

---

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

<b>1</b>	<b>Introduction to Module</b>	<b>2</b>
<b>2</b>	<b>Binary Arithmetic</b>	<b>3</b>
<b>3</b>	<b>Worksheet 1</b>	<b>9</b>
<b>4</b>	<b>Negative Numbers</b>	<b>16</b>
<b>5</b>	<b>Worksheet 2/1</b>	<b>18</b>
<b>6</b>	<b>Worksheet 2/2</b>	<b>23</b>
<b>7</b>	<b>Digital Gates</b>	<b>24</b>
<b>8</b>	<b>Worksheet 03</b>	<b>28</b>
<b>9</b>	<b>Adders and Subtractors</b>	<b>33</b>
<b>10</b>	<b>Boolean Algebra Simplification I</b>	<b>37</b>
<b>11</b>	<b>Boolean Algebra Simplification II</b>	<b>41</b>
<b>12</b>	<b>Karnaugh Maps</b>	<b>43</b>
<b>13</b>	<b>Exam Preperation</b>	<b>49</b>
<b>14</b>	<b>Computer Structure and Function</b>	<b>50</b>
<b>15</b>	<b>WORKSHEET 10 Computer Structure and Function</b>	<b>53</b>
<b>16</b>	<b>IAS Computer</b>	<b>56</b>
<b>17</b>	<b>WORKSHEET 11 IAS Computer</b>	<b>60</b>
<b>18</b>	<b>Fetch Execute Cycle</b>	<b>62</b>
<b>19</b>	<b>Interconnections</b>	<b>66</b>
<b>20</b>	<b>WORKSHEET 13 Interconnections</b>	<b>69</b>
<b>21</b>	<b>Computer Memory Systems</b>	<b>71</b>
<b>22</b>	<b>WORKSHEET 14 Computer Memory Systems</b>	<b>73</b>
<b>23</b>	<b>WORKSHEET: Guest Lecture</b>	<b>75</b>
<b>24</b>	<b>Operating Systems - Introduction</b>	<b>78</b>

# Page 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.

## Page 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:

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 1 &= 1 \\ 1 + 1 &= 10 \end{aligned}$$

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

$$\begin{array}{r} 1 \ 1 \ 0 \\ + \ 0 \ 1 \ 1 \\ \hline \end{array}$$

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

$$\begin{array}{r}
 1\ 1\ 0 \\
 +\ 0\ 1\ 1 \\
 \hline
 1
 \end{array}$$

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.

$$\begin{array}{r}
 1\ 1\ 1\ 0 \\
 +\ 0\ 1\ 1 \\
 \hline
 0\ 1
 \end{array}$$

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!

$$\begin{array}{r}
 1\ 1\ 1\ 1\ 0 \\
 +\ 0\ 1\ 1 \\
 \hline
 0\ 0\ 1
 \end{array}$$

5. Add the final column

$$\begin{array}{r}
 1\ 1\ 1\ 1\ 0 \\
 +\ 0\ 1\ 1 \\
 \hline
 1\ 0\ 0\ 1
 \end{array}$$

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

## Binary Multiplication

There are 4 basic rules for binary multiplication.

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

### Binary Multiplication Example

Multiply  $10 \times 11$

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

$$\begin{array}{r}
 11 \\
 \times 10 \\
 \hline
 \end{array}$$

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.

$$\begin{array}{r}
 11 \\
 \times 10 \\
 \hline
 00
 \end{array}$$

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

$$\begin{array}{r}
 11 \\
 \times 10 \\
 \hline
 00 \\
 11
 \end{array}$$

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

$$\begin{array}{r}
 11 \\
 \times 10 \\
 \hline
 00 \\
 + 11 \\
 \hline
 110
 \end{array}$$

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.

$$\begin{aligned}
 0 - 0 &= 0 \\
 1 - 0 &= 1 \\
 1 - 1 &= 0 \\
 10 - 1 &= 1
 \end{aligned}$$

**Binary Subtraction Example**Subtract  $110 - 001$ 

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

$$\begin{array}{r} 1 \quad 1 \quad 0 \\ - \quad 0 \quad 0 \quad 1 \\ \hline \end{array}$$

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.

$$\begin{array}{r} 1 \quad \overset{0}{\cancel{1}} \quad \overset{1}{0} \\ - \quad 0 \quad 0 \quad 1 \\ \hline \quad \quad 1 \end{array}$$

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

$$\begin{array}{r} 1 \quad \overset{0}{\cancel{1}} \quad \overset{1}{0} \\ - \quad 0 \quad 0 \quad 1 \\ \hline \quad 0 \quad 1 \end{array}$$

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

$$\begin{array}{r} 1 \quad \overset{0}{\cancel{1}} \quad \overset{1}{0} \\ - \quad 0 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 1 \end{array}$$

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.

$$10 \overline{) 110}$$

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.

$$\begin{array}{r} 1 \\ 10 \overline{) 110} \\ \underline{10} \end{array}$$

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

$$\begin{array}{r} 11 \\ 10 \overline{) 110} \\ \underline{- 10} \\ 01 \end{array}$$

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

$$\begin{array}{r} 11 \\ 10 \overline{) 110} \\ \underline{10} \\ 010 \end{array}$$


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.

$$\begin{array}{r} 11 \\ 10 \overline{) 110} \\ \underline{10} \\ 010 \\ \underline{- 10} \\ 0 \end{array}$$

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

## Page 3

# Worksheet 1

 20-09-22 Worksheet

### Basic Exercises

1. Convert the following numbers to binary

(a) 12 = 1100

(b) 103 = 1100111

(c) 97 = 1100001

(d) 55 = 0110111

(e) 395 = 110001011

2. Convert the following binary numbers to decimal

(a) 1101 = 13

(b) 101001 = 41

(c) 110111 = 55

(d) 1000011 = 135

(e) 11111110 = 254

3. Convert the following decimal numbers to hexadecimal numbers

(a) 1026

0100	0000	0010
------	------	------

4	0	2
---	---	---

= 402

(b) 5678

0001	0110	0010	1110
------	------	------	------

1	6	2	E
---	---	---	---

= 162E

(c) 9567

0010	0101	0101	1111
------	------	------	------

2	5	5	F
---	---	---	---

= 255E

(d) 72627

0001	0001	1011	1011	0011
1	1	B	B	3

= 11BB3

(e) 115497

0001	1100	0011	0010	1001
1	C	3	2	9

= 1C329

4. Convert the following hexadecimal numbers to binary numbers

(a) 2D = 0010 1101

(b) F3A = 1111 0011 1010

(c) 1BD = 0001 1011 1101

(d) ABC = 1010 1011 1100

(e) D3F2 = 1101 0011 1111 0010

**Core Exercises**

a) 11 + 11

$$\begin{array}{r}
 \phantom{+} \phantom{1} \phantom{1} \phantom{0} \\
 \phantom{+} \phantom{1} \phantom{1} \phantom{0} \\
 + \phantom{1} \phantom{1} \phantom{0} \phantom{0} \\
 \hline
 1 \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 1 \phantom{0} \phantom{0}
 \end{array}$$

= 110

b) 100 + 10

$$\begin{array}{r}
 \phantom{+} \phantom{1} \phantom{0} \phantom{0} \\
 \phantom{+} \phantom{1} \phantom{0} \phantom{0} \\
 + \phantom{1} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 1 \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 1 \phantom{0} \phantom{0}
 \end{array}$$

= 110

c) 111 + 11

$$\begin{array}{r}
 \phantom{+} \phantom{1} \phantom{1} \phantom{0} \\
 \phantom{+} \phantom{1} \phantom{1} \phantom{0} \\
 + \phantom{1} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 1 \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 1 \phantom{0} \phantom{0}
 \end{array}$$

= 1010

d) 110 + 100

$$\begin{array}{r}
 \phantom{+} \phantom{1} \phantom{1} \phantom{0} \\
 \phantom{+} \phantom{1} \phantom{1} \phantom{0} \\
 + \phantom{1} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 1 \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 1
 \end{array}$$

= 1010

e)

$$\begin{array}{r}
 \phantom{+} \phantom{1} \phantom{1} \phantom{1} \phantom{1} \\
 \phantom{+} \phantom{1} \phantom{1} \phantom{1} \phantom{1} \\
 + \phantom{1} \phantom{1} \phantom{1} \phantom{0} \phantom{0} \\
 \hline
 \phantom{1} \phantom{1} \phantom{0} \phantom{1} \phantom{1} \\
 \hline
 \phantom{1} \phantom{1}
 \end{array}$$

= 11011

g) 11 - 10

$$\begin{array}{r}
 \phantom{1} \phantom{1} \\
 - \phantom{1} \phantom{0} \\
 \hline
 \phantom{0} \phantom{1} \\
 \hline
 \phantom{0} \phantom{1}
 \end{array}$$

= 10

i) 101 - 011

$$\begin{array}{r}
 \phantom{0}1 \phantom{0} \phantom{1} \\
 - \phantom{0} \phantom{1} \phantom{1} \\
 \hline
 \phantom{0} \phantom{1} \phantom{0} \\
 \hline
 \phantom{0} \phantom{1} \phantom{0}
 \end{array}$$

= 010

k) 101 × 111

$$\begin{array}{r}
 \phantom{\times} \phantom{1} \phantom{0} \phantom{1} \\
 \phantom{\times} \phantom{1} \phantom{1} \phantom{1} \\
 \times \phantom{1} \phantom{0} \phantom{1} \\
 \hline
 \phantom{1} \phantom{0} \phantom{1} \\
 + \phantom{1} \phantom{0} \phantom{1} \\
 \hline
 \phantom{1} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \\
 \hline
 \phantom{1} \phantom{1} \phantom{1}
 \end{array}$$

= 100011

f) 11 - 01

$$\begin{array}{r}
 \phantom{1} \phantom{1} \\
 - \phantom{0} \phantom{1} \\
 \hline
 \phantom{1} \phantom{0} \\
 \hline
 \phantom{1} \phantom{0}
 \end{array}$$

= 10

h) 111-100

$$\begin{array}{r}
 \phantom{1} \phantom{1} \phantom{1} \\
 - \phantom{1} \phantom{0} \phantom{0} \\
 \hline
 \phantom{0} \phantom{1} \phantom{1} \\
 \hline
 \phantom{0} \phantom{1} \phantom{1}
 \end{array}$$

= 011

j) 11 × 11

$$\begin{array}{r}
 \phantom{\times} \phantom{1} \phantom{1} \\
 \times \phantom{1} \phantom{1} \\
 \hline
 \phantom{1} \phantom{1} \\
 + \phantom{1} \phantom{1} \\
 \hline
 \phantom{1} \phantom{0} \phantom{0} \phantom{1} \\
 \hline
 \phantom{1} \phantom{0} \phantom{0} \phantom{1}
 \end{array}$$

= 1001

l) 1101 × 1010

$$\begin{array}{r}
 \phantom{\times} \phantom{1} \phantom{1} \phantom{0} \phantom{1} \\
 \phantom{\times} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \\
 \times \phantom{1} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 + \phantom{1} \phantom{1} \phantom{0} \phantom{1} \\
 \hline
 \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{0} \\
 \hline
 \phantom{1} \phantom{1} \phantom{1}
 \end{array}$$

= 10000010

m)  $110 \div 11$

$$\begin{array}{r}
 \phantom{11}10 \\
 11 \overline{) 110} \\
 \underline{11} \phantom{0} \\
 000
 \end{array}$$

$= 10$

n)  $110 \div 10$

$$\begin{array}{r}
 \phantom{11}11 \\
 10 \overline{) 110} \\
 \underline{10} \phantom{0} \\
 010 \\
 \phantom{0}10 \\
 \underline{\phantom{0}0}0
 \end{array}$$

$= 11$

o)  $1100 \div 100$

$$\begin{array}{r}
 \phantom{11}11 \\
 100 \overline{) 1100} \\
 \underline{100} \phantom{0} \\
 0100 \\
 \phantom{0}100 \\
 \underline{\phantom{0}0}00
 \end{array}$$

$= 11$

## More Core Exercises

a)  $11 + 01$

$$\begin{array}{r}
 \phantom{11}11 \\
 + \phantom{11}01 \\
 \hline
 100 \\
 \hline
 11
 \end{array}$$

$= 100$

b)  $10 + 10$

$$\begin{array}{r}
 \phantom{11}10 \\
 + \phantom{11}10 \\
 \hline
 100 \\
 \hline
 1
 \end{array}$$

$= 100$

c)  $101 + 11$

$$\begin{array}{r}
 \phantom{11}101 \\
 + \phantom{11}11 \\
 \hline
 1000 \\
 \hline
 111
 \end{array}$$

