LOGBOOK`

Computer Architecture Logbook

Name: Kaloyan Lalov

Student ID: 21368509

Computer Games Technology

University of West London

LOGBOOK`

Table of Contents

LOGBOOK`

Introduction

Welcome to my logbook! This Logbook is written to have a better understanding of the Computer Architecture module, which is a part of the Computer Games Technology course. In this module we will learn more about binary coding and digital circuits. For humans, information can be pictures, symbols, words, sounds, movements, and more. A typical computer has a keyboard and mouse so that words and movements can be sent to the processor as information. The information must be converted into electrical off-on (``0 and 1"). A general-purpose computer has these identical parts:

– Processor, the “brain” of the computer. that does arithmetic, responds to incoming information, and generates outgoing information

– Primary storage (memory or RAM) is used to remember the information given by the processor.

– System – the system connects the processor to the primary storage and input/output devices.

LOGBOOK`

Week 1

1.1 Explain the idea that "all computers can do exactly the same things".

A computer can be designed to do anything with given information. The way it functions is that it follows a code and it gives specific instructions what to do and how to do it. The computer requires RAM to be able to complete the tasks it's given in a short amount of time. It should also be stated that software like the operating system have influence on the computer's performance. If a computer doesn't have enough RAM it will still do the same tasks as a better computer, although you will need additional time for the tasks you are commanding the computer to do.

1.2 Name one of the characteristics of natural languages that prevents them from being used as programming languages.

The difference between natural language (ex. - English language) and programming language(ex. - Java) is that whilst the natural language can be ambiguous (can mean more than one thing), in the programming language, one code means only one thing.

1.3 Write a statement in a natural language and offer two different interpretations of that statement.

Example: Look at the man with the camera.
The example can mean both him holding a camera or looking at a man through a camera.

1.4 Two computers, A and B, are identical except for the fact that A has a subtract instruction and B does not. Both have add instructions.

LOGBOOK`

Both have instructions that can take a value and produce the negative of that value. Which computer is able to solve more problems, A or B? Prove your results.

Both computers can solve the same amount of problems. B needs to use negation of a positive number in order for it to subtract. A doesn't require to use the same method as it has subtraction built-in, although it could do both operations.

1.5 Identify one advantage of programming in a higher-level language compared to a lower-level language. Identify one disadvantage.

Advantage:

A high-level language is more similar to a natural language (The english language), thus it is easier to learn and understand.

Disadvantage:
A high-level language is slightly slower compared to a low-level language when it is being compiled.

1.6 List the level of transformation and name an example for each level.

Similarly to any real-life problem, in order to invent something that will solve it, you need to come up with a plan. First of all, however, you need to acknowledge the problem. That is the first level of transformation known as "Problem Statement". It can be expressed using a natural language, thus it can be ambiguous.

Algorithm is the next level of transformation. The algorithm should be finite, effective and briefly explain the actions of the program you're trying to create as well as the outcome. It can be pseudocode, which is preferred; a way of writing in a natural language that resembles a programming language. That way it will be easier to perform the next level of transformation.

LOGBOOK`

Next up, we convert the pseudocode to programming language code. This is the part where a high/low-level language is picked. Luckily, there are specific keywords that are used in the pseudocode that are similar to the syntax of a programming language. Because of that, this conversion is much easier. This process is known as creating a Program.

Now comes the part where we have to make it computer-readable. The programming language is converted to machine language through ISA **(**Instruction Set Architecture**)**. We go through levels such as Microarchitecture where we can choose from different implementations of ISA, Logic Circuits; where you combine basic operations to implement a single function. For instance, addition for subtraction.

Finally, the lowest level is Process Engineering & Fabrication which is interconnected with the development and manufacture of lowest-level components.

Week 2

2.1 Given n bits, how many distinct combinations of the n bits exist?

Two to the power of n.

2.2 There are 26 characters in the alphabet we use for writing English. What is the least number of bits needed to give each character a unique bit pattern? How many bits would we need to distinguish between upper and lowercase of all 26 characters?

Only for lowercase letters, we would need 5 bits , which is equal to 32 possible outcomes, which would fill the 26 characters in the alphabet.
In the situation of both uppercase and lowercase letters, we would need 6 bits, which is equal to 64 possible outcomes, which would fill the 52 characters in the alphabet.

2.3 a. Assume that there are about 400 students in your class. If every student is to be assigned a unique bit pattern, what is the minimum number of bits required to do this?

If every student is to be assigned a unique bit pattern, we would need 9 bits which calculates to 512 possible outcomes of 400 students in total.

LOGBOOK`

2.3 b. How many more students can be admitted to the class without requiring additional bits for each student's unique bit pattern?

112 additional bits.

2.4 Given n bits, how many unsigned integers can be represented with the n bits? What is the range of these integers?

Since we start with 0 and go up, we have to negate that 0 by using "-1". Therefore, 2^n-1 can be unassigned integers. The range of those integers would be from 0 to 2^n-1

2.5 Using 5 bits to represent each number, write the representations of 7 to -7 in 1's complement, signed magnitude, and 2's complement integers.

Java Binary	Signed magnitude	1's Complement	2's Complement
7	00111	00111	00111
6	00110	00110	00110
5	00101	00101	00101
4	00100	00100	00100
3	00011	00011	00011
2	00010	00010	00010
1	00001	00001	00001
0	00000	00000	00000
-0	10000	(-7)10111	(-8)11000

LOGBOOK`

-1	10001	(-6)10110	(-7)10111
-2	10010	(-5)10101	(-6)10110
-3	10011	(-4)10100	(-5)10101
-4	10100	(-3)10011	(-4)10100
-5	10101	(-2)10010	(-3)10011
-6	10110	(-1)10001	(-2)10010
-7	10111	(-0)10000	(-1)10001

2.6 Write the 6-bit 2's complement representation of -32.

100000 => $2^6 = 32$

2.7 Create a table showing the decimal values of all 4-bit 2's complement numbers.

Decimal	1's complement	2's complement
7	7	0111
6	6	0110
5	5	0101
4	4	0100
3	3	0011
2	2	0010
1	1	0001
0	0	0000
-8	-1	1111
-7	-2	1110
-6	-3	1101
-5	-4	1100

LOGBOOK`

-4	-5	1011
-3	-6	1010
-2	-7	1001
-1	-8	1000

2.8 a. What is the largest positive number one can represent in an 8-bit 2's complement code. Write your result in binary and decimal.

