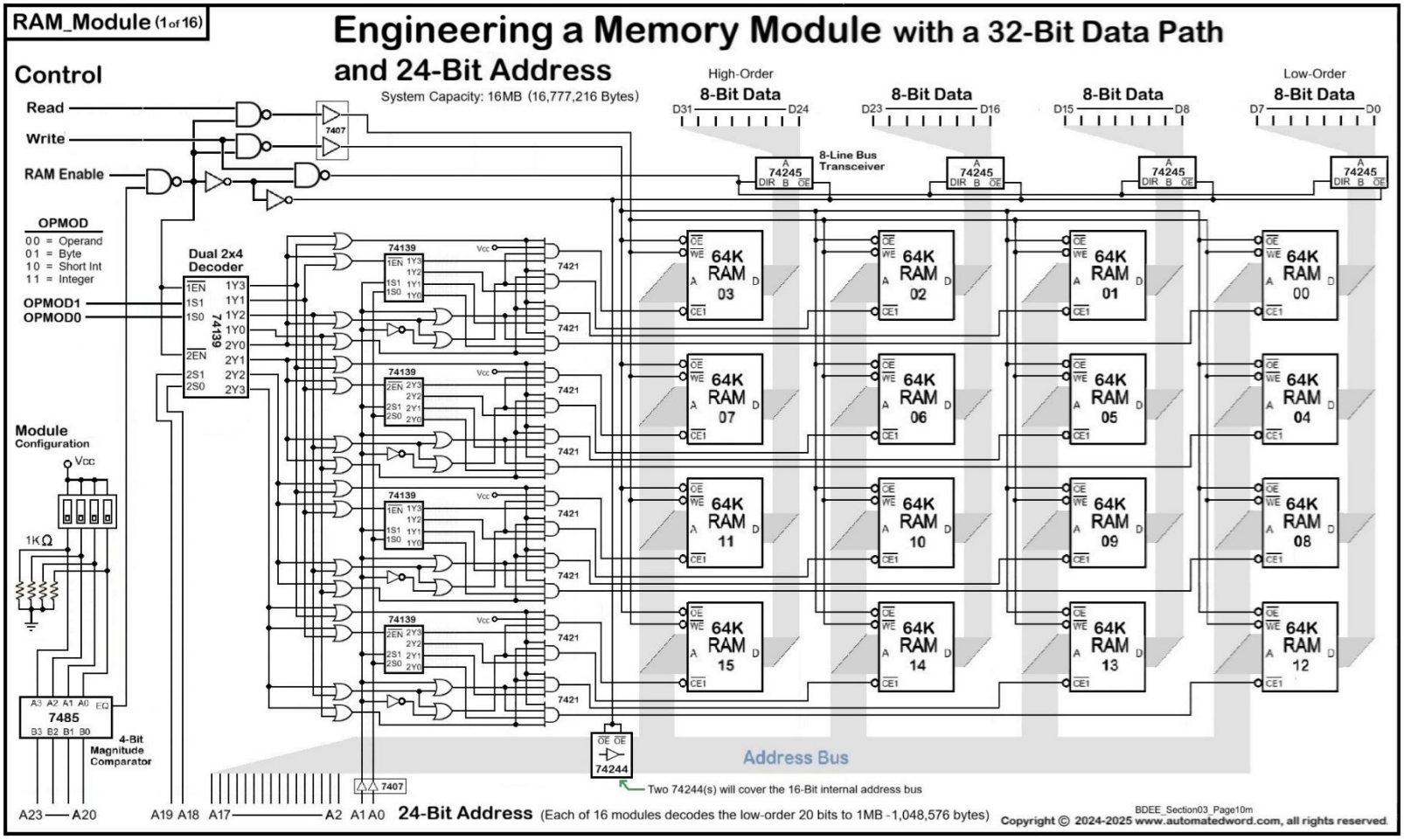
There came a point (I think, in the late 1970's / early 1980's) that a huge leap in computer processing power needed to happen in order for us to experience the kind of hardware we have today (laptops and cell phones). After much development in software (programming) concepts, it became apparent that for the most part, the "Integer" was a very useful size of numerical data in terms of developing the vast majority of applications. In other words, the 32-bit word length was able to represent a sizable enough number to facilitate the most common algorithms for the typical programs of the most common everyday uses of computers. The 32-bit number happens to be 2,147,483,248 – approximately 2.15 billion.

