

Binary Arithmetic and Algorithms Using C and Assembly Language

Converting Decimal Inputs into Various Outputs

Module: Computer Systems Architecture

Focus: Understanding Computer Based Logic

A report originally created to fulfil the degree of

BSc (Hons) in Computer Science



**University
of Brighton**

Submitted 12/11/2018

Abstract

This paper demonstrates how decimal numbers can be converted into alternate number formats, including binary and hexadecimals, through the use of arithmetic and algorithms. These decimal inputs can be positive or negative, as each task outlined below also has a subtask which allow for signed binary converts, allowing for the use of negative inputs. The paper then goes on to use the arithmetic to add or subtract two 8-bit binary numbers, as well as developing binary functions in C.

Contents

Task 1: 7-Bit Decimal to Binary Converter	2
Task 1a: Signed 8-Bit Decimal to Binary Converter	3
Task 2: Decimal to Hexadecimal Converter.....	3
Task 2a: Signed Decimal to Hexadecimal Converter	4
Task 3: Decimal to Octal Converter	5
Task 3a: Signed Decimal to Octal Converter	6
Task 4: 7-Bit Binary Converter	7
Task 4a: Signed 8-Bit Binary Converter	8
Task 5a: Signed 11-Bit Binary to Decimal Converter	10
Task 6: 8-Bit Binary Addition and Subtraction	10
Task 7: Converting Inputs into Integers	13
Task 8: Converting Decimal into Binary	14
Task 9: Debugging Dumped Registers	15
Reflection.....	16
Appendices	17

Task 1: 7-Bit Decimal to Binary Converter

The first task was to make a 7-Bit decimal to binary converter for values in the range of 0 to 127. I started off by making a table of values which I would be able to look up at any point using HLOOKUP. I did this as I knew I would most likely have to repeat this action for multiple tasks. Once I had my table of bits corresponding to their decimal equivalent, I then used HLOOKUP to automatically take the correct values and input them into my decimal to binary converter, ready for the next process.

Binary Numbers	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	-128	64	32	16	8	4	2	1

The second step involved taking the users input and subtracting it from each bit in the table, starting from the highest bit number. I used IF functions to determine if the decimal was greater or equal to the value of the selected bit number. A 1 would be outputted in the cell below if true, otherwise the output would be 0, and the same decimal number was used for the next bit number. However, if a 1 was outputted, then the looked-up value from the table was subtracted from the current decimal number, and then the result would be used as the next decimal number. Once all 7-bits had been calculated, the final step was to concatenate all the 0's and 1's in the order they were outputted, starting from the most significant bit. This output was then fed back to the user as the final result.

Task 1: 7-Bit Decimal to Binary Converter (For Values 0 to 127)	
Decimal Input	Binary Output
31	0011111

Binary Conversion								Result	Check
Bit Number	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0011111	0011111
Binary Bit Lookup Number	64	32	16	8	4	2	1		
Current Decimal Number	31	31	31	15	7	3	1		
Is Bit No. < Decimal No.?	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE		
Conversion to Binary	0	0	1	1	1	1	1		
Resulting Decimal Number	31	31	15	7	3	1	0		

Binary Conversion	
Bit Number	Comments
Binary Bit Lookup Number	Looks up the value of the bit in a table of binary numbers in decimal form.
Current Decimal Number	The decimal number that is compared to the above value.
Is Bit No. < Decimal No.?	If the bit number looked up from the table is less than the decimal above, output True.
Conversion to Binary	If the above value is True, output a 1, otherwise output a 0.
Resulting Decimal Number	If the above output is 1, subtract the Binary Bit Lookup Number from the Current Decimal Number.

The area that the user inputs their decimal input is kept on a separate sheet to where the conversion takes place. The final result from the conversion table is then sent back to the sheet where the user inputted their number. The conversion table contains comments and descriptions of the step by step process that occurs. The result produced is also checked against functions built into Excel, such as DEC2BIN (31) in this example, to make sure that is correct. Any values including a decimal point are converted to integers using the INT function. If a value outside the range is inputted into the system, the output prints "Value out of Range", to let the user know the conversion cannot be processed.

Task 1: 7-Bit Decimal to Binary Converter (For Values 0 to 127)	
Decimal Input	Binary Output
128	Value Out of Range

Task 1a: Signed 8-Bit Decimal to Binary Converter

In order for negative values to be inputted into the converter, another bit was needed to show if the binary output was positive or negative. If the decimal input was positive, then the process would unfold the same as before, except with an extra 0 simply slotted in the front of the result. If the decimal input was negative however, then a 1 would be added to the front instead. This is because the largest bit number in decimal form is always negative when using two's complement. Another way of achieving the result would be to invert all the 0's and 1's, and then adding 1 to the answer, however this would require a carry in the event of two 1's being added together. This would have required more steps to achieve the same outcome, and therefore would have been less efficient.

Task 1a: Signed 8-Bit Decimal to Binary Converter (For Values -127 to 127)	
Decimal Input	Binary Output
-15	11110001

Binary Conversion									Result	Check
Bit Number	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	11110001	11110001
Binary Bit Lookup Number	-128	64	32	16	8	4	2	1		
Current Decimal Number	-15	113	49	17	1	1	1	1		
Is Bit No. < Decimal No.?	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE		
Conversion to Binary	1	1	1	1	0	0	0	1		
Resulting Decimal Number	113	49	17	1	1	1	1	0		

Binary Conversion	
Bit Number	Comments
Binary Bit Lookup Number	Looks up the value of the bit in a table of binary numbers in decimal form.
Current Decimal Number	The decimal number that is compared to the above value.
Is Bit No. < Decimal No.?	If the bit number looked up from the table is less than the decimal above, output True.
Conversion to Binary	If the above value is True, output a 1, otherwise output a 0.
Resulting Decimal Number	If the above output is 1, subtract the Binary Bit Lookup Number from the Current Decimal Number.

Task 2: Decimal to Hexadecimal Converter

The second task involved taking a decimal number between the values 0 and 127, and converting it into a hexadecimal equivalent. The first step was to find the modular of the user's decimal number, using the MOD function with a divisor of 16. The output of this would later form the second half of the hexadecimal output. The second step was to subtract this result from the original decimal input. This output would be used to form the first half of the hexadecimal. Once the two numbers were found, they were checked to see if they equalled or were greater than 10. If either result contained multiple digits, they were then converted to a single hexadecimal digit using HLOOKUP on another predefined table. Once the two halves were calculated, they were simply concatenated together to form a hexadecimal output, similar to how the result of the first task was formed.

Hex Numbers	10	11	12	13	14	15	16
	A	B	C	D	E	F	0

Task 2: Decimal to Hexadecimal Converter (For Values 0 to 127)	
Decimal Input	Hexadecimal Output
83	53

Binary Conversion			Result	Check
Step by Step Guide	Number	Hex	53	53
The Decimal Input	83	N/A		
Modular of the Input (Divisor of 16)	3	3		
Decimal Input - The Modular Value	80	N/A		
Modular of the Above (Divisor of 16)	5	5		

Binary Conversion	
Step by Step Guide	Comments
The Decimal Input	This is the number that will be converted to hexadecimal.
Modular of the Input (Divisor of 16)	This number will form the second half of the hexadecimal.
Decimal Input - The Modular Value	This equation is used to find the first half of the hexadecimal.
Modular of the Above (Divisor of 16)	This number will form the first half of the hexadecimal.