$2^{0+}2^1+2^2+2^3+2^4+2^5+2^6+2^7$ = 1 + 2 + 4 + 8 + 16 + 32 + 64= **127** in decimal, **0111 1111** in binary.

2.8 b. What is the greatest magnitude negative number one can represent in an 8-bit 2's complement code. Write your result in binary and decimal.

$(1000\ 000)_2$ and -128, would be the greatest magnitude negative number.

2.9 Convert the following 2's complement binary numbers to decimal:

a. 1010
b. 01011010
c. 11111110
d. 0011100111010011

a. 1010 = $2^1 + (-2^3)$= 2 + (-8) = -6
b. 01011010 = $2^1 + 2^3 + 2^4 + 2^6$ = 2+8+16+64 = 90
c. 11111110 = -2^1 = -2
d. 0011100111010011 = $2^0 + 2^1 + 2^4 + 2^6 + 2^7 + 2^8 + 2^{11} + 2^{12} + 2^{13}$=
1+2+16+64+128+256+2048+4096+8192= 14803

2.10 Convert these decimal numbers to 8-bit 2's complement binary numbers:

LOGBOOK`

a. 102
b. 64
c. 33
b. -128
e. 127

a. 102 = 0110 0110 => $2^1 + 2^2 + 2^5 + 2^6 = 102$

b. 64 = 0100 0000 => $2^6 = 64$

c. 33 = 0010 0001 => $2^0 + 2^5 = 33$

d. -128 = 1000 0000 => $-2^7 = -128$

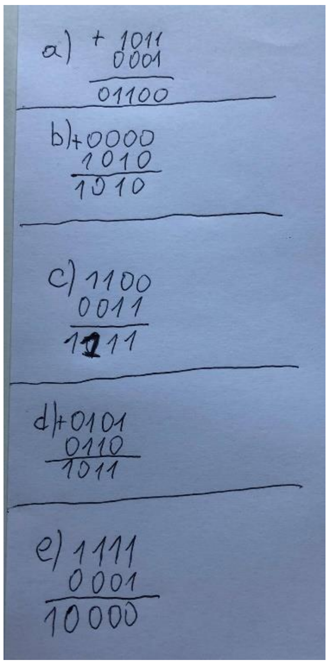
e. 127 = 0111 1111 => $2^0 + 2^1+2^2+2^3\ 2^4+2^5\ 2^6 = 127$

LOGBOOK`

Week 3

3.1 Add the following bit patterns. Leave your results in binary form.

a. 1011 + 0001
b. 0000 + 1010
c. 1100 + 0011
d. 0101 + 0110
e. 1111 + 0001



```
a) + 1011
     0001
    -----
    01100

b) + 0000
     1010
    -----
     1010

c)   1100
     0011
    -----
     1111

d) + 0101
     0110
    -----
     1011

e)   1111
     0001
    -----
    10000
```

3.2 When is the output of an AND operation equal to 1

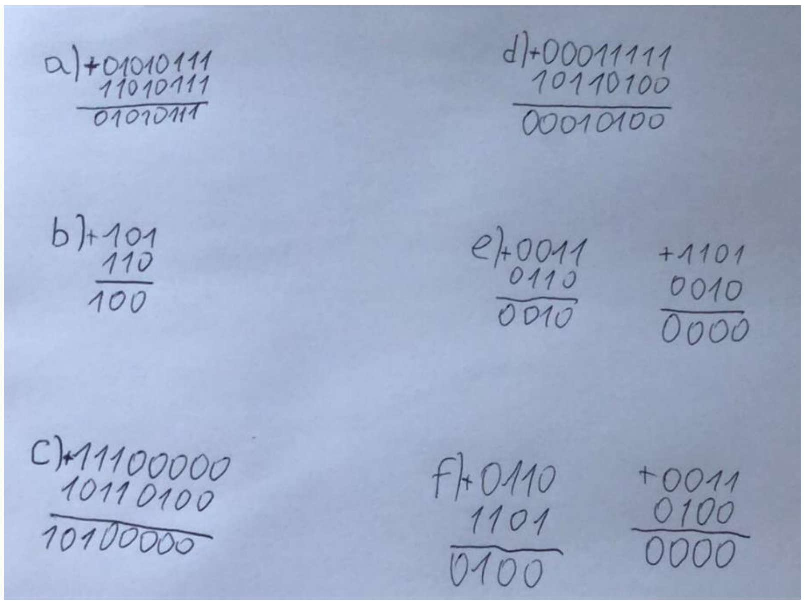
When both A and B are true, then it’s true, false otherwise

LOGBOOK`

3.3 Fill in the following truth table for a one-bit AND operation.

X	Y	X and Y
0	0	0
0	1	0
1	0	0
1	1	1

3.4 Compute the following. Write your results in binary.
a. 01010111 AND 11010111
b. 101 AND 110
c. 11100000 AND 10110100
d. 00011111 AND 10110100
e. (0011 AND 0110) AND 1101
f. 0011 AND (0110 AND 1101)



```
a)+01010111                    d)+00011111
   11010111                       10110100
  ----------                     ----------
   01010111                       00010100

b)+101                         e)+0011        +1101
   110                            0110         0010
  ----                           -----        -----
   100                            0010         0000

c)+11100000                    f)+0110        +0011
   10110100                       1101         0100
  ----------                     -----        -----
   10100000                       0100         0000
```