What would cause the giant leap in processing power (mostly in terms of speed) is the ability for the CPU to retrieve and store, as well as, process "Integers" (4-byte words) all at the same time in a single cycle of operation as opposed to going through many more cycles of processing individual bytes. And, part of this leap forward is to re-design memory (RAM) to transport, store and retrieve four bytes at a time (during one memory access cycle. This is what this slide is all about – designing a concept for this type of RAM.

In this concept we also retain the ability for the CPU to access and store smaller chunks of data because they have their uses as well. For example, it would become a terrible waste of RAM space if each number used in an algorithm that would never exceed, say, 65,536 had to be stored in four bytes or, double-byte characters, the same, when they could be more efficiently (space-wise) stored in two bytes at a time. Ultimately, this concept of retaining capabilities also applies to byte sized data – it is better not to lose it. This concept, you might say, is a good example of "maintaining backwards compatibility" – for hardware.

Anyway, the long and short of it is we provide 4 different ways for the CPU to access different sized data. Leftmost in the layout above is the retained standard singe-byte access which uses the entire address (all of its bits). Next (to the right), is the Word (Short Integer) and then Double-Word (Integer) and finally, the Operand – a method for efficiently modifying actual machine instructions we will talk about later.

  (Back to TOC)



This is the implementation of our 32-bit wide data access memory module designed specifically for our BDEE Computer. There are some engineering concepts that will be covered in more detail to follow but, for now, let's start with this: All inputs into this module must connect with a single (one, and only one) logic input on this board. The reason for this is pretty simple. At the other end of those lines is the CPU or other component of the system which is driving the bus lines with a TR-State Buffer/Line-Driver. Those buffer/line-drivers have a nice wide fan-out capability as they are expected to be connected to many inputs along the system-wide busses they will be driving.

