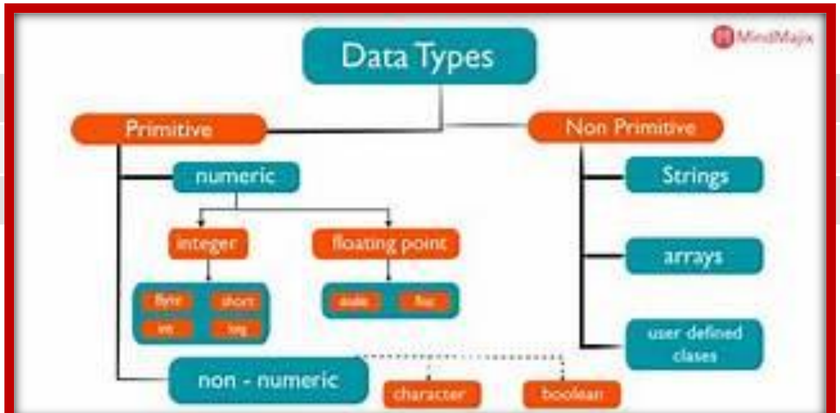
Specification & learning objectives

A Level	Specification point description
1.4.1a	Primitive data types, integer, real/floating point, character, string and Boolean
1.4.1b	Represent positive integers in binary
1.4.1c	Use of sign and magnitude and two's complement to represent negative numbers in binary
1.4.1d	Addition and subtraction of binary integers
1.4.1e	Represent positive integers in hexadecimal
1.4.1f	Convert positive integers between binary hexadecimal and denary
	Positive and negative real numbers using normalised floating-point representation
1.4.1g	Representation and normalisation of floating-point numbers in binary
1.4.1h	Floating point arithmetic, positive and negative numbers, addition and subtraction
1.4.1i	Bitwise manipulation and masks: shifts, combining with AND, OR, and XOR
1.4.1j	How character sets (ASCII and UNICODE) are used to represent text

Resources

PG Online textbook page ref: 155-177
Hodder textbook page ref: 136-141, 146-155
[CraignDave videos for SLR 13]



Key question: What is meant by the term, 'data type'?

Primitive data types	The most basic data types within a language. Integers, characters, floats and Booleans are all examples of this. A string, however, is a composite data type.
Integer	A whole number (eg, 3, 4, 65465)
Real / floating point	A number with a decimal (eg. 3.14, 64.78)
Character	A single letter, number or symbol. (e.g., A, 1, !)
String	A combination of characters (eg, “Hello”, “DY10 1XA”)
Boolean	1 of 2 possible given values (eg. True/False, Yes/No)

<table>
<tr><th colspan="2">Number representation</th></tr>
<tr><td>Denary</td><td>Numbering system which uses base 10 (0-9) – these are our normal numbers that we use every day. (Otherwise known as decimal)</td></tr>
<tr><td>Binary</td><td>Numbering system which uses base 2 (0s & 1s) – the only language that computers truly understand. 0 means off, 1 means on.</td></tr>
<tr><td>Signed</td><td>A binary number which has 1 bit allocated to determining the sign of the number.</td></tr>
<tr><td>Unsigned</td><td>A binary number which does not have 1 bit allocated to determining its sign.</td></tr>
<tr><td>Sign and magnitude</td><td>Uses the left hand bit to represent the sign (0 being + and 1 being -).
<table>
<tr><th>SIGN</th><th>64</th><th>32</th><th>16</th><th>8</th><th>4</th><th>2</th><th>1</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Fixed point</td><td>A binary number whereby the decimal point always appears in the same place.</td></tr>
<tr><td>Floating point</td><td>A binary number whereby an exponent determines where the decimal point should be.</td></tr>
<tr><td>Two's complement</td><td>The most significant bit is considered negative. Meaning that the largest column value is -128.
<table>
<tr><th>-128</th><th>64</th><th>32</th><th>16</th><th>8</th><th>4</th><th>2</th><th>1</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Hexadecimal</td><td>Numbering system which uses base 16 (0-9 and A-F). These numbers are used to represent colours and code in assembly language, as they are easier for humans to understand than binary.</td></tr>
<tr><td>ASCII</td><td>A character set which uses 7 bits to store a maximum of 128 characters. This uses the binary numbers 0 to 127.</td></tr>
<tr><td>Unicode</td><td>The modern standard for representing characters in a computer system. Uses 16 bits to allow 65,536 characters to be represented.</td></tr>
<tr><td>Character set</td><td>A set of characters used in a language, which are each represented using a unique binary number.</td></tr>
</table>