3.5 When is the output of an OR operation equal to 1

When both A and B are false, then it is false, true otherwise

LOGBOOK`

3.6 Fill in the following truth table for a one-bit OR operation.

X	Y	X or Y
0	0	0
0	1	1
1	0	1
1	1	1

3.7 Compute the following:

a. 01010111 OR 11010111

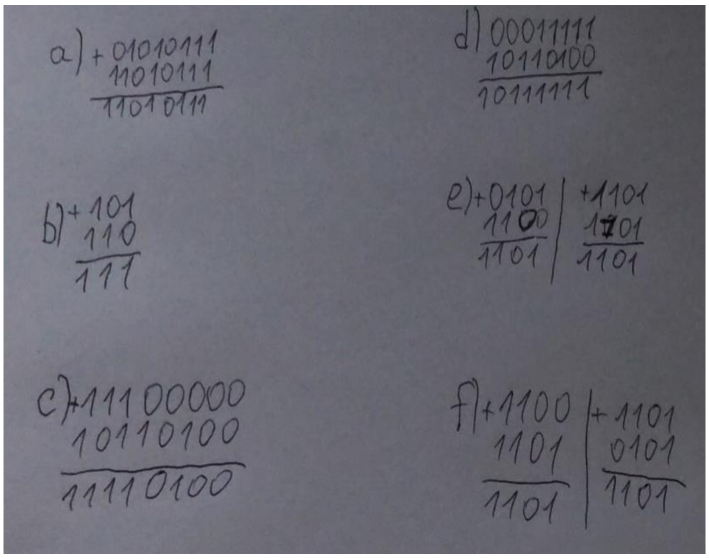
b. 101 OR 110

c. 11100000 OR 10110100

d. 00011111 OR 10110100

e. (0101 OR 1100) OR 1101

f. 0101 OR (1100 OR 1101)



```
a) + 01010111          d) 00011111
     11010111             10110100
     --------             --------
     11010111             10111111

b) + 101               e) + 0101 | + 1101
     110                    1100 |   1101
     ---                    ---- |   ----
     111                    1101 |   1101

c) + 11100000          f) + 1100 | + 1101
     10110100               1101 |   0101
     --------               ---- |   ----
     11110100               1101 |   1101
```

LOGBOOK`

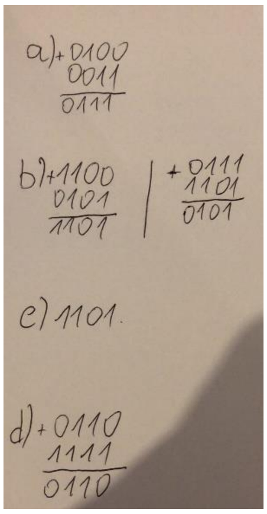
3.8 Compute the following:

a. NOT (1011) OR NOT(1100)

b. NOT (1000) AND (1100 OR 0101))

c. NOT (NOT (1101))

d. (0110 OR 0000) AND 1111



```
a)+0100
   0011
   ----
   0111

b)+1100   |  +0111
   0101   |   1101
   ----   |   ----
   1101   |   0101

e) 1101.

d)+0110
   1111
   ----
   0110
```

3.9 Convert the following unsigned binary numbers to hexadecimal.

a. 1101 0001 1010 1111

b. 001 1111

c. 1

d. 1110 1101 1011 0010

a) – xD1AF

b) – x1F

c) – x1

d) – xEDB2

LOGBOOK`

3.10 Convert the following hexadecimal numbers to binary.
Note: The way this exercise works is that you turn the numbers into decimals first and then you turn them into hexadecimal.

a. x10
b. x801
c. xF731
d. x0F1E2D
e. xBCAD

a) – 1010
b) – 1000 0000 0001
c) – 1111 0111 0011 0001
d) – 0000 1111 0111 0011 0001
e) – 1011 1100 1011 1101

LOGBOOK`

Week 4

4.1 Using the information provided in the slides, explain how the NOT, OR, NOR, AND, NAND gates work.

NOT GATE

When the input voltage has a truth value of 0 the output of the gate will be 1, and vice versa. This method is known as a Inverter.

OR GATE

We use an OR gate when we want the output truth to be 0, only when all input values are 0, otherwise for all input values of 1, the truth value will also be 1.

NOR GATE

We use a NOR gate when usually all input values are 0 so the output value is 1, however, when one input truth value changes to 1 or all input values are 1, the output value will become 0.

AND GATE

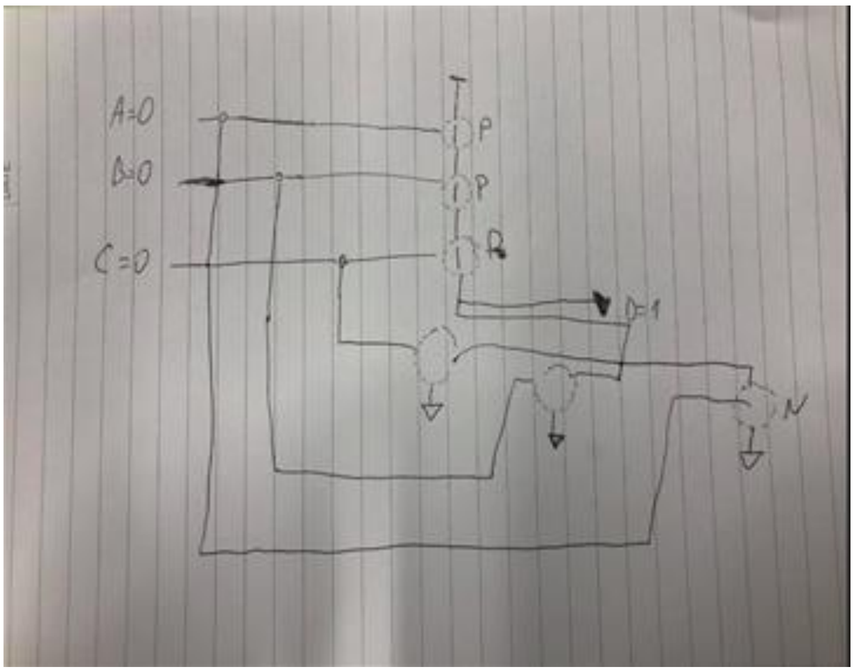
We use an AND gate when we want to output of the gate to be 1 only when all inputs have a truth value of 1.