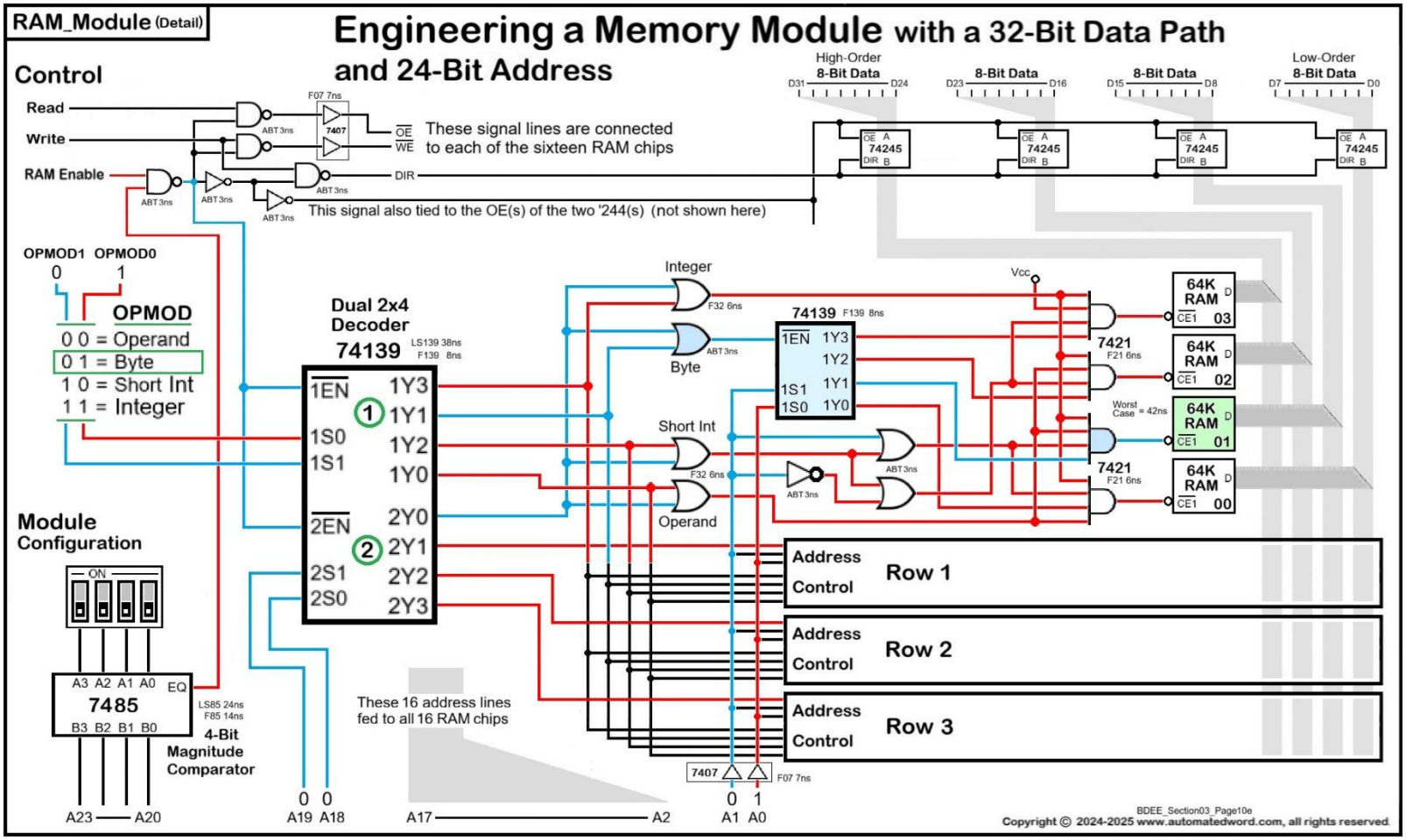
As with the ALU_ADDSTORE and ALU_COMPARE modules, this module has engineering considerations that must be addressed in order for it to fit the overall design of accommodating a 200ns instruction cycle. We will be using a different family of 7400 TTL chips. Manufacturers of 7400 chips also produce what is called the "F" series and we will be using "F" series chips in the design of this memory module. They operate much faster than the "LS" versions In fact, most 74F series chips are at least twice as fast as their 74LS versions and in several cases can be as much as three or four times faster.

Here includes the list of parts for the memory module including links to the datasheets for each chip. You should compare speeds between the LS and F version of these chips (just look for "ns" in the "Unit" column of the tables)…

7485	(1)	'F85	4-Bit Magnitude Comparator	
74244	(2)	'F244	Octal TRI-State Buffer	(use 'LS244)
74245	(4)	'F245	Octal TRI-State Bus Transceiver	(use 'LS245)
74139	(3)	'F139	Dual 2-Bit to 4-Line Decoder	
7421	(8)	'F21	Dual 4-Input AND Gate	
7400	(3)	'F00	Quad 2-Input NAND Gate	
7432	(6)	'F32	Quad 2-Input OR Gate	
7404	(1)	'F04	Hex Inverter	Here are three possible SRAM chips that could be used…
7407	(1)	'F07	Hex Non-Inverting Buffer	

SRAM (16) 64Kx8 – 512K-Bit (use 25ns) ISSI– IS61C512 Winbond– W24M512 UMC– UM61512A

  (Back to TOC)



Since the RAM_Module has four rows of RAM chips to select from, this slide is a detailed view of the circuitry used to select which of the chips in that row will be selected dependent upon the signals from the CPU (OPMOD0 and OPMOD1). Each row of RAM chips in this module have exactly the same circuitry for selecting the individual chips. One row at a time is selected by address A18 and A19.

The above slide depicts the first row under selection by A18 and A19 going to the second 2x4 decoder of the main 74139. As the CPU has issued a “01” on OPMOD0 and OPMOD1, a single individual byte has been selected because the first 2x4 decoder in the main 74139 is reading those signals. The 74139 highlighted in blue is responsible for selecting the individual byte by enabling RAM chip “01” which is selected by the low-order address bits A0 and A1. This is when the RAM chip will read or write the byte.