$= 1000$

d)  $111 + 110$

$$\begin{array}{r}
 \phantom{11}111 \\
 + \phantom{11}110 \\
 \hline
 1101 \\
 \hline
 11
 \end{array}$$

e)  $1001 + 101$ 

$$\begin{array}{r}
 1 \quad 0 \quad 0 \quad 1 \\
 + \quad \quad 1 \quad 0 \quad 1 \\
 \hline
 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 1
 \end{array}$$

= 1110

g)  $11-1$ 

$$\begin{array}{r}
 1 \quad \quad 1 \\
 - \quad \quad 1 \\
 \hline
 1 \quad \quad 0 \\
 \hline
 \end{array}$$

= 10

i)  $110 - 101$ 

$$\begin{array}{r}
 1 \quad \quad 0^1 \quad \quad 10^0 \\
 - 1 \quad \quad 0 \quad \quad 1 \\
 \hline
 0 \quad \quad 0 \quad \quad 1 \\
 \hline
 \end{array}$$

= 001

k)  $1100 - 1001$ 

$$\begin{array}{r}
 1 \quad \quad 0^1 \quad \quad 1^0 \quad \quad 10^0 \\
 - 1 \quad \quad 0 \quad \quad 0 \quad \quad 1 \\
 \hline
 0 \quad \quad 0 \quad \quad 1 \quad \quad 1 \\
 \hline
 \end{array}$$

= 0011

f)  $1101 + 1011$ 

$$\begin{array}{r}
 1 \quad 1 \quad 0 \quad 1 \\
 + \quad \quad 1 \quad 0 \quad 1 \quad 1 \\
 \hline
 1 \quad 1 \quad 0 \quad 0 \quad 0 \\
 \hline
 1 \quad 1 \quad 1 \quad 1
 \end{array}$$

= 11000

h)  $101 - 100$ 

$$\begin{array}{r}
 1 \quad \quad 0 \quad \quad 1 \\
 - 1 \quad \quad 0 \quad \quad 0 \\
 \hline
 0 \quad \quad 0 \quad \quad 1 \\
 \hline
 \end{array}$$

= 001

j)  $1110 - 11$ 

$$\begin{array}{r}
 1 \quad \quad 0^1 \quad \quad 1^1 \quad \quad 10^0 \\
 - \quad \quad \quad \quad 1 \quad \quad 1 \\
 \hline
 1 \quad \quad 0 \quad \quad 1 \quad \quad 1 \\
 \hline
 \end{array}$$

= 1011

l)  $11010 - 10111$ 

$$\begin{array}{r}
 1 \quad \quad 0^1 \quad \quad 1^0 \quad \quad 10^1 \quad \quad 10^0 \\
 - 1 \quad \quad 0 \quad \quad 1 \quad \quad 1 \quad \quad 1 \\
 \hline
 0 \quad \quad 0 \quad \quad 0 \quad \quad 1 \quad \quad 1 \\
 \hline
 \end{array}$$

= 00011

n)  $100 \times 10$

[illegible]

$$= 1000$$

p)  $1000 \times 110$

$$\begin{array}{r} \phantom{0000}1\phantom{0}1\phantom{0}0\phantom{0}1 \\ \times \phantom{000}1\phantom{0}1\phantom{0}0 \\ \hline \phantom{0000}0\phantom{0}0\phantom{0}0\phantom{0}0 \\ \phantom{00}1\phantom{0}0\phantom{0}0\phantom{0}1 \\ + \phantom{00}1\phantom{0}0\phantom{0}0\phantom{0}1 \\ \hline \phantom{00}1\phantom{0}1\phantom{0}0\phantom{0}1\phantom{0}1\phantom{0}0 \end{array}$$

= 110110

r)  $1110 \times 1101$

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

$$= 10110110$$

s)  $100 \div 10$

$$\begin{array}{r} \phantom{10}10 \phantom{0} \overline{) 100} \\ \phantom{10}10 \phantom{0} \\ \hline \phantom{10}000 \end{array}$$

= 10

t)  $1001 \div 11$

$$\begin{array}{r} \phantom{100}11 \phantom{00} \overline{) 1001} \\ \phantom{100}11 \phantom{00} \\ \hline \phantom{100}011 \\ \phantom{100}11 \\ \hline \phantom{100}001 \end{array}$$

= 11

u)  $1100 \div 100$

$$\begin{array}{r} \phantom{100}11 \phantom{00} \overline{) 1100} \\ \phantom{100}11 \phantom{00} \\ \hline \phantom{100}0100 \\ \phantom{100}100 \\ \hline \phantom{100}000 \end{array}$$

= 11



## Page 4

# 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
<hr/>				
	0	0	1	1
<hr/>				
1				

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

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

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.

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

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

## Page 5

# Worksheet 2/1

📅 04-10-22

👁 Worksheet

### Core Exercises

Find the negative numbers using the sign and method for the following.

- (a)  $-7 = 10000111$
- (b)  $-12 = 10001100$
- (c)  $-15 = 1001111$
- (d)  $-46 = 10101110$
- (e)  $-57 = 10111001$
- (f)  $-112 = 11110000$
- (g)  $-149 = 100010010101$
- (h)  $-179 = 100010110011$
- (i)  $-216 = 100011011000$
- (j)  $-406 = 100110010110$
- (k)  $-1001 = 11111101001$
- (l)  $-1645 = 111001101101$

Find 1's Complement for the following.

- (a)  $01 = 10$
- (b)  $10 = 01$
- (c)  $111 = 000$
- (d)  $011 = 100$
- (e)  $100 = 011$
- (f)  $101 = 010$
- (g)  $0011 = 1100$
- (h)  $1001 = 0110$
- (i)  $10111 = 01000$

Find 2's Complement for the following.

(a) 01

$$\begin{array}{r} 1 \quad 0 \\ + \quad 1 \\ \hline 1 \quad 1 \\ \hline \end{array}$$

= 11

(b) 10

$$\begin{array}{r} 0 \quad 1 \\ + \quad 1 \\ \hline 1 \quad 0 \\ \hline 1 \end{array}$$

= 10

(c) 111

$$\begin{array}{r} 0 \quad 0 \quad 0 \\ + \quad 1 \\ \hline 0 \quad 0 \quad 1 \\ \hline \end{array}$$

= 001

(d) 011

$$\begin{array}{r} 1 \quad 0 \quad 0 \\ + \quad 1 \\ \hline 1 \quad 0 \quad 1 \\ \hline \end{array}$$

= 101

(e) 100

$$\begin{array}{r} 0 \quad 1 \quad 1 \\ + \quad 1 \\ \hline 1 \quad 0 \quad 0 \\ \hline 1 \quad 1 \quad 0 \end{array}$$

= 100

(f) 101

$$\begin{array}{r} 0 \quad 1 \quad 0 \\ + \quad 1 \\ \hline 0 \quad 1 \quad 1 \\ \hline \end{array}$$

= 011

(g) 0011

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 0 \\ + \quad 1 \\ \hline 1 \quad 1 \quad 0 \quad 1 \\ \hline \end{array}$$

= 1101

(h) 1001

$$\begin{array}{r} 1 \quad 1 \quad 1 \quad 0 \\ + \quad 1 \\ \hline 0 \quad 1 \quad 1 \quad 1 \\ \hline \end{array}$$

= 0111

(i) 10111

$$\begin{array}{r} 0 \quad 1 \quad 0 \quad 0 \quad 0 \\ + \quad 1 \\ \hline 0 \quad 1 \quad 0 \quad 0 \quad 1 \\ \hline \end{array}$$

= 01001

Carry out the following subtractions using (i) 1's complement and then (ii) 2's complement.  
*1's complement shown on the left and 2's complement shown on the right.*

(a) 11-01

$$\begin{array}{r}
 1 \quad 1 \quad 1 \\
 + \quad 1 \quad 0 \\
 \hline
 \quad 0 \quad 1 \\
 + \quad \quad 1 \\
 \hline
 1 \quad 0 \\
 \hline
 1
 \end{array}$$

= 10

$$\begin{array}{r}
 11 \quad 1 \\
 + \quad 1 \quad 1 \\
 \hline
 \cancel{1} \quad 1 \quad 0 \\
 \hline
 = 10
 \end{array}$$

(b) 11-10

$$\begin{array}{r}
 1 \quad 11 \quad 1 \\
 + \quad 0 \quad 1 \\
 \hline
 \quad 0 \quad 0 \\
 + \quad \quad 1 \\
 \hline
 0 \quad 1 \\
 \hline
 \hline
 \end{array}$$

= 01

$$\begin{array}{r}
 1 \quad 1 \quad 1 \\
 + \quad 1 \quad 0 \\
 \hline
 \cancel{1} \quad 0 \quad 1 \\
 \hline
 = 01
 \end{array}$$

(c) 101-011

$$\begin{array}{r}
 \quad 1 \quad 0 \quad 1 \\
 + \quad 1 \quad 0 \quad 0 \\
 \hline
 \quad 0 \quad 0 \quad 1 \\
 + \quad \quad \quad 1 \\
 \hline
 0 \quad 1 \quad 0 \\
 \hline
 \hline
 \end{array}$$

= 010

$$\begin{array}{r}
 1 \quad 1 \quad 10 \quad 1 \\
 + \quad 1 \quad 0 \quad 1 \\
 \hline
 \cancel{1} \quad 0 \quad 1 \quad 0 \\
 \hline
 = 010
 \end{array}$$

(d) 101-100

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

= 001

(e) 110-101

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

= 001

(f) 1110 - 0011

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

= 1011

(g) 1100 - 1001

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

= 001

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

= 001

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

= 1011

$$\begin{array}{r}
 1 \quad 11 \quad 1 \quad 0 \quad 0 \\
 + \quad 0 \quad 1 \quad 1 \quad 0 \\
 \hline
 \quad 0 \quad 0 \quad 1 \quad 0 \\
 + \quad \quad \quad 1 \\
 \hline
 \quad 0 \quad 0 \quad 1 \quad 1 \\
 \hline
 \end{array}$$

= 0011

(h) 11110 - 10011

$$\begin{array}{r}
 1 \quad 11 \quad 11 \quad 1 \quad 1 \quad 0 \\
 + \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \\
 \hline
 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \\
 + \quad \quad \quad 1 \\
 \hline
 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \\
 \hline
 \end{array}$$

= 01011

(i) 10111 - 10110

$$\begin{array}{r}
 1 \quad 11 \quad 10 \quad 11 \quad 11 \quad 1 \\
 + \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \\
 \hline
 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\
 + \quad \quad \quad 1 \\
 \hline
 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \\
 \hline
 \end{array}$$

= 00001

$$\begin{array}{r}
 1 \quad 11 \quad 1 \quad 0 \quad 0 \\
 + \quad 0 \quad 1 \quad 1 \quad 1 \\
 \hline
 \cancel{1} \quad 0 \quad 0 \quad 1 \quad 1 \\
 \hline
 \end{array}$$

= 0011

$$\begin{array}{r}
 1 \quad 11 \quad 11 \quad 1 \quad 1 \quad 0 \\
 + \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \\
 \hline
 \cancel{1} \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \\
 \hline
 \end{array}$$

= 01011

$$\begin{array}{r}
 1 \quad 11 \quad 10 \quad 1 \quad 1 \quad 1 \\
 + \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \\
 \hline
 \cancel{1} \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \\
 \hline
 \end{array}$$

= 00001

## Page 6

# Worksheet 2/2

📅 05-10-22

👁 Worksheet

1. Convert the decimal numbers to signed and unsigned 8-bit binary numbers.

Decimal	Unsigned Binary	Signed		
		Sign-Magnitude	1's Complement	2's Complement
11	00001011	00001011	00001011	00001011
-11	—	10001011	11110100	11110101
29	00011101	00011101	00011101	00011101
-29	—	10011101	11100010	11100011
114	01110010	01110010	01110010	01110010
-114	—	11110010	10001101	10001110
111	01101111	01101111	01101111	01101111
-111	—	11101111	10010000	10010001

2. Convert the binary numbers using the required methods.

Binary Numbers	Unsigned	Signed			
		Sign-Magnitude	1's Complement	2's Complement	Complement
00001101	13	13	13	13	
10001001	137	-9	-118	-119	
11000100	196	-68	-59	-60	
01000100	68	68	68	68	
11111111	255	-127	0	-1	
01111111	127	127	127	127	
10100101	165	-37	-90	-91	
11001001	201	-73	-54	-55	



## Page 7

# 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).

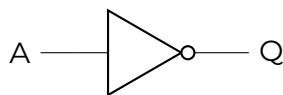
Digital gates can be represented by a circuit symbol. Their inputs and outputs can be mapped onto a Truth Table and they have symbols which can be used in expressions to describe the circuit.

