Lecture 6: Signed Numbers & Arithmetic Circuits

Professor Peter Cheung
Department of EEE, Imperial College London

(Floyd 2.5-2.7, 6.1-6.7)
(Tocci 6.1-6.11, 9.1-9.2, 9.4)

Points Addressed in this Lecture

- Representing signed numbers
- Two's complement
- Sign Extension
- Addition of signed numbers
- Multiplication by -1
- Multiplication and division by integer powers of 2
- Adder & subtractor circuits
- Comparators
- Decoders
- Encoders

Binary Representations (Review)

- We have already seen how to represent numbers in binary
- Review

$(179)_{10}$ is $(10110011)_2$ is $(B3)_{16}$ is $(263)_8$

– HEX: 1 0 1 1 | 0 0 1 1 → B | 3

– OCTAL 1 0 | 1 1 0 | 0 1 1 → 2 | 6 | 3

- BCD (Binary Coded Decimal)
  - Each digit of a decimal number is coded using Binary
  - The 4 bit binary words are joined to make the full decimal number
  - E.g.
    - 987 in decimal
    - 9 : 1001
    - 8 : 1000
    - 7 : 0111
  - So 987 in decimal becomes 1001 1000 0111 in BCD

Summary

Decimal	Binary	HEX	BCD	Octal
0	00000	0	0000 0000	0
1	00001	1	0000 0001	1
2	00010	2	0000 0010	2
3	00011	3	0000 0011	3
4	00100	4	0000 0100	4
5	00101	5	0000 0101	5
6	00110	6	0000 0110	6
7	00111	7	0000 0111	7
8	01000	8	0000 1000	10
9	01001	9	0000 1001	11
10	01010	A	0001 0000	12
11	01011	B	0001 0001	13
12	01100	C	0001 0010	14
13	01101	D	0001 0011	15
14	01110	E	0001 0100	16
15	01111	F	0001 0101	17
16	10000	10	0001 0110	20

Signed numbers Basics

- So far, numbers are assumed to be unsigned (i.e. positive)
- How to represent signed numbers?
- Solution 1: **Sign-magnitude** - Use one bit to represent the **sign**, the remain bits to represent **magnitude**

7 6 0

0 = +ve
1 = -ve

| s | magnitude |

+27 = 0001 1011$_b$
-27 = 1001 1011$_b$

  - Problem: need to handle sign and magnitude separately.
- Solution 2: **One's complement** - If the number is negative, invert each bits in the magnitude

+27 = 0001 1011$_b$
-27 = 1110 0100$_b$

  - Not convenient for arithmetic - add 27 to -27 results in 1111 1111$_b$
  - Two zero values

Two's complement

- Solution 3: **Two's complement** - represent negative numbers by taking its magnitude, invert all bits and add one:

Positive number	**+27 = 0001 1011$_b$**
Invert all bits	**1110 0100$_b$**
Add 1	**-27 = 1110 0101$_b$**

Unsigned number: 2^7 2^6 … 2^0

Signed 2's complement: -2^7 2^6 … 2^0 (s)

$$x = -b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \bullet\bullet\bullet + b_1 2^1 + b_0 2^0$$

Examples of 2's Complement

- A common method to represent -ve numbers:
  - use half the possibilities for positive numbers and half for negative numbers
  - to achieve this, let the MSB have a negative weighting
- Construction of 2's Complement Numbers
  - 4-bit example

Decimal	2's Complement (Signed Binary)			
	-8	+4	+2	+1
5	0	1	0	1
-5	1	0	1	1
7	0	1	1	1
-3	1	1	0	1

Why 2's complement representation?

- If we represent signed numbers in 2's complement form, subtraction is the same as addition to negative (2's complemented) number.