NAND GATE

We use a NAND gate when we want the output of the gate to be 1 only when all inputs have a truth value of 0. For example, in a 2-input system when both or either values are 0 and not 1.

LOGBOOK`

4.2 Implement a 3-input NOR gate with MOS transistors.

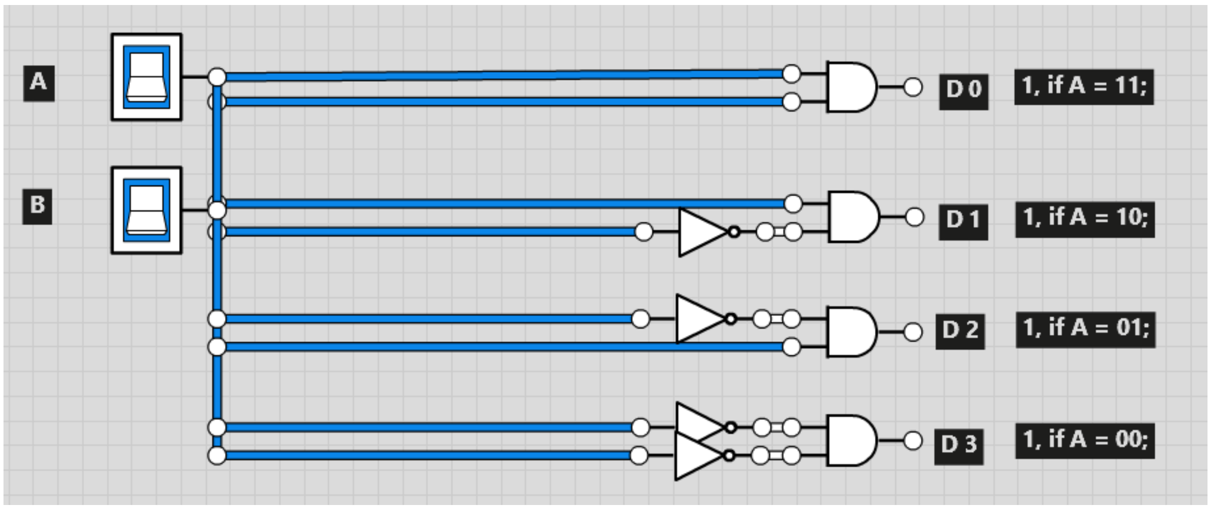


LOGBOOK`

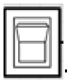
Week 5

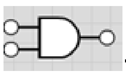
5.1 Create and document the following series of combinational logic circuits. You can test your design against the truth table using logic.ly.x`

a. A two-input decoder.

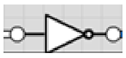


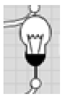
This is the two-input decoder. This picture has been made with the program Logicly. A binary decoder is a multi-input, multi-output combinational circuit that converts a binary code of n input lines into a one out of 2n output code. These are used when there is need to activate exactly one of 2n output based on an n-bit input value.

- This is a switch that can either be turned on and have positive voltage, or be turned off and have negative voltage.

- This is a AND gate. The only possible way for the outcome to be 1, is if there is positive voltage from both sides connected to the AND gate

LOGBOOK`

 - This is a NOT gate. If there is negative voltage on the left side and it goes through a NOT gate, the outcome will be positive.