Task 2a: Signed Decimal to Hexadecimal Converter

The process for allowing negative inputs didn't require any change in the process, except for adding 00 on the front of positive numbers, and FF on the front of negative ones. The reason FF is slotted in front of the result is because one F represents the 16th and largest digit of hexadecimal. Two F's are used because 16 multiplied by 16 is 256, which is the total number of possible outputs when using an input that can be represented as an 8-bit number.

Task 2a: Signed Decimal to Hexadecimal Converter (For Values -127 to 127)	
Decimal Input	Hexadecimal Output
-83	FFAD

Binary Conversion			Result	Check
Step by Step Guide	Number	Hex	FFAD	FFAD
The Decimal Input	-83	N/A		
Modular of the Input (Divisor of 16)	13	D		
Decimal Input - The Modular Value	-96	N/A		
Modular of the Above (Divisor of 16)	10	A		

Binary Conversion	
Step by Step Guide	Comments
The Decimal Input	This is the number that will be converted to hexadecimal.
Modular of the Input (Divisor of 16)	This number will form the second half of the hexadecimal.
Decimal Input - The Modular Value	This equation is used to find the first half of the hexadecimal.
Modular of the Above (Divisor of 16)	This number will form the first half of the hexadecimal.

Task 3: Decimal to Octal Converter

Converting a decimal input to an octal output was similar to the previous task, apart from requiring a few more steps to produce an output. To start off, the input was converted to an absolute value, so any negative values would be treated as a positive until later on in the process. A modular of this result was then taken, with a divisor of 8, which would later form the last digit of the octal output. To find the middle digit, an absolute value of the decimal input was taken again, but this time was divided by 8. If this resulted in the fraction, the number was simply rounded down to become an integer. A modular of this result was then taken, and like the previous time a divisor of 8 was used. This number would later form the second digit of the output. To find the first digit, an absolute value of the decimal input was found yet again, this time dividing it by 64 (equal to 8^2). A modular of the rounded down value, with a divisor of 8, produced the first digit. The 3 individual digits were then concatenated together in the order previously stated to form the final octal output.

Task 3: Decimal to Octal Converter (For Values 0 to 127)	
Decimal Input	Octal Output
83	123

Binary Conversion			Result	Check
Step by Step Guide		Number	123	123
Input	The Decimal Input	83		
Oct No.	Absolute Value of the Input	83		
Part 3	Modular of the Above (Divisor of 8)	3		
Oct No.	Abs. Dec. Input / 8 (Rounded Down)	10		
Part 2	Modular of the Above (Divisor of 8)	2		
Oct No.	Abs. Dec. Input / 64 (Rounded Down)	1		
Part 1	Modular of the Above (Divisor of 8)	1		

Binary Conversion		
Step by Step Guide		Comments
Input	The Decimal Input	This is the number that will be converted to octal.
Oct No.	Absolute Value of the Input	The input needs to be positive for it to be converted.
Part 3	Modular of the Above (Divisor of 8)	This will be the last digit in the converted octal number.
Oct No.	Abs. Dec. Input / 8 (Rounded Down)	To find the next digit, the input must be divided by 8^1 .
Part 2	Modular of the Above (Divisor of 8)	This will be the middle digit in the converted octal number.
Oct No.	Abs. Dec. Input / 64 (Rounded Down)	To find the next digit, the input must be divided by 8^2 .
Part 1	Modular of the Above (Divisor of 8)	This will be the first digit in the converted octal number.

Task 3a: Signed Decimal to Octal Converter

Two more steps were needed when adapting the converter for negative numbers, the first being to check if the original input was indeed negative. If this was true, the second step was to subtract the concatenated value from 778. I discovered after some research that the reason for it being this number is due to the nature of base 8 only containing the digits 0 – 7, and the process of finding the modular of the decimal input is completed three times, so the smallest negative input creates an output of 777. Since the smallest negative input possible is -1, the number that the concatenated value is subtracted from must be 777 minus -1, which is equal to 778.

Task 3a: Signed Decimal to Octal Converter (For Values -127 to 127)	
Decimal Input	Octal Output
-83	655

Binary Conversion			Result	Check
Step by Step Guide		Number	655	655
Input	The Decimal Input	-83		
Oct No.	Absolute Value of the Input	83		
Part 3	Modular of the Above (Divisor of 8)	3		
Oct No.	Abs. Dec. Input / 8 (Rounded Down)	10		
Part 2	Modular of the Above (Divisor of 8)	2		
Oct No.	Abs. Dec. Input / 64 (Rounded Down)	1		
Part 1	Modular of the Above (Divisor of 8)	1		
Final Step	Concatenated Number (Parts 1, 2 & 3)	123		
	Is the Input a Positive Number?	FALSE		

Binary Conversion		
Step by Step Guide		Comments
Input	The Decimal Input	This is the number that will be converted to octal.
Oct No.	Absolute Value of the Input	The input needs to be positive for it to be converted.
Part 3	Modular of the Above (Divisor of 8)	This will be the last digit in the converted octal number.
Oct No.	Abs. Dec. Input / 8 (Rounded Down)	To find the next digit, the input must be divided by 8^1 .
Part 2	Modular of the Above (Divisor of 8)	This will be the middle digit in the converted octal number.
Oct No.	Abs. Dec. Input / 64 (Rounded Down)	To find the next digit, the input must be divided by 8^2 .
Part 1	Modular of the Above (Divisor of 8)	This will be the first digit in the converted octal number.
Final Step	Concatenated Number (Parts 1, 2 & 3)	Slots the individual parts together to make a final number.
	Is the Input a Positive Number?	If the input is negative, subtract the number above from 778.

Task 4: 7-Bit Binary Converter

The next task involved taking a 7-bit binary number and converting it into multiple outputs, including decimal, hexadecimal and octal. To convert the input into a decimal number, the process of using HLOOKUP to find the equivalent table values was used again. This meant the value of the looked-up bits were only outputted if the same bit from the input was equal to 1. All of the decimal values that had bits equal to 1 were then added up to produce a final decimal output.

Binary Numbers	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	-128	64	32	16	8	4	2	1

Task 4: 7-Bit Binary Converter (For Values 0000000 to 1111111, or 0 to 127)									
Binary Input								Equivalent Outputs	
No Sign	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Binary Number	Decimal
Bit Used	1	1	1	0	0	1	1	1110011	115

Binary Conversion								Result	Check
Bit Number	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	1110011	1110011
Binary Input	1	1	1	0	0	1	1		
Dec Conversion	64	32	16	0	0	2	1	115	115

The next output was a hexadecimal value, which involved breaking down the 7-bit binary input into 2 smaller binary inputs, a 3-bit and 4-bit binary number respectively. The first 3 bits were used to form one value, the largest being 7 if all bits were 1's, while the last 4 bits were used to form another, with the largest being 15 if all bits were 1's. Because hexadecimal values convert numbers with multiple digits into a single value, anything over 10 is converted into a corresponding letter using HLOOKUP, similarly to task 2. In this case, the 15 would be converted into an F.