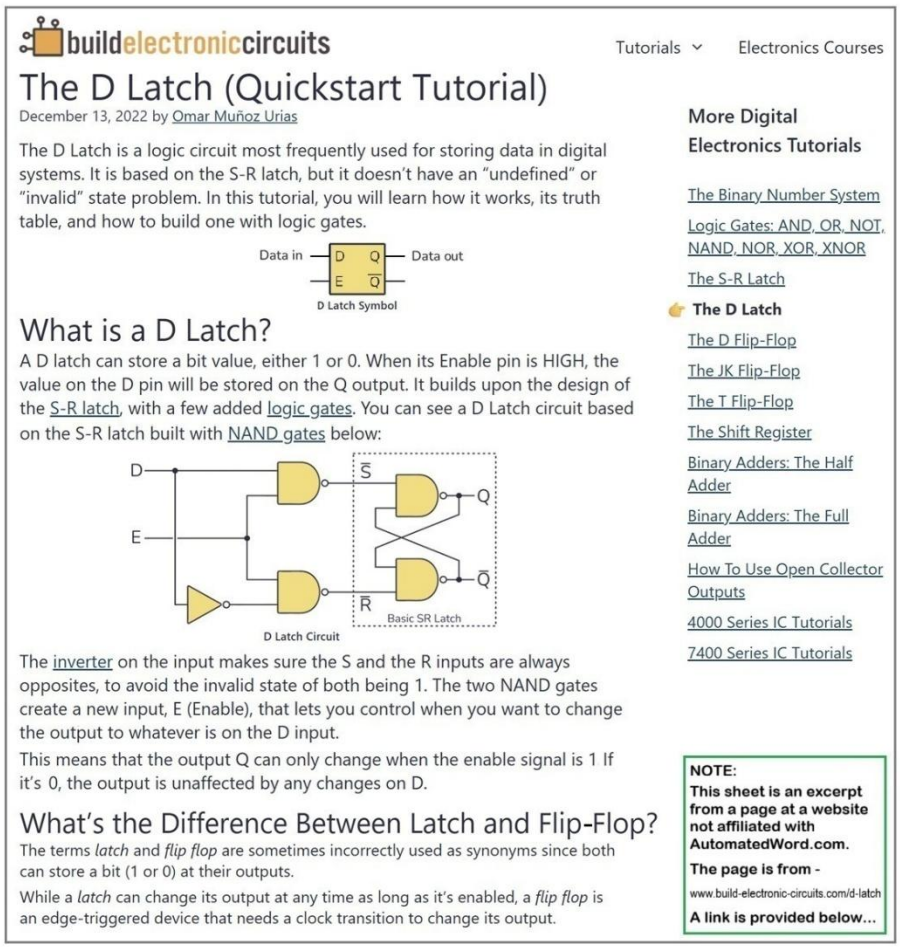
I have noted the chip propagation delay times for the “F” versions of the chips in the above schematic and you should notice that the worst case scenario of 50ns to select individual RAM chips is when the CPU is asking to select the high-order two bytes during a “Short Integer” (Word) access.