Key question: How are numbers stored in memory?

In order to run efficiently, computers need to be able to handle all forms of data. When a variable is defined, a data type usually also needs to be declared.

This gives the computer an understanding of how much memory needs to be allocated as well as what operations can be applied to an item of data. For example, you cannot store an integer in a variable designated for storing text and vice versa.

Key question: How does an arithmetic logic unit (ALU) perform arithmetic?

An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers.

Key question: How does an arithmetic logic unit (ALU) perform arithmetic?

+/-	64	32	16	8	4	2	1	Sign and magnitude
0	0	1	0	0	1	1	1	=39
1	0	1	0	0	1	1	1	=-39

The number remains the same but the largest bit turns in to a + or – and a 1 means negative and a 0 in that column is a positive.

-128	64	32	16	8	4	2	1	Two's complement
0	0	1	0	0	1	1	1	=39
1	1	0	1	1	0	0	1	=-39

Use two's complement for addition etc... As sign and magnitude won't work. A method of converting positive number to negative or vice versa. By making the most significant bit a negative and adding up the other bits to equal the desired number.

-128	64	32	16	8	4	2	1	Ans
0	0	1	0	1	0	0	1	=41
1	1	0	1	0	1	1	1	=-41
0	1	0	0	0	1	1	0	=70
0	0	0	1	1	1	0	1	=29
1				1	1			

The carry column, there is one to the far left but you ignore it.

Addition and subtraction

- 70-41
- To do this you change it so it's 70+ -41
- All you do is convert 41 into -41 using two's complement.
- Then you add.

To add do the same but don't convert one of the numbers.

In the carry column for example 1+1=10 the 0 goes in the current column and the 1 is carried along. Another example is 1+1+1=11 (the third 1 is carried from previous sum) so the final 1 stays in the current column and the first 1 is carried.

Component 1 | 1.4.1| Data types

Key question: What examples are there where working with large binary numbers is a problem, and what is the solution?

Converting to hexadecimal

0-9
A=10
B=11
C=12
D=13
E=14
F=15

128	64	32	16	8	4	2	1	
0	0	1	0	1	1	0	1	=45

8	4	2	1
0	0	1	0

2

8	4	2	1
1	1	0	1

14=E

So 2E is 45 in hexadecimal

- Break down in to bits of 4.
- Then find the values for that section.
- Take the total and convert it to hexadecimal using the scale on the left of the table.
- Final place the letters/numbers next to one another.

Two's complement binary floating point format

The number of bits in the mantissa and exponent can change. But the following format is used.

-1	$\bullet\,^{1}/_{2}$	$^{1}/_{4}$	$^{1}/_{8}$	$^{1}/_{16}$	$^{1}/_{32}$	$^{1}/_{64}$	$^{1}/_{128}$	$^{1}/_{256}$	$^{1}/_{256}$	-32	16	8	4	2	1
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
10 bit mantissa In two's complement										6 bit exponent in two's complement					

Example – Using a positive exponent

The number 0100101000 000100 uses 10 bits mantissa and 6 bits exponent. The exponent is positive as it begins with a 0. We begin by calculating the exponent value.

$$000100 \rightarrow 4$$

(point moves 4 steps to the right ➔)