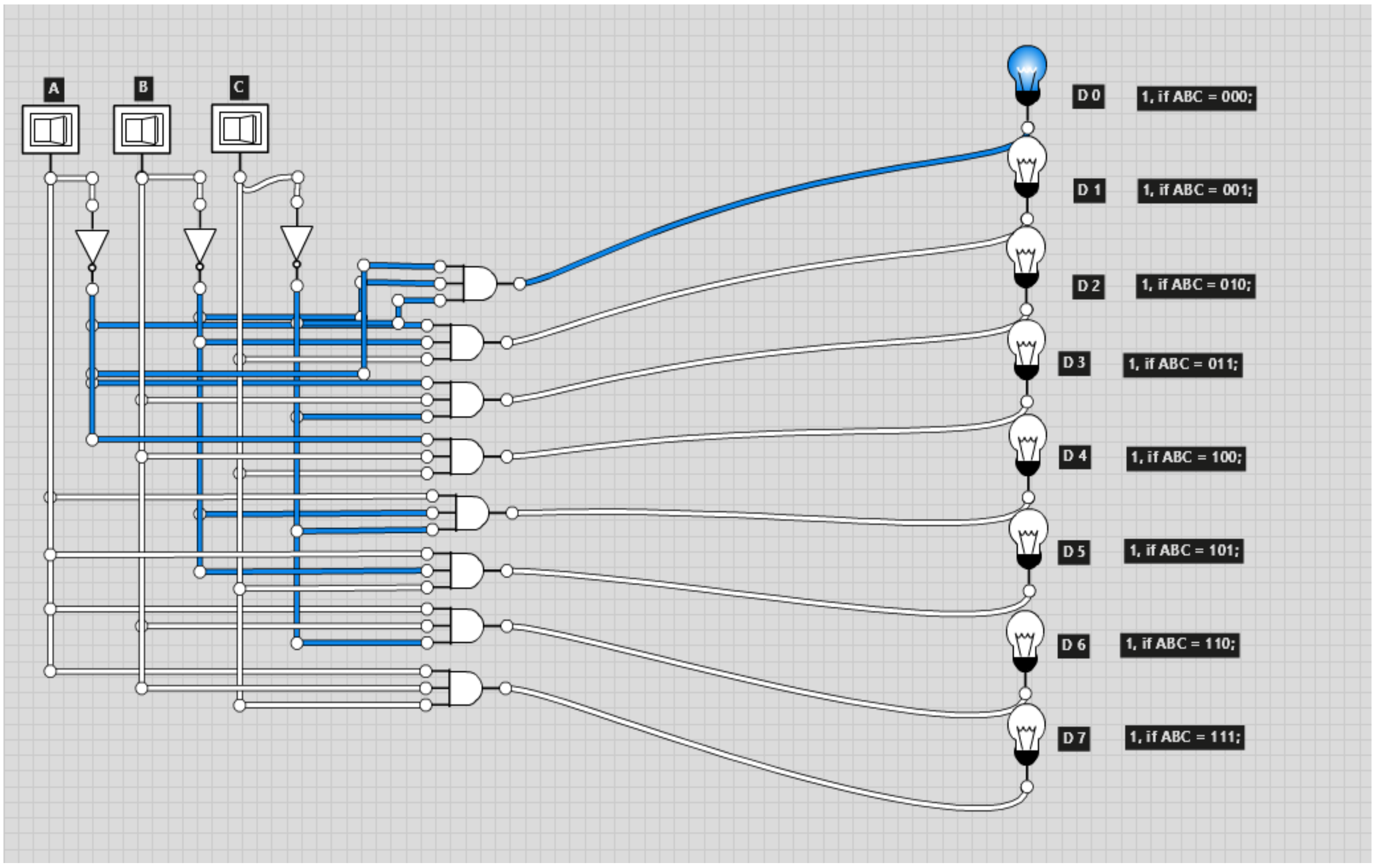
 - This is a lightbulb. We use it to prove whether the voltage is positive or negative. This lightbulb is with negative voltage.

- This is a lightbulb with positive voltage.

A	B	D 0	D 1	D 2	D 3
1	1	1	0	0	0
1	0	0	1	0	0
0	1	0	0	1	0
0	0	0	0	0	1

LOGBOOK`

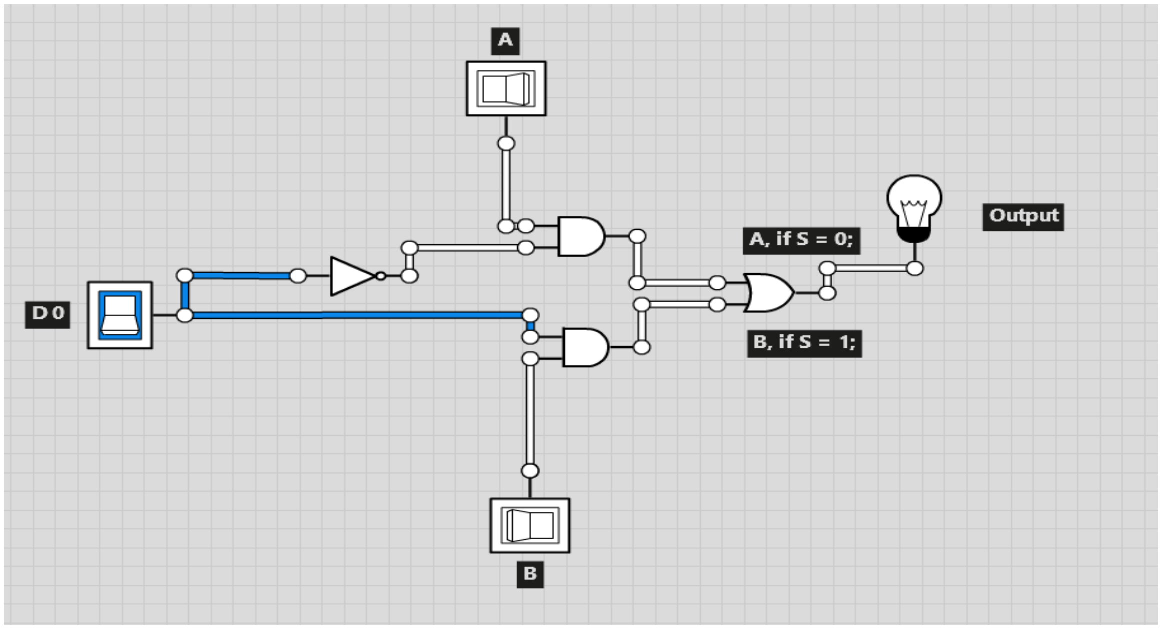
b. A three-input decoder.



A	B	C	D 0	D 1	D 2	D 3	D 4	D 5	D 6	D 7
1	1	1	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1	0	0	0
0	1	1	0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0