## NOT Gate

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.



$$Q = \bar{A}$$

$$Q = A'$$

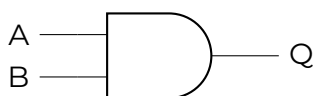
A	Q
0	1
1	0

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.



$$Q = A \wedge B$$

$$Q = A \cdot B$$

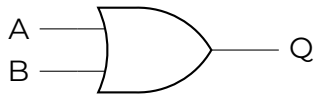
$$Q = AB$$

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.



$$Q = A \vee B$$

$$Q = A + B$$

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

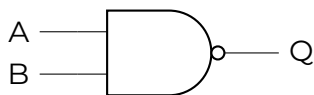
Truth table for the OR gate.

## 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.



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

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

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

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

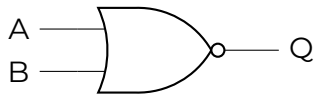
Truth table for the NAND gate.

## 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.



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

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

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

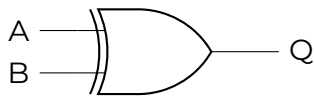
Truth table for the NOR gate.

## 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.



$$Q = A \oplus B$$

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

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

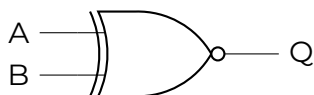
B	A	Q
0	0	0
0	1	1
1	0	1
1	1	0

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.



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

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

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

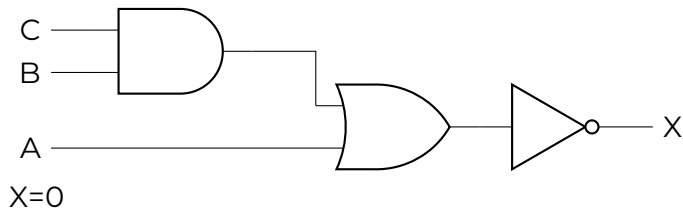
Truth table for the XNOR gate.

## Practice Questions

### Question 1

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

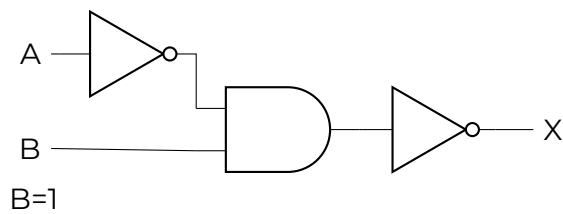
$$A = 0 \quad B = 1 \quad C = 1$$



### Question 2

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

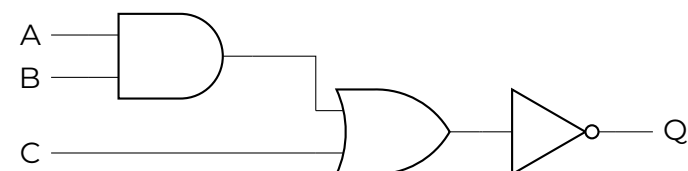
$$X = 0 \quad A = 0$$



### Question 3

Convert the following boolean logic expression to a circuit diagram.

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



Page 8

Worksheet 03

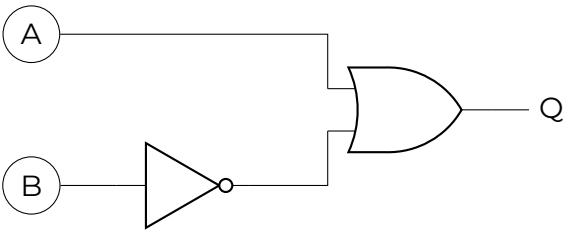
15-10-22

Worksheet

Basic Exercises

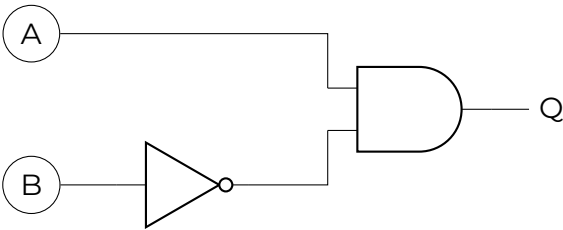
1. Translate the following boolean expressions into a digital gate circuit and construct a truth table.

a.  $A + \overline{B}$



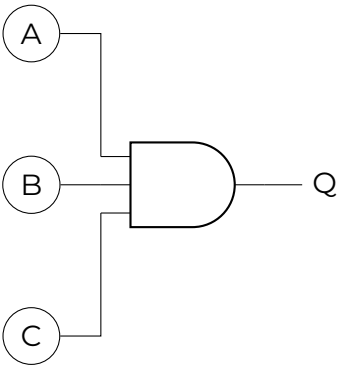
B	A	Q
0	0	1
0	1	1
1	0	0
1	1	1

b.  $A \cdot \overline{B}$



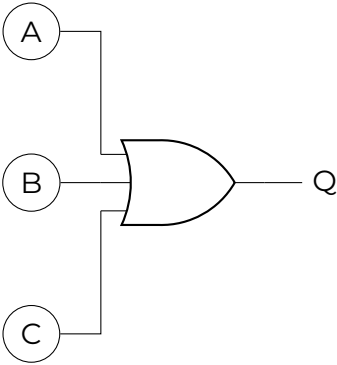
B	A	Q
0	0	0
0	1	1
1	0	0
1	1	0

c.  $A \cdot B \cdot C$



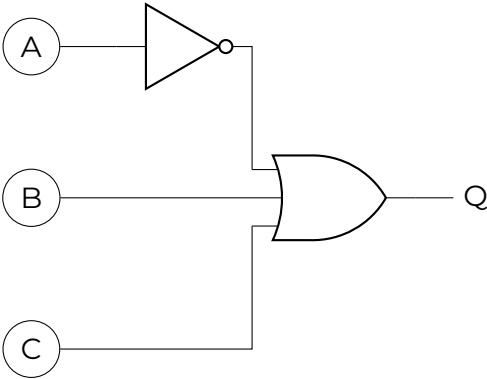
C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

d.  $A + B + C$



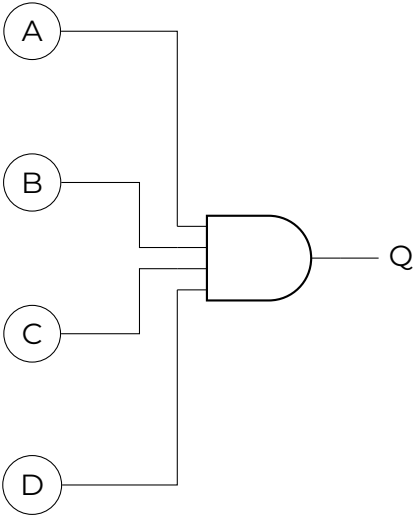
C	B	A	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

e.  $\overline{A} + B + C$



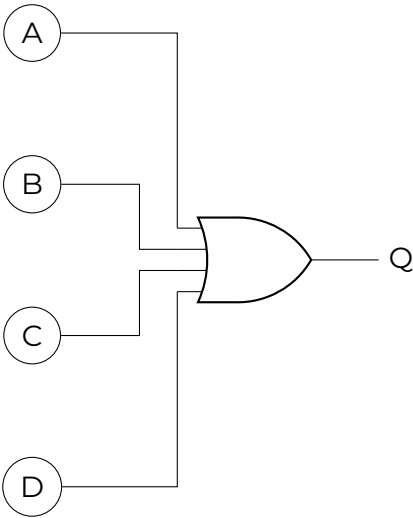
C	B	A	Q
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

f.  $A \cdot B \cdot C \cdot D$



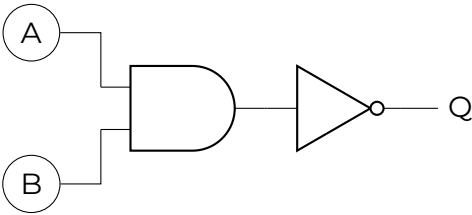
D	C	B	A	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

g.  $A + B + C + D$



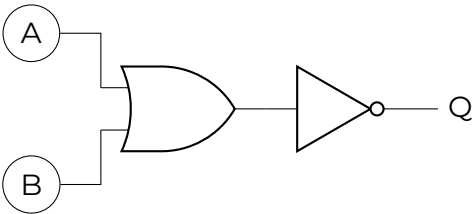
D	C	B	A	Q
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

h.  $\overline{A \cdot B}$



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

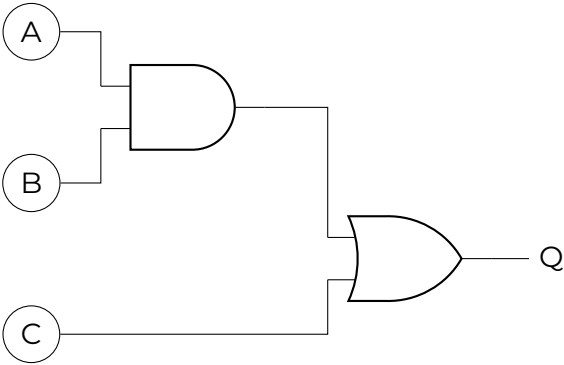
i.  $\overline{A + B}$



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

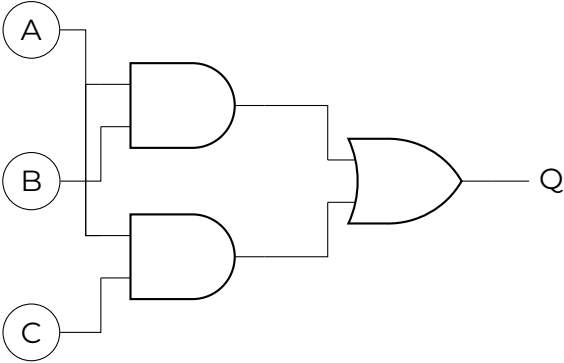


j.  $A \cdot B + C$



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

k.  $A \cdot B + A \cdot C$



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

## Page 9

# Adders and Subtractors

📅 25-10-22

🕒 16:00

🎓 Farzad

📍 RB LT1

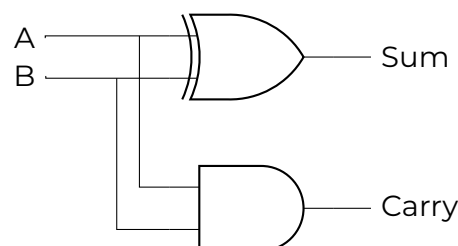
Within computers, there needs to be a way that numbers can be added together and subtracted. This is done using something called an adder subtractor. There are a number of different versions of the circuit which we need to look at first.

## 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.

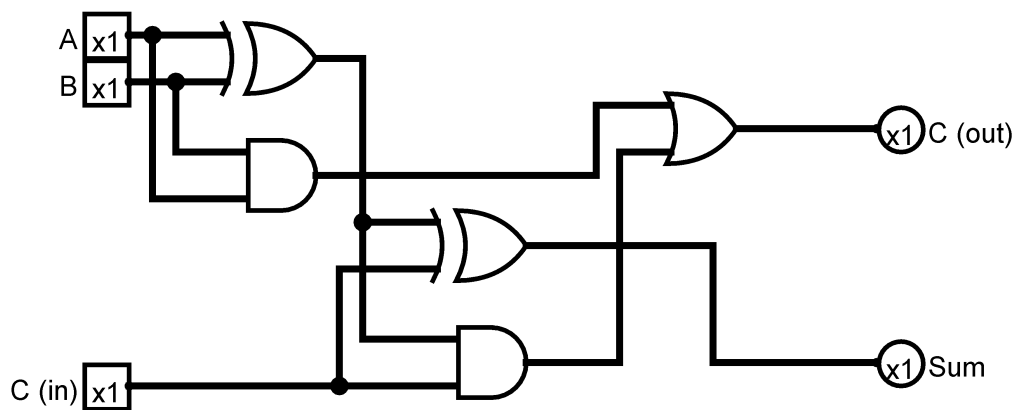


## Full Adder

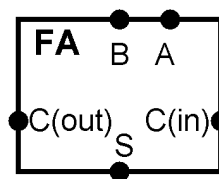
The circuit above is all well and good for adding one bit to another bit, but what if we are trying to do two bit addition (e.g.  $11 + 01$ ). The table below shows how we would express this in a truth table.

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.