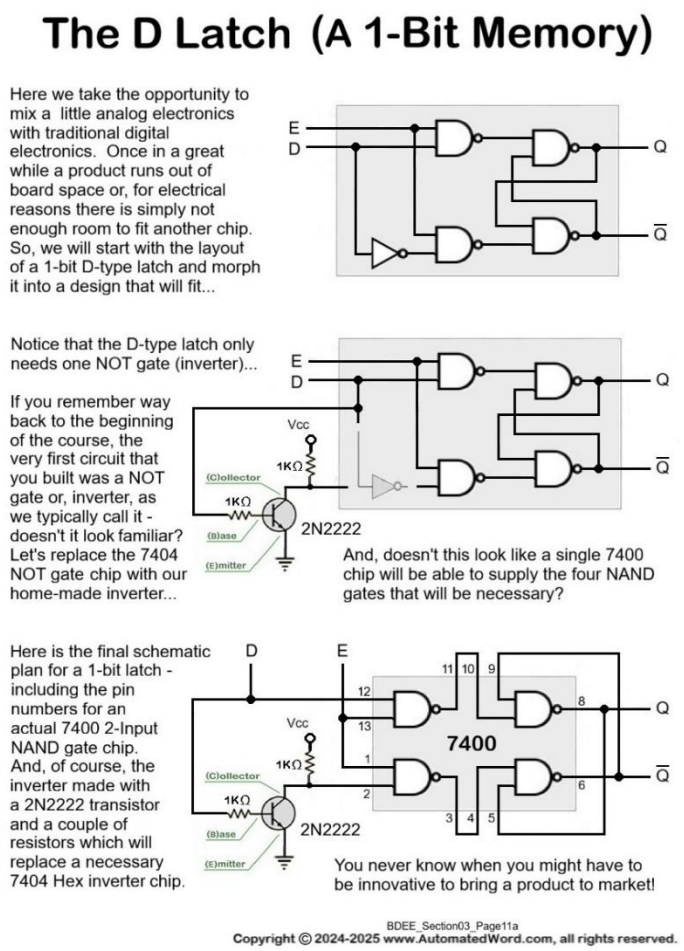
During any memory read or write cycle (which are performed in exactly the same amount of time), the CPU has set all of the signal, address and data line to this module – there is no intervening clock cycle with these new RAM chips. It has set all control lines for Read or Write and Access-Mode, it has placed the intended address on the address bus as well as any data on the data bus (if it is a memory write cycle) all at the same time at the very beginning of the memory access cycle.

Apparently, for the past few years, there has been advancement in Dynamic RAM (DRAM) as opposed to Static RAM (SRAM) and DRAM is now fast enough to become the predominant technology in use…

SRAM vs DRAM vs SDRAM YouTube - RAM Timing Explained The Full Guide to RAM Speeds
YouTube - Does RAM Speed REALLY Matter? Best DRAMs For AI DDR5 RAM in 2025 |GamerTech

  (Back to TOC)





Link to BuildElectronicCircuits - [The D Latch (Quickstart Tutorial)]

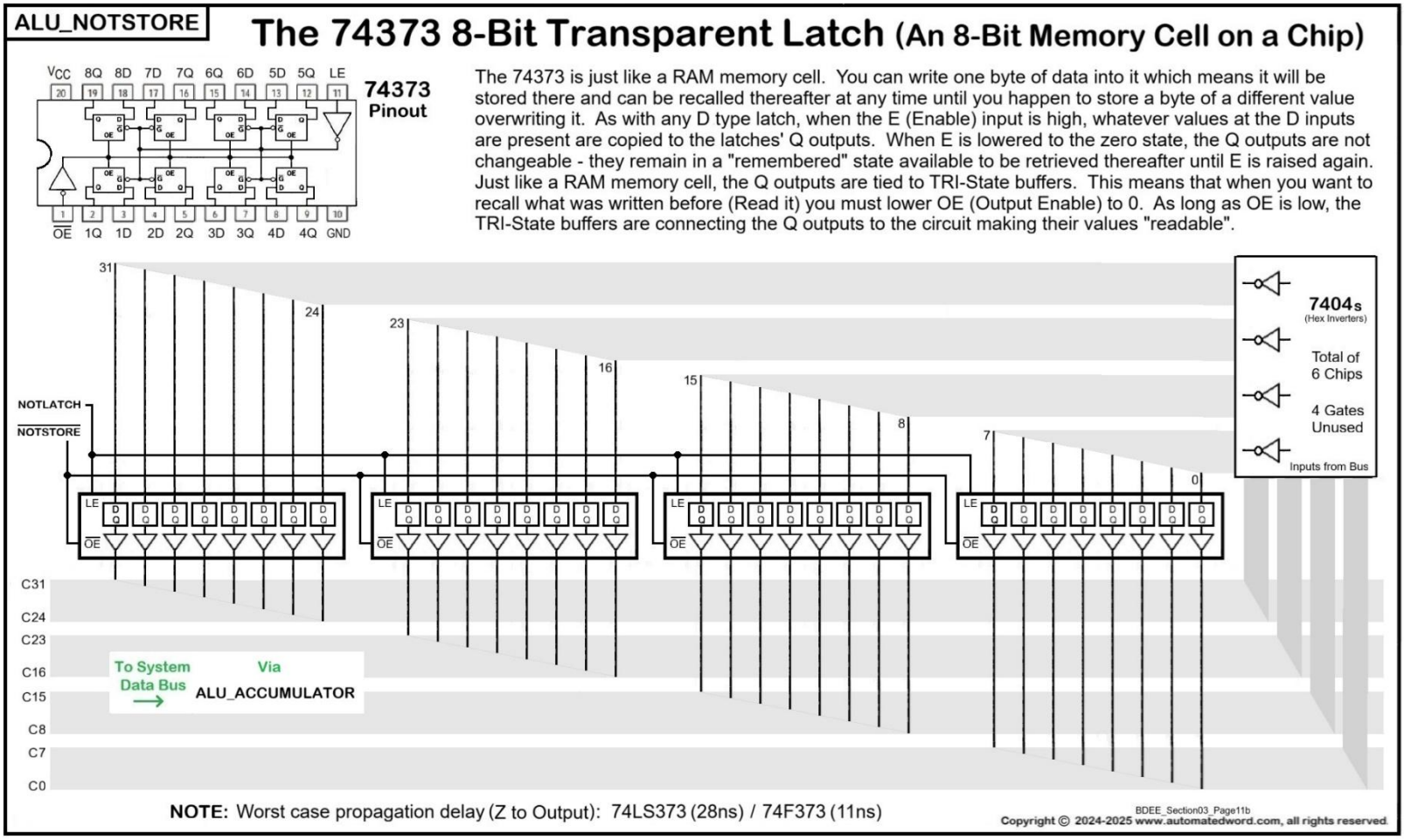
This is the fundamental way that a bit of data (1 or 0) is remembered in digital electronics. It starts with the S-R Latch (also explained in the set of tutorials if you go to the website). The S-R produces a 1 at the Q output and a 0 at the Q-compliment output when S ("Set") is brought low temporarily. Likewise, the opposite occurs when R ("Reset") is brought low temporarily – that is that the Q is set to 0 and Q-compliment is set to 1. Q and Q-compliment remain unchanged until S or R are momentarily lowered.