LOGBOOK`

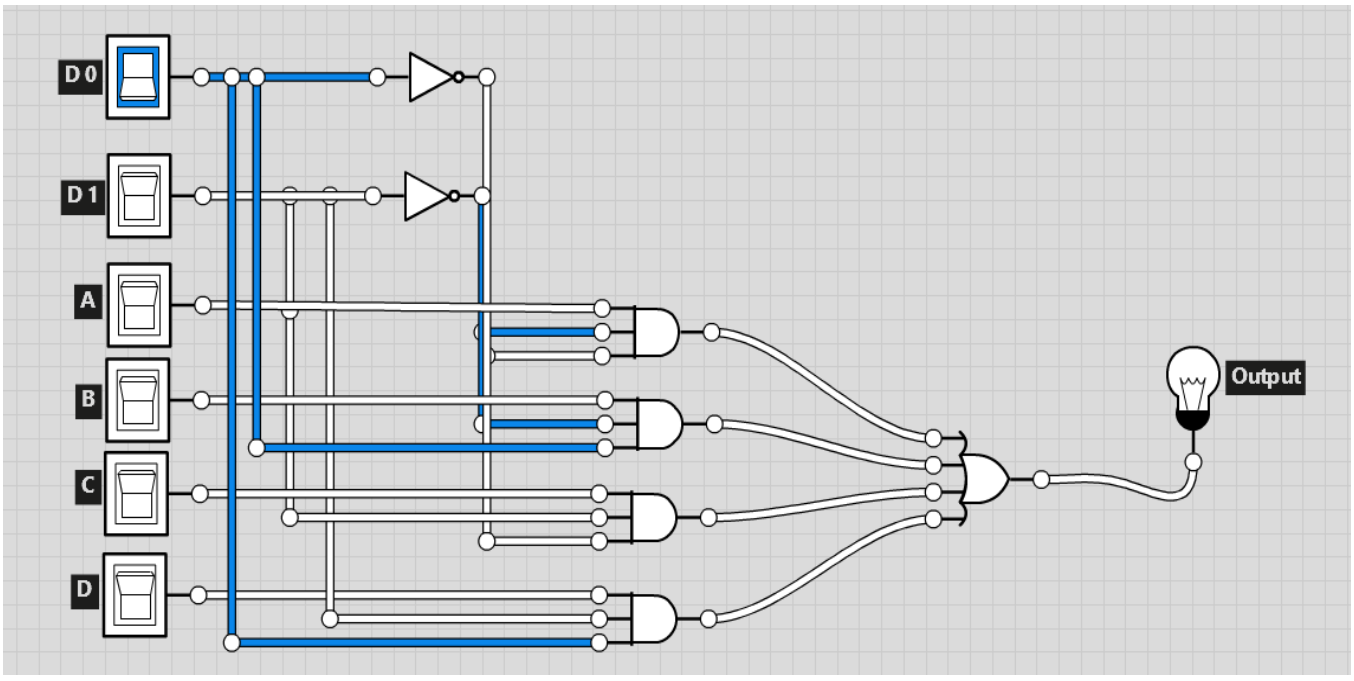
c. A 2-to-1 Mux.



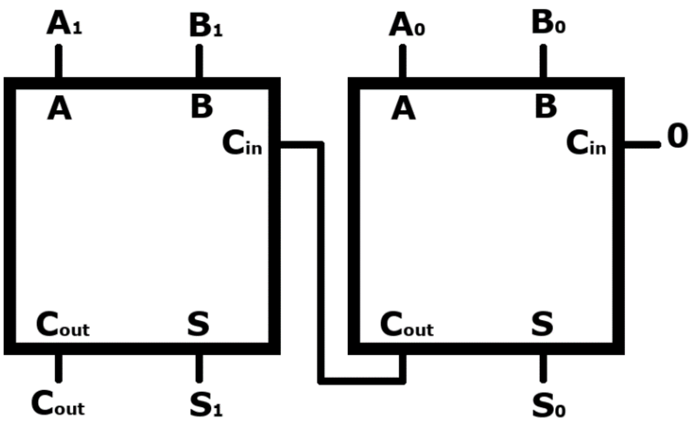
A	B	D 0	Output
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

LOGBOOK`

d. A 4-to-1 Mux.

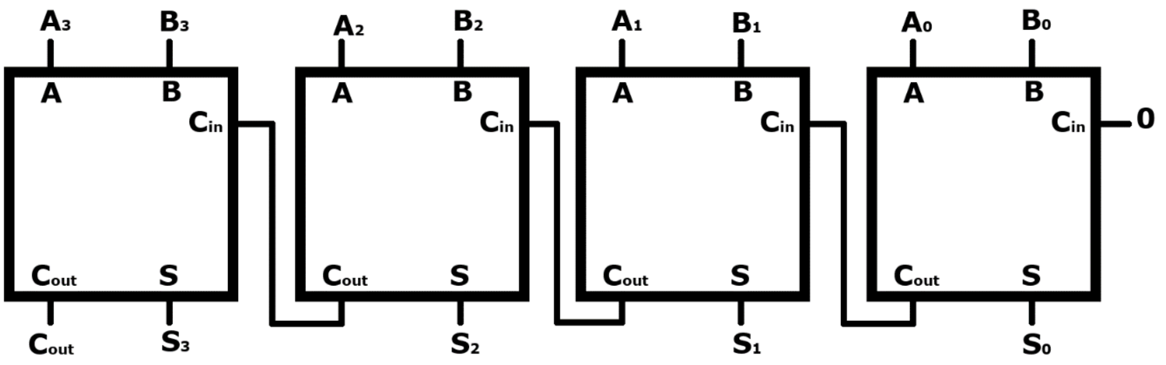


e. A 2-bit Adder.