Task 4: 7-Bit Binary Converter (For Values 0000000 to 1111111, or 0 to 127)									
Binary Input								Equivalent Outputs	
No Sign	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Binary Number	Hex
Bit Used	1	1	1	1	1	1	1	1111111	7F

Binary Conversion								Result	Check
Bit Number	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	1111111	1111111
Binary Input	1	1	1	1	1	1	1		
Hex Conversion	4	2	1	8	4	2	1	7F	7F
	7			15					
	7			F					

The final output for this task was an octal value, which meant breaking down the 7-bit binary input into 2 smaller binary inputs yet again. This time however the 2 smaller numbers were the other way around, with a 4-bit binary number followed by a 3-bit one. The first 4 bits were used to form one value, the largest being 15 if all bits were 1's, while the last 3 bits were used to form another, with the largest being 7 if all bits were 1's. This value of 7 would later be used to form the last digit of the octal output. A modular of the first value, in this case 15, was used with a divisor of 8 to produce a remainder of 7. This would later form the middle digit of the octal output. Finally, the first digit was found by taking the first value, which again would be 15 in this case, and dividing it by 8. The result is rounded down to the nearest integer, resulting in a value of 1 on this occasion. Therefore, the final octal output would be a concatenated value of the 3 outputs above, resulting in a value of 177.

Task 4: 7-Bit Binary Converter (For Values 0000000 to 1111111, or 0 to 127)									
Binary Input								Equivalent Outputs	
No Sign Bit Used	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Binary Number	Octal
	1	1	1	1	1	1	1	1111111	177

Binary Conversion								Result	Check
Bit Number	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	1111111	1111111
Binary Input	1	1	1	1	1	1	1		
Oct Conversion	8	4	2	1	4	2	1	177	177
	15			7					
	7			1					

After I completed all three functions separately, they were then slotted together to use the same input for all three conversions. This allowed for a side by side comparison between binary, decimal, hexadecimal and octal values, ranging from 0 to 127 in the decimal equivalent. The final version of this task can be seen below, with each of the three functions working simultaneously.

Task 4: 7-Bit Binary Converter (For Values 0000000 to 1111111, or 0 to 127)											
Binary Input								Equivalent Outputs			
No Sign Bit Used	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Binary Number	Decimal	Hex	Octal
	1	1	1	0	0	1	1	1110011	115	73	163

Binary Conversion								Result	Check
Bit Number	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	1111111	1111111
Binary Input	1	1	1	1	1	1	1		
Dec Conversion	64	32	16	8	4	2	1	127	127
Hex Conversion	4	2	1	8	4	2	1	7F	7F
	7			15					
	7			F					
Oct Conversion	8	4	2	1	4	2	1	177	177
	15			7					
	7			1					

Binary Conversion	
Bit Number	Comments
Binary Input	This is the input that is used when converting to different number types.
Dec Conversion	Looks up the value of the bit in a table of binary numbers in decimal form.
Hex Conversion	The 7-Bit binary number is first of all converted to a 3-Bit number and a 4-Bit number.
	The sum of the numbers above are converted to form the first and second half of the hexadecimal.
	If the result is between 10 and 15 then it will converted to it's corresponding hexadecimal letter.
Oct Conversion	The 7-Bit binary number is first of all converted to a 3-Bit number and a 4-Bit number.
	The sum of the numbers above are converted to form the first and second half of the octal number.
	A modular of 8 for the first value above (14 in this case), are used to form the final octal number.

Task 4a: Signed 8-Bit Binary Converter

Like the previous tasks, each process had to be slightly modified to allow for negative values to be converted. The decimal conversion simply needed another bit added on the front to represent the sign of the number. If the first bit is equal to 1, the corresponding value is equal to -128. The reason the number is negative unlike the others is for the same reason talked about in task 1, where two's complement inverts all the 0's and 1's and then adds a 1 onto the result to give the final answer.

Binary Conversion								Result	Check
Bit Number	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	11111111	11111111
Binary Input	1	1	1	1	1	1	1		
Dec Conversion	-128	64	32	16	8	4	2	-1	-1

To be able to produce a negative hexadecimal value, the same idea of splitting the binary input into 2 smaller binary numbers still occurs. However, since the input is now an 8-bit number due to an extra bit used to show if the number is positive or negative, it is split into two 4-bit numbers instead. The first 4 bits were used to form one value, while the last 4 bits were used to form another. Just like the previous iteration of this process, any value that is 10 or over is converted into a corresponding letter using HLOOKUP. In this case, both of the 15's would be converted into F's. Just like task 2a, 00 would then be added on the front of positive numbers, and FF on the front of negative ones.

Binary Conversion								Result	Check
Bit Number	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	11111111	11111111
Binary Input	1	1	1	1	1	1	1		
Dec Conversion	-128	64	32	16	8	4	2	-1	-1
Hex Conversion	8	4	2	1	8	4	2	FFFF	FFFF
	15				15				
	F				F				

The process of finding a negative octal value compared to just a positive one has a slight change, but otherwise it is practically the same. If the most significant bit of the binary input is a 1, then the other 7-bits are inverted for the rest of the process, with all 0's becoming 1's, and all 1's becoming 0's. The converter then follows the same steps as the previous iteration, splitting up the remaining 7-bits into 4-bit and 3-bit binary numbers. The first 4 bits were used to form one value, while the last 3 bits were used to form another. The steps remain the same up until the output is about to be produced. If the original binary input was a positive number, the output steps are the same as before. However, if the input is negative, then the result is subtracted from 777, similar to task 3a.

Binary Conversion								Result	Check
Bit Number	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	11111111	11111111
Binary Input	1	1	1	1	1	1	1		
Dec Conversion	-128	64	32	16	8	4	2	-1	-1
Hex Conversion	8	4	2	1	8	4	2	FFFF	FFFF
	15				15				
	F				F				
Oct Conversion	777	0	0	0	0	0	0	777	777
		0	0	0	0	0	0		
		0				0			
		0				0			
		0				777			

Binary Conversion	
Bit Number	Comments
Binary Input	This is the input that is used when converting to different number types.
Dec Conversion	Looks up the value of the bit in a table of binary numbers in decimal form.
Hex Conversion	The 8-Bit binary number is first of all converted to a 3-Bit number and a 4-Bit number.
	The sum of the numbers above are converted to form the first and second half of the hexadecimal.
	Each result is converted to the matching letter. If the binary number is negative, FF is slotted in front.
Oct Conversion	If the input is negative, the 0's and 1's are inverted in this step before moving on with the process.
	The 8-Bit binary number is next converted to a 3-Bit number and a 4-Bit number.
	The sum of the numbers above are converted to form the first and second half of the octal number.
	A modular of 8 for the first value above (0 in this case), are used to get to the final octal number.
	If the input is positive, the answer is from the process above (0), otherwise it is subtracted from 777.

Task 5a: Signed 11-Bit Binary to Decimal Converter

The next conversion was similar to task 1a in that it involved taking a signed binary number in order to produce a decimal equivalent, however this time it involved using values less than 1 in the 3 least significant bits via the use of a decimal point. The exact same process was performed, including using HLOOKUP to find the bit values, except when it came to calculating the last 3-bits. The values of the decimal digits were not inverted like the others if the binary input was negative.