The D Latch builds upon the S-R to conform to a common usage as a persisted bit of memory by adding inputs that are conducive to common circuitry for memory operations in CPU circuitry (which is basically identical to circuitry found in static RAM memory cells. The inputs to a D Latch are "D" (for a bit of Data) and "E" (for Enable). Essentially, in this D-type configuration, the state of the D input (1 or 0) is carried to the Q output when E is raised to 1. What makes this a "Transparent" latch is the fact that as long as E is high, the value of D is copied to Q meaning that if D changes value while E is high, that changed value is also reflected in Q. Finally, when E is lowered, D is captured ("Latched") and Q is persisted until E goes high again. As long as E is low, a change in D's value will not be "Followed" to Q – Q is "remembered".

THIS IS IMPORTANT! – Watch these three videos in order… 1) [YouTube – Basic SR Latch]
Moving into State Logic, you need to know Latches and Flip-Flops 2) [YouTube – Gated SR Latch Examples]
which comes later: [Latches in Digital Logic - GeeksforGeeks] 3) [YouTube – The D Type Latch]

At this point you are very much encouraged to visit https://www.buildelectroniccircuits.com and explore the Tutorials. Having perused thousands of websites regarding digital electronics, I have found this one (among several others) to be one of the best written explanations of fundamental concepts. It may not be the most comprehensive library but, it is very good at explaining not only digital logic components but does a great job of describing some of the underlying electronics behind digital logic (something that is intentionally left out of this course). You should now have enough of a handle on the terminology of digital electronics to glean the essence of some technical documentation/discussion of the underlying electronic concepts.