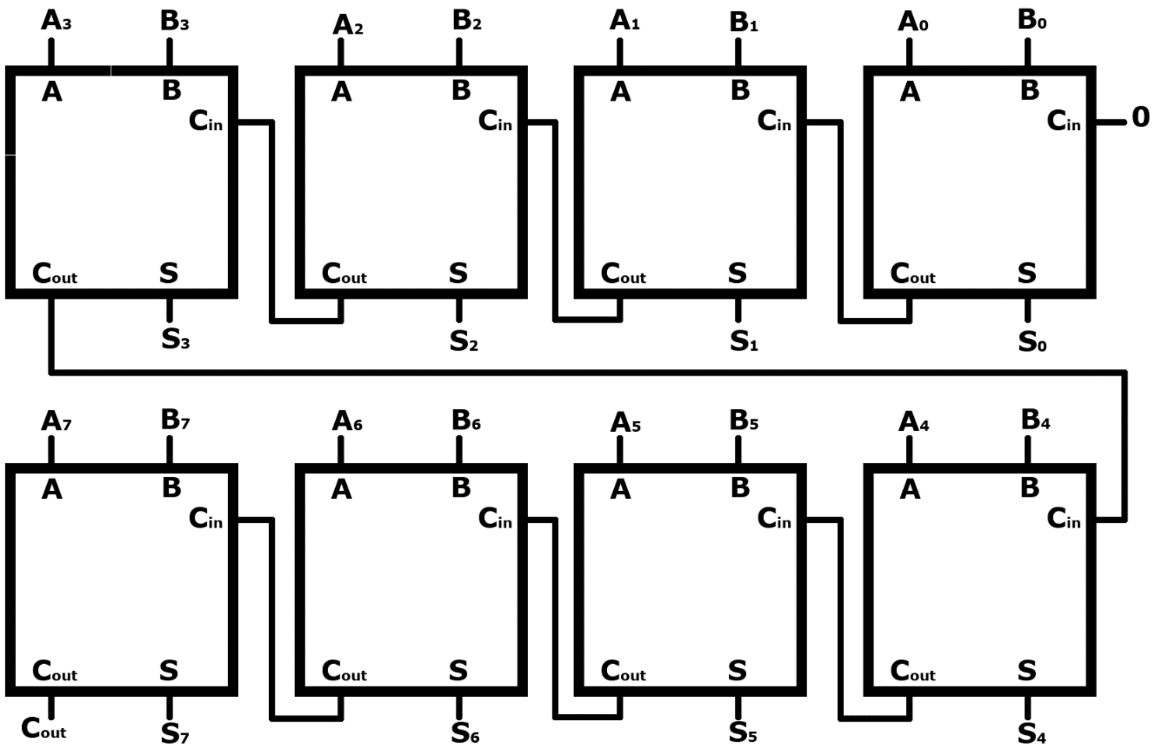


LOGBOOK`

f. A 4-bit Adder.



g. A 8-bit Adder.



LOGBOOK`

Week 6

6.1 Explain the difference between logic structures that include the storage of information and those that do not.

Combinational circuits are defined as the time independent circuits which do not depends on previous inputs to generate any output. **Sequential circuits** are those which are dependent on clock cycles and depends on present and past inputs to generate any output.

- Combinational circuits depend only upon present input whereas the Sequential circuits depend upon present as well as past inputs.
- The combinational circuit are quite faster than the Sequential circuits, and also they're designed way easier.
- The combinational circuit are time independent while the Sequential circuits are time dependant.
- The combinational circuits are used for arithmetic as well as boolean operations, whereas the sequential circuits are used for storing useful data.
- The combinational circuits don't have a memory element, which the sequential circuits have.

6.2. Using the information provided in the slides, explain how the R-S Latch, Gated D Latch, and Register work.

R-S Latch:

An R-S Latch is a simple storage device that saves information by using inputs such as "S" that sets the element to 1 and "R" that resets it to 0. If both of R and S is 1, the output can be either 1 or 0 no matter whether A = 1 and B = 0 or vice-versa. If both R, S are equal to 0, the output is always equal to 1 and the final state is determined by the electrical properties of the gates and that isn't good.

Gated D-Latch:
The Gated D-latch is a sequential circuit. It has two inputs, which are D (data) and WE(write enable). Similarly to the R-S Latch, it can hold values. However, thanks to WE it also has the potential of holding past values, which means it stores information. While WE is 1, the value of D is passed on to the input of an R-S Latch, thus storing it. Then, if WE is 0, the latch now holds the previous value.

Register:
The Register is a collection of D-latches controlled by a common WE input. When WE is 1, the Gated D-latch's D values are stored, but in this case, they are written to register instead. This gives the ability to hold multiple bit values.

6.3 Explain the concept of Memory, Address space and Addressability using current technologies and architectures as examples.

Memory resembles a logical array of k × m stored bits where:

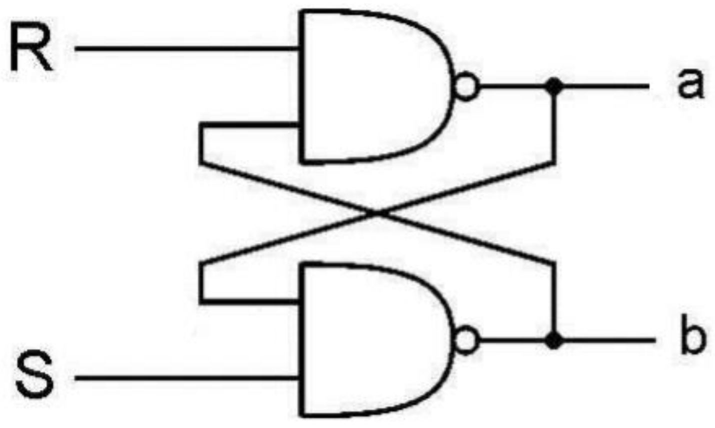
k represents the number of locations (Address space), and **m** represents the number of bits per location(Addressability)

Extra information:
An **address space** is a range of valid addresses in memory that are available for a program or process and the **Addressability** is the way how a computer identifies memory locations.

