

- What is Computer Science ??
- What is an Algorithm ?
- Bits and Number Bases
- Binary/Octal/Hexadecimal
- Logic Gates
- Truth Tables and Design
- Binary Addition
- Reading and Self Assessment

*These slides were based on those of C Johnson, E McCreath and W Liang.

What is Computer Science?

- A discipline which seeks to build a scientific foundation for things like:
 - Computer design.
 - Computational programming.
 - Information processing.
 - The algorithmic solution of problems.

What is an Algorithm?

- A set of steps that describe how to perform some task.
 - Cooking.
 - Finding your way.
 - Operating a machine.
 - Playing music.

What is an Algorithm?

- Algorithms are represented using programs.
- Algorithms + Programs = Software.
- The machine itself is the Hardware.

What is Computer Science?

- Computer Science has established itself as the “science of algorithms”.
- It's scope is broad; drawing from:
 - Mathematics.
 - Engineering.
 - Psychology.
 - Business Administration.
 - Linguistics.

What is Computer Science?

- Some questions that define the science of computing:
 - Which problems CAN be solved by algorithmic processes?
 - Can the discovery of algorithms be made easier?
 - How can the representation and communication of algorithms be improved?
 - How can our knowledge of algorithms be applied to provide better machines?
 - How can the characteristics of different algorithms be analysed and compared?

Number Bases

- 110 binary (base 2) expressed as a decimal is
$$1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 4 + 2 + 0 = 6.$$
- 345 octal (base 8) expressed as a decimal is
$$3 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 = 192 + 32 + 5 = 229.$$
- 345 decimal (base 10) is
$$3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 = 300 + 40 + 5 = 345.$$

Bits

- Inside today's computers, information is stored as patterns of 0's and 1's.
- These digits are called bits (binary digits).
- Although you might be inclined to associate bits with numeric values, they are really only symbols whose meaning depends on the application at hand.

Bits

- Sometimes, patterns of bits are used to represent numeric values; other symbols (such as characters in an alphabet and punctuation); images; sounds.
- Storing a bit in a machine requires a device that can be in one of two states, such as a switch, an electronically controlled relay, or other device.
- One state is used to represent 0, the other to represent 1.

(Brookshear)

Information

- A computer is a machine that operates upon information.
- This information must be stored, processed and transferred via a physical medium.
- Binary information is the simplest way of storing information.

Information Storage, Processing, and Transfer

- The physical medium that information is stored in places limitations on the amount that can be stored.
- The physical medium that information is processed within places bounds on the speed at which it may be processed.
- Finally, the physical medium that information is transferred via places limitations on the speed (or rate) it can be transferred.

Binary Information

- A single bit is in one of two states, these are usually labeled either 0 or 1.
- By collecting a number of bits together more than two states can be represented within a computer.
- 0000 → 0, 0001 → 1, 0010 → 2, . . . , 1111 → 15.
- How many states can n bits represent?

Binary Information

- A single bit is in one of two states, these are usually labeled either 0 or 1.
- By collecting a number of bits together more than two states can be represented within a computer.
- 0000 → 0, 0001 → 1, 0010 → 2, . . . , 1111 → 15.
- n bits can represent 2^n states.

The Byte

- A byte consists of 8 bits.
- A byte forms a basic unit of binary information.
- A byte can represent 256 states.
- A nibble refers to 4 bits.
- 2, 4, or 8 bytes are placed together to form a word.
- The word size is dependent on the architecture in question.

Decimal

- Binary is cumbersome for humans to manipulate.
- One approach is to use decimal.
- Binary can be changed to decimal:

$$b_{n-1}b_{n-2}\dots b_1b_0 \rightarrow \sum_{k=0}^{n-1} b_k 2^k.$$

- Decimal can be changed to binary:

$$b_k = \frac{d}{2^k} \bmod 2.$$

- However, this is a slow and error prone task for humans to undertake.