  (Back to TOC)



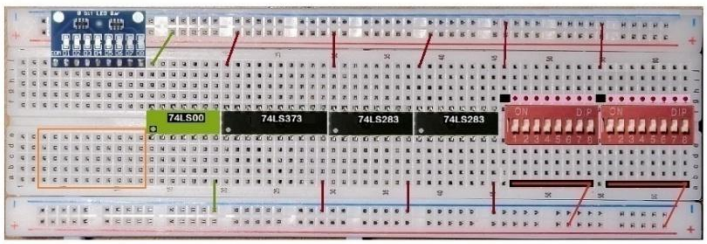
This slide represents another installment of the design sheets for our BDEE Computer. This sheet is the design for the ALU_NOTSTORE function which will need a little further explanation. If I have not mentioned it before, our BDEE computer will have an ALU (an Arithmetic/Logic Unit) which is a part of the CPU (the Central Processing Unit). The ALU contains a number of functions specifically designed for the manipulation of data – essentially meaning, the processing of data. The ALU is divided into sections, each of which perform a function. For example, one section is devoted entirely to adding two numbers together and another to subtracting one number from another – we have already designed and created the sheets for adding numbers (ALU_ADDSTORE) as well as comparing two numbers (ALU_COMPARE). All in all, there can be several dozen separate functions in a modern computer but, our BDEE computer will be much simpler in order to better understand the overall operational circuitry.

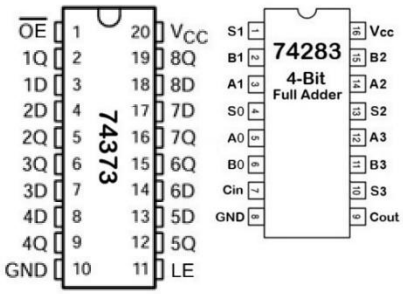
The ALU_NOTSTORE function boils down to the following process… At times during the operation of the computer, the CPU will perform an ALU function and want to store the outcome (result) of that function into memory (RAM). The resulting data will be placed on an internal 32-bit data bus as can be seen in the lower part of the above schematic as four grey colored pathways which represent four 8-bit data channels. These pathways (bus channels) will ultimately connect to the system data bus to the right but, before doing so, notice that there is a tie into a section of 7404 chips which are inverters and, in turn, the inverter outputs are tied to the inputs of four 74373s where the values of what are on those input lines can be stored. What is happening here is described as simply this: Whatever data the CPU sends to RAM will also be inverted and stored in the bank of 74373s. This is called “the data’s compliment” and the purpose for storing the data’s compliment in 74373s is explained below… Datasheet Viewer: 74LS373

Fairly often in programming algorithms it is desired to form the compliment of a result of a mathematical or logic operation. By performing this operation automatically when any math or logic function is performed, it makes it very convenient for the CPU to then store that complimented value into RAM in a later step.

  (Back to TOC)

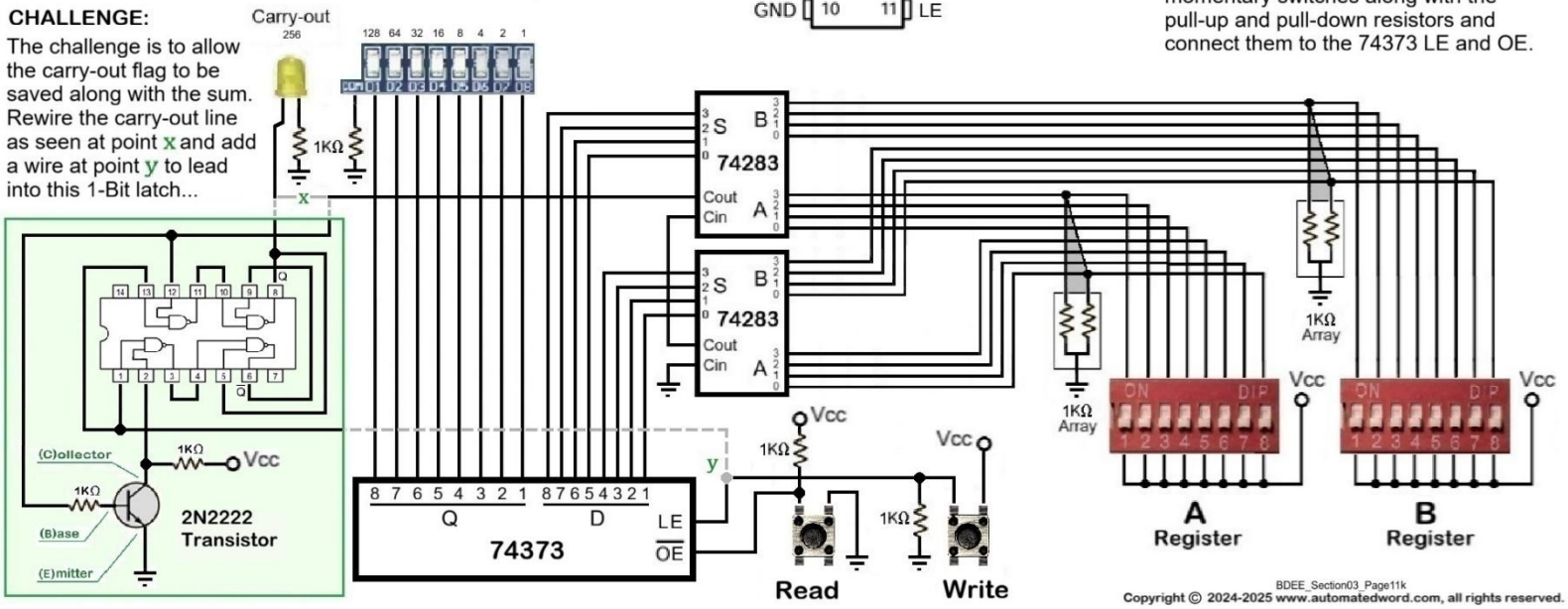
Exercise: 10 Data Latching Process (74373 8-Line Latch with TRI-State Output)