6.4 Create and document the following logic structures. You can test your design against the truth table using logic.ly.

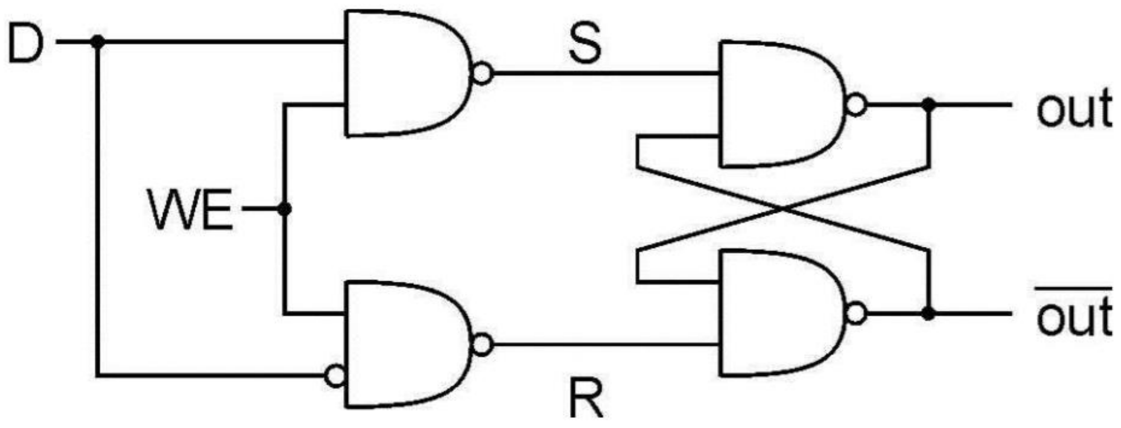
LOGBOOK`

a. S-R Latch



S	R	a	b
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

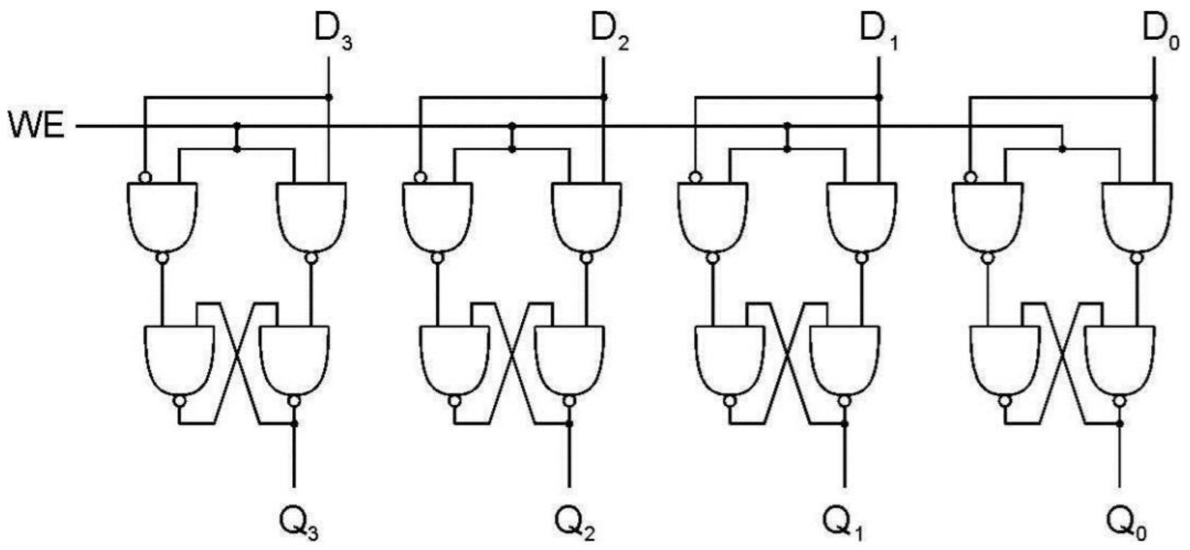
b. Gated D-Latch



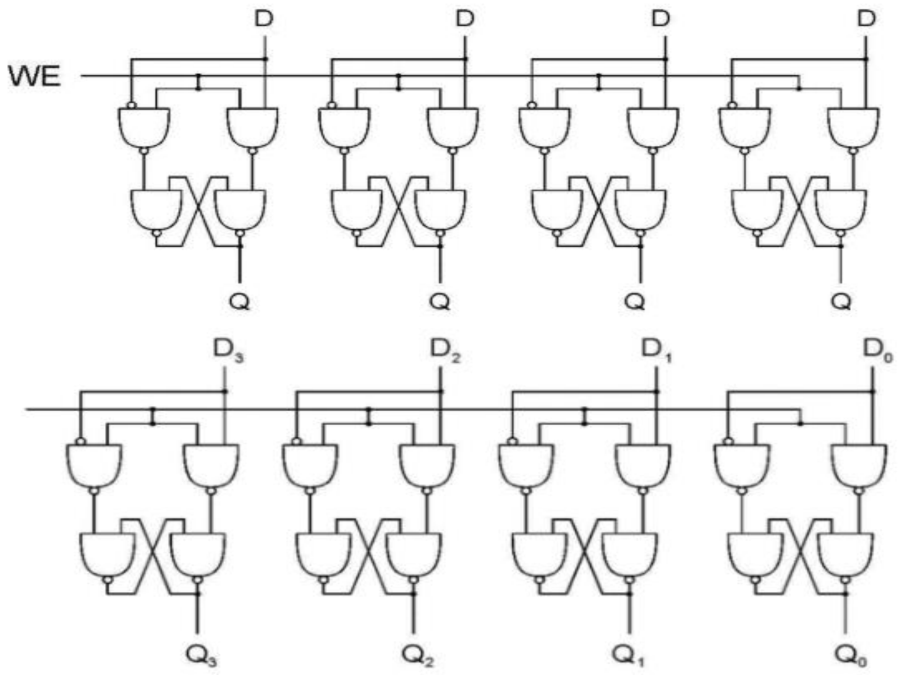
LOGBOOK`

WE	D	out	$\overline{out}$
0	0	latch	Latch
0	1	latch	latch
1	0	0	1
1	1	1	0

c. A 4-bit Register



d. An 8-bit Register



LOGBOOK`

References

The information for this Logbook has been taken from my notes in the Computer Architecture lectures and from the slides on our Blackboard.