This means that our mantissa changes to become

$$01001.01000 = 8 + 1 + \frac{1}{4} = 9\frac{1}{4}$$

Example – Using a negative exponent

The number 0101000000 11110 uses 10 bits mantissa and 6 bits exponent. The exponent is negative as it begins with a 1. We begin by calculating the exponent value.

$$111110 \rightarrow 000001 \rightarrow 000010 = -2$$

(point moves 2 steps to the left 🡐)

This means that our mantissa changes to become

$$0.00101 = \frac{1}{8} + \frac{1}{32} = \frac{5}{32}$$

Example – Negative mantissa and negative exponent

The number 1011000000 111110 uses 10 bits mantissa and 6 bits exponent. The exponent is negative as it begins with a 1. We begin by calculating the exponent value.

$$111110 \rightarrow 000001 \rightarrow 000010 = -2$$

(point moves 2 steps to the left ←)

The mantissa is negative so we find the positive value then move the point into position

$$1.011000000 \rightarrow -0.100111111 \rightarrow -0.101000000$$
$$-0.101000000 -> -0.00101000000$$

This means that our mantissa changes to become

$$-0.00101000000 = -\frac{1}{8} + -\frac{1}{32} = -\frac{5}{32}$$

Key question: How does a computer store fractions (real numbers)?

Example – Negative mantissa and negative exponent

The number 1011000000 111110 uses 10 bits mantissa and 6 bits exponent. The exponent is negative as it begins with a 1. We begin by calculating the exponent value.

$$111110 \rightarrow 000001 \rightarrow 000010 = -2$$

(point moves 2 steps to the left ←)

The mantissa is negative so we find the positive value then move the point into position

$$1.011000000 \rightarrow -0.100111111 \rightarrow -0.101000000$$
$$-0.101000000 -> -0.00101000000$$

This means that our mantissa changes to become

$$-0.00101000000 = -\frac{1}{8} + -\frac{1}{32} = -\frac{5}{32}$$

Example – Normalising a positive number

The mantissa of a normalised positive number begins with 01. To get this we must identify where the first 01 pattern is, and adjust the mantissa and exponent to suit. For this example we shall use a mantissa of 8 bits and exponent of 4 bits.

Mantissa: 00010011 Exponent : 0011 (3)

The point must end up here 0001.0011

The normalised Mantissa is 0.10011000, therefore the exponent must be 1 as the point has to move 1 place to the right → To find its true value.

Our final answer is 010011000 0001

When normalising a positive floating point number, the value is padded with 0s to fill the mantissa.

Example – Normalising a negative number

The mantissa of a normalised positive number begins with 10. To get this we must identify where the first 10 pattern is, and adjust the mantissa and exponent to suit. For this example we shall use a mantissa of 8 bits and exponent of 4 bits.

Mantissa: **11100100** Exponent : **0011 (3)**

The point must end up here **1110.0100**

The normalised Mantissa is **1.0010011**, therefore the exponent must be 1 as the point has to move 1 place to the right ➔ To find its true value.

Our final answer is **10010011 0001**

When normalising a negative floating point number, the value is padded with 1s to fill the mantissa.

Key question: How does a computer store text in memory?

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Typical exam questions

1. Convert the denary numbers 96 and 204 into unsigned binary and then calculate the addition of the numbers. Store your answer in 8-bits and show your working. **[5]**

96:

204:

2. Explain your answers to part 1. **[2]**

3. Convert the denary number -96 into binary using sign and magnitude notation**. [2]**

4. Demonstrate how you subtract two binary numbers using 8-bit two's complement notation. Use the equivalent denary calculation of 120 – 47. Make sure to show all your working. **[4]**

Component 1 | 1.4.1| Data types

Target: ☐ Overall grade: ☐

Minimum expectations & learning outcomes

☐	Terms 154-174 from your A Level Key Terminology should be included and formatted.
☐	You must include a table which summarises the characteristics of the primitive data types.
☐	You must include some fully worked examples of conversion between denary, hex and binary.
☐	You must include some fully worked examples of arithmetic (addition & subtraction), use of carries, lost carries, why computers don't use sign and magnitude for arithmetic, and performing floating point addition and subtraction.
☐	You must include a diagram which clearly explains bitwise manipulation and masks.
☐	You must include an explanation of how the character sets ASCII and UNICODE are used to represent text.
☐	Answer the exam questions.

