MONASH University

FIT1047 Applied Session Week 2

REPRESENTATION OF CHARACTERS AND NUMBERS, BASIC BOOLEAN LOGIC

OBJECTIVES

- The purpose of this applied session is to work on number systems conversion, apply Boolean logic, logic gates to build a circuit in Logisim

INSTRUCTIONS

- Download Logisim evolution from Moodle
- You may work in a small group.

Activity 1: Representation of Characters (using ASCII) and Numbers

In our computers, characters are represented using ASCII/Unicode and numbers are represented using 2's complement representation.

(Note: Using MARIE we can show how the computer keyboard characters are represented in computer memory.)

Task 1.1: Representation of Characters.

If your computer uses 7-bit ASCII, how would the computer represent the string "FA5"?

Sample Solution:
Please refer to the table at the end of this document.
F-> 100 0110
A-> 100 0001
5-> 011 0101

"FA5" will be represented as "100 0110 100 0001 011 0101" in the computer.

MONASH University

Task 1.2: Integer Numbers (4-bit) Representing Unsigned, Signed, 1's & 2's complement representations.

Create a table with all possible 4-bit binary values in one column and the (equivalent) decimal they represent in different columns using the following different notations:

(i) unsigned integer,
(ii) sign-magnitude notation,
(iii) 1's complement representation,
(iv) 2's complement representation.

	4-bit Binary Number 16 Numbers	Unsigned Representation 16 Numbers [0-15]	Signed Representation 16 Numbers [-7 -0][+0 +7]	1's Complement Representation 16 Numbers [-7 -0][+0 +7]	2's Complement Representation 16 Numbers [-8 -1][0 +7]
1	0000	0	+0	+0	0
2	0001	1	+1	+1	+1
3	0010	2	+2	+2	+2
4	0011	3	+3	+3	+3
5	0100	4	+4	+4	+4
6	0101	5	+5	+5	+5
7	0110	6	+6	+6	+6
8	0111	7	+7	+7	+7
9	1000	8	-0	-7	-8
10	1001	9	-1	-6	-7
11	1010	10	-2	-5	-6
12	1011	11	-3	-4	-5
13	1100	12	-4	-3	-4
14	1101	13	-5	-2	-3
15	1110	14	-6	-1	-2
16	1111	15	-7	-0	-1

Task 1.3: Integer Numbers (8-bit)

(a) Assume you have 8 bits to represent binary numbers. What decimal value does the binary number 10110101_2 have in the different notations?

(i) unsigned binary number
(ii) sign-magnitude notation
(iii) 1's complement
(iv) 2's complement

Sample Solution:

(i) In an unsigned binary number, all the bits represent the number.

$1011\ 0101_2$

$= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 181_{10}$

(ii) In this notation, the left-most bit represents the sign of the number: 0 for positive and 1 for negative. Thus, the number is negative.

The actual value of the number is then just the binary representation of the remaining 7 bits:.

$1011\ 0101_2$

$= -(0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)$

$= -53_{10}$

(iii) 1's complement Representation

Let N = given number 10110101_2.

The 1 in the most significant bit indicates that N is a negative number. Thus, the number was derived by flipping all bits of the positive number with the same magnitude. 1's complement of N will give this corresponding positive number (i.e. flipping all 1s to 0s and 0s to 1s).

1's complement of N is 01001010.

$01001010_2 = 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 74_{10}$

Thus, the actual value is -74.

(iv) 2's complement representation

Let N = given number 10110101_2.

The 1 in the most significant bit indicates that N is a negative number. Calculating 2's complement of N backwards (which is the same operation as the 2's complement) will give the corresponding positive number (i.e. flipping all bits and then adding one bit, just discarding any carry bit).

2's complement of N is $01001010 + 1 = 01001011$.

(b) Represent the number -92_{10} in

(i) 8-bit signed magnitude

(ii) 8-bit 2's complement

Sample Solution:

(i) 8-bit signed magnitude

Just convert +92 to binary (getting 01011100) and put sign bit 1 out the front to get −92 (11011100).

Note: *Here we don't need to pad with leading 0s, but the students are reminded the need for that;* ***i.e.*** *in order to get required number of bits.*

(ii) 8-bit 2's complement

Again, start with +92 in 8 bits. (For positive numbers, signed magnitude is same as 2's complement. For NEGATIVE numbers, they are quite different.)

Then negate, by complementing each bit then numerically adding 1.

10100011+1 = 10100100

MONASH University

Task 1.4: Addition and subtraction of signed numbers.

(a) Perform the following tasks using binary arithmetic with 8-bit 2's complement representation:
(i) 33 + 92
(ii) 33 - 92

Sample Solution:

(i) 33 + 92

```
92      01011100
33      00100001
----------------
        01111101
```

(ii) 33 - 92

Compute 33 - 92 by negating 92 (which we did in 1(b)) and finding 33 + (-92) by addition.

```
-92     10100100
33      00100001
----------------
        11000101
```

(b) Convert the following numbers to 6-bit 2's complement notation and add them:
(i) -16+11
(ii) 16-11

Sample Solution:

(i) -16 + 11

SOLUTION:

-16 = 110000; 11 = 001011

```
110000
001011
-------
111011
```

(ii) 16 - 11

16 = 010000; -11= 110100+1 = 110101

```
010000
110101
-------
000101
```

MONASH University

Task 1.4: Concept of "Ranges of Numbers and Overflow".