Binary Numbers (Inc. Decimal Point)	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	-128	64	32	16	8	4	2	1	0.5	0.25	0.125

Task 5: Signed 11-Bit Binary to Decimal Converter (For Values 00000000.000 to 11111111.111, or 0 to 127.875)												
Binary Input											Binary Number	Decimal Output
Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
0	0	0	0	1	1	0	0	1	0	0	00001100.100	12.5

Binary Conversion												Result	Check
Bit Number	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
Binary Number	0	0	0	0	1	1	0	0	1	0	0	00001100.100	00001100.100
Dec Conversion	0	0	0	0	8	4	0	0	0.5	0	0	12.5	12.5

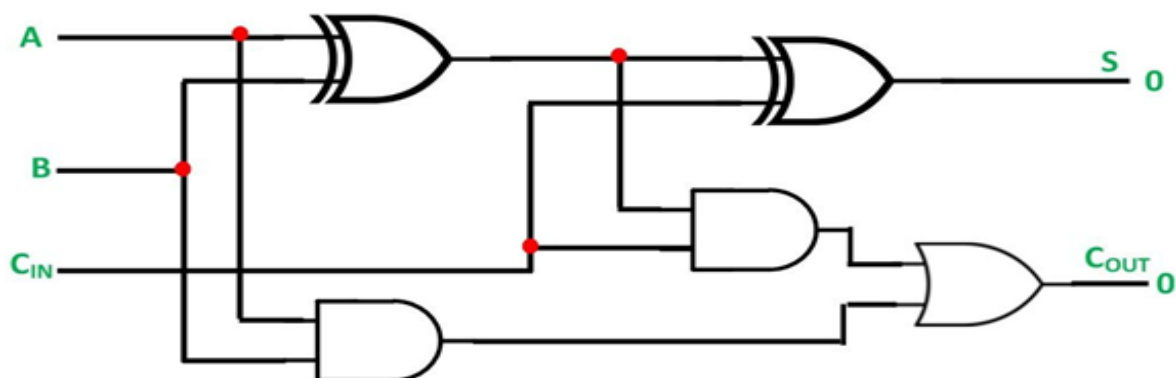
Binary Conversion	
Bit Number	Comments
Binary Number	This takes the binary input and concatenates it into a single number.
Dec Conversion	Each bit is matched to its corresponding value and then added together.

An example of a negative binary inputted into the converter can be seen below. Whilst the bits to the left of the decimal point are inverted from 0's to 1's and 1's to 0's, like in the previous tasks that use two's complement to produce negative values, the bits to the right remain the same.

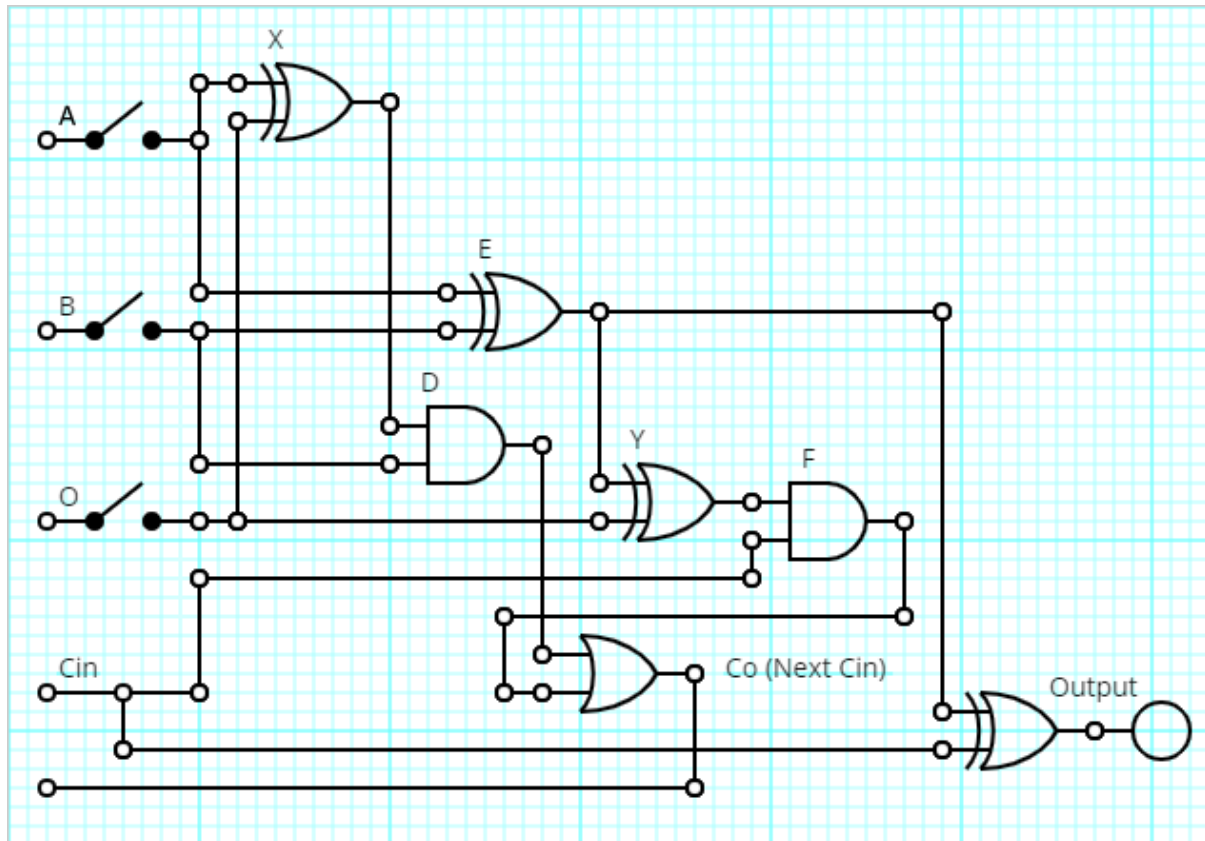
Binary Conversion												Result	Check
Bit Number	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
Binary Number	1	1	1	1	1	1	1	1	1	0	0	11111111.100	11111111.100
Dec Conversion	-128	64	32	16	8	4	2	1	0.5	0	0	-1.5	-1.5

Task 6: 8-Bit Binary Addition and Subtraction

The following task was to make a binary calculator that could add or subtract two 8-bit binary inputs. I found this to be more challenging than the previous tasks so far, as it consisted of a following a long list of steps in order to produce a working calculator. Due to this, I decided to break down the task into two smaller steps, creating an adder, followed by a subtractor. After researching more about adders online, I discovered a few diagrams that made it easier to visualise.



I was then able to successfully create a full adder that I could use for my binary calculator. The next step was to then find a circuit diagram of a subtractor that I could recreate using logic functions. I was on my way to completing it when I realised that the logic behind the subtractor was quite similar to the adder I had previously built for the first half of the task. If I was building the calculator using logic gates on a circuit board, I would try to use the least number possible in order to make the process as efficient as possible. Due to this, I attempted a similar approach with both my adder and subtractor. Although this meant I required a third input, I was able to remove almost half of the logic by combined the two processes together in order for them to both run on the same logic gates. The logic circuit I used in the end was one I drew myself after seeing the parallels between the processes.



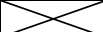
This logic gate was created with the Logic Gate Simulator by Dominic Ford, on ScienceDemos.org.uk. The result was then screenshots and edited in photoshop, in order to display the gate symbols.

