Boolean Algebra Applications in Computer Processors

Eginata Kasan 13517030
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517030@std.stei.itb.ac.id

***Abstract*—The study of boolean function manipulation is a branch of discrete mathematics named Boolean Algebra, invented by George Boole. Logic Gates are devices (physical or not) that receive one or more binary inputs and performs logical operations to produce one (or sometimes more) binary output. Logic gates are a physical implementation of boolean logic. A logic gate requires at least one diode or transistor which acts like a switch in order to performits decision making using boolean logic. Logic gates are a necessity for a digital computer where they serve as processors. They can produce either 1 (high) or 0 (low) current depending on the input current given. A modern computer can contain more than 100 million logic gates. The main boolean functions of a logic gates are: AND, OR, NOT, NAND, NOR, XOR, and XNOR. The main logic gates can be combined and combinations of these logic gates can make variations of new logic functions.**

Keywords—boolean algebra, computer architecture, logic gate, transistor.

I. INTRODUCTION

[1] Boolean Algebra is the branch of mathematics that is known as modern algebra or abstract algebra. In Boolean algebra, the value of variables and the results are either true (1) or false (0). It was invented by George Boole in 1854. Boolean Algebra is usually used for analyzing or simplifying circuit that uses boolean logic. In Boolean Algebra there exist laws as a guide to show which manipulations are legit.

Identity $a + 0 = 1$ $a.1 = 1$	Idempotent $a + a = a$ $a . a = a$
Complement $a + a' = 1$ $a . a' = 0$	Annulment $a.0 = 0$ $a + 1 = 1$
Double Negation $(a')' = a$	Absorptive $a + ab = a$ $a(a + b) = a$
Commutative $a + b = b + a$ $ab = ba$	Associative $a + (b + c) = (a + b) + c$ $a(bc) = (ab)c$
de Morgan´s Theorem $(a + b)' = a'b'$ $(ab)' = a' + b'$	Distributive $a + (bc) = (a + b)(a + c)$ $a(b + c) = (ab + ac)$

Table 1 Laws of Boolean Algebra

Our computer's processors are one of the examples of a complicated logic circuit, where these circuits are called the *logic gate*.

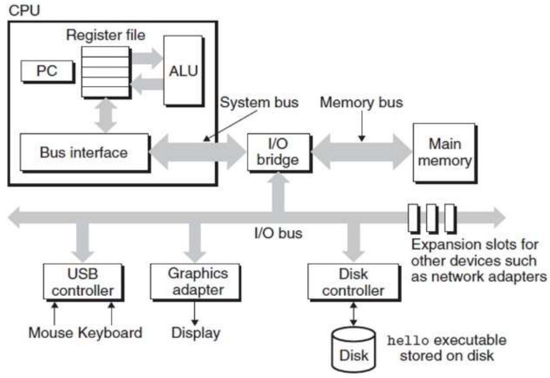

Figure 1 Simplified illustration of computer system

A computer must have at lease these three components in order to work: the power supply, Central Processing Unit (CPU), and memory. The *Central Processing Unit* (CPU), is usually called the brain of the computer, that is located on the motherboard. It is in the CPU that all calculations (arithmetic and logical operations), instructions decoding, and instructions execution.

The Central Processing Unit (CPU) has three main components: (registers and caches), datapaths (ALU), and Control Units. The ALU handles all the arithmetic and logical calculations, whereas the Control Units handle the instructions from memory, and calls the ALU whenever any calculation is needed. Caches and registers are small memory that saves information/instructions as the CPU can access them at a much higher speed rate than to access the hardware. These components of a CPU that were once separated are now constructed as an all-in-one *microprocessor*.

A processor's world is made of bits of 1 and 0, which is machine language instructions, so in order to do calculation, a processor must receive instructions through an electric current/signal, and change it into 0's and 1's using a switch-like component, that is a *transistor*, whereas a high voltage level, for example 2V or 5V (these voltages may vary), is translated into 1's and a voltage near 0 are translated as 0's. Besides storing inputs of 0's and 1's, transistors are also capable of controlling the electric current flow. The key of calculation and decision making of a microprocessor are held by these transistors, implemented in *logic gates*. Therefore, logic gates are basically circuits that manipulates the electric current that flows through it. This is where Boolean Algebra comes in, the study of manipulating various logic gates in order to make some smart computations, for example: addition and subtraction.

