University Of Portsmouth
BSc (Hons) Computer Science
First Year

Architecture and Operating Systems - Computer

M30943
September 2022 - May 2023
20 Credits

Thomas Boxall
up2108121@myport.ac.uk

Contents

S.1. Introduction to Module

26-09-22 16:00 Farzad RB LT1

Division of the Module

This module is split into two parts: computer (this part) which is worth 70% and maths (the other part) which is worth 30%. The two parts are run completely independently of each other. The only time they come together is when the final overall score is calculated.
There are two separate Moodle pages (one for Computer and one for Maths)

Computer Module assessments

For the Computer section of the module, there are two assessments. One is in January 2023, which will be a Computer Based Test (covering content taught in the first teaching block). It is worth 30% of the over module score. The second is in the May/June 2023 assessment period. It will be computer based. This assessment will be worth 40% of the overall module score.
Both assessments are closed book however a formula sheet will be provided for the January assessment. Nothing is provided for the May/June assessment.
The pass mark for the entire module is 40%, this score is generated from all the computer assessments AND all the maths assessments.

Module structure

There will be a one hour lecture per week, where content is introduced to us. This will be delivered using worksheets for the first 10 weeks.
There will also a practical session each week where the cohort is split into smaller groups. These sessions will be a chance to practice the ideas introduced in the lectures. There will be more members of staff around at the practical sessions to help out.

More Information on Practical Sessions

There are practical session guidelines available in the induction slides or on Moodle.

Content in each Week

There is a teaching plan on Moodle which outlines the content covered each week as well as the weeks in which the exams will be held.

S.2. Binary Arithmetic

26-09-2022 16:15 Farzad RB LT1

Number Systems

There are a number of different number systems and different methods to convert between them.

Denary (Base 10)

Used most commonly, this is the one most people learn.

10^x	10^3	10^2	10^1	10^0
$10^x =$	1000	100	10	1
	4	2	5	1

The total of the numbers above would be calculated in the following way:
$4251 = (1000 \times 4) + (100 \times 2) + (10 \times 5) + (1 \times 1)$
Denary is also known as base 10, this means each column can have one of ten possible values (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

Binary (Base 2)

This is base 2, this means each column can have one of two possible values (0, 1). The columns are also different. Moving from right to left, the columns double each time.

2^x	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$2^x =$	128	64	32	16	8	4	2	1
	1	0	1	1	0	0	1	1

The largest value which can be stored in 8-bits of binary is 11111111_2 or 255_{10}.

Hexadecimal (Base 16)

Also known as Hex. Using this method, numbers up to 255 can be stored in two characters. This is used a lot in computing, especially in graphics and website development. Each column can have one of 16 values (1 2 3 4 5 6 7 8 9 A B C D E F). The letters are used to represent two-digit numbers as seen below.

Hex:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

To calculate the value held in a Hex number, we calculate in a similar way to Denary and Binary as seen below.

16^x	16^3	16^2	16^1	16^0
$16^x =$	4096	256	16	1
	D	3	C	E

$D3CE = (13 \times 4096) + (3 \times 256) + (12 \times 16) + (14 \times 1) = 54222$

Converting Between Number Systems

Binary To Denary

Add together all the columns in which there is a 1. Using the example shown in the binary section, the total would be 179.

Denary To Binary

This is the reverse of binary to denary. Work from right to left seeing if the value will fit into the column, if it won't then mark down an zero and move onto the next.

Denary to Hex

The easiest way to do this is to go via Binary. Convert the number into binary, then split the binary into two nibbles. The values inputted in the previous step don't need to change. With the two nibbles of (4, 2, 1, 0), convert each of them back into denary, giving two individual digits, then convert each of those into Hex.

Binary Addition

Basic Rules

There are four basic rules to binary addition:

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

The last one (1+1) is a special case; strictly speaking, the answer is 0 with the 1 carried over. This is particularly useful in digital circuitry.