The first step I took when merging the two components together was to create the three inputs, with the first and second being the two 8-bit binary numbers, whilst the third was for the operation. A logic state of 0 indicated the calculator would be performing addition, while a logic state of 1 indicated it would be performing subtraction. If something else was inputted into the calculator for the operation, the process would simply output "Incorrect Output Detected", so that the user knew to change it before the calculator could move on the next step.

This is where the merge between addition and subtraction first needed to be worked out. If the user wanted to add their two binary numbers, this step would simple produce a copy of the first input. If the user wanted to perform subtraction, then this step would produce an inverted copy instead, with all 0's becoming 1's and all 1's becoming 0's. This was called process X, so that it was easier to understand the logic when looking back through the various steps. It is important to note that the original inputs are stored and remain unchanged, as they are needed later on further down the line.

Binary Calculator									
Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
A	Input 1	0	0	1	1	0	0	1	1
B	Input 2	0	0	0	0	1	0	1	0
O	Input 3	1							
X	XOR(A, O)	1	1	0	0	1	1	0	0

The next process involved creating a working carry, which allowed the correction of any overflow from adding two 1's together or subtracting a 1 from a 0. This was done by taking the resulting carry out from the previous bit and using it for the next carry in. Since the first carry has no value to take from, it defaults to 0, symbolled as a cross in the picture in order to clearly show it is not in use.

Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Cin	Carry: Last Co Out.	0	0	0	1	0	0	0	

The next step involved finding the values of B and process X. If both of them were 1, the output would be 1, otherwise it would be 0. This was called process D.

Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
D	AND(X, B)	0	0	0	0	1	0	0	0

Afterwards, the bits from the two binary inputs are checked against each other using an XOR gate. If the bits are both 1 or 0, then a result of 0 is produced, but if they are different, a result of 1 will be outputted instead. This was called process E.

Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
E	XOR(A, B)	0	0	1	1	1	0	0	1

The result of this process is then checked against the third input, the one determining the operation, to see if they are also the same using another XOR gate. If the resulting bits are both 1 or 0, the result will be 0, but if there are different then the output will be 1. This was called process Y.

Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Y	XOR(E, O)	1	1	0	0	0	1	1	0

If both the carry in and process Y bits are equal to 1, then the result via the use of an AND gate will be a 1. This was called process F.

Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
F	AND(Cin, Y)	0	0	0	0	0	0	0	0

The carry output was then worked out via the use of an OR gate, using process D and process F and inputs. If either of the processes produced a 1, the result would be a 1.


Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Co	OR(D, F)	0	0	0	0	1	0	0	0

Finally, the bit being outputted was calculated with another XOR gate, this time using the carry in and process E as inputs. If either of the inputs were 1, the bit being outputted would be 1. This chain of processes was completed for each of the 8-bits, until finally an 8-bit result was produced.

Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Output	XOR(Cin, E)	0	0	1	0	1	0	0	1

Once each of the 8-bits had been calculated, the cells were concatenated to create the final 8-bit binary number. Both the result and the individual 8-bits were then sent back to the input screen, directly under the two binary inputs and the operation used. The final version of this screen, the binary calculator and the comments for each process can be seen below.

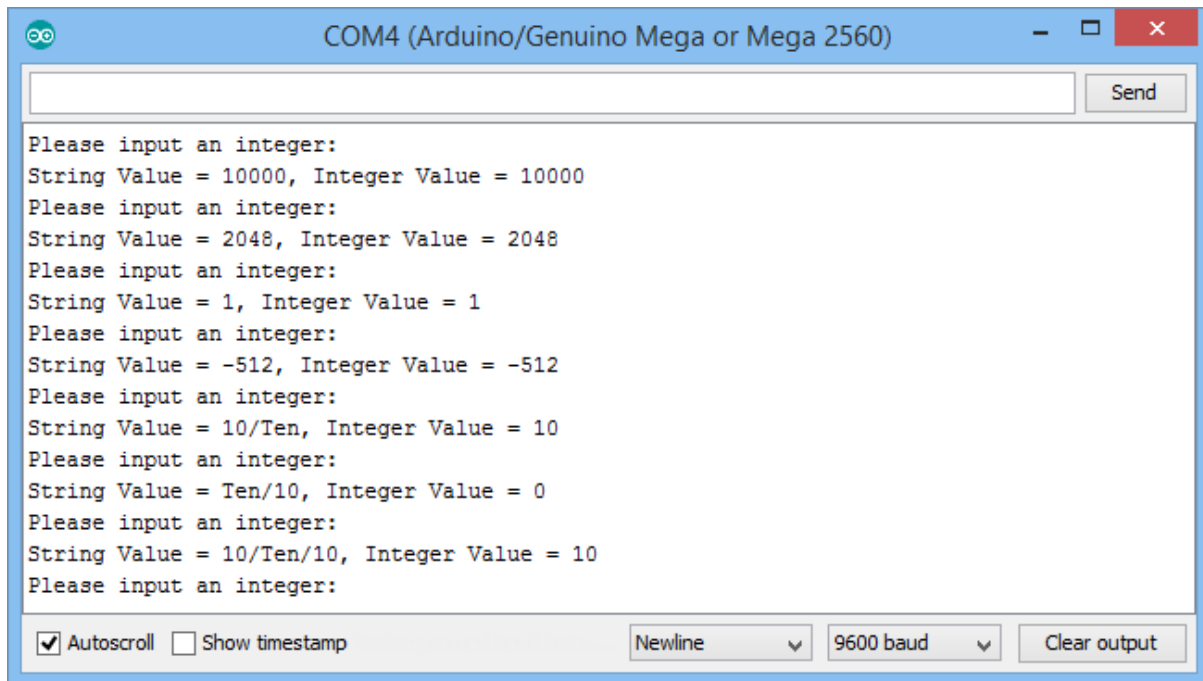
Task 6: 8-Bit Binary Addition and Subtraction									
8 Bit Adder	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Result
Input 1	0	0	1	1	0	0	1	1	00110011
Input 2	0	0	0	0	1	0	1	0	00001010
Operation	-								Subtraction
Output	0	0	1	0	1	0	0	1	00101001

Binary Calculator										Result	Check
Symbol	Process	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	00101001	00101001
A	Input 1	0	0	1	1	0	0	1	1		
B	Input 2	0	0	0	0	1	0	1	0		
O	Input 3	1									
X	XOR(A, O)	1	1	0	0	1	1	0	0		
Cin	Carry: Last Co Out.	0	0	0	1	0	0	0			
D	AND(X, B)	0	0	0	0	1	0	0	0		
E	XOR(A, B)	0	0	1	1	1	0	0	1		
Y	XOR(E, O)	1	1	0	0	0	1	1	0		
F	AND(Cin, Y)	0	0	0	0	0	0	0	0		
Co	OR(D, F)	0	0	0	0	1	0	0	0		
Output	XOR(Cin, E)	0	0	1	0	1	0	0	1		

Binary Calculator		
Symbol	Process	Comments
A	Input 1	This is the first input used in the binary calculator, representing the first value.
B	Input 2	This is the second input used in the binary calculator, representing the second value.
O	Input 3	This is the third input used in the binary calculator, with a 0 for addition or a 1 for subtraction.
X	XOR(A, O)	If the operation is addition, the result will simply be the first input, otherwise it will be the inverse.
Cin	Carry: Last Co Out.	This is used for when values carry over, i.e. when both input bits are set to 1.
D	AND(X, B)	If the bit values of X and B are both 1, then the output is 1, otherwise it will be 0.
E	XOR(A, B)	If A and B have different states then the result will be 1, otherwise it will be 0.
Y	XOR(E, O)	If the operation is addition, the result will simply be the result above, otherwise it will be the inverse.
F	AND(Cin, Y)	If the bit values of Cin and Y are both 1, then the output is 1, otherwise it will be 0.
Co	OR(D, F)	If either the bit values of D or F are 1, then the output is 1, otherwise it will be 0.
Output	XOR(Cin, E)	If Cin and E have different states then the result will be 1, otherwise it will be 0. This is the final output.