II. Logic Gates

[2, pp 3.1] Logic gates are the basic element that makes up digital system. A logic gate is a device that performs logical operations on one or more binary inputs, that is 0's or 1's, and outputs one binary input in exchange (with the exception of some special cases where the output is more than just one). Logic gates are an absolute necessity for computation and decision making, they use only boolean operations to solve problems (for example: addition, subtraction, negation of binary digits).

There are 3 basic boolean functions in logic gates: AND, OR, NOT. There is also XOR which is a very useful boolean function. There are also popular combinations of the basic boolean functions: NAND, NOR, and XNOR.

a. AND gate

The AND gate produces an output of 1 (high) if all of the inputs are 1, otherwise it will output a 0 (low).

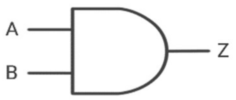

Figure 2.1 AND gate

A	Z
1	1
0	0

A	B	Z
1	1	1
1	0	0
0	1	0
0	0	0

Table 2.1 Truth table of AND gate

b. OR gate

The OR gate produces an output of 0 (low) if any of the inputs (just one or more) are high. It will only output 0 (low) if all the outputs are also 0.

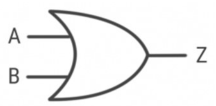

Figure 2.2 OR gate

A	Z
1	1
0	0

A	B	Z
1	1	1
1	0	1
0	1	1
0	0	0

Table 2.2 Truth table of OR gate

c. NOT gate

The NOT gate is an inverter gate, meaning it will output a 1 (high) if the input is 0 (low), and will output 0 (low) if the input is 1 (high). The NOT gate accepts only one input.

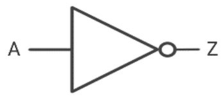

Figure 2.3 NOT gate

A	Z
1	0
0	1

Table 2.3 Truth table of NOT gate

d. NAND gate

The NAND gate is a NOT-AND gate, it yields an output that is the opposite of the AND gate (an inverted output from AND gate). The NAND gate only accepts two or more inputs.

Figure 2.4 NAND gate

A	Z
1	0
0	1

A	B	Z
1	1	0
1	0	1
0	1	1
0	0	1

Table 2.4 Truth table of NAND gate

e. NOR gate

The NOR gate is a NOT-OR gate, it yields an output that is the opposite of the OR output (an inverted output from OR gate).

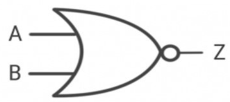

Figure 2.5 NOR gate

A	Z
1	0
0	1

A	B	Z
1	1	0
1	0	0
0	1	0
0	0	1

Table 2.5 Truth table of NOR gate

f. XOR gate

The XOR gate is Exclusive Or gate, in the case of 2 inputs, it produces an output of 1 if one of the inputs are 1. If both of the inputs are 1, it will produce a 0.

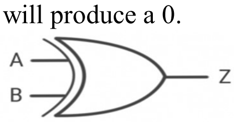

Figure 2.6 XOR gate

A	Z
1	0
0	1

A	B	Z
1	1	0
1	0	1
0	1	1
0	0	0

Table 2.6 Truth table of XOR gate

g. XNOR gate
The XOR gate is a combination of NOT and XOR gate, it inverts the output of the XOR gate. It produces 1 (high) if both of the inputs have the same value, else it produces a 0 (low).