Binary Addition Example

Add 100 + 011

1. Draw out the binary addition columns

$$\begin{array}{cccc} & 1 & 1 & 0 \\ + & 0 & 1 & 1 \\ \hline & & & \end{array}$$

2. Start with the right-most column and add the digits

$$\begin{array}{cccc} & 1 & 1 & 0 \\ + & 0 & 1 & 1 \\ \hline & & & 1 \end{array}$$

3. Move to the next column. Add those digits together. As this is 1+1 = 0 carry 1, we write a little 1 in the next column as a carry.

$$\begin{array}{cccc} & {}^{1}1 & 1 & 0 \\ + & 0 & 1 & 1 \\ \hline & & 0 & 1 \end{array}$$

4. Move to the next column and add that. Remember to add the carry (making the sum $1 + 1 + 0$). This results in 0 carry 1, so, again, we add a little 1 in the next column. It doesn't matter that there aren't any other numbers there to be added, we will make a column!

$$\begin{array}{cccc} & {\scriptstyle 1} & {\scriptstyle 1}1 & 1 & 0 \\ + & & 0 & 1 & 1 \\ \hline & & 0 & 0 & 1 \end{array}$$

5. Add the final column

$$\begin{array}{cccc} & {\scriptstyle 1} & {\scriptstyle 1}1 & 1 & 0 \\ + & & 0 & 1 & 1 \\ \hline & 1 & 0 & 0 & 1 \end{array}$$

This gives us out final answer of $110 + 011 = 1001$

Binary Multiplication

There are 4 basic rules for binary multiplication.

```
0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1
```

Binary Multiplication Example

Multiply 10×11

1. Draw out the multiplication grid as you would for a standard column multiplication with decimal numbers.

$$\begin{array}{ccc} & 1 & 1 \\ \text{x} & 1 & 0 \\ \hline \end{array}$$

2. Take the right-most digit of the bottom binary number, we will multiply it with each of the digits above and place their results directly underneath.
   $0 \times 1 = 0$, place this directly under the 0
   $0 \times 1 = 0$, place this to the left.

$$\begin{array}{ccc} & 1 & 1 \\ \text{x} & 1 & 0 \\ \hline & 0 & 0 \end{array}$$

3. Next, move to the next digit on the bottom row, repeat the same process as before.

$$\begin{array}{ccc} & 1 & 1 \\ \text{x} & 1 & 0 \\ \hline & 0 & 0 \\ 1 & 1 & \end{array}$$

4. We then add our two answer rows together, using the rules of binary addition.

$$\begin{array}{cccc} & & 1 & 1 \\ \text{x} & & 1 & 0 \\ \hline & & 0 & 0 \\ + & 1 & 1 & \\ \hline & 1 & 1 & 0 \end{array}$$

This gives us the final answer of $11 \times 10 = 110$

Binary Subtraction

At this stage, there are four basic rules for binary subtraction. At a later stage, there will be negative numbers introduced when we look at signed binary so more rules will be introduced.

```
 0 - 0 = 0
 1 - 0 = 1
 1 - 1 = 0
10 - 1 = 1
```

Binary Subtraction Example

Subtract $110 - 001$

1. Draw out the subtraction columns as you would for a standard decimal column subtraction

$$\begin{array}{cccc} & 1 & 1 & 0 \\ - & 0 & 0 & 1 \\ \hline \end{array}$$

2. Start at the right hand most column $(0 - 1)$. This is something which we can't do, so we have to borrow 1 from the left hand column. To represent this, cross out the borrowed digit, replace with a 0 and in the current column, add a little 1 to the left.
This leaves us with $10 - 1$, which we can do and know equals 1.

$$\begin{array}{cccc} & 1 & {}^{0}\cancel{1} & {}^{1}0 \\ - & 0 & 0 & 1 \\ \hline & & & 1 \end{array}$$

3. Now move to the next column, and perform that operation. This is the middle column which is $0 - 0 = 0$.

$$\begin{array}{cccc} & 1 & {}^{0}\cancel{1} & {}^{1}0 \\ - & 0 & 0 & 1 \\ \hline & & 0 & 1 \end{array}$$

4. Move to the next column, and perform that operation. This is the left column which is $1 - 0 = 1$

$$\begin{array}{cccc} & 1 & {}^{0}\cancel{1} & {}^{1}0 \\ - & 0 & 0 & 1 \\ \hline & 1 & 0 & 1 \end{array}$$

This gives us the final answer of $110 - 001 = 101$

Binary Division

Binary division follows much the same procedure as 'bus stop' decimal division.