Task 7: Converting Inputs into Integers

The next program I created involved reading an input from the keyboard and converting it into an integer, which is then displayed on the terminal. This is simply done by reading the string into a null terminated char array, stopping only when it reaches the end of the input once a newline character ('\n') is read. After this occurs, the program then detects if the input is an integer using the built-in *atoi()* function. If the input is an integer, the program prints the converted number to the terminal, however if it is not then a '0' is printed instead. An example of the program in use with the integers 10000, 2048, 1 and -512, as well the inputs '10/Ten', 'Ten/10' and '10/Ten/10' is shown below:

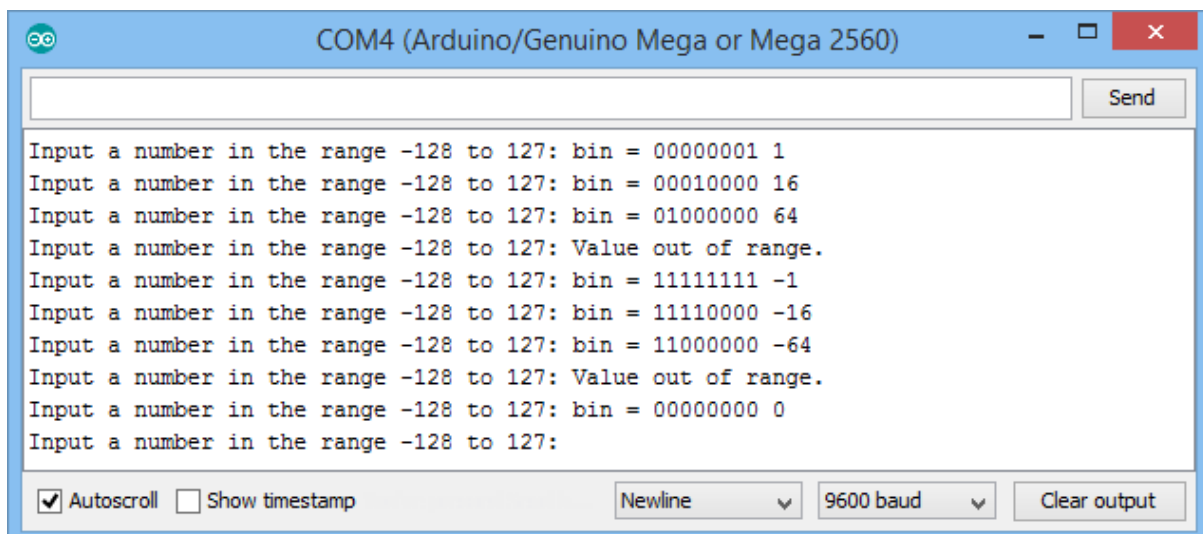


```
Please input an integer:
String Value = 10000, Integer Value = 10000
Please input an integer:
String Value = 2048, Integer Value = 2048
Please input an integer:
String Value = 1, Integer Value = 1
Please input an integer:
String Value = -512, Integer Value = -512
Please input an integer:
String Value = 10/Ten, Integer Value = 10
Please input an integer:
String Value = Ten/10, Integer Value = 0
Please input an integer:
String Value = 10/Ten/10, Integer Value = 10
Please input an integer:
```

As you can see from the test, numbers are converted into integers, including negative numbers. If, however, anything other than a number or negative sign is inputted, then a '0' is printed to the terminal as previously mentioned. In the event of a number being inputted before a string of text, the number will be converted, with anything after the number being discarded. This can be seen in the last line of testing, where the input '10/Ten/10' causes only the first '10' to be converted.

Task 8: Converting Decimal into Binary

The second program involved taking the *readInt()* function I created in the previous task and using it to convert decimal numbers into binary. Like the previous task, the program first checks if the input is valid, in this case checking if the input is both an integer and in the range of -128 to 127. If this is true, the program converts it into its binary equivalent using a function called *decimalToBinary()*. However, if this is not the case, then the program will instead output that the value is out of range. An example with the inputs 1, 16, 64, 256, -1, -16, -64, -256 and 'one' can be seen below.

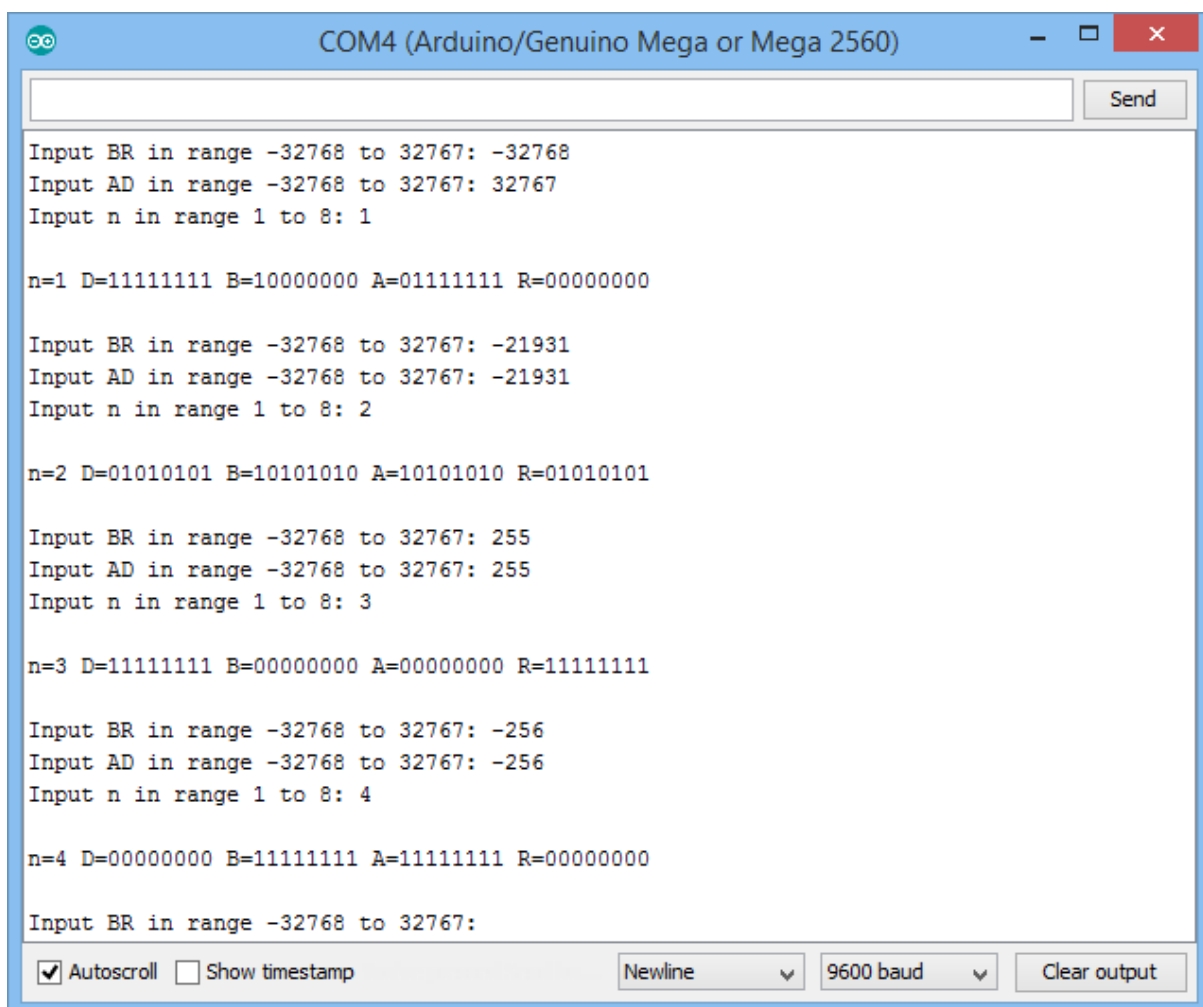


```
Input a number in the range -128 to 127: bin = 00000001 1
Input a number in the range -128 to 127: bin = 00010000 16
Input a number in the range -128 to 127: bin = 01000000 64
Input a number in the range -128 to 127: Value out of range.
Input a number in the range -128 to 127: bin = 11111111 -1
Input a number in the range -128 to 127: bin = 11110000 -16
Input a number in the range -128 to 127: bin = 11000000 -64
Input a number in the range -128 to 127: Value out of range.
Input a number in the range -128 to 127: bin = 00000000 0
Input a number in the range -128 to 127:
```