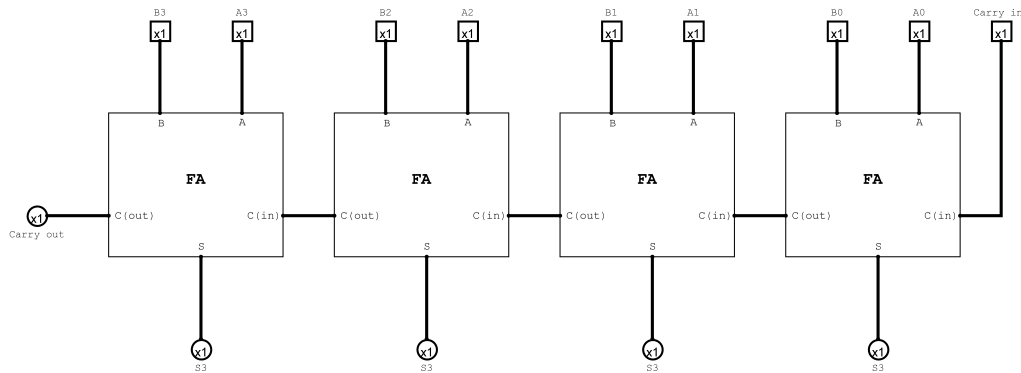


We can turn this circuit into a block for diagram simplicity.



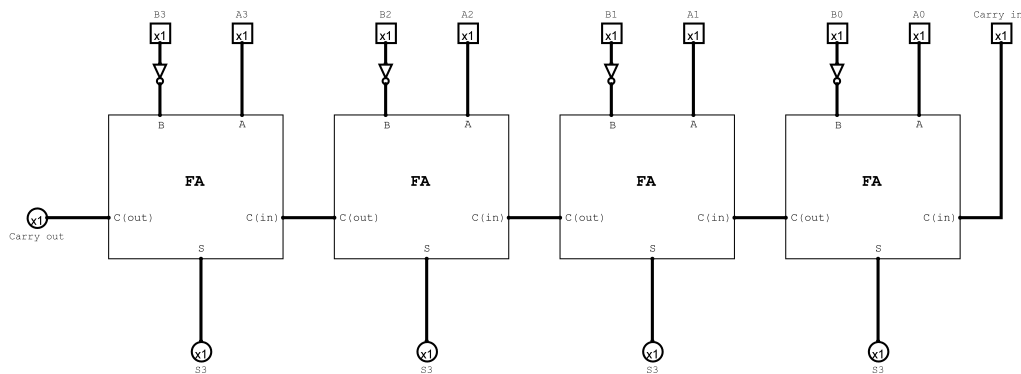
So, using 5 gates we can add 2 bits and give a carry out (this is 2 XOR, 2 AND and 1 OR). What happens if we want to add 1101 and 0110 (4 bits).

We will need a bigger adder. The general rule of thumb is one full adder for each bit we want to add.



## Subtracting

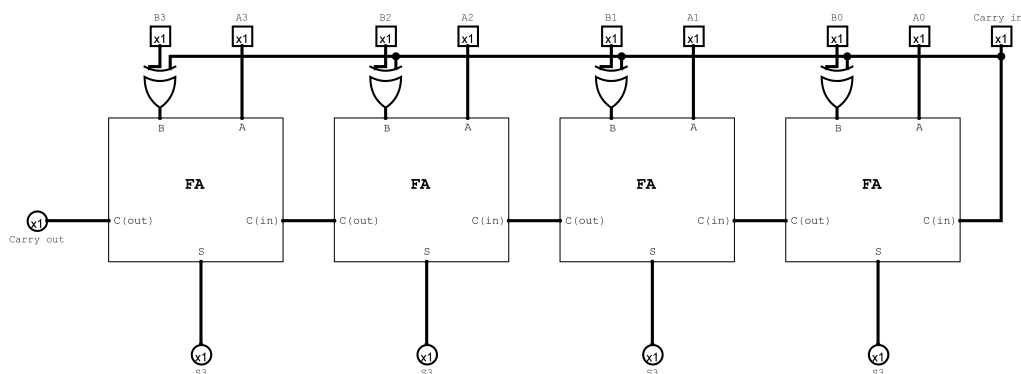
As we know from earlier weeks, computers do not subtract, they add a negative number. This poses a challenge as we need to convert the second number in the sum to a negative number using a two's complement number. To do this, we have to first flip the bits then add a 1 to the carry in of the least significant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

## Page 10

# 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 interchangeable 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	1	Q
0	1	0
1	1	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**



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

**Rule 11**

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

**Derivation of Rule 11**

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

**Rule 12**

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

**Derivation of Rule 12**

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

## Page 11

# 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**  $(A \cdot B + C)'$

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

Apply DeMorgan's to OR  
Apply DeMorgan's to bracket  
Expand bracket

## 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

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

### Example 4: Simplify

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

## Page 12

# Karnaugh Maps

📅 28-11-2022

🕒 16:00

🎓 Farzad

📍 RB LT1

This is the last topic we will cover before the Christmas break. There will be an exam in January, which will be a Computer Based Test. In the exam, we can use either a Karnaugh map or the rules & laws to simplify boolean algebra.

## Sum Of Products Form

Sum Of Products (SOP) form is a form in which there are no brackets. A boolean algebra equation needs to be in SOP form before it can be put into a Karnaugh map.

### Converting equation to SOP Form

Convert the following equation to SOP form.

$$A(B + C) + (CD)' + (A + B)'$$

We use the Distributive laws and DeMorgan's Theorem to remove the brackets. This results in

$$AB + AC + C' + D' + A'B'$$

## Karnaugh Maps

A Karnaugh map is a special arrangement of a truth table which can be used to simplify boolean algebra equations. It is important to recognise the order in which the terms are written in the grid.

The Karnaugh map to the right is a two input karnaugh map. This has two inputs  $A$  and  $B$ .

$B \backslash A$	0	1
0		
1		

The Karnaugh map to the right is a three input karnaugh map. This has three inputs  $A$ ,  $B$  and  $C$ .

Note the order of the inputs on the top side.

$C \backslash BA$	00	01	11	10
0				
1				

The Karnaugh map to the right is a four input karnaugh map. This has four inputs  $A$ ,  $B$ ,  $C$  and  $D$ . Note the order of the inputs on the top side.

$\begin{smallmatrix} B \\ A \end{smallmatrix}$	00	01	11	10
$\begin{smallmatrix} D \\ C \end{smallmatrix}$				
00				
01				
11				
10				

The orientation that the karnaugh map is drawn in and the order that the letters are put on the karnaugh map do not matter. What is important is that the numbers along the side only change by one digit between each cell.

It is possible to have more than 4 inputs to a Karnaugh Map however this then becomes three-dimensional. This is not covered in this module.

### Simplification using Karnaugh Maps

*Simplify the following expression.*

$$AB + A(B + C) + B(B + C)$$

The first step is to get the equation into SOP form.

$$AB + AC + B + BC$$

Now we can take each term one by one and find where that fits into the Karnaugh Map.

We are dealing with a three input expression so we need to use a three input Karnaugh map.

$\begin{smallmatrix} B \\ A \end{smallmatrix}$	00	01	11	10
$\begin{smallmatrix} C \end{smallmatrix}$				
0				
1				

Now we take  $AB$  where  $A = 1$  and  $B = 1$ . On the Karnaugh map, we mark a 1 where this is true.

$\begin{smallmatrix} B \\ A \end{smallmatrix}$	00	01	11	10
$\begin{smallmatrix} C \end{smallmatrix}$				
0			1	
1			1	

Now we take  $AC$  where  $A = 1$  and  $C = 1$ . On the Karnaugh map, we mark a 1 where this is true.

$\begin{smallmatrix} C \\ BA \end{smallmatrix}$	00	01	11	10
0			1	
1		1	1	

Now we take  $B$  where  $B = 1$ . On the Karnaugh map, we mark a 1 where this is true.

$\begin{smallmatrix} C \\ BA \end{smallmatrix}$	00	01	11	10
0			1	1
1		1	1	1

Now we take  $BC$  where  $B = 1$  and  $C = 1$ . On the Karnaugh map, we mark a 1 where this is true.

$\begin{smallmatrix} C \\ BA \end{smallmatrix}$	00	01	11	10
0			1	1
1		1	1	1

Finally, we need to group the 1s according to the grouping rules (shown below)

$\begin{smallmatrix} C \\ BA \end{smallmatrix}$	00	01	11	10
0			1	1
1		1	1	1

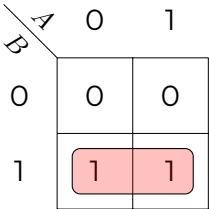
Finally, we pull out the groupings which gives us the answer.

$$B + AC$$

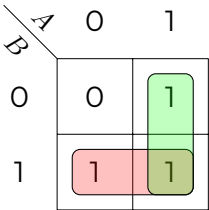
## Karnaugh Map Groupings

There are a number of rules which govern the groupings of Karnaugh Map digits.

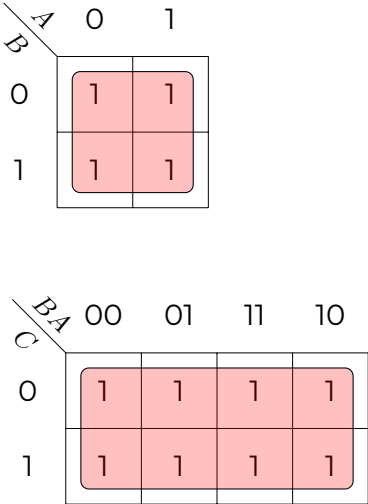
Groups must only contain 0s



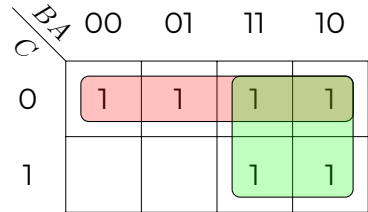
Groups may be horizontal or vertical but not diagonal



Groups must contain 1, 2, 4, 8 ( $2^n$ ) cells.



Groups should be as large as possible



Groups can overlap. Where there is overlap, each group must have one unique 1 to that group

BA \ C	00	01	11	10
0	1	1	1	1
1		1	1	

Groups can wrap around the table.

BA \ C	00	01	11	10
0	1			1
1	1			1

Groups can wrap around the entire table on multiple axis

DC \ BA	00	01	11	10
00	1			1
01				
11				
10	1			1

There should be as few groups as possible, as long as this doesn't contradict any of the previous rules

BA \ C	00	01	11	10
0	1	1	1	1
1	1	1		1

These Karnaugh Map grouping rules can be summarised as:

1. No groups containing 0s are allowed
2. No diagonal grouping
3. Group sizes should only be powers of 2 (1, 2, 4, 8, 16)
4. Groups should be as large as possible
5. All the ones must be in at least one group
6. Overlapping is allowed



7. Wrap around is allowed
8. Least possible number of groups is preferable

### Simplify Complex Boolean Algebra Expression using Karnaugh Maps

Simplify the following expression using Karnaugh Maps.


$$ABC'D' + ABC'D + AB'C'D' + AB'C'D + A'B'CD + A'B'CD' + A'BCD'$$


BA \ DC	00	01	11	10
00		1	1	
01	1			1
11	1			
10		1	1	

$$= D'CA' + C'A + CB'A'$$

## Page 13

# Exam Preperation

 12-12-22

 17:00

 Fazad

 PO 1.74

### Information about Exam

- Exam is on 13th January 1t 9am and will last one hour
- It takes place in a number of different computer labs across university buildings
- It will cover all content taught so far
- A good revision tactic would be to go through all the worksheets and practice sheets given out so far
- There is a mock test available on Moodle
- There are about 21 questions in the exam
- A very basic calculator is permitted, however nothing scientific
- We will be given a formula sheet (containing the boolean algebra simplification rules) and paper for working out.

## Page 14

# Computer Structure and Function

📅 23-01-23

🕒 16:00

🎓 Farzad

📍 RB LT1

*We have now finished content relating to Binary and Boolean Algebra. Until Easter (approx 5 weeks) we will be covering the CPU and how it works then until the end of the year, we will look at operating systems and how they work.*

## Computers as Complex Systems

A computer is a complex system. There are two key concepts to understand when referring to how a computer works, computer architecture and computer organisation.

### Computer Architecture

This is the attributes that have a direct impact on the logical execution of a program.

### Computer Organisation

This refers to the operational units and their interconnections that realise the architectural operation.

The difference between these two concepts can be described with the analogy of a car: the architecture of the car will always be the same (cars always need an engine and wheels) however precise organisation (implementation) of the architecture will vary from model to model (some cars may have electric engines and some may have petrol).