27	$0001\ 1011_b$
- 17	$0001\ 0001_b$
+ 10	$0000\ 1010_b$

+27	$0001\ 1011_b$
+ - 17	$1110\ 1111_b$
+10	$0000\ 1010_b$

- Note that the range for 8-bit unsigned and signed numbers are different.
  - 8-bit unsigned: **0 …… +255**
  - 8-bit 2's complement signed number: **-128 …… +127**

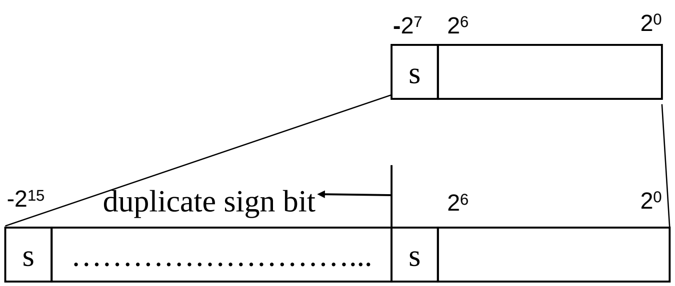
Comparison Table

- Note the "**wrap-around**" effect of the binary representation
  - i.e. The top of the table wraps around to the bottom of the table

Unsigned	Binary	2' comp
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000	0
15	1111	-1
14	1110	-2
13	1101	-3
12	1100	-4
11	1011	-5
10	1010	-6
9	1001	-7
8	1000	-8

Sign Extension

- How to translate an 8-bit 2's complement number to a 16-bit 2's complement number?



- This operation is known as **sign extension**.

Sign Extension

- Sometimes we need to extend a number into more bits
- Decimal
  - converting 12 into a 4 digit number gives 0012
  - we add 0's to the left-hand side
- Unsigned binary
  - converting 0011 into an 8 bit number gives 00000011
  - we add 0's to the left-hand side
- For signed numbers we duplicate the sign bit (MSB)
- Signed binary
  - converting 0011 into 8 bits gives 00000011 (duplicate the 0 MSB)
  - converting 1011 into 8 bits gives 11111011 (duplicate the 1 MSB)
  - Called "**Sign Extension**"

Imperial College London

Signed Addition

- The same hardware can be used for 2's complement signed numbers as for unsigned numbers
  - this is the main advantage of 2's complement form
- Consider 4 bit numbers:
  - the Adder circuitry will "think" the negative numbers are 16 greater than they are in fact
  - but if we take only the 4 LSBs of the result (i.e. ignore the carry out of the MSB) then the answer will be correct providing it is with the range: -8 to +7.
- To add 2 n-bit signed numbers without possibility of overflow we need to:
  - sign extend to n+1 bits
  - use an n+1 bit adder

Imperial College London

Multiplication of Signed Numbers by -1

- Inverting all the bits of a 2's complement number X gives: -X-1 since adding it back onto X gives -1
- E.g.

0101	5	X
1010	-6	-X-1
1111	-1	-1

- Hence to multiply a signed number by -1:
  - first invert all the bits
  - then add 1
- Exception:
  - doesn't work for the maximum negative number
  - e.g. doesn't work for -128 in a 8-bit system

Imperial College London

Multiplication and Division by 2^N

- In decimal, multiplying by 10 can be achieved by
  - shifting the number left by one digit adding a zero at the LS digit
- In binary, this operation multiplies by 2
- In general, left shifting by N bits multiplies by 2^N
  - zeros are always brought in from the right-hand end
  - E.g.

Binary	Decimal
1101	13
11010	26
110100	52

Imperial College London

- Right shifting by N bits divides by 2^N
  - the bit which "falls off the end" is the remainder
  - sign extension must be maintained for 2's complement numbers
  - Decimal:

    $(486)_{10}$ divided by 10 gives 48 remainder 6
  - Unsigned:

    $(110101)_2$ divided by 2 gives 11010 remainder 1
    $(53)_{10}$ $(26)_{10}$

    $(110101)_2$ divided by 4 gives 1101 remainder 01
    $(53)_{10}$ $(13)_{10}$
  - Signed 2's Complement:

    $(110101)_2$ divided by 2 gives 111010 remainder 1
    $(-11)_{10}$ $(-6)_{10}$
    $(110101)_2$ divided by 4 gives 111101 remainder 01
    $(-11)_{10}$ $(-3)_{10}$

Imperial College London

Summary of Signed and Unsigned Numbers

Unsigned	Signed
MSB has a positive value (e.g. +8 for a 4-bit system)	MSB has a negative value (e.g. -8 for a 4-bit system)
The carry-out from the MSB of an adder can be used as an extra bit of the answer to avoid overflow	To avoid overflow in an adder, need to sign extend and use an adder with one more bit than the numbers to be added
To increase the number of bits, add zeros to the left-hand side	To increase the number of bits, sign extend by duplicating the MSB
Complementing and adding 1 converts X to (2^N - X)	Complementing and adding 1 converts X to -X

Imperial College London

Binary Addition

- Recall the binary addition process

A	1	0	0	1
+B	0	0	1	1
S	1	1	0	0

- LS Column has 2 inputs 2 outputs
  - Inputs: $A_0 \quad B_0$
  - Outputs: $S_0 \quad C_1$
- Other Columns have 3 inputs, 2 outputs
  - Inputs: $A_n \quad B_n \quad C_n$
  - Outputs: $S_n \quad C_{n+1}$
  - We use a "half adder" to implement the LS column
  - We use a "full adder" to implement the other columns
  - Each column feeds the next-most-significant column.