Feedback

Breadth	Depth	Presentation	Understanding
☐ All	☐ Analysed	☐ Excellent	☐ Excellent
☐ Most	☐ Explained	☐ Good	☐ Good
☐ Some	☐ Described	☐ Fair	☐ Fair
☐ Few	☐ Identified	☐ Poor	☐ Poor

Comment & action required

Component 1 | 1.4.1| Data types

Reflection & Revision checklist

Confidence	Clarification
☹ 😐 ☺	Candidates need to have an understanding of programming data types such as integer, real, Boolean, character, string etc.
☹ 😐 ☺	Candidates need to be able to choose appropriate data types for a situation or given data.
☹ 😐 ☺	Candidates should have experience of programming solutions using these data types.
☹ 😐 ☺	Candidates should have knowledge of how to convert from one data type to another (casting).
☹ 😐 ☺	Candidates should understand how and why computers store data as binary, and that a binary number can have a variety of different interpretations depending on what is being stored (e.g. numeric, text, image, sound).
☹ 😐 ☺	Candidates should be able to convert positive whole numbers to binary and from binary to denary.
☹ 😐 ☺	Candidates should know how to store negative numbers using Sign and Magnitude and Two's Complement.
☹ 😐 ☺	Candidates should be able to convert denary numbers to sign and magnitude, and two's complement – and vice-versa.
☹ 😐 ☺	Candidates should be able to perform addition and subtraction on integer binary numbers. (These numbers could be positive or negative using two's complement representation.)
☹ 😐 ☺	Candidates need to have an understanding of the purpose and potential uses of hexadecimal for example where and why they are used instead of binary and the benefits of using hexadecimal over alternatives such as binary.
☹ 😐 ☺	Candidates should be able to convert denary numbers to hexadecimal and vice-versa and from binary to hexadecimal and vice-versa.
☹ 😐 ☺	Candidates should have an understanding of how (positive and negative) real numbers are represented in a binary floating-point representation and should be able to convert between a denary number and a real binary number. (NB the representation used for the exam is the mantissa and exponent both represented using two's complement.)
☹ 😐 ☺	Candidates should understand the need for normalised floating-point numbers.
☹ 😐 ☺	Candidates should be able to normalise a floating-point number.
☹ 😐 ☺	Candidates should have an understanding of how characters are represented in binary.
☹ 😐 ☺	Candidates should understand the need for a character set and how a computer makes use of a character set.
☹ 😐 ☺	Candidates should be aware of the ASCII and UNICODE character sets and be able to explain the differences between these and the benefits of each.
☹ 😐 ☺	Candidates should be able to use a character set, or part of a character set, to translate characters into binary and vice-versa. (Candidates are not expected to memorise any values in a character set)

Component 1 | 1.4.1| Data types

Reflection & Revision checklist

Confidence	Clarification
☹ 😐 ☺	Candidates should be able to normalise a floating point number.
☹ 😐 ☺	Candidates should be able to perform addition and subtraction floating point arithmetic including addition and subtraction of both positive and negative numbers.
☹ 😐 ☺	Candidates should be able to perform right and left logical shifts.
☹ 😐 ☺	Candidates should understand the effect of right and left shifts on a binary numbers.
☹ 😐 ☺	Candidates should understand the purpose of using masks with bitwise operators, and should have experience of applying masks using AND, OR and XOR.