This same analogy can be applied to computers: all computers require the same architecture to work however the precise organisation varies from machine to machine (historically vacuum tubes, now we use transistors).

### Computer Structure

The way in which the components are interrelated.

### Computer Function

The operation of each individual component as part of the structure.

## Computer Function

There are four main functions of the computer.

- Data Processing
- Data Storage
- Data Movement

- Control

Data processing relates to the manipulation of data for a specific function.

Data storage has two different types, long term and short term. Long term is the Hard Drives or SSDs where data can be stored for a long period of time at relatively low costs. Short term is the RAM within a computer where data which is currently or soon-to-be needed by the CPU is stored, this is at higher cost than the long term storage. The storage of data requires movement of data from one place to another to store it.

Data Movement concerns data entering or leaving a system. This will most probably be coming from an input device (keyboard, mouse or microphone) and will most probably be going to an output device (monitor, speakers or printer). Data movement also concerns peripherals and communication from device to device.

Control is required with everything as computers must coordinate all their actions.

## Computer Structure

Within a computer, there are a number of interrelated components.

- Peripherals
- Communication Lines
- Storage
- Processing

There are also a number of key components within a computer.

- Central Processing Unit (CPU) - controls everything in the computer
- Main Memory - short term memory
- I/O - input & output devices
- System Interconnection (buses) - communicate between the main memory, I/O and CPU

## Central Processing Unit

The CPU has a number of components within it.

The *Arithmetic & Logic Unit (ALU)* is an adder/subtractor and performs the arithmetic and logical operations which we learnt about in Teaching Block 1.

The *Registers* are very small amounts of extremely fast memory. Register memory is very expensive. Registers are used to hold the data which is going to/from main memory.

The *Internal Bus* is the internal communications line within the CPU.

The *Control Unit (CU)* is the 'heart' of the CPU. It signals to other components of the CPU to keep them in sync. The CU contains sequencing logic, CU registers and decoders, and control memory; these are beyond the scope of this course.

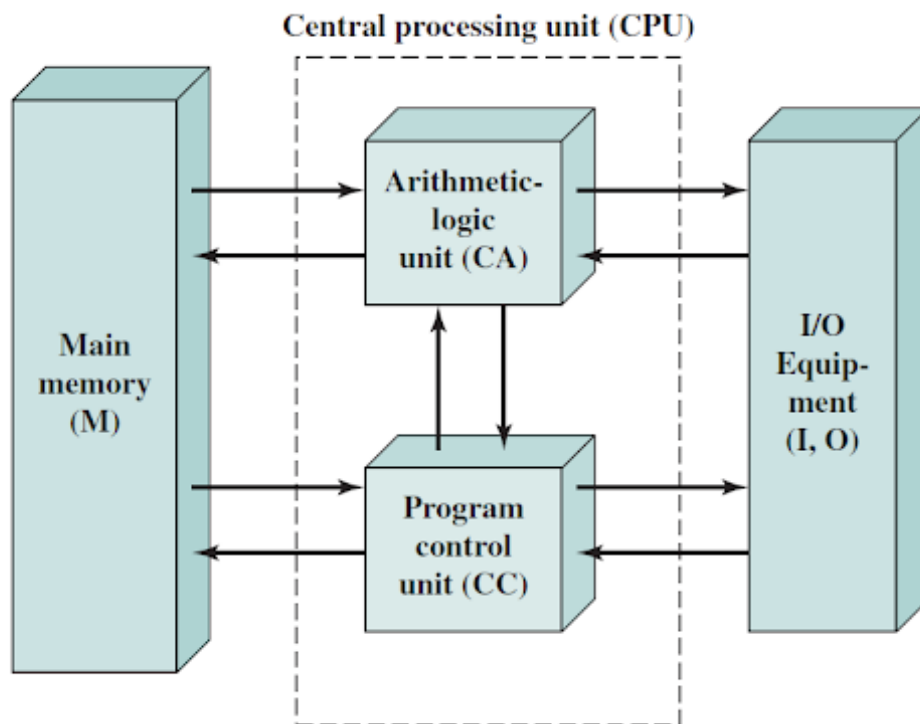
## History Of Computers

### First Generation

The first generation of computers used vacuum tubes. The first computer (ENIAC, Electronic Numerical Integrator and Computer) was created during WWII out of need for automating some of the missile launching procedures at the University Of Pennsylvania. It was manually programmed using decimal (not binary) and could do 5000 additions per second. It weighed 30 tones, took up 1500 square feet and contained 18000 vacuum tubes.

## Next Generation

The next generation of computers were designed between 1946 and 1952 by Von Neumann. This machine now incorporates the 'stored-program concept'. This computer was called an Institute of Advanced Study (IAS) Computer.



Institute of Advanced Study Computer

Memory within the IAS computer had also been re-designed since ENIAC. The memory now had 1000 locations, each 1 word (40 binary bits) long. This could either be used in Number word where the MSB acted as a sign bit or Instruction Mode where there were two instructions per word, each comprised of an 8-bit opcode (what the instruction is) and a 12 bit address (memory location of data to be used in the operation).

## Page 15

# WORKSHEET 10 Computer Structure and Function

📅 27-01-23

👁 Worksheet

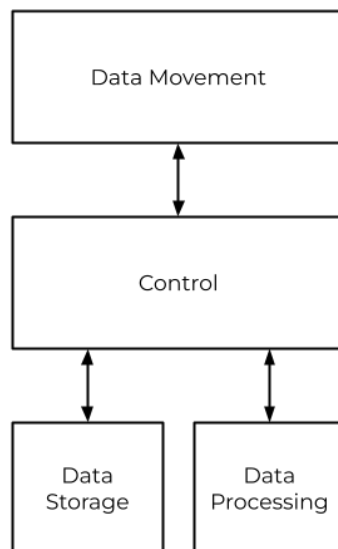
1. **What is Computer Architecture? What is Computer Organisation? What are the main differences between them, explain them using examples?**

Computer architecture is the attributes of the computer that have a direct impact on the logical execution of a program (for example there will need to be some form of memory and some form of addition unit). Computer organisation is the operational units and their interconnections (for example the vacuum tubes or transistors).

2. **What are the four main functions of a computer?**

Data processing, data storage, data movement, control.

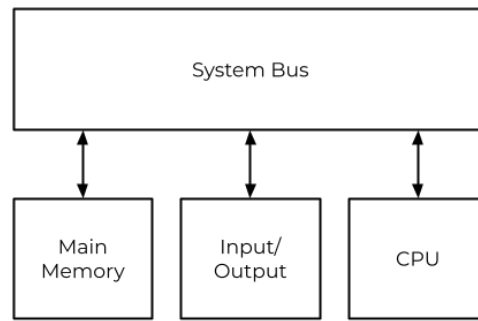
3. **Explain the functional view of a computer with the help of a flow chart.**



Functional view of a computer

In the above diagram, the arrows represent the movement of data throughout the blocks. The data moves from the 'movement block' where it interfaces with external peripherals. The data moves through the 'control block' where it can either go to the 'data storage' or 'data processing' blocks.

4. **Explain with a simple diagram the internal structure of the computer.** The diagram below shows the internal structure of a computer. The main memory, input and output devices, and CPU are connected together by the 'system bus'. Through this, they can share data between them.



Internal structure of a computer

5. **List and define the main structural components of a processor.**

The CPU has a number of components within it.

The *Arithmetic & Logic Unit (ALU)* is an adder/subtractor and performs the arithmetic and logical operations which we learnt about in Teaching Block 1.

The *Registers* are very small amounts of extremely fast memory. Register memory is very expensive. Registers are used to hold the data which is going to/ from main memory.

The *Internal Bus* is the internal communications line within the CPU.

The *Control Unit (CU)* is the 'heart' of the CPU. It signals to other components of the CPU to keep them in sync. The CU contains sequencing logic, CU registers and decoders, and control memory; these are beyond the scope of this course.

6. **What was the main technology used in the first generation of computers? Describe its technology.**

The first generation of computers used vacuum tubes. Vacuum tubes work by heating up a filament which releases electrons (negatively charged). These move towards the positively charged plate (anode) which causes a current to flow.

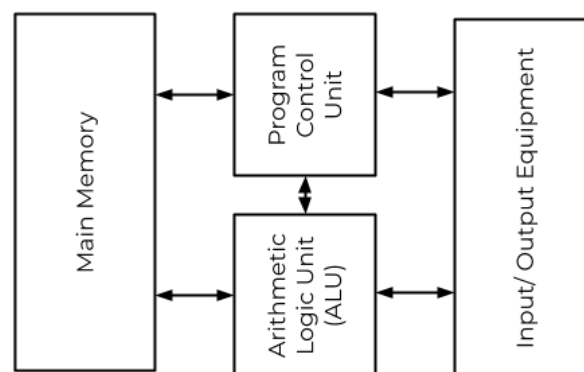
7. **What is ENIAC and explain its working process(e.g. accumulators)?**

ENIAC was manually programmed and worked using decimal, not binary.

8. **What is a stored-program concept?**

The stored program concept is where the program contents is stored in main memory then during run-time it is fetched, executed and decoded.

9. **Explain with a simple diagram the von Neumann machine architecture.**



Von Neumann architecture

**10. What is the difference between ALU and control units?**

The ALU (Arithmetic Logic Unit) is where the arithmetic and logical operations are performed whereas the Control Unit is the part of the CPU which synchronises all the other operations within the CPU.

**11. Explain IAS memory structure.**

Memory has 1000 locations, each 1 word (40 bits) long. Could either store one number (where MSB is sign bit) or two instructions (each 20 bits long).

**12. What are the differences between Number and Instruction words in IAS computers?**

Number words are 40 bits long (1 word) however instructions are 20 bits long (1/2 word). This means two instructions can fit into one word.

**13. Why are two different kinds of words used in von Neumann architecture?**

We need to store instructions and numbers in the same memory.

**14. What are Opcodes and addresses in instruction words?**

Opcodes (8-bits) are the operation which is to be performed; for example, add, subtract. Addresses (12-bits) is the address which the opcode is to be performed on.

**15. What are the advantages and disadvantages of von Neumann architecture?**

As there is a shared instructions and data memory, you can only access one or the other at once which decreases performance. As the memory is shared, both the instructions and data have to be the same size.



## Page 16

# IAS Computer

📅 2023-01-30

🕒 16:00

🎓 Farzad

📍 RB LT1

## IAS Computer

The IAS Computer uses the stored program concept. This is where the instructions and data required to execute the program are stored in the computer's main memory then during runtime moved to the CPU where each instruction is executed one by one. Further details on the IAS computer's structure can be found in the previous lecture. There are three key concepts of the IAS Computer

- Single read-write memory (data and instructions both stored here)
- Memory is addressable
- Execution occurs in a sequential fashion (unless explicitly modified)

## Registers

Registers are very small amounts of very fast and expensive memory which is found within the CPU. There are a number of registers and each have a specific purpose.

**Memory Buffer Register (MBR)** contains a word to be stored in memory or sent to the input/output unit; or to be received from either.

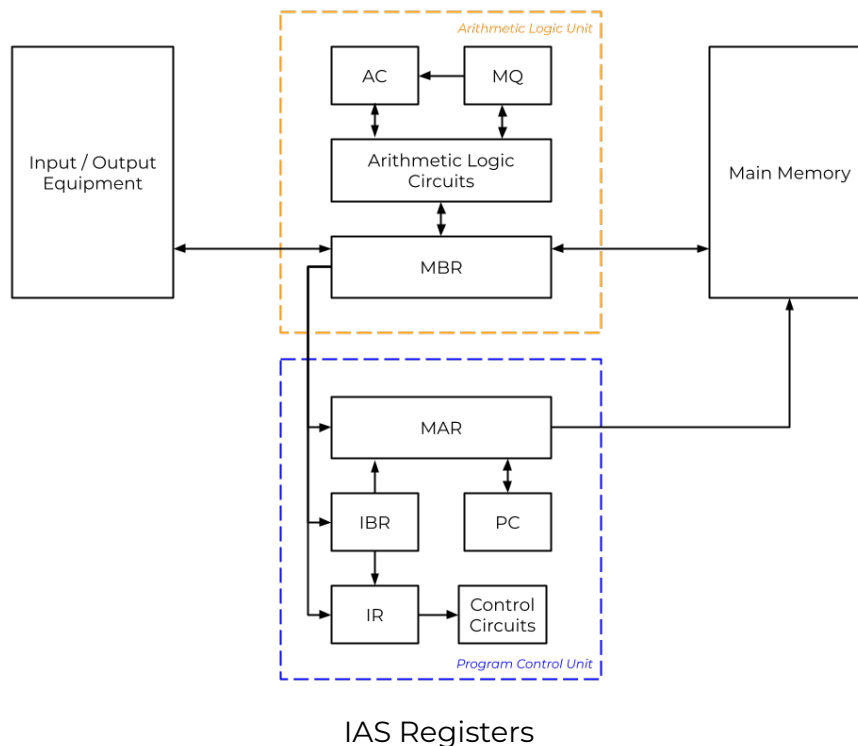
**Memory Address Register (MAR)** contains the address in memory of the word to be written to or read into the MBR.