Hexadecimal

- A more common way of representing binary information is using base 16.
- This is known as Hexadecimal.
0000→0 0100→4 1000→8 1100→C
0001→1 0101→5 1001→9 1101→D
0010→2 0110→6 1010→A 1110→E
0011→3 0111→7 1011→B 1111→F
- It is easy and quick to change between Hex and binary. e.g. 0001101011000100 → 1AC4

Octal

- Octal is sometimes used. Octal is base 8.
000→0
001→1
010→2
011→3
100→4
101→5
110→6
111→7
- Changing between Octal and Binary is also simple.
e.g. 010111001 → 271

Memory

- A contiguous list of bytes forms the memory within the computer.
- These bytes are indexed with binary addresses.
- Larger amounts of memory are specified using the prefixes: K(kilo), M(mega), G(giga) and T(tera).
- These have different meanings from their standard scientific meaning.

Memory

Prefix	Scientific	Computer
<i>K</i>	$10^3 = 1000$	$2^{10} = 1024$
<i>M</i>	$10^6 = 1000000$	$2^{20} = 1048576$
<i>G</i>	$10^9 = 1000000000$	$2^{30} = 1073741824$
<i>T</i>	$10^{12} = 1000000000000$	$2^{40} = 1099511627776$

Memory

- These prefixes are used inconsistently.
 - *2KB* means 2×2^{10} bytes.
 - whereas *2Km* means 2×10^3 meters.
 - *5MB* means 5×2^{20} bytes.
 - whereas *5MHz* means 5×10^6 Hertz.
 - however *5Mb/s* means 5×10^6 bits per second!?

ASCII

- ASCII is the American Standard Code for Information Interchange.
- It is a 7 bit code providing 128 bit patterns.
- The International Organisation for Standardisation (ISO) 8859 standard includes several 8 bit extensions to the ASCII character set.
- This provides an additional 128 bit patterns.
- Unicode uses 16 bits providing 65,536 patterns.

Boolean Data

- The boolean data type is another basic data type within a computer system.
- A boolean variable is either "true" or "false".
- The operations performed on booleans include: not, or, and, xor, ...
- Booleans are central to conditional operations.
- A single bit would be sufficient to capture a boolean, however, an entire word is often used.

Interpretation of Binary Information

- An interpretation is placed upon a binary pattern.
- This gives the binary information meaning.
- It is important that the intended interpretation is used on this information.
- Consider the binary pattern: 01100001
- If it was interpreted as an ASCII character then it would be the letter 'a'.
- However, if it is interpreted as an integer then it would be the number 97 !!!!

Non-negative integers in binary

- The non-negative integers are the numbers 0,1,2,...
- This is an infinite set and hence it can't be represented with a fixed sized binary pattern.
- However, we can simply bound the integers represented.
- A binary pattern can be interpreted as a binary number. e.g. 01100001 is 97.

A Problem

- One problem that often arises is determining what order to interpret the bit pattern.
- Is the first bit the least or most significant bit?
- Also a question remains over the ordering of bytes when larger integers are considered.
- It is important that there is consistency across all interpretations.
- Fortunately the architecture/compiler does most of this work.

Fractions in binary

- Use a radix point (like a decimal point in decimal).
- Digits to the left represent the integer part.
- Digits to the right represent the fractional part.

$$101.101 =$$

$$\begin{aligned} & 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ & = 4 + 0 + 1 + \frac{1}{2} + 0 + \frac{1}{8} = 5\frac{5}{8} \end{aligned}$$

Negative integers in binary

- We also want to represent negative numbers.
- A simple way of doing this is to use the first bit to indicate the sign of the number.
- With 0 meaning a positive number and 1 meaning that it is negative.
- But we end up with two zeros!
 $10001010 \rightarrow -10$ $00000000 \rightarrow 0$ $10000000 \rightarrow 0$
- This wastes a bit pattern.

Two's Complement

- The most common approach for representing negative numbers is two's complement.
- The first bit also indicates that the number is negative.
- To negate a number you "invert all bits and add one".
- This removes the problem of multiple zeros.

Two's Complement

0111	→ 7	1111	→ -1
0110	→ 6	1110	→ -2
0101	→ 5	1101	→ -3
0100	→ 4	1100	→ -4
0011	→ 3	1011	→ -5
0010	→ 2	1010	→ -6
0001	→ 1	1001	→ -7
0000	→ 0	1000	→ -8

Two's Complement

- To alternate between positive and negative:
 - Copy bits from right to left until a '1' has been copied.
 - Complement (flip/toggle/1→ 0, 0→ 1) the remaining bits.