Note that the last test produced the output 'bin = 00000000 0' instead of 'Value out of range'. This is because the same logic is used from the previous question, where in the event of a number being inputted before a string of text, only the number will be converted. Since no number is inputted before the input 'one', the input is treated as being empty as opposed to being out of range.

Task 9: Debugging Dumped Registers

Similar to the previous two tasks, this program starts by using the *readInt()* function to check for two valid integer inputs. After each input is verified, the *decimalToBinary()* function was used to convert them both into their respective binary outputs. Unlike the previous program however, these binary numbers were both 16-bit instead of 8-bit. The program followed the same logic when converting the numbers, except for the end where each 16-bit binary number was split into 2 separate 8-bit binary numbers. These values are then printed to the terminal, alongside a third integer input in the range of 1 and 8. An example with various BR, AD and n inputs can be seen below.



```
COM4 (Arduino/Genuino Mega or Mega 2560)

Input BR in range -32768 to 32767: -32768
Input AD in range -32768 to 32767: 32767
Input n in range 1 to 8: 1

n=1 D=11111111 B=10000000 A=01111111 R=00000000

Input BR in range -32768 to 32767: -21931
Input AD in range -32768 to 32767: -21931
Input n in range 1 to 8: 2

n=2 D=01010101 B=10101010 A=10101010 R=01010101

Input BR in range -32768 to 32767: 255
Input AD in range -32768 to 32767: 255
Input n in range 1 to 8: 3

n=3 D=11111111 B=00000000 A=00000000 R=11111111

Input BR in range -32768 to 32767: -256
Input AD in range -32768 to 32767: -256
Input n in range 1 to 8: 4

n=4 D=00000000 B=11111111 A=11111111 R=00000000

Input BR in range -32768 to 32767:
```

As you can see from the testing above, each 16-bit input is split into two 8-bit outputs, known as the high byte (variable 'B') and the low byte (variable 'A'). These variables will be used during the signed multiplication of two binary numbers in the next program, along the variables 'R' and 'D' which will be used as multipliers. The final variable 'n' will record how many times the program has looped.

Reflection

Upon reflection of my work, I believe I have expanded on my knowledge of representing binary as equivalent outputs, alongside my ability to represent said outputs through the use of functions in C. While I was able to create solutions for the first five tasks using logic functions with relative ease, the sixth task in particular took some time to produce a solution. I found that the more I worked through a function, either through the spreadsheets or through C, the more I found myself able to see ways of improving them. This was especially helpful when trying to debug my C code, as some solutions would not work initially due to a simple error that only became obvious thorough rigorous testing.

If I were to attempt a similar activity in this future, I would take the approach of drawing the logic circuit of the final task and apply it to the earlier tasks. Although I was able to produce solutions for them, I believe they could have been solved slightly easier if I had produced logic circuits for them beforehand. I would also spend more time researching online the different assembly registers, as I suspect this would be the best way for me to fix the main issue that I had with creating my final program. Overall, I am very happy with the knowledge I have gained from this assignment.

Appendices

Appendix A: Task 7 Code

```
const byte numChars = 32;           // Creates a char array called 'numChars' with length 32.
char inputtedChars[numChars];       // An array to store the received chars from the user.

boolean allowConversion = false;

void setup() {
    Serial.begin(9600);
    Serial.println("Please input an integer: ");
}

int readInt(void) {
    if (allowConversion == true) {    // Allows the function to continue if input is fully read.

        int convertedToInt;
        char copyStr[32];             // Creates a char array called 'copyStr' with length 32.

        strcpy(copyStr, inputtedChars); // Copies 'inputtedChars' into a String called 'copyStr'.
        convertedToInt = atoi(copyStr); // Converts the String 'copyStr' into an int.

        Serial.print("String Value = ");
        Serial.print(inputtedChars);
        Serial.print(", Integer Value = ");
        Serial.println(convertedToInt);
        Serial.println("Please input an integer: ");

        allowConversion = false;       // Sets 'allowConversion' back to false so the function
                                        // won't run again until the next number is inputted.
        return(0);
    }
}

void detectInput() {
    static byte spaceLimit = 0;
    char stoppingPoint = '\n';
    char inputChar;

    while (Serial.available() > 0 && allowConversion == false) {

        inputChar = Serial.read();

        if (inputChar != stoppingPoint) { // If the program has yet to reach the end of the users
                                            // input, the program will continue to loop until so.

            inputtedChars[spaceLimit] = inputChar;
            spaceLimit++;

            if (spaceLimit >= numChars) {
                spaceLimit = numChars - 1;
            }
        }
        else {

            inputtedChars[spaceLimit] = '\0'; // Terminates string, reaching the end of the input.
            spaceLimit = 0;
            allowConversion = true;           // Causes the program to convert the user's input.
        }
    }
}

void loop() {
    detectInput();
    readInt();
}
```