Binary Division Example

Divide $110 \div 10$

1. Draw out the division columns as you would for a standard decimal 'bus stop' division.

$$\begin{array}{cc|ccc} 1 & 0 & 1 & 1 & 0 \end{array}$$

2. Start by looking for factors and find 11 is greater than 10. We then write the number of times the value goes into 11 at the top, and the value itself underneath.

$$\begin{array}{cc|ccc} & & & 1 & \\ \hline 1 & 0 & 1 & 1 & 0 \\ & & 1 & 0 & \end{array}$$

3. We then subtract to see if there is a remainder. $(11 - 10 = 01)$
The remainder is written up on the top line

$$\begin{array}{cc|ccc} & & & 1 & 1 \\ \hline 1 & 0 & 1 & 1 & 0 \\ & - & 1 & 0 & \\ \hline & & 0 & 1 & \end{array}$$

4. We then bring down the final digit in the division (0), to where we are working.

$$\begin{array}{cc|ccc} & & & 1 & 1 \\ \hline 1 & 0 & 1 & 1 & 0 \\ & & 1 & 0 & \\ \hline & & 0 & 1 & 0 \end{array}$$

5. Now, we look to see if our divisor can fit in again. It does fit again, so we subtract it. $(010 - 10 = 0)$ It leaves no remainder.

$$\begin{array}{cc|ccc} & & & 1 & 1 \\ \hline 1 & 0 & 1 & 1 & 0 \\ & & 1 & 0 & \\ \hline & & 0 & 1 & 0 \\ & - & & 1 & 0 \\ \hline & & & & 0 \end{array}$$

This gives us the answer of $110 \div 10 = 11$.

S.3. Negative Numbers

03-10-22 16:00 Farzad RB LT1

In computers, subtraction is not possible. We must convert the calculation to be an addition. For example `5-3` is not possible, so it becomes `5+(-3)`. This means we need to be able to represent negative numbers in binary; there are three methods we can use to do this.

Sign and Magnitude

In this method, the Most Significant Bit (MSB) is replaced to show the sign rather than a number. A 0 represents a positive number and a 1 represents a negative number. The other bits behave the same.
Converting to and from sign and magnitude binary and decimal is the same as unsigned binary.

	+/−	64	32	16	8	4	2	1
27	0	0	0	1	1	0	1	1
-27	1	0	0	1	1	0	1	1
+13	0	0	0	0	1	1	0	1
-34	1	0	1	0	0	0	1	0

1's Complement

To convert to 1's complement, first you need to convert to unsigned binary. You then invert the bits so that 0s become 1s and 1s become 0s.
When doing a 1s complement addition, its important that any overflow bits are carried around to the least significant bit and added on there.

1's Complement subtraction example

Perform the calculation 10-6 = 1010 - 0110.
First, convert the second value to 1s complement = 1010 + 1001. Then draw out the addition grid and perform the addition

	1	0	1	0
+	1	0	0	1
	0	0	1	1
1				

As we have an overflowing carry, we have to add this to the least significant bit of the answer.

	1	0	1	0
+	1	0	0	1
	0	0	1	1
+				1
	0	1	0	0
		1	1	

And here we have our final answer, 4.

2's Complement

To convert to decimal to 2's complement binary, first convert to unsigned binary. Then work from right to left, inverting the bits so that 0 becomes 1 and 1 becomes 0. However, don't flip any bits to the right of or including the first 1. All bits to the left of should be flipped.

2's Complement subtraction example

Perform the calculation 6-1 = 110-001.
First, convert the second value to 2's complement = 111. Then draw out the addition grids and perform the addition.

	1	1	0
+	1	1	1
	1	0	1
1	1		

We have an overflow carry, we discard this. This gives us our final answer of 5.

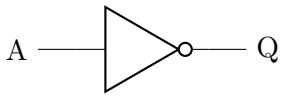
S.4. Digital Gates

10-10-22 16:00 Farzad RB LT1

Digital gates are used as the building block of digital systems. They manipulate inputs to provide outputs.
Throughout this lecture, its important to remember that a signal of 1 represents on (or high voltage) and a signal of 0 represents off (or low voltage).
Digital gates can be represented by a circuit symbol. Their inputs and outputs can be mapped onto a Truth Table and they have symbols which can be used in expressions to describe the circuit.