**Instruction Register (IR)** contains the 8-bit opcode of the instruction currently being executed.

**Instruction Buffer Register (IBR)** Temporarily holds the right hand instruction from a word in memory while the left hand instruction is being executed.

**Program Counter (PC)** holds the address of the next instruction pair to be fetched from memory.

**Accumulator (AC) and Multiplier Quotient (MQ)** is used to hold temporary operands and results from the ALU. The MQ will be used if the result from the ALU is too big to fit in the AC.



## Instructions

The IAS Computer had a very limited set of instructions.

Opcode	Symbolic Representation	Description
<b>Data Transfer</b>		
00001010	LOAD MQ	Transfer the contents of register MQ to the accumulator AC
00001001	LOAD MQ,M(X)	Transfer the contents of memory location X to MQ
00100001	STOR M(X)	Transfer contents of accumulator to memory location X
00000001	LOAD M(X)	Transfer M(X) to the accumulator
00000010	LOAD-M(X)	Transfer -M(X) to the accumulator
00000011	LOAD  M(X)	Transfer absolute value of M(X) to the accumulator
00000100	LOAD - M(X)	Transfer - M(X)  to the accumulator

## Unconditional Branch

00001101	JUMP $M(X, 0:19)$	Take next instruction from left half of $M(X)$
00001110	JUMP $M(X, 20:39)$	Take next instruction from right half of $M(X)$

## Conditional Branch

00001111	JUMP+ $M(X, 0:19)$	If number in the accumulator is non-negative, take next instruction from left half of $M(X)$
00010000	JUMP+ $M(X, 20:39)$	If number in the accumulator is non-negative, take the next instruction from the right half of $M(X)$

## Arithmetic

00000101	ADD $M(X)$	Add $M(X)$ to AC; put result in AC
00000111	ADD $ M(X) $	Add $ M(X) $ to AC; put result in AC
00000110	SUB $M(X)$	Subtract $M(X)$ from AC; put result in AC
00001000	SUB $ M(X) $	Subtract $ M(X) $ from AC; put the remainder in AC
00001011	MUL $M(X)$	Multiply $M(X)$ by MQ; put MSBs in AC, put LSBs in MQ
00001100	DIV $M(X)$	Divide AC by $M(X)$ ; put quotient in MQ and remainder in AC
00010100	LSH	Multiply AC by 2 (left shift by 1 position)
00010101	RSH	Divide AC by 2 (right shift by 1 position)

## Address Modify

---

00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

---

## Programs

There are two different methods of programming computers, that we need to know about at this stage.

### Hardwired Programs

This is like what we did in `logic.ly`. This is where the individual logic gates inputs are hand-manipulated and the connections are put together by hand to generate outputs.

### Software Programming

This works by the code being written in a language which then gets passed through an interpreter. The interpreter will translate the high level language into a low level language which the components of the computer are able to understand and use.

## Page 17

# WORKSHEET 11 IAS Computer

📅 2023-02-03

👁 Worksheet

1. **What are the three key concepts of Von Neumann architecture?**

There is a single read-write memory; the memory is addressable; and execution of instructions occurs in a sequential fashion unless explicitly modified.

2. **What is a stored-program computer?**

A stored program computer is one where the instructions which get executed are stored within memory then loaded into the Central Processing Unit (CPU) one-by-one at run-time.

3. **What are the four main components of any general purpose computer?**

Peripherals; communication lines; storage and processing

4. **Why are registers used in CPU?**

Registers are used within the CPU to hold data which is being transmitted from one component of the CPU to another. They have a very small capacity and transmit data at very high rates.

5. **What is the overall function of a processor's control unit?**

The control unit synchronises actions within the CPU and signals to components of the CPU if they need to enable/ disable a flag. This is used, for example, to signal to the ALU that the next operation it is performing is a subtraction.

6. **Explain the use of the following registers**

(a) **Memory Buffer Register (MBR)**

Contains the word which is being sent to/ received from main memory; or being sent to/ received from input & output devices.

(b) **Memory Address Register (MAR)**

Contains the address in memory of the word to be written to or read into the MBR.

(c) **Instruction Register (IR)**

Contains the opcode (8-bits) of the instruction currently being executed.

(d) **Instruction Buffer Register (IBR)**

Holds the right hand instruction while the left hand instruction from the same word is being executed.

(e) **Program Counter (PC)**

Holds the address in memory of the next instruction pair to be fetched.

(f) **Accumulator (AC) & Multiplier Quotient (MQ)**

AC holds temporary operands and results from the ALU (*Arithmetic & Logic Unit*). MQ is used if the result from the ALU is too big to fit in the AC.

7. **What are the main two phases of instruction execution**

Fetch & Execute

**8. Explain the Fetch cycle and the registers that are used in that?**

The program counter holds the address of the next instruction. Copy the contents of the PC to the MAR. Increment PC. Load instruction pointed to by the MAR to the MBR.

**9. Explain the Execute cycle and the registers that are used in that?**

Fetch data from the address pointed to by the operand or perform the function stated in the opcode.

**10. What is a hardwired program?**

A program where the individual logic gates and circuitry are manipulated to give the desired inputs.

**11. Explain the idea of programming in software**

Programming in software works by writing the code in a high level language (for example, C, Python, etc). This code is then translated to machine code which can be processed by the hardware.

**12. What are the differences between programming in hardware and software?**

Programming in hardware directly manipulates the electrical circuitry whereas programming in software involves additional signals before the electrical circuits can be manipulated.

**13. What do control signals do?**

Control signals are used to signal to different components within the CPU to instruct them to perform their actions and to synchronise them.

## Page 18

# Fetch Execute Cycle

📅 2023-02-06

🕒 16:00

🎓 Farzad

📍 RB LT1

## Computer Function

One of a computer's main functions is to execute a program (this is a pre-written set of instructions). The processors execute the instructions in two stages, the first stage is to 'fetch' the instruction from memory and the second stage is to execute the instruction. Program execution is these two stages repeated until a `halt` command is executed.

## Fetch & Execute Example

This example will perform the operation  $2 + 3$  then store the result in memory. This uses a single data register, the memory has 16-bit words of which the opcodes are 4-bits. As the Opcodes are 4 bits in length, there are 16 possible opcodes ( $2^4$ ). At the start of the example, the program counter is set to 300.

### More detailed example

The example shown below is available as a step-by-step walk through through the slides linked on Moodle.

### Step 0

The data stored in memory is represented as hexadecimal as we can store one nibble with one character.

CPU Registers					Main Memory				
PC	3	0	0	0	300	1	9	4	0
MAR					301	5	9	4	1
MBR					302	2	9	4	1
IR					...	...	...	...	...
AC					940	0	0	0	2
					941	0	0	0	3

### Step 1 - Fetch I

1. The contents of the PC is copied to the MAR
2. The contents of the memory address stored in the MAR is copied into the MBR
3. The contents of the MBR is copied into the IR

CPU Registers				
PC	3	0	0	0
MAR	3	0	0	0
MBR	1	9	4	0
IR	1	9	4	0
AC				

Main Memory				
300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	2

## Step 2 - Execute I

1. The PC is incremented
2. The opcode is analysed and the instruction is performed. In this case, the opcode is 1 which means `load`. The address is 940 so the contents of memory address 940 is copied into the AC.

CPU Registers				
PC	3	0	0	1
MAR	3	0	0	0
MBR	1	9	4	0
IR	1	9	4	0
AC	0	0	0	3

Main Memory				
300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	2

## Step 3 - Fetch II

1. The contents of the PC is copied to the MAR
2. The contents of the memory address stored in the MAR is copied into the MBR
3. The contents of the MBR is copied into the IR

CPU Registers				
PC	3	0	0	1
MAR	3	0	0	1
MBR	5	9	4	1
IR	5	9	4	1
AC	0	0	0	3

Main Memory				
300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	2



**Step 4 - Execute II**

1. The PC is incremented
2. The opcode is analysed and the instruction is performed. In this case, the opcode is 5 which means `add to AC from memory`. The address is 941 so the contents of memory address 941 is added to the contents of the AC.

CPU Registers				
PC	3	0	0	2
MAR	3	0	0	1
MBR	5	9	4	1
IR	5	9	4	1
AC	0	0	0	5

Main Memory				
300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	2

**Step 5 - Fetch III**

1. The contents of the PC is copied to the MAR
2. The contents of the memory address stored in the MAR is copied into the MBR
3. The contents of the MBR is copied into the IR

CPU Registers				
PC	3	0	0	2
MAR	3	0	0	2
MBR	2	9	4	1
IR	2	9	4	1
AC	0	0	0	5

Main Memory				
300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	2

**Step 6 - Execute III**

1. The PC is incremented
2. The opcode is analysed and the instruction is performed. In this case, the opcode is 2 which means `store to AC to memory`. The address is 941 so the contents of the AC is written to memory address 941.

CPU Registers					Main Memory				
PC	3	0	0	3	300	1	9	4	0
MAR	3	0	0	2	301	5	9	4	1
MBR	2	9	4	1	302	2	9	4	1
IR	2	9	4	1	...	...	...	...	...
AC	0	0	0	5	940	0	0	0	3
					941	0	0	0	5

This cycle will then continue on to the next instruction and so-on.

## Page 19

# Interconnections

📅 2023-02-13

🕒 16:00

🎓 Farzad

📍 RB LT1

## Interconnection Structure

Computers are a network of basic modules, connected together by paths (such as the system bus). The structure of these paths depends on the exchanges taking place.

### Memory

Within memory there are n-words, each of equal length. Each word has a unique numerical address. The operation which the memory is currently performing is controlled by read and write control signals. The location for the operation is specified by an address. Data can be written to or read from memory.

### Input/ Output Module

The Input/ Output (I/O) Module is similar to memory from an internal point of view, in that data can be read-from and written-to it. The I/O module can control more than one external device (each port has a unique address which can be used to identify an individual device). The I/O module has internal data input and output ports, this is for data coming from/ going to other places inside the computer; it also has external data in and out ports, which are used for interfacing with an external device. The I/O module also has the ability to send interrupt signals to the CPU which signals to the CPU to stop doing what it was doing as something more important has happened (for example, printer has run out of paper) which it needs to deal with.

### Processor

The processor reads instructions and data and writes out data after processing. It uses control signals to control the overall operations. The processor also receives interrupt signals.

### Types of Transfer

**Memory to Processor** The processor reads an instruction or unit of data from memory.

**Processor to Memory** The processor writes a unit of data to memory.

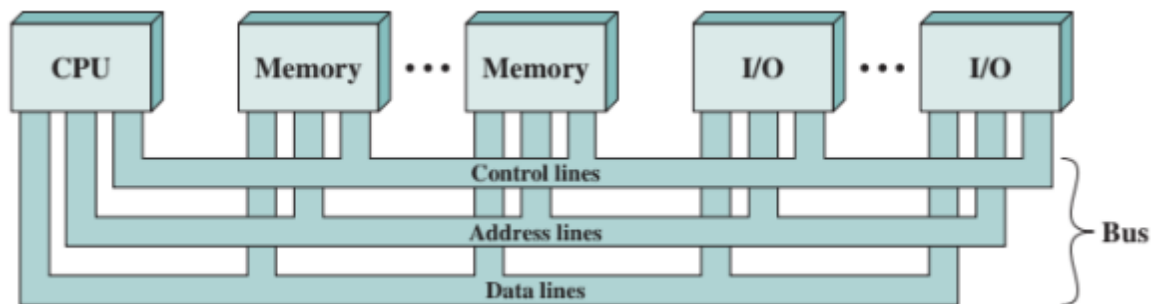
**I/O to Processor** The processor reads data from an I/O device via an I/O module.

**Processor to I/O** The processor sends data to the I/O device.

**I/O to or from Memory** I/O module is allowed to exchange data directly with memory, without going through the processor using direct memory access.