The purpose of this exercise is to test the 74373 as a memory device. We will build upon exercise 8 - utilizing the two 74283 adder chips which will first entail removing any extra circuitry beyond exercise 8. In this circuit, you will rewire the outputs from the 74283s that go directly to the LED panel and plug them into the appropriate inputs of the 74373. Next, you will add the two momentary switches along with the pull-up and pull-down resistors and connect them to the 74373 LE and OE.

CHALLENGE:
The challenge is to allow the carry-out flag to be saved along with the sum. Rewire the carry-out line as seen at point x and add a wire at point y to lead into this 1-Bit latch...



This exercise will demonstrate how the 74373 chip performs its functions as a D latch memory. It will have all the functions that a part of a memory cell has in a RAM chip's stack of memory cells. The only difference is that a RAM memory cell has selection logic as it is, of course, in a huge collection of cells where one has to be selected first.

So, to test this circuit, please turn all switches OFF on the A and B registers before turning on the power. When you turn the power on, you should see that all LED lights are off – This should be the case because the 74373 is a TRI-State device and, if you have wired everything correctly, there should be a 1 at the OE input (Output Enable) of the 74373. This means the Q outputs are basically disconnected from the lines going to the LED panel and, therefore, not supplying any voltage to the LEDs. This will always be the case until you press the momentary "Read" button. Now, press the "Read" button momentarily and you should see that still, no LEDs turn on. This is because the initial state of the 74373 after turning power on should be all eight D latches contain 0's.

Next, without pressing the "Read" or "Write" buttons, enter any two numbers on your A and B registers and then press the "Read" button again and you should still see no LEDs turn on. This is because the 74373 is still holding the value 0 in its memory. So, release the "Read" button and now momentarily press the "Write" button – it will only take a very quick tap that will instantly write the summation that is already on the 74283 Adder outputs into the 74373's D latches. Then, the next time you press the "Read" button you should finally see the sum of your addition.

Now, as a final test, without pressing the "Read" or "Write" buttons, enter completely different numbers on your A and B registers and then press the "Read" button again. Hopefully, you are not surprised to see that the LED lights still show results of your previous addition. You must "Write" the results of additions in order to "Read" them.

The CHALLENGE part of this exercise is to build a 1-bit D latch from a single 7400 NAND gate chip and a 2N2222 transistor so that the carry-out line can be remembered also for the CPU to be able to read it with the sum results. After all, the 74373 only provides 8 bits of memory but a summation of two 8-bit numbers may also produce a carry.

And now, for something completely different - a diverse usage of a 74373 – The problem is: There are 8 players in a quiz type game, they might all think they have the answer and are all reaching out to press their buzzer-buttons simultaneously! – Who will be First? - YouTube - Quiz Buzzer Circuit for Eight players using a 74LS373

  (Back to TOC)