Figure 2.8 XNOR gate

A	Z
1	0
0	1

A	B	Z
1	1	1
1	0	0
0	1	0
0	0	1

Table 2.8 Truth table of XNOR gate

III. APPLICATIONS

There are a huge number of logic gates applications: arithmetic calculator, digits display, automatic shutdown circuit, used for making combinatorial circuits, a three-way light switch, flow directors, or anything that depends on "switches" to make the desired output. There are also more practical applications for logic gates. For example, the one installed in every house: the doorbell. The doorbell circuit needs the logic gate OR in case of multiple doorbells in one house (for example: one doorbell for the front door, one for the back door, etc.) so that when the front door bell and the back doorbell are pressed at the same time or a short time after the other was just pressed, the output stays as 1 (high) and it will ring. This circuit will also make it ring for if only one of the doorbells are pressed due to the logic gate OR.

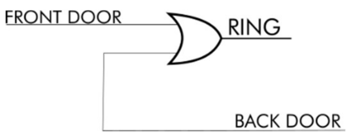

Figure 3.1 Illustration for the use of logic gate in doorbells

Logic gates are crucial in ALU (*Arithmetic Logic Unit*) in the CPU. Examples of the logic gates used in ALU are multiplexors, bitwise AND gate, bitwise OR gate, adders, subtractors, overflow output, negative output, and zero output. The multiplexors are for choosing inputs based on the control line. The bitwise gates have many useful applications, for example the AND is to calculate an IP network's identity. The adders and subtractors, as the names suggest, is used to do calculations of addition and subtraction of binary digits.

The discussion and the details of ALU logic gates in this paper will be limited to only adders and subtractors and showing the boolean algebra applications in it. The half-adder is one of the simple yet most important part of arithmetic computation. The half-adder can do simple addition of two single-digit binaries.

Here are some examples of single binary digit addition:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

The 1+1 yielded a two-digit binary output (10), so a circuit that outputs two digits is needed in this case. In other words, addition of two bits will be done when the instruction says addition of two numbers whereas both numbers only consist of one digit. The half-adder consists only of a XOR and an AND gate, where the output is 1 bit for each. [2, pp. 4.3 – 4.4] The AND gate will output CARRY that will hold the higher significant byte, whereas the XOR gate will output SUM, that is the least significant byte.

Below is an illustration of how the logic gates are used in the half-adder.

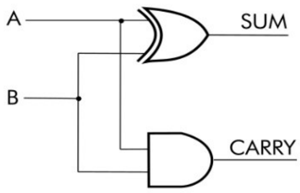

Figure 3.2 Logic diagram of half-adder

The half-adder receives 2 inputs (A and B), that represents the digits that needs to be added and outputs two values, sum and carry value. The XOR and the AND gate are connected to both A and B. Using the XOR output as the SUM, and the AND gates as the CARRY, we get:

A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 3.1 Truth Table of half-adder

The CARRY acts as the most significant bit of the two-digit output in half-adders, so the table shows that

0 + 0 = 00
0 + 1 = 01
1 + 0 = 01
1 + 1 = 10

for the SUM, the boolean expression is:

$$S = A \oplus B$$

while for the CARRY, it is:

$$C = A . B$$

However, there is a limitation of using half-adders that is the half-adder only works for addition of single bits (it cannot do additions like 11+11). This is because the half-adder has room of input for only 2 bits, whereas to do addition of, for example, 4-digit inputs, 3 bits of inputs are needed: two for the binary digits, and one for the carry-out from the digit before them. Here are some examples of addition of 4 bits:

0000 + 0001 = 0001
1100 + 1000 = 10100
1111 + 1111 = 11110

In the case of half-adder, there is no consideration for carry-outs, therefore, a more complex circuit is needed. The full-adder was made exactly for this purpose. [4] A full-adder is a combinatorial circuit that forms the arithmetic sum of three input bits. The full-adder is more or less the combination of two half-adders, and an OR gate. It can do addition for 1 or more number of bits. Below is the illustration of the full-adder circuit.