## Bus Interconnection

Bus interconnections are not used much in modern systems as they are a shared transmission medium which means there is only a single route which all data must travel on. Multiple devices connect to the bus and a signal transmitted by one device is available for reception by all other devices attached to the bus. If signals overlap, they become garbled therefore only one device at a time can successfully transmit. A bus consists of multiple communication pathways (called lines); each transmits a single bit at a time which means if we want to transmit 8-bits simultaneously, we need a bus with a bus-width of 8.



Bus interconnections

### Data Lines

The data lines provide a path for moving data among system modules. The width of the data bus is a key factor in determining overall system performance. For example, if the data bus is 32 bits wide and the instruction to be transmitted is 64-bits then you need to use the data bus twice whereas if the data bus was 64-bits wide then you would only need to use it once.

### Address Lines

The address bus carries the source/ destination of the data which is being transmitted on the data bus. The bus width determines the maximum possible memory capacity. The high order bits select the module and the low order bits select the memory location or I/O port.

### Control Lines

Control lines control the access to and use of the data and address lines because the data and address lines are shared by all components and there must be a means to control their use. Control signals are transmitted on control lines, these include command signals which specify the operation to be performed and timing signals which confirm the validity of data and address information. Control lines include memory write & read, I/O write & read etc.

### Bus Operation

If one module wishes to send data to another, it must first obtain use of the bus then it can transfer the data via the bus.

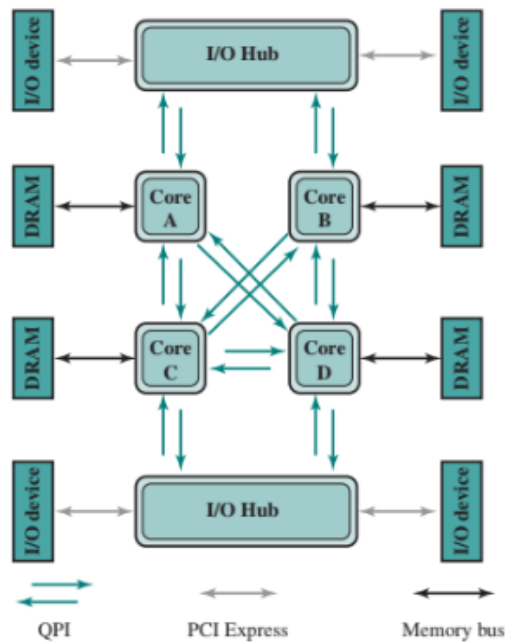
If one module wishes to request data from another module, it must first obtain the use of the bus. Then it can transfer a request to the other module over the appropriate control and address lines.

## Point-To-Point Interconnection

Contemporary systems increasingly rely on point-to-point interconnections. This removes the bus system problems as well as lowering the latency, increasing the data rate and improving the scalability.

### QuickPath Interconnect

QuickPath Interconnect (QPI) was released in 2008. It includes multiple direct connections, a layered protocol architecture and packetized data transfer.



Point-To-Point Interconnection diagram showing QuickPath

## Page 20

# WORKSHEET 13 Interconnections

📅 2023-02-17

👁 Worksheet

1. **Why do computers need interconnection systems?**  
Computers need interconnection systems so that data and information can be transmitted between the different components which comprise the computer.
2. **What does the interconnection structure depend on?**  
The exchanges taking place.
3. **What are the types of exchanges from/to the main memory?**  
The processor can read and write data/ instructions to the main memory and the IO can read and write data to the main memory.
4. **What are the types of exchanges from/to the I/O module?**  
The I/O module can send and receive data from the main memory and send data to and receive data from the processor.
5. **What are the types of exchanges from/to the CPU?**  
The CPU can send and receive data to/ from the main memory and I/O module.
6. **What types of transfer does the interconnection structure support?**  
Bus interconnection supports transfers between the CPU, main memory and I/O module which travels through all of them. The Point-to-Point interconnection supports transfer between the same components but directly between them.
7. **What is a bus in computer systems? What are its key characteristics?**  
A bus is an interconnection line between modules. Its key characteristic is its width which determines how many bits can be transmitted in parallel.
8. **What does shared transmission medium mean?**  
Multiple data items can be transmitted down the medium at once.
9. **How do the signals overlap during transmission?**  
If two signals become overlapped then they become garbled and corrupt.
10. **What is a line in computer systems? What does it transmit?**  
A line is the name of the physical connection between components.
11. **How many lines does a bus need to transmit 2 Bytes of data?**  
16
12. **What is a system bus?**  
Collective name for the address, control and data buses.
13. **What are the different types of bus lines?**  
Address, control and data.

**14. What do the data lines do? What is a data bus?**

The data lines provide a path for moving data among system modules. The width of the data bus is a key factor in determining overall system performance. For example, if the data bus is 32 bits wide and the instruction to be transmitted is 64-bits then you need to use the data bus twice whereas if the data bus was 64-bits wide then you would only need to use it once.

**15. What do the address lines do? What is an address bus?**

The address bus carries the source/ destination of the data which is being transmitted on the data bus. The bus width determines the maximum possible memory capacity. The high order bits select the module and the low order bits select the memory location or I/O port.

**16. What do the control lines do? What is a control bus?**

Control lines control the access to and use of the data and address lines because the data and address lines are shared by all components and there must be a means to control their use. Control signals are transmitted on control lines, these include command signals which specify the operation to be performed and timing signals which confirm the validity of data and address information. Control lines include memory write & read, I/O write & read etc.

**17. How does the address bus determine the maximum possible memory capacity?**

Its width is the maximum size of the numerical value assigned to an address.

**18. How are different I/O ports addressed in computers?**

High order bits in the address lines.

**19. How does a module (memory, I/O,... ) send data to another module in computers?**

If one module wishes to send data to another, it must first obtain use of the bus then it can transfer the data via the bus.

**20. How does a module (memory, I/O,... ) request data from another module in computers?**

If one module wishes to request data from another module, it must first obtain the use of the bus. Then it can transfer a request to the other module over the appropriate control and address lines.

**21. What is a point to point interconnection system?**

A system where all the different components connect to each other. This means that a data transfer can go directly to where the data is intended for rather than having to go 'the long way around'.

**22. What are the main advantages of using a point to point interconnection system?**

The data being transferred can go directly to where the data is intended for rather than having to go the 'long way around'.

**23. What are the significant characteristics of QuickPath Interconnect (QPI)?**

Multiple direct connections; layered protocol architecture; packetized data transfer.

## Page 21

# Computer Memory Systems

📅 2023-02-20

🕒 16:00

🎓 Farzad

📍 RB LT1

Computer memory is the part of the computer where the computer stores (remembers) data and instructions (stored as binary values). Memory stores the information temporarily or permanently and provides the CPU with data and instructions. There are a number of different types of memory, each of which have a different purpose and can be found in different parts of the computer.

## Memory Key Characteristics

### Location

Memory can be *internal*, which means it is inside the core of the machine. This may include processor registers, cache memory, or main memory. Memory can also be *external*, which means it is not in the core of the machine, or even outside the machine all together. External memory can include optical disks, magnetic disks, tapes, or USB drives.

### Capacity

The capacity of the memory concerns how much data can be stored within it. This may be measured in *words*, *bytes* or *bits*.

### Unit of Transfer

The unit of transfer is the number of bits read out of or written into memory at a time. *Bits* are generally moved in main memory, *words* are used in registers, and *blocks* are used when moving from one memory location to the other. A *block* is multiple words together.

### Access Method

There are a number of different access methods which can be used to access data from memory. *Sequential Access* starts reading from the start of the memory unit and continues to read until it finds the location which has been requested, this is very slow and is used for tape units. *Direct Access* moves directly to the address which needs to be read and can read it directly, this is generally used for disk units. *Random Access*, when used in block mode, will move the requested words as well as some from either side of the target word, this increases the efficiency and is often found in main memory and cache systems. *Associative Access* is a hybrid between random access and direct access, it searches through the entire memory and returns all words with a similar contents to what you requested, this is generally found in cache systems.

### Performance

The *Access Time* is the time it takes to access the memory - When this is high, the machine will be slow. The *Cycle Time* is the time which it takes for one cycle of the Fetch-Decode-



Execute Cycle. The *Transfer Rate* is calculated by dividing 1 by the cycle time; and is the frequency of memory accesses.

## Physical Type

*Semiconductor* memory uses different types of material to create polarised areas which can be used to represent 1s and 0s. *Magnetic* uses magnetic technology, which can be polarised to create 1s and 0s. *Optical* uses lasers to read/ write data in the form of pits and lands. *Magneto-optical* is a hybrid of magnetic and optical.

## Physical Characteristics

Memory can either be *volatile* (when the power cuts, memory empties) or *non-volatile* (when power cuts, memory contents stays). Memory can also be *erasable* (where it can be completely emptied, for example RAM) or *non-erasable* (where it cannot be completely emptied, for example ROM).

## Hierarchy of Memory

As you move down the hierarchy, the cost decreases, the capacity increases, however the access time increases which decreases the frequency of access of the memory by the processor.

Location	Memory Type
Inboard Memory	Registers
	Cache
	Main Memory
Outboard Storage	Magnetic Disk
	CD-ROM/ CD-RW
	DVD-RW/ DVD-RAM
Off-line storage	Magnetic tape

Smaller, more expensive, faster memory is generally supplemented by larger, cheaper and slower memory.

## Cache Memory

The concept of cache memory is to combine fast and expensive memory with less expensive and lower speed memory. By doing this, you end up with the cache containing a copy of a portion of memory. It works by blocks of data transferring to & from main memory and the cache memory; this transfer is slow. Words of data are then transferred between the cache memory to & from the CPU, this is much faster.

There are three levels of cache memory. The speed of transfer is fastest between the CPU and level 1 cache and is slowest between the main memory and level 3 cache. Level 2 cache is in the middle.

## Page 22

# WORKSHEET 14 Computer Memory Systems

📅 2023-02-24

👁 Worksheet

1. **What is computer memory?**  
The part of the computer where the computer stores data and instructions, can be stored either temporarily or permanently.
2. **Why do computers need memory?**  
To be able to store data and instructions, either temporarily or permanently.
3. **How is memory capacity usually measured?**  
In terms of words, bits or bytes (or multiples of them, eg terabytes).
4. **What is the memory unit of transfer? What are the units?**  
Bits are used when referring to main memory, words are used in registers and blocks are used when moving from one memory location to another.
5. **What does memory access mean?**  
The computer retrieving data from or sending data to main memory.
6. **What is the sequential access method in memory? Give an example.**  
The memory starts being read from the first location and all locations are examined until the desired location is found, this is used in tape units.
7. **What is the direct access method in memory? Give an example.**  
The read/write head moves directly to the address which needs to be accessed. Disk units generally use this method.
8. **What is the random access method in memory? Give an example.**  
The memory will transfer the desired word as well as some words from either side of the desired word. This is often found in main memory or cache systems.
9. **What is the associative access method in memory? Give an example.**  
Hybrid between direct and random access. It searches through the entire memory and returns all the words similar to what you have requested. Often found in cache systems.
10. **How is memory performance measured?**  
Through a number of metrics: access time, cycle time and transfer rate.
11. **What is memory access time?**  
The time it takes to access the memory.
12. **What is the memory cycle time?**  
The time it takes for one cycle of the Fetch-Decode-Execute cycle to complete
13. **What is the transfer rate in memory?**  
The maximum frequency of memory access.

14. **What are different physical types of memory?**

Semiconductor, magnetic, optical and magneto-optical.

15. **What is a volatile memory? What is non-volatile memory?**

Memory is volatile when it empties when the power is cut. Memory is non-volatile when its contents stays when the power cuts.

16. **What is ROM? Is it volatile memory? Is it erasable?**

Read Only Memory (ROM) is non-volatile and is non-erasable.

17. **What are the relationships between cost, capacity, and access time in memory design?**

As cost decreases, the capacity increases however access time increases which decreases the frequency of access of the memory.

18. **What is a good computer memory system? How it should be in terms of a combination of different types of memory?**

Smaller amounts of more expensive, fast memory (eg registers, cache) are combined with larger, cheaper slower memory (eg magnetic disk).

19. **How do you explain memory hierarchy?**

Location	Memory Type
Inboard Memory	Registers
	Cache
	Main Memory
Outboard Storage	Magnetic Disk
	CD-ROM/ CD-RW
	DVD-RW/ DVD-RAM
Off-line storage	Magnetic tape

20. **What is cache memory? Why is it important?**

Small amounts of very fast memory which is used between the CPU and registers. This allows the cache memory to contain a copy of a portion of the memory which decreases access time of the CPU getting data.