1011 \longleftrightarrow 0101

Decoding Two's Complement

- If the leading bit (the sign bit) is 0
 - Evaluate as a normal positive number.
- Otherwise
 - Note that the number is negative.
 - Negate the number.

Two's complement

- The two's complement approach simplifies arithmetic operations as an ordinary adder (operates on non-negatives) will correctly handle negative numbers.

$$\begin{array}{r}
 0011 \quad 3 \\
 +1011 \quad +(-5) \\
 \hline
 1110 \quad -2
 \end{array}$$

- Note: 3 minus 1 is the same as 3 plus -1.

Overflow

- There is a limit to the size of values that may be represented.
- Try adding 5+4 using 4-bit two's complement.
- Occurs when adding two positive or two negative numbers.
- Use larger bit patterns.
- 32 bits is common today.
- Can represent positive integers up to 2,147,483,647.

Excess Notation

- Start by writing down the 2^n n -bit binary numbers in order from 0 to 2^n-1 .
- The pattern consisting of a '1' followed by all 0's represents the number zero.
- Patterns following this represent $1, 2, 3, \dots, 2^{n-1} - 1$.
- Patterns preceding this represent $-1, -2, -3, \dots, -2^{n-1}$.
- Known as excess 2^{n-1} notation when dealing with n bits.
- Why?

Excess Notation

Excess eight	
Bit Pattern	Value Represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

Excess four	
Bit Pattern	Value Represented
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

Floating Point Numbers

- Floating point numbers provide a way of approximating continuous/real numbers.
- The binary pattern is divided up into three parts; the sign bit, the exponent and the mantissa.



- This representation works in a similar way to that of scientific notation.

$$0.314 \times 10^1$$

Floating Point Numbers

- The IEEE Standard Double is 64 bits with a 52 bit mantissa and a 11 bit exponent.
- For simplicity we will use a 8 bits with a 3 bit exponent and a 4 bit mantissa.
- If m is the mantissa, and e the exponent, the binary representation of the floating point number is

$$0.m \times 2^{e'}$$

- where e' is the excess four notation of e .
- Note in general use excess $2^{\text{number of bits in exponent}-1}$.

Floating Point Numbers

- Evaluate the binary number 01101011
- Sign is 0 (so positive)
- Exponent is 110
- Mantissa is 1011
- 110 (exponent) in excess four is 2
- Answer is $0.1011 \times 2^2 = 10.11 = 2.75$

Floating Point Numbers

- Evaluate the binary number 00111100
- Sign is 0 (so positive)
- Exponent is 011
- Mantissa is 1100
- 011 (exponent) in excess four is -1
- Answer is $0.1100 \times 2^{-1} = 0.01100 = 0.375$

Floating Point Numbers

- To store a value using floating point notation, reverse the process.
- Encode into binary (call this x).
- Copy the bit pattern into the mantissa field from left to right **starting with the leftmost '1'** in the binary representation.

Floating Point Numbers

- Imagine a radix point at the start of the mantissa field.
- How many times does this point need to move in order to obtain the original binary number (x).
- The exponent is this number in excess four notation.
- The number is positive if you need to move the radix point to the right.
- Negative if you need to move it to the left.

Floating Point Numbers

- Store 1.125 (decimal) as binary
- In binary this is 1.001
- The mantissa will be 1001
- To get from .1001 to 1.001 we need to move the radix point to the right by one place
- +1 in excess four notation is 101 (this is the exponent)
- Sign bit is 0 as number is positive
- So the answer is 01011001

Floating Point Numbers

- Store 0.375 (decimal) as binary
- In binary this is 0.011
- The mantissa will be 1100
- To get from .1100 to .011 we need to move the radix point to the left by one place
- -1 in excess four notation is 011 (this is the exponent)
- Sign bit is 0 as number is positive
- So the answer is 00111100