(a) What are the ranges of numbers that can be represented in a computer if it is using:
- (i) 4-bit 2's complement representation
- (ii) 6-bit 2's complement representation
- (iii) 8-bit 2's complement representation

Sample Solution:

4-bit Number System ->

Total Numbers: 16 ($=2^4$), Range: [-8 -> -1], [0 -> +7]

6-bit NS ->

Total Numbers: 64 ($=2^6$), Range: [-32 -> -1], [0 -> + 31]

8-bit NS ->

Total Numbers: 256 ($=2^8$), Range: [-128 -> -1], [0 -> + 127]

(b) What happens if you try to calculate 92+92 (using binary arithmetic with 8-bit 2's complement representation)? Can the result of this operation be accepted? How would you explain this operation referring to the "ranges of numbers" in 8-bit 2's complement representation?

Sample Solution:

92 + 92 = 184 (beyond the range of 8-bit number system). This operation will cause overflow and the result must not be accepted. We can also verify the overflow situation by looking at the signs of these three numbers, two operands and the result

92 -> 0101 1100
92 -> 0101 1100
—----------
1011 1000

MONASH University

Activity 2: Boolean Logic, Logic Gates and Logisim

Start logisim by clicking on the file or executing:

java -jar logisim-evolution.jar

Task 2.1: Using Logisim

Make yourself familiar with logisim by doing a few very easy tasks:

- place one AND gate, 2 inputs and one output
- connect them
- poke the inputs to try simulation
- do the same with OR and NOT
- try some combinations of gates

Sample Solution:
Draw any simple logic circuit in logisim.

Task 2.2: Build logic circuits using Logisim

In addition to AND, OR and NOT, there are a number of other Boolean logic operators. Examples are XOR (the exclusive OR), and NAND (a negated AND) operation. Their truth tables are shown in the tables below.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

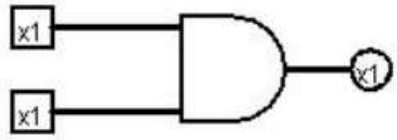
(a) XOR

A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

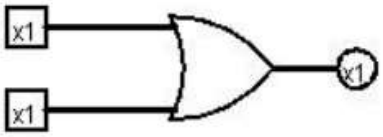
(b) NAND

Build XOR and NAND logic in logisim only using AND, OR, and NOT gates. Use the simulation to test your result.
Sample Solution:

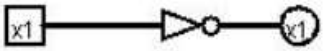
AND Gate



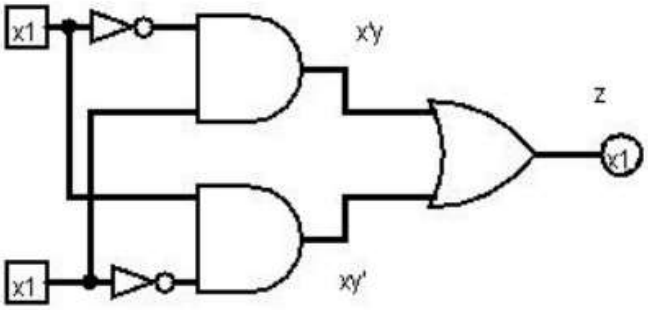
OR Gate



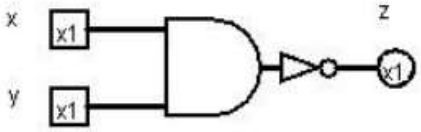
NOT Gate



x XOR y = x AND y' OR x' AND y



x NAND y = NOT (x AND y)



==========

ASCII Table (7-bit)

MONASH University

Bit pattern	Character	Decimal value	Bit pattern	Character	Decimal value
0100000	SPACE	32	0100001	!	33
0100010	"	34	0100011	#	35
0100100	$	36	0100101	%	37
0100110	&	38	0100111	'	39
0101000	(	40	0101001	)	41
0101010	*	42	0101011	+	43
0101100	,	44	0101101	-	45
0101110	.	46	0101111	/	47
0110000	0	48	0110001	1	49
0110010	2	50	0110011	3	51
0110100	4	52	0110101	5	53
0110110	6	54	0110111	7	55
0111000	8	56	0111001	9	57
0111010	:	58	0111011	;	59
0111100	<	60	0111101	=	61
0111110	>	62	0111111	?	63
1000000	@	64	1000001	A	65
1000010	B	66	1000011	C	67
1000100	D	68	1000101	E	69
1000110	F	70	1000111	G	71
1001000	H	72	1001001	I	73
1001010	J	74	1001011	K	75
1001100	L	76	1001101	M	77
1001110	N	78	1001111	O	79
1010000	P	80	1010001	Q	81
1010010	R	82	1010011	S	83
1010100	T	84	1010101	U	85
1010110	V	86	1010111	W	87
1011000	X	88	1011001	Y	89
1011010	Z	90	1011011	[	91
1011100	\	92	1011101	]	93
1011110	^	94	1011111	_	95
1100000	`	96	1100001	a	97
1100010	b	98	1100011	c	99
1100100	d	100	1100101	e	101
1100110	f	102	1100111	g	103
1101000	h	104	1101001	i	105
1101010	j	106	1101011	k	107

1101100	l	108	1101101	m	109
1101110	n	110	1101111	o	111
1110000	p	112	1110001	q	113
1110010	r	114	1110011	s	115
1110100	t	116	1110101	u	117
1110110	v	118	1110111	w	119
1111000	x	120	1111001	y	121
1111010	z	122	1111011	{	123
1111100	|	124	1111101	}	125