Appendix B: Task 8 Code

```
const byte numChars = 32;          // Creates a char array called 'numChars' with length 32.
char inputtedChars[numChars];      // An array to store the received chars from the user.

void setup() {

  Serial.begin(9600);
  Serial.print("Input a number in the range -128 to 127: ");

}

String decimalToBinary(String message, byte n) {

  int c = 0;
  int intInput = n;
  int intCorrected = intInput;
  char binaryNumber[8] = {0};

  if (intInput >= 128) {

    intCorrected = (256 - intInput) / -1;

  }

  Serial.print(message);

  if (intInput != intCorrected) {

    Serial.print("1");          // Prints a '1' to the terminal as part of the binary output.

  } else {

    Serial.print("0");          // Prints a '0' to the terminal as part of the binary output.

  }

  while (c < 7) {

    Serial.print(intInput >> (6-c)&1); //Gets bit c of int check.

    n /= 2;
    c += 1;

  }

  c = 0;

  Serial.print(" ");
  Serial.print(intCorrected);
  Serial.println(binaryNumber);

  Serial.print("Input a number in the range -128 to 127: ");

}

int readInt(void) {

  int convertedToInt;
  char copyStr[32];                // Creates a char array 'copyStr' with length 32.
  strcpy(copyStr, inputtedChars); // Copies 'inputtedChars' into the String 'copyStr'.
  convertedToInt = atoi(copyStr);   // Converts the String 'copyStr' into an int.

  if (convertedToInt < -128 || convertedToInt > 127) {

    Serial.println("Value out of range.");
    Serial.print("Input a number in the range -128 to 127: ");

  }

  else {

    decimalToBinary("bin = ", convertedToInt);    // Calls the decimalToBinary() function.

  }

}
```

```

}

void detectInput() {

    static byte spaceLimit = 0;
    char stoppingPoint = '\n';
    char inputChar;

    while (Serial.available() > 0) {

        inputChar = Serial.read();

        if (inputChar != stoppingPoint) {           // If the program has yet to reach the end of the
                                                    users input, the program continues to loop.

            inputtedChars[spaceLimit] = inputChar;
            spaceLimit++;

            if (spaceLimit >= numChars) {
                spaceLimit = numChars - 1;
            }
        }

        else {

            inputtedChars[spaceLimit] = '\0';       // Terminates the string, reaching the end of the
                                                    users input.

            spaceLimit = 0;
            readInt();                               // Calls the readInt() function.
        }
    }
}

void loop() {

    detectInput();

}

```

Appendix C: Task 9 Code

```
boolean startMessage = true;

int loopNumber = 1;

int binaryInt1;
int binaryInt2;
int binaryInt3;

String binaryBR;
String binaryAD;

String B;
String R;
String A;
String D;

const byte numChars = 32;           // Creates a char array called 'numChars' with length
32.
char inputtedChars[numChars];       // An array to store the received chars from the user.

void setup() {
    Serial.begin(9600);
}

void trace(word BR, word AD, byte n) {
    binaryBR = decimalToBinary(BR);  // Calls the decimalToBinary(int) function to produce
                                     a binary value in the form of a string labelled
                                     'binaryBR' from an interger input.
    binaryAD = decimalToBinary(AD);  // Calls the decimalToBinary(int) function to produce
                                     a binary value in the form of a string labelled
                                     'binaryAD' from an interger input.

    for (int x = 0; x < 8; x++) {
        B.concat(binaryBR.charAt(x)); // Amends the first 8 chars (the high byte) from the
                                     string 'binaryBR' to the variable B.
        R.concat(binaryBR.charAt(x+8)); // Amends the last 8 chars (the low byte) from the
                                     string 'binaryBR' to the variable R.
        A.concat(binaryAD.charAt(x));  // Amends the first 8 chars (the high byte) from the
                                     string 'binaryAD' to the variable A.
        D.concat(binaryAD.charAt(x+8)); // Amends the last 8 chars (the low byte) from the
                                     string 'binaryAD' to the variable D.
    }

    Serial.print("\nn=");
    Serial.print(n);
    Serial.print(" D=");
    Serial.print(D);
    Serial.print(" B=");
    Serial.print(B);
    Serial.print(" A=");
    Serial.print(A);
    Serial.print(" R=");
    Serial.print(R);
    Serial.print("\n\n");
}

String decimalToBinary(byte n) {
    int c = 0;
    int bitNum = 16;
    int intInput = n;
    String binaryString = "";

    while (c < (bitNum - 1)) {        // Loops until all of the bits have been checked.

        int digit = (intInput >> ((bitNum - 2) - c) & 1); // Checks if the digit for the
                                                         selected bit should be 0 or 1.
        String digitString = String(digit);               // Converts the digit to a string.
    }
}
```

```

        binaryString += digitString; // Adds the previously
converted digit onto the final binary output string.

        c += 1;

    }

    if (intInput >= 0) {
        binaryString = "0" + binaryString; // Places a '0' on the start
of the binary output if the number is positive.
    }
    else {
        binaryString = "1" + binaryString; // Places a '1' on the start
of the binary output if the number is negative.
    }

    return(binaryString); // Returns the final binary
number in the form of a string.
}

int readInt(void) {

    int convertedToInt;
    char copyStr[32]; // Creates a char array called
'copyStr' with length 32.

    strcpy(copyStr, inputtedChars); // Copies the String
'inputtedChars' into another String called 'copyStr'.
    convertedToInt = atoi(copyStr); // Converts the String
'copyStr' into an int.

    if (loopNumber == 1 || loopNumber == 2) {

        if (convertedToInt < -32768 || convertedToInt > 32767) {

            Serial.println("Value out of range.");
            loopNumber -= 1;

        }
        else {

            return(convertedToInt);

        }
    }
    else {

        if (convertedToInt < 1 || convertedToInt > 8) {

            Serial.println("Value out of range.");
            return(-1);

        }
        else {

            Serial.println(convertedToInt);
            return(convertedToInt);

        }
    }
}

void detectInput() {

    static byte spaceLimit = 0;
    char stoppingPoint = '\n';
    char inputChar;

    while (startMessage == true) {

        if (loopNumber == 1) {
            Serial.print("Input BR in range -32768 to 32767: ");
        }
    }
}

```

```

else if (loopNumber == 2) {
    Serial.print("Input AD in range -32768 to 32767: ");
}
else {
    Serial.print("Input n in range 1 to 8: ");
}

startMessage = false;
}

while (Serial.available() > 0) {

    inputChar = Serial.read();

    if (inputChar != stoppingPoint) {        // If the program has yet to reach the end of
                                                the input, the program will continue to loop.

        inputtedChars[spaceLimit] = inputChar;
        spaceLimit++;

        if (spaceLimit >= numChars) {
            spaceLimit = numChars - 1;
        }
    }

    else {

        inputtedChars[spaceLimit] = '\0';    // Terminates the string, reaching the end of the
                                                users input.

        spaceLimit = 0;

        if (loopNumber == 1) {

            binaryInt1 = readInt();           // Calls the readInt() function, converting the
                                                users string input into an interger and storing
                                                it as 'binaryInt1'.

            Serial.println(binaryInt1);
            B = "";
            R = "";

        } else if (loopNumber == 2) {

            binaryInt2 = readInt();           // Calls the readInt() function, converting the
                                                users string input into an interger and storing
                                                it as 'binaryInt2'.

            Serial.println(binaryInt2);
            A = "";
            D = "";

        } else {

            binaryInt3 = readInt();           // Calls the readInt() function, converting the
                                                users string input into an interger and storing
                                                it as 'binaryInt3'.

            if (binaryInt3 != -1) {

                loopNumber = 0;
                trace(binaryInt1, binaryInt2, binaryInt3);    // Calls the trace(int, int, int)
                                                                function, taking the inputs from
                                                                the user to produce an output.

            } else

                loopNumber = -1;

        }

        loopNumber += 1;
        startMessage = true;

    }
}

void loop() {

    detectInput();    // Constantly calls detectInput() to check if the user has inputted.

}

```