Imperial College London

Half Adder

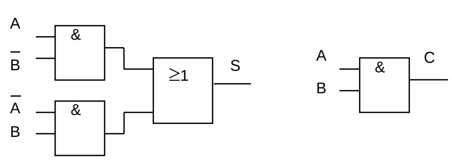
- Truth Table

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Boolean Equations

$$S = \overline{A}B + A\overline{B} = A \oplus B$$
$$C = AB$$

- Implementation



  - Note also XOR implementation possible for S

Imperial College London

Full Adder

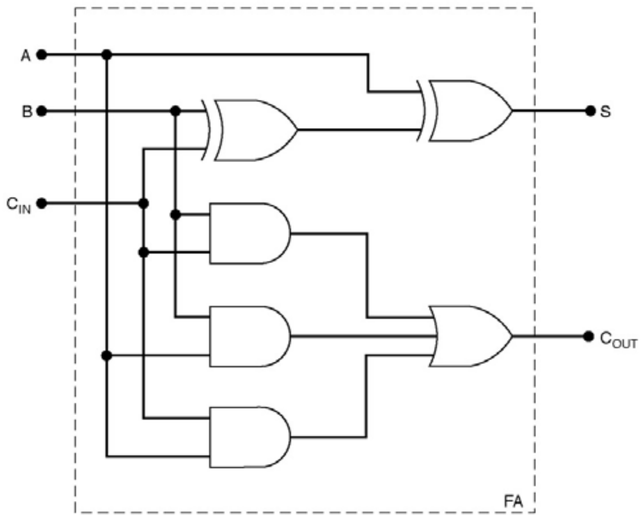
- Truth Table

A	B	C_i	S	C_o
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

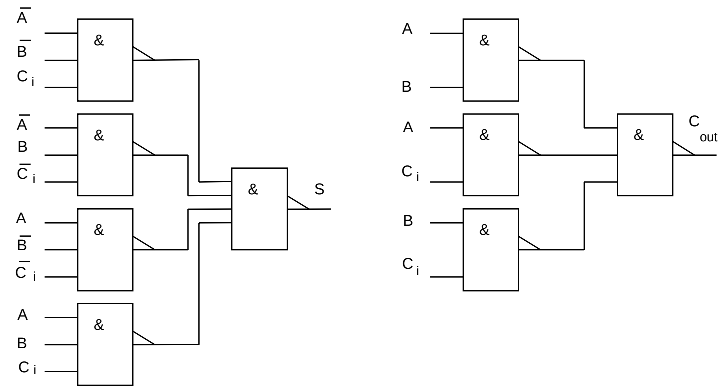
- Boolean Equations

$$S = \overline{A}.\overline{B}.C_i + \overline{A}.B.\overline{C_i} + A.\overline{B}.\overline{C_i} + A.B.C_i$$
$$= A \oplus B \oplus C_i$$
$$C_o = \overline{A}BC_i + A\overline{B}C_i + AB\overline{C_i} + ABC_i$$
$$= AB + AC_i + BC_i$$
$$= AB + C_i(A + B)$$

Complete circuitry for a FA



- Implementation (using NAND gates only)