Normalised Form

- Evaluate the binary numbers
 - 01001000
 - 01010100
 - 01100010
 - 01110001

Normalised Form

- Evaluate the binary numbers
 - 01001000
 - $01001000 = +.1000 \times 2^0 = 0.1 = \frac{1}{2}$
 - 01010100
 - $01010100 = +.0100 \times 2^1 = 0.1 = \frac{1}{2}$
 - 01100010
 - $01100010 = +.0010 \times 2^2 = 0.1 = \frac{1}{2}$
 - 01110001
 - $01110001 = +.0001 \times 2^3 = 0.1 = \frac{1}{2}$

Floating Point Numbers

- Store 0.375 (decimal) as binary
- In binary this is 0.011
- **The mantissa will be 1100**
- To get from .1100 to .011 we need to move the radix point to the left by one place
- -1 in excess four notation is 011 (this is the exponent)
- Sign bit is 0 as number is positive
- So the answer is 00111100

Truncation Errors

- Try to store $2\frac{5}{8}$ (decimal) as an 8 bit binary number.
- In binary this is 10.101
- The mantissa becomes 1010 (losing the final '1' and 0.125 from the number!).
- The exponent is 110
- The sign bit is 0.
- We end up with 01101010 which is $2\frac{1}{2}$ not $2\frac{5}{8}$!
- This is called a truncation error (or round-off error).

Layers

- A computer system can be divided up into layers.
- Each layer is built upon the resources the layer below provides.
- This is a common design approach for complex systems, as it simplifies and partitions the overall design.

Layers

Applications
Operating System
Higher Level Language
Assembly Language
Machine Code
Architecture
Logic Components
Logic Gates
Micro-electronics
Semiconductor physics

Digital Logic

- Digital logic is concerned with the design of components such as: memory, adders, instruction decoders, . . . hence digital logic is the foundation for computer hardware.
- The electronics layer provides the basic gates.
- These gates are combined to form logic components.
- The logic is simple and based on Boolean logic.
(George Boole (1815–1864) was first to frame logic as an algebra.)

The Transistor

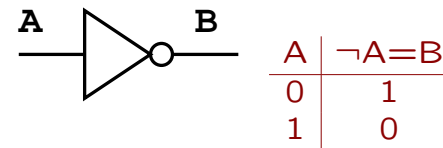
- The invention of the transistor in 1947 opened up the way for devices with millions of switching elements.
- There are a number of different commonly used technologies. These include:
 - Transistor-transistor logic (TTL) family.
 - Complementary Metal Oxide Semiconductors (CMOS).
- These different technologies all provide the same logic gates.

Logic Gates

- A device that produces the output of a Boolean when given the operation's input values is called a gate.
- The different logic gates are given names according to their function in Boolean algebra.
- NOT, AND, OR are basic functions.
- NAND, NOR, XOR are convenient alternatives.
- The choice of electronics determines which sets of logic gates are easier to create.

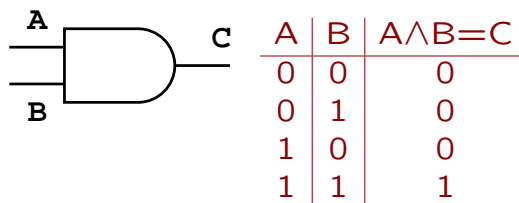
NOT

- The NOT operator evaluates to true when the input is false, otherwise it evaluates to false.
- NOT has one input and one output.
- NOT can be written as the operator \neg .



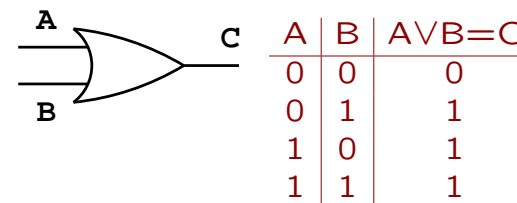
AND

- The AND operator evaluates to true when all of its inputs are true.
- AND has more than one input.
- AND can be written as the operator \wedge .



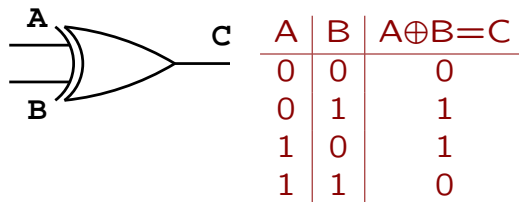
OR

- The