NOT Gaate

This can also be called an inverter.
As the name inverter suggests, the NOT gate inverts the bits. This means a 0 inputted, will output a 1 and a 1 inputted will output a 0.
NOT gate can only have one input.



$$Q = \overline{A}$$

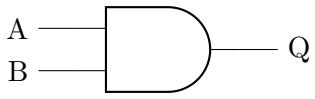
$$Q = A'$$

A	Q
0	1
1	0

Truth table for the NOT gate.

AND Gate

This gate compares two or more inputs. If all its inputs are 1, then it outputs 1; otherwise it outputs 0.
The rules of binary multiplication are the same of the AND gate.



$$Q = A \bigwedge B$$

$$Q = A \cdot B$$

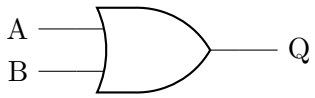
$$Q = AB$$

B	A	Q
0	0	0
0	1	0
1	0	0
1	1	1

Truth table for the AND gate.

OR Gate

This gate compares two or more inputs. If one or more input is 1, then it outputs 1; otherwise it outputs 0.



$$Q = A \bigvee B$$

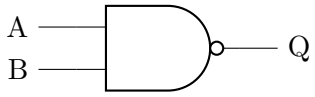
$$Q = A + B$$

B	A	Q
0	0	0
0	1	1
1	0	1
1	1	1

Truth table for the OR gate.

NAND Gate

Not AND
This gate compares two or more inputs. It outputs 1 where at least one of the inputs are not 1 and 0 where all the inputs are 1.
Through combinations of this gate, all the other logic gates can be made, due to this, it is called a Universal Gate.



$$Q = \overline{AB} = (AB)'$$

$$Q = \overline{A \bigwedge B} = (A \bigwedge B)'$$

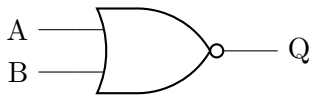
$$Q = \overline{A \cdot B} = (A \cdot B)'$$

B	A	Q
0	0	1
0	1	1
1	0	1
1	1	0

Truth table for the NAND gate.

NOR Gate

Not OR
This gate compares two or more inputs. Where all the inputs are 0, it outputs 1; otherwise it outputs 0.
Through combinations of this gate, all other logic gates can be constructed, due to this it can be called a Universal Gate.



$$Q = \overline{A + B} = (A + B)'$$

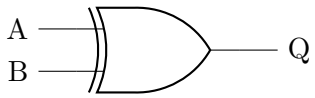
$$Q = \overline{A \bigvee B} = (A \bigvee B)'$$

B	A	Q
0	0	1
0	1	0
1	0	0
1	1	0

Truth table for the NOR gate.

XOR Gate

eXclusive OR
This gate compares two or more inputs. For a 2-input XOR gate, where the two inputs are different, it outputs 1; otherwise it outputs 0.
This gate is used for adder circuits.



$$Q = A \oplus B$$

$$Q = AB' + A'B$$

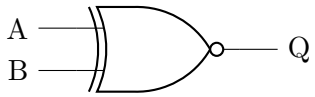
$$Q = A \cdot \overline{B} + \overline{A} \cdot B$$

B	A	Q
0	0	0
0	1	1
1	0	1
1	1	0

Truth table for the XOR gate.

XNOR Gate

eXclusive Not OR
This gate compares two or more inputs and where the inputs are the same, it outputs 1 and where the inputs are different, it outputs 0.



$$Q = \overline{A \oplus B}$$

$$Q = \left(AB' + A'B\right)'$$

B	A	Q
0	0	1
0	1	0
1	0	0
1	1	1

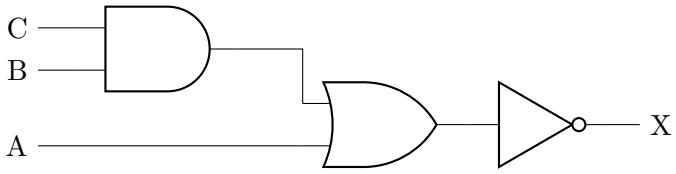
Truth table for the XNOR gate.