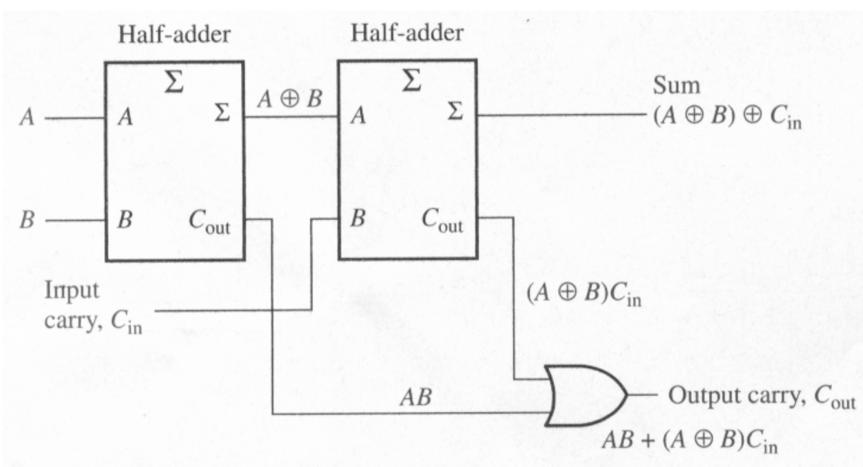


Full Adder from Half Adders

- Truth Table

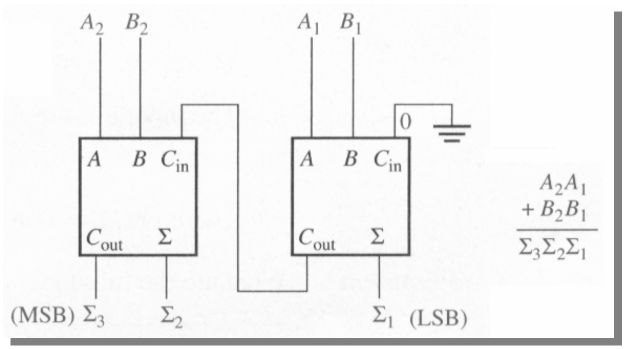
A	B	HA_s	HA_c	C_i	S	C_o
0	0	0	0	0	0	0
0	0	0	0	1	1	0
0	1	1	0	0	1	0
0	1	1	0	1	0	1
1	0	1	0	0	1	0
1	0	1	0	1	0	1
1	1	0	**1**	0	0	**1**
1	1	0	**1**	1	1	**1**

Full Adder from Half Adders

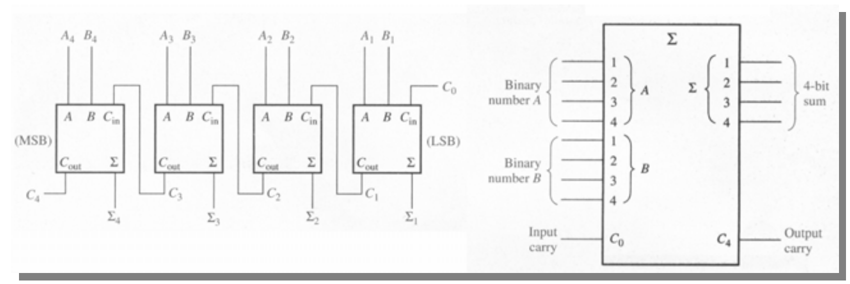


Parallel Adder

- Uses 1 full adder per bit of the numbers
- The carry is propagated from one stage to the next most significant stage
  - takes some time to work because of the carry propagation delay which is n times the propagation delay of one stage.

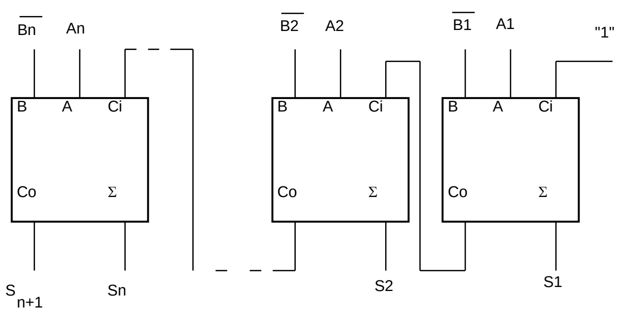


4-bit Parallel Binary Adders



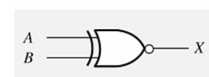
Parallel Subtraction using Parallel Adder

- Subtraction can be achieve by adding the complement
  - E.g.: 6 - 3 = 6 + (-3) = 3
- 2's complement :- invert all bits and then add 1
  - Use Carry-in of first stage for the "add 1"
  - Invert all the inputs bits of B