21. **How does multi-level cache organisation work?**

There are three levels of cache memory, the fastest speed of transfer is between the CPU and Level 1 cache (which is the smallest); the slowest speed of transfer is between Level 3 cache and Main Memory (L3 cache is the biggest). Level 2 cache is found in the middle of L1 and L3.

22. **How does word and block transfer work in cache memory?**

Whole blocks of data are transferred from main memory to level 3 cache. Sub blocks are transferred onto level 2. Smaller sub-parts of blocks are transferred onto level 1 where words are then transferred onto the CPU as needed.

## Page 23

# WORKSHEET: Guest Lecture

📅 2023-03-03

👁 Worksheet

*NB: ChatGPT used for research as suggested.*

1. **Operating Systems have 'entry points' - the various fine-grained, one might almost call, 'atomic' operations upon which applications and drivers may call. Look-up the different calling conventions used to access them; e.g., pascal; cdecl; fast-call etc. Why are these different ways of access used; what are the pros/cons that are being mitigated for here?**

Calling conventions are sets of rules that determine how functions should be called and how parameters are passed between a caller and a callee in a computer program.

- Cdecl (C calling convention): This convention is used in the C programming language and is widely used on many platforms. In this convention, the caller is responsible for cleaning up the stack after a function call.
- Pascal calling convention: This convention is used in Pascal and Delphi programming languages. In this convention, the callee is responsible for cleaning up the stack after a function call.
- Fastcall calling convention: This convention is used on x86 processors and is designed to make function calls faster by passing some of the function's arguments in registers rather than on the stack.
- Stdcall calling convention: This convention is used in the Microsoft Windows operating system and is similar to the cdecl convention, except that the callee is responsible for cleaning up the stack after a function call.

2. **What are "Systems' Programming Languages"? Why are they so named; what are their attributes, pros and cons?**

Systems programming languages are programming languages that are designed for writing system-level software, such as operating systems, device drivers, and system utilities. These languages are typically low-level and provide direct access to system resources, such as memory, hardware, and input/output devices. They are so named because they are used to build and maintain the core systems of a computer or other computing device.

3. **Why use Assembler/Assembly language today, especially when many say that constraints on the availability of memory-usage is no longer an issue?**

Assembler/Assembly language is still used today for a few reasons, despite the availability of modern programming languages with higher-level abstractions and more user-friendly syntax: performance, control, legacy code and debugging.

4. **Why will there never be a 128-bit operating system? I mean, NEVER. Ok, you might create an 'OS' with 128-bit wide registers, but that would only be for fun. Right?**

It would be extremely unlikely that a 128-bit operating system will never exist. This is due to hardware limitations, diminishing returns, and complexity. It is not impossible to create a 128-bit OS, the limitations make it unlikely that it would ever be made.

5. **What's your favourite Dilbert/XKCD cartoon? You may elect to give one from the 'Far Side' if you wish.**
6. **What's your Slashdot ID number? (<https://slashdot.org/>) - You know 'News for Nerds, Stuff that Matters'. Your 'ID' says a lot. Really!**
7. **You're at Microsoft, interviewing folks (four scenarios) - and each is a critical hire. What do you ask? You can have two questions if you like the scenario you choose.**
  - **You're recruiting someone to work on the OS**
  - **You're recruiting someone to work on Excel**
  - **You're recruiting someone to work on UI/UX**
  - **You're recruiting a manager today (assume their background, but then expand upon it)**

OS Developer: what experience do you have developing operating systems?; how do you approach debugging complex issues in an operating systems?

Excel Developer: what experience do you have developing applications similar to Excel?; how would you approach designing and testing an application like Excel?

UI/UX: what design tools and methodologies do you use to create engaging and intuitive user interfaces?; Can you walk me through your design process from initial concept to final design, how do you include stakeholders and users in the process?

Manager: How do you approach managing a diverse team of individuals with different backgrounds and skill sets, what methods have you found to be effective in promoting collaboration and communication?; How do you balance the needs and requirements of different stakeholders?

8. **Edsger Dijkstra once said: "Computer science is no more about computers than astronomy is about telescopes." What he meant was that, in his view, Computer Science is about the 'Theory of Computation' - what a mere machine may achieve (within some stated constraints). What do you think of Dijkstra's view today? Justify.**

Computer science encompasses more than just the physical machines that we use. It encompasses a wide range of theoretical and practical topics including algorithms, programming languages, data structures and more. The study of computer science is ultimately about understanding what can and cannot be computed.

9. **In most operating systems, one can either link one's object code to library code either statically, or dynamically (linking to LIB files, or DLLs - to use Windows' terminology). What are the dis/advantages to either model?**

Statically linking object code to library code means that the library code is compiled and included in the final executable file. On the other hand, dynamically linking object code to library code means that the library code is loaded at runtime by the operating system, and multiple executable files can share the same library code. The advantages of static linking are: portability, stability, and performance. The advantages of dynamic linking are smaller executable size, memory usage, and flexibility.

10. **In the Hallmarks of the Portsmouth Graduate, which hallmarks are most aligned to those you think a <your computing related subject here> needs?**

1, 2, 3, 4, 5, 7, 9, 10.

11. **If you could write a book on some aspect of Computing; perhaps software development/principles; mathematical-underpinnings, what would be a few items listed in its Contents?**

Introduction to software development, programming fundamentals, testing and debugging, software development methodologies.

12. **Most programming languages are good at certain things, e.g., Prolog is excellent for Logic programming (goal seeking etc), whilst Scala and Functional languages may be used to write building blocks that are side-effect-free. What is your favourite language, explain why?**

Out of the Languages I've used - C# because of its power and versatility.

Unfinished.

## Page 24

# Operating Systems - Introduction

📅 2023-03-13

🕒 16:00

🎓 Farzad

📍 RB LT1

## Development of Operating Systems

### Late 1940s

There were no operating systems, the programmer was also the user. Programs and data were entered in binary through switching switches on the front of the machine (each switch represented one bit). Output was given through lights (each light represented one bit). The programmer did everything that an operating system does today.

### 1950s

Specialist operators were introduced. These people are not programmers. They tend to the machine, feed the programs to the machine and deliver back to the output.

Programmers use punch machines to encode their programmer as a series of holes in stiff cards. Computers can read and interpret the holes in the punch cards.

The operators acted as an human interface between the programmer and the hardware.

### 1960s

The need for a human (to load and unload punched cards, starting and stopping devices) added delay into the system.

The programmers now punched instructions onto control cards which were inserted at the appropriate places before and after the program cards and data cards

The commands were written in specially developed job control languages.

### Late 1960s and 1970s

Computer users found that programs were growing in size and required more memory. The initial solution was to break programs up into small chunks, each fitting in the available memory.

We now have larger memories, however we need to timeshare this between different programs.

The issue of protecting one program from another became more and more important. This is a task which the OS performs.

### 1980s

Computers now have the ability to communicate with one another.

### 1990s

Previously, operating systems have been developed specifically for particular hardware platforms (for example, MS-DOS for the PC).

This move allowed generic operating systems (for example Linux) to run on any hardware.

## What Is An Operating System?

The *Operating System* sits between the application and the hardware. It allows the application to interface with the hardware, so that there is only one thing interfacing with the hardware. The users interface with the applications, and at times the operating system.

Operating systems, at heart, are computer programs. They are written, tested and compiled just like any other program. Operating systems will begin running as soon as the computer has turned on, this usually happens automatically.

The operating system makes the hardware more useful and user friendly.

## What Does The Operating System Do?

Operating systems provide an environment which helps other programs do productive work; helps the user to develop & run programs, by providing a convenient environment; and starts and stops applications by sharing the CPU between them.

The operating system is also responsible for managing memory, this involves keeping track of which parts of memory are in use and which are free; and providing a mechanism by which applications can ask for more memory to be allocated to them or give back memory which they no longer need.

The operating system handles inputs and outputs; differences between hardware devices (so that all applications can interface with hardware the same); and controlling input/output and processing so that they can happen at the same time.

The operating system also provides data management, which involves managing the different physical drives and how to move data between them; protection to the CPU of overlapping processes; networking support through covering up differences between machines; and error handling & recovery which provides a way for the user to interface with errors.

## Operating System Interfaces

The operating system can be interfaced with in a number of ways. This includes through a Graphical User Interface - where icons and pictures are used to control what happens in the OS or through a Command Interpreter - where text commands are entered at a prompt which the operating system interprets.

Modern operating systems will come with both a GUI and command interpreter.

The GUI, command interpreter and application programs don't directly interface with the operating system. They all interface with the system services, which provide a set of functions which request services of the operating system.

## Kernel

The *kernel* is the central component (the heart) of an operating system. It interfaces between hardware components and software applications - this means it makes the software interact with the hardware to get a specific task done.

The kernel decides the amount of resources (RAM, GPU, etc) to be used by every application and the order of execution of programs. The kernel has a separate memory space so that its functions are independent (this is the bit of memory, both primary and secondary, which isn't listed as usable in OS).

The kernel is the central authority which guides memory and keeps an eye on all the hardware and software data flow. The kernel responds to system calls - which are calls where processes can demand resources.



## CPU Operation Modes

There are two modes of CPU operation, each of which have different limitations on instructions.

By default, machines run in user mode; when it wants to do something special, it has to change into kernel mode.

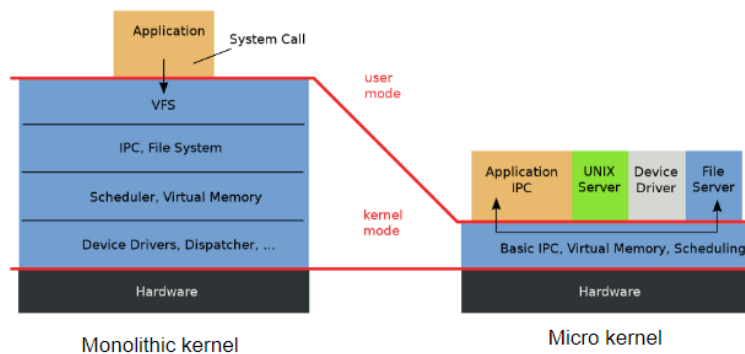
### User Mode

The CPU can only execute a subset of its instructions - the more common ones (add, subtract, load, store, etc). When a program is executed, it doesn't have access to memory, hardware and such resources.

### Kernel Mode

The CPU can execute all of its instructions, including extra privileged instructions. When a program is executed, it has direct access to memory, hardware and such resources.

## Types of Kernel



### Monolithic Kernel

Both the user services and kernel services are kept in the same address space (this means the entire OS is in the kernel space). It is larger than micro kernel, less access time and fast execution, hard to extend. It has higher performance, however higher risk of a systems crash.

Linux uses the monolithic kernel.

### Microkernel

The user services and kernel services are kept in separate address spaces. This means the kernel is smaller in size, there is minimum code in kernel space, greater access time with slow execution. It is easily extendible and has lower responsibility.

Windows uses a hybrid of the two types of kernel.

## Types of Operating System

Different types of operating system are required for different purposes.

### Batch Systems

These were the earliest systems developed. Data and commands to manipulate the program and data are all submitted together to the computer in the form of a *job*. There is little-to-no interaction between the users and an executing program.

These systems are used where there is no need for operator intervention once the job has started, for example in payroll processing or bank/ credit card statement generation.

## **Interactive Systems**

The most common mode of using a computer is through using a keyboard, mouse and screen.

Interactive systems are a significant improvement over *batch systems* as it is now possible to intervene directly while a program is being developed or as it is running. Single user interactive systems are available, these provide multitasking and interactive computing on a single-user basis such as Windows, MS-DOS and OS/2. Also, multi user interactive systems are available, these provide interactive computing on a multi-access or multi-user basis (using different terminals) such as Unix, Linux, Windows 10, Ubuntu, Mac OS.

## **General Purpose**

A given environment may want a bit of everything - for example a timesharing system may support interactive users and also include the ability to run programs in batch mode.

## **Network**

To share resources such as printers and databases across a network, such as Windows NT Server.

## **Distributed**

The most recent development in operating systems, this meets the requirements of a multi-user system. It consists of a group of machines acting together as one. When a user starts a program, it may run on the local machine. However if that computer is heavily loaded and the operating system knows that another machine is idle then the job may be transferred to that idle machine.

## **Specialist**

Dedicated to processing large volumes of data which are maintained in an organized way such as a real-time operating system would require for example a banking system.

## **Design of Operating Systems**

An operating system is a very large piece of software. To design, build and maintain large software systems like this requires a high-level view of how the system is structured and how its different components work together.