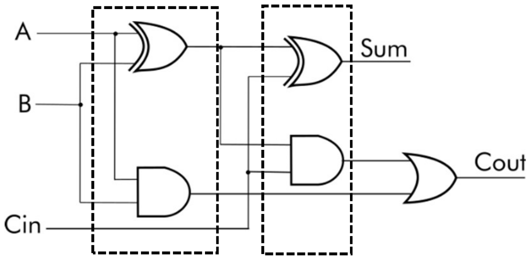

Figure 3.3 Logic diagram of half-adder

Understanding the full-adder may not be as simple as understanding the half-adder. There is an OR gate that receives input from two AND gates, it is the one that will give the final COUT. The full-adder treats input A and B the same way as the half-adder does, but there is an input for carry-outs in full-adders (CIN), which will affect the SUM and also COUT. The third input CIN, represents the carry from the previous lower significant position. The truth table for the full-adders is shown as below:

A	B	CIN	COUT	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 3.2 Truth Table of full-adder

For the SUM in full-adder, it is

$$S = (A \oplus B) \oplus Cin$$

and for the COUT:

$$Cout = \left((A \oplus B).Cin\right) + (A.B)$$

But unlike the half-adder where the COUT is used directly as the most significant bit, the COUT of the full-adder will be used as CIN for the addition of the next digit until the last digit, where the resulting value size will be plus one bit more than the size of the inputs, like the half-adder (1+1 actually means addition of two bits).

Take 11+10 for an example of addition using full-adder. The rightmost bit of 11 and 10 will be the first A and B accordingly, with the CIN being 0.

Input:

A	1
B	0
CIN	0

Output:

COUT	0
SUM	1

Next, we take the second bit from the left out of the two inputs. We then take the COUT from the step before, which is 0, and save it as CIN for this next step.

Input:

A	1
B	1
CIN	0

Output:

COUT	1
SUM	0

Now the two binary numbers have been calculated, but 11+10 is 101, so far, the results yielded a 01. Though there is COUT with the value of 1 in the step before. Thus, that is why the true output digit count has to be 1 bit more than the inputs, so that there are no COUTs left uncalculated. For example, if the input is a 2 bit, then the output must be a 3 bit. For the final step, the next A and B will be 0s, because the numbers have been calculated. Using the last COUT as CIN, we then continue.

Input:

A	0
B	0
CIN	1

Output:

COUT	0
SUM	1

The final answer of 11+10 is now 101.

The subtractor in processors, instead of producing SUM and CARRY, it instead produces DIFFERENCE and BORROW. Like in adders, there are Half-Subtractors and Full-Subtractors. [3, pp. 240-242] A Half-Subtractor is a combinatorial circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. DIFFERENCE has the value of 1 if the A and the B has a difference of 1, whereas BORROW indicates whether the minuend needs borrowing because it is smaller than the subtrahend.

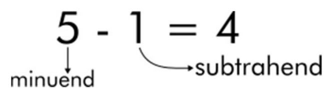

Figure 3.4 minuend and subtrahend

Here are some examples of very simple single binary digit subtraction:

0 – 0 = 0
1 – 1 = 0
1 – 0 = 1

The subtraction above are cases of simple subtractions with no borrowing, so they are rather simple to do. A case of subtraction with borrows are when the subtrahend is smaller than the minuend, like: 0 – 1 (0 is the minuend, 1 is the subtrahend). Like how we do 31 – 8 = 23, the 1 in "31" borrows a 10 from 3 in "31" in order to be subtracted to 8, and the 3 becomes a 2, thus the answer is 28. This method is the same with binary numbers, but instead of a 10, a binary number borrows a 1 (meaning 2 in decimals).

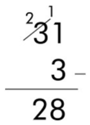

Figure 3.5 Illustration of subtraction using borrows

The logic diagram of the half-subtractor and the truth table is shown below:

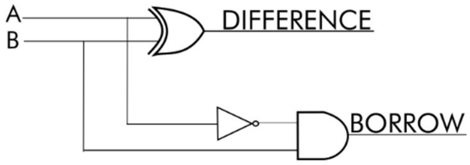

Figure 3.6 Logic diagram of half-subtractor

A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Table 3.3 Truth table of half-subtractor

The boolean expression for the DIFFERENCE is:

$$D = A \oplus B$$

whereas for the BORROW, it is:

$$B = \bar{A} . B$$

The half-subtractor only works for subtractions of one digit since it cannot receive inputs of borrow-ins from the previous digit. [3, pp. 242] The full-subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into considering whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not. The full-subtractor considers the borrows from the digit before and uses it as input "Borrow In". Like how the full-adders are made by two half-adders, the full-subtractor is also made of two half-subtractors.