Comparators

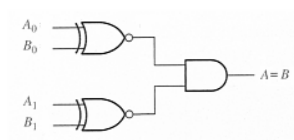
- 1-Bit Comparator



A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

The output is 1 when the inputs are equal

- 2-Bit Comparator



The output is 1 when $A_0 = B_0$ AND $A_1 = B_1$

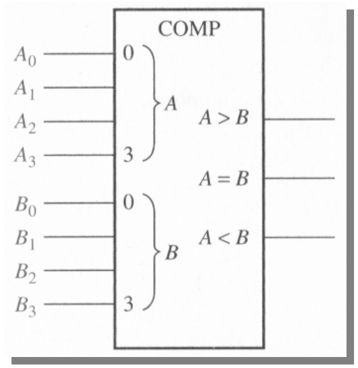
Imperial College London

Comparators

- 4-Bit Comparator

One of three outputs will be HIGH:

- A greater than B (A > B)
- A equal to B (A = B)
- A less than B (A < B)



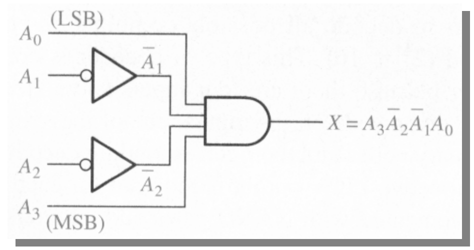
Imperial College London

Decoders

- Binary decoder

The output is 1 only when:

$A_0 = 1$
$A_2 = 0$
$A_3 = 0$
$A_4 = 1$



This is only one of an infinite number of examples

Imperial College London

Decoders

- 4-bit decoder

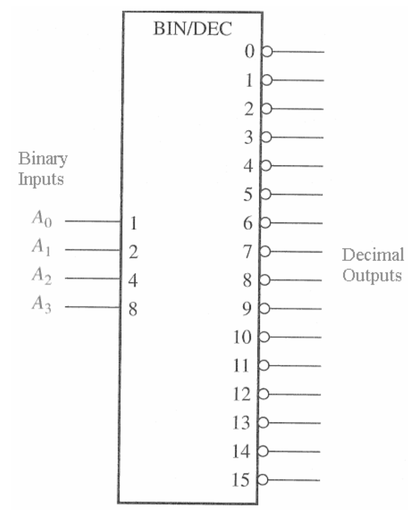
BINARY INPUTS				DECODING	OUTPUTS															
A_3	A_2	A_1	A_0	FUNCTION	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	1	1	1	$\bar{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	0	1	0	$A_3\bar{A}_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
1	0	1	1	$A_3\bar{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	0	0	$A_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	1	0	1	$A_3A_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	0	$A_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Logic Diagram

Imperial College London

Decoders

- 4-bit decoder
  - Binary inputs
  - Active-low outputs

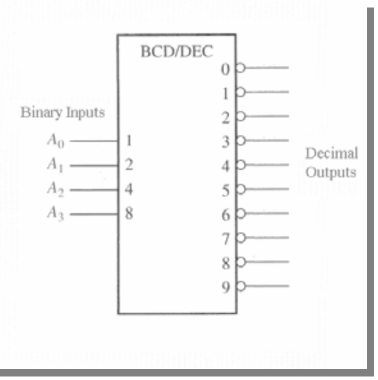


Truth Table

Decoders

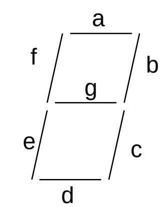
- BCD-to-decimal decoder

DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0	DECODING FUNCTION
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$
4	0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$
5	0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$
6	0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$
9	1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$



BCD-to-7 Segment Display Decoder

- LCD or LED displays can display digits made of up to 7 segments or lines
- Decode 4 bits BCD into 7 control signals using a BCD/7SEG decoder



Decoders

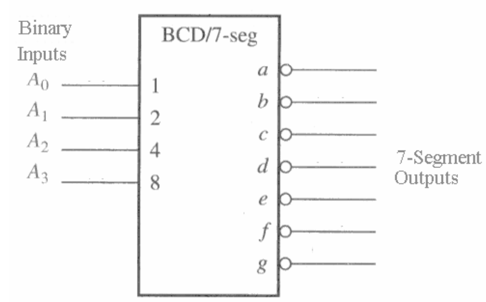
- BCD-to-7-segement decoder

DECIMAL DIGIT	INPUTS D	C	B	A	SEGMENT OUTPUTS a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

Logic Diagram

Decoders

- BCD-to-7-segement decoder



Truth Table

Imperial College London

Encoders

- Decimal-to-BCD encoder

DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1