Practice Questions

Question 1

Find the value of X where the inputs have the following values.

$$A = 0 \quad B = 1 \quad C = 1$$

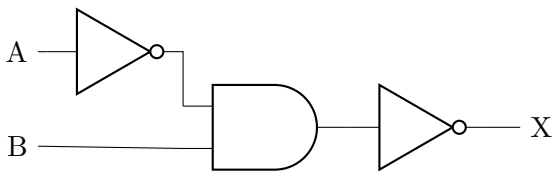


X=0

Question 2

Find the value of B where the logic system is as follows and has the following values.

$$X = 0 \quad A = 0$$

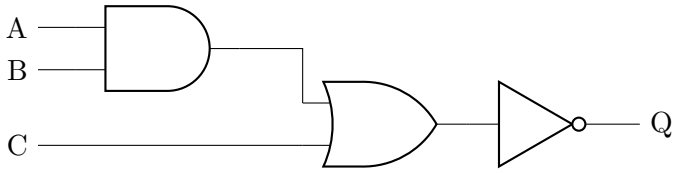


B=1

Question 3

Convert the following boolean logic expression to a circuit diagram.

$$\overline{A \cdot B + C}$$



S.5. Adders and Subtractors

25-10-22 16:00 Farzad RB LT1

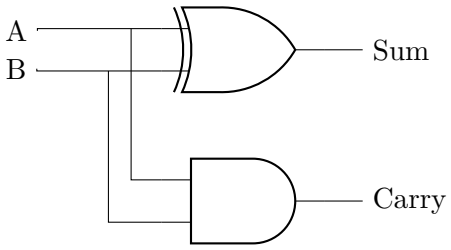
Within computers, there needs to be a way that numbers can be added together and subtracted. This is done using something called an adder subtractor. There are a number of different versions of the circuit which we need to look at first.

Half Adder

To start with, if we look at a truth table showing two inputs (B and A) and two outputs (sum and $carry$), we can show the combinations of gates which we need for a half adder.

B	A	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the table, we can see that the sum column represents the truth table for an XOR gate and the carry column represents the truth table for an AND gate. We can draw this as a circuit diagram.

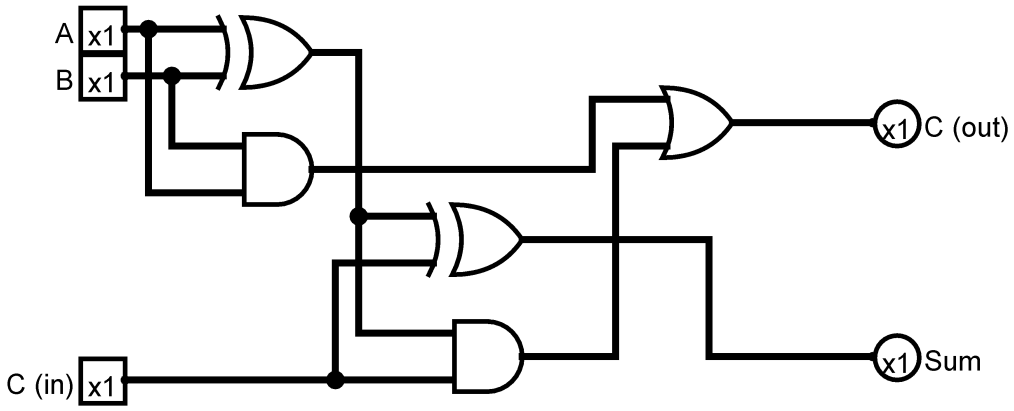


Full Adder

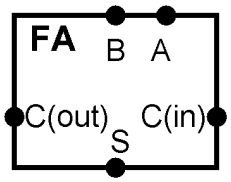
The circuit above is all well and good for adding one bit to another bit, but what if we are trying to do two bit addition (e.g. 11 + 01). The table below shows how we would express this in a truth table.

A	B	C_{in}	sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

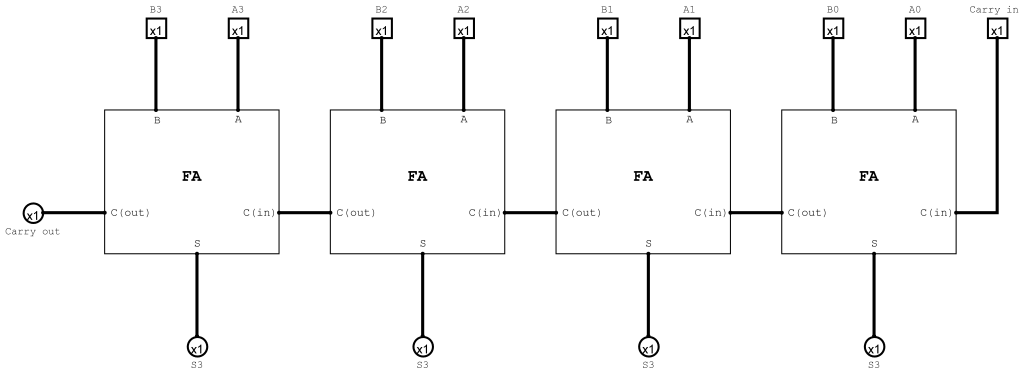
This addition is a two stage. First add $A+B = S_{interim}+C_{interim}$. Then add $C_{in+S} = S_{out}+C_{interim2}$. This can be represented in the following circuit.



We can turn this circuit into a block for diagram simplicity.

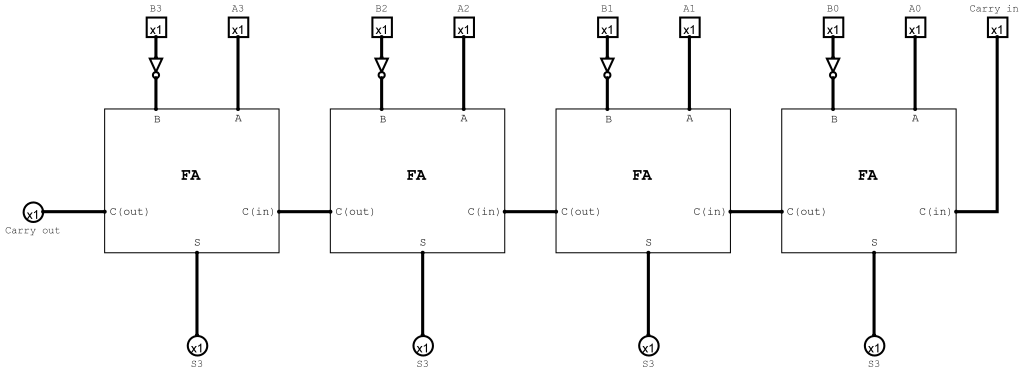


So, using 5 gates we can add 2 bits and give a carry out (this is 2 XOR, 2 AND and 1 OR). What happens if we want to add 1101 and 0110 (4 bits).
We will need a bigger adder. The general rule of thumb is one full adder for each bit we want to add.



Subtracting

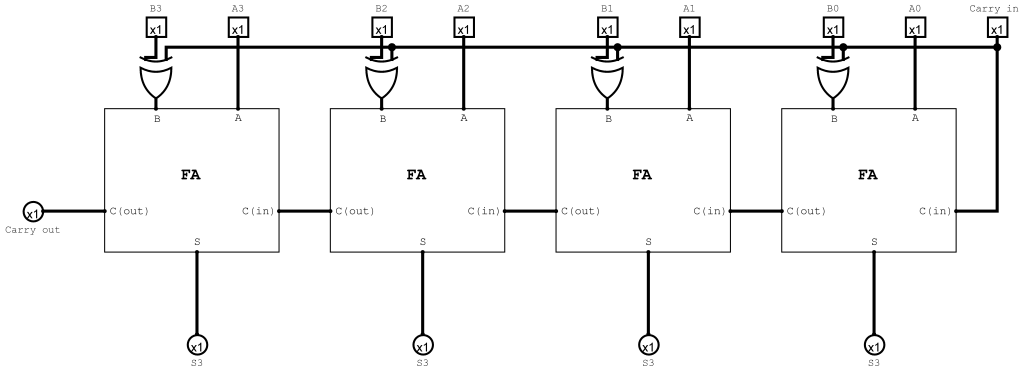
As we know from earlier weeks, computers do not subtract, they add a negative number. This poses a challenge as we need to convert the second number in the sum to a negative number using a two's complement number. To do this, we have to first flip the bits then add a 1 to the cary in of the least significant bit adder. This would generate the following circuit diagram.



We use two's complement to avoid having positive and negative zero.
Inside a computer, it is a bad use of space to have two circuits, we instead want one 'General Purpose Circuit'. This should add and subtract.

Adder Subtractor

The subtractor bit of the circuit will always flip B. We need to find a combination of gates that when set to add, doesn't invert and when set to subtract, it inverts B. This gate is the XOR gate, this circuit diagram is shown below.



If the cary in switch is 0, B is kept the same and $C_{in} = 0$. If the carry in switch is 1, B is flipped $= \overline{B}$ and $C_{in} = 1$, which adds the 1 needed for the addition part of two's complement.

S.6. Boolean Algebra Simplification I

07-11-22 16:00 Farzad RB LT1

Now we have covered all the different gates and how they interact, we need to look at ways to make the circuits simpler. There are a number of ways this can be done, either through Boolean Algebra Simplification or through Karnaugh Maps. We'll be looking at the former in this lecture.
As a general principle of simplification, we need to get rid of the brackets to see exactly what we are dealing with then simplify (which may involve returning some things to brackets).

Laws of Boolean Algebra

Law 1: Commutative

This law states that the order of terms is interchangable without changing the overall expression permitting the gates between them do not change.

$$A + B = B + A$$

$$A \cdot B = B \cdot A$$

Law 2: Associative

This law states that where there are multiple of the same operator used brackets, and therefore gates, can be placed anywhere.

$$\begin{aligned} A + (B + C) &= (A + B) + C \\ &= A + B + C \end{aligned}$$

$$\begin{aligned} A \cdot (B \cdot C) &= (A \cdot B) \cdot C \\ &= A \cdot B \cdot C \end{aligned}$$

Law 3: Distributive (Associative)

This rule acts much like simplification in standard maths where a common term is 'factored' out so that it only appears once in the equation.

$$A(B + C) = A \cdot B + A \cdot C$$

Rules of Boolean Algebra

Operations with 0 and 1

Rule 1

In this rule, the 0 bit means that the output will always be A.

$$A + 0 = A$$

A	0	Q
0	0	0
1	0	1

Rule 2

In this rule, the 1 bit means that the output will always be 1.

$$A + 1 = 1$$

A	1	Q
0	1	1
1	1	1

Rule 3

In this rule, the 0 bit means that the output will always be 0.

$$A \cdot 0 = 0$$

A	0	Q
0	0	0
1	0	0

Rule 4

In this rule, the 1 bit means that the output will always be A.

$$A \cdot 1 = A$$

A	0	Q
0	0	0
1	0	1

Idempotent Rules

Rule 5

$$A + A = A$$

A	A	Q
0	0	0
1	1	1

Rule 6

$$A \cdot A = A$$

A	0	Q
0	0	0
1	1	1

Laws Of Complementarity

Rule 7

$$A + \overline{A} = 1$$

A	$\overline{A}$	Q
0	1	1
1	0	1

Rule 8

$$A \cdot \overline{A} = 0$$

A	$\overline{A}$	Q
0	1	0
1	0	0

Other Laws

Rule 9: Double Inversion

In this rule, the 0 bit means that the output will always be A.

$$\overline{\overline{A}} = A$$

$\overline{\overline{A}}$	Q
0	0
1	1

Rule 10

This rule is applicable in both directions. The bits removed can also be added back where needed. This is useful for some simplifications.

$$A + A \cdot B = A$$

$$A = A \cdot B + A$$

Derivation of Rule 10

$$\begin{aligned}
&A = A && \textit{rule } 10\\
&A = A + A \cdot B\\
&A = A + (A + A \cdot B) \cdot B && \textit{rule } 10\\
&A = A + A \cdot B + A \cdot B \cdot B && \textit{rule } 6\\
&A = A + A \cdot B + A \cdot B\\
&A = A + nA \cdot B+\\
&A = A + A \cdot B+
\end{aligned}$$

Rule 11

$$A + \overline{A} \cdot B = A + B$$

Derivation of Rule 11

$$\begin{aligned}
A + \overline{A} \cdot B = &\\
&= (A + A \cdot B) + \overline{A} \cdot B && \textit{rule } 10\\
&= A \cdot A + A \cdot B + \overline{A} \cdot B && \textit{rule } 7\\
&= A \cdot A + A \cdot B + A \cdot \overline{A} + \overline{A} \cdot B && \textit{rule } 8\\
&= A \cdot (A + B) + \overline{A} \cdot (A + B)\\
&= (A + \overline{A}) \cdot (A + B) && \textit{rule } 6\\
&= 1 \cdot (A + B)\\
&= A + B
\end{aligned}$$

Rule 12

$$(A + B) \cdot (A + C) = A + B \cdot C$$

Derivation of Rule 12

$$
\begin{aligned}
(A+B)\cdot(A+C) &= \\
&= A\cdot A + A\cdot C + A\cdot B + B\cdot C && \textit{rule } 7\\
&= A + A\cdot C + A\cdot B + B\cdot C \\
&= A\cdot(1+C) + A\cdot B + B\cdot C && \textit{rule } 2\\
&= A + A\cdot B + B\cdot C \\
&= A\cdot(1+B) + B\cdot C && \textit{rule } 2\\
&= A + B\cdot C
\end{aligned}
$$

S.7. Boolean Algebra Simplification II

14-11-22 16:00 Farzad RB LT1

NEW: Drop-in's this week are there for support. They are timetabled as Drop-Ins therefore not compulsory.
NB: From here-on, the $\overline{X}$ will be changing to X' as I now know that this is the format which we will *have* to use in the exam.

DeMorgan's Theorem

Theorem 1

$$(A \cdot B)' = A' + B'$$

Theorem 2

$$(A + B)' = A' \cdot B'$$

Multi-Part DeMorgan's

Simplify $(A \cdot B + C)'$

$$\begin{aligned}
(A \cdot B + C)' &= (A \cdot B)' \cdot C' && \text{Apply DeMorgan's to OR} \\
&= (A' + B') \cdot C' && \text{Apply DeMorgan's to bracket} \\
&= A' \cdot C' + B' \cdot C' && \text{Expand bracket}
\end{aligned}$$

How To Simplify

1. Apply De-Morgan's Theorem where possible. This will result in as single negated terms rather than bracketed negated terms
2. Remove brackets where possible by applying distributive laws
3. Apply rules & laws to simplify
4. Factorise the expression
5. Iterate through steps 3 and 4 until the expression is simplified.

Examples

Example 1: Simplify

$$\begin{aligned}
A \cdot B \cdot C' + B \cdot C' + B' \cdot A = & \\
&= B \cdot C' \cdot (A + 1) + B' \cdot A \\
&= B \cdot C' \cdot 1 + B' \cdot A \\
&= B \cdot C' + B' \cdot A
\end{aligned}$$

Example 2: Simplify

$$\begin{aligned} A \cdot B + A \cdot (B + C) + B \cdot (B + C) = & \\ &= A \cdot B + A \cdot B + A \cdot C + B \cdot B + B \cdot C \\ &= A \cdot B + A \cdot C + B + B \cdot C \\ &= A \cdot B + A \cdot C + B(1 + C) \\ &= A \cdot B + A \cdot C + B \\ &= B + A \cdot C \end{aligned}$$

Example 3: Simplify

$$\begin{aligned} \big([(B \cdot D + C) \cdot (A \cdot B')] + (B' \cdot A')\big) \cdot C = & \\ &= ([(B \cdot D + C)A \cdot B'] + A' \cdot B') \cdot C \\ &= [(A \cdot B' \cdot B \cdot D + A \cdot B' \cdot C) + A' \cdot B'] \cdot C \\ &= A \cdot B \cdot B' \cdot C \cdot D + A \cdot B' \cdot C \cdot C + A' \cdot B' \cdot C \\ &= A \cdot B' \cdot C + A' \cdot B' \cdot C \\ &= B' \cdot C \cdot (A + A') \\ &= B' \cdot C \cdot (1) \\ &= B' \cdot C \end{aligned}$$

Example 4: Simplify

$$\begin{aligned} A \cdot B' + A \cdot (B + C)' + B \cdot (B + C)' = & \\ &= A \cdot B' + A \cdot B' \cdot C' + B \cdot B' \cdot C' \\ &= A \cdot B' \cdot (1 + C') \\ &= A \cdot B' \cdot 1 \\ &= A \cdot B' \end{aligned}$$