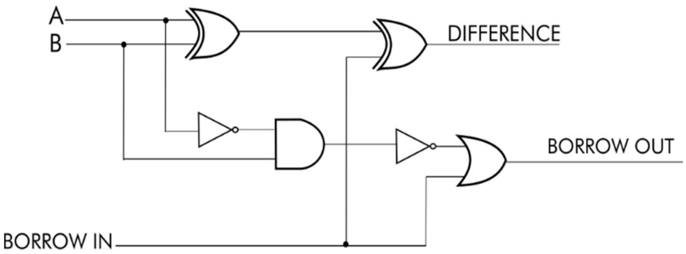

Figure 3.7 logic diagram of full-subtractor

A	B	B_{in}	B_{out}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 3.4 Truth Table of full-subtractor

The B_{in} is used to store if there are any borrows from the digit before, the value depends on the B_{out} on the previous subtraction of the same number. Take figure 3.3 for example, the 3 in "31" has the B_{in} of 1 when it was subtracted by 0 in "08".

In Full-Subtractors, the boolean expression for Difference (D) is

$$D = (A \oplus B) \oplus B_{in}$$

Whereas for the B_{out} the boolean expression is

$$B_{out} = (\overline{\bar{A} . B}) + B_{in}$$

In order to show how the full-subtractor works and prove it, take 101-10 as a case for this example (translated to decimal, 5-2 = 3).

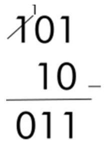

Figure 3.7 Illustration of binary subtraction

First, the A and B should be the least significant bit of the binary numbers, that is 1 and 0 respectively. The B_{in} will be a 0 since there are no B_{out} saved yet

input

A	1
B	0
B_{in}	0

output

B_{out}	0
D	1

Next the A and B take a step towards the next binary digit. The Bout is used as the Bin for this next step

input

A	0
B	1
B_{in}	0

output

B_{out}	1
D	1

Because 0 – 1 is subtraction where the 0 needs borrowing, the Bout is now a 1. Then we take another step with the Bout used as the next Bin.

input

A	1
B	0
B_{in}	1

output

B_{out}	0
D	0

Thus, the final answer of the subtraction of 101-10 is now 011.

IV. ACKNOWLEDGMENT

Author would like to express her deepest appreciation to all those who provided me the possibility to complete this report. Thank God, for His blessings, for it is His grace that made this paper can be finished in time. A thanks to my friends who gave great advices for this paper. A thanks to Dr. Judhi Santoso, M.Sc. and to Dra. Harlili M.Sc, Author's lecturers, who taught classes about discrete mathematics, for their knowledge are passed well to the students and thanks to Dr. Ir. Rinaldi Munir, MT. for his loving support of all the students, his informative website, and for his ever glowing spirit he shows everyone in his works.

REFERENCES

[1] J. Eldon Whitesitt , "Boolean Algebra and Its Applications" , New York: Dover Publications, pp. 1.

[2] A.P.Godse, "Digital Logic Circuits" , D.A.Godse, India:Technical Publications Pune, chapter 3 pp.1-2

[3] Anil K. Maini, "Digital Electronics: Principles, Devices and Applications", India: John Wiley & Sons.

[4] John, *Half Adder and Full Adder* on July 31, 2018, retrieved December 7, 2017 from http://www.circuitstoday.com/half-adder-and-full-adder

[5] Administrator, *Implementation of Boolean Functions using Logic Gates* on August 7, 2015 retrieved December 7, 2018 from https://www.electronicshub.org/implementation-of-boolean-functions-using-logic-gates/

[6] *IEEE Referencing: Getting started with IEEE referencing* retrieved December 9, 2018, from http://libraryguides.vu.edu.au/ieeereferencing/gettingstarted#s-lg-box-wrapper-9929081

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Desember 2017

13517030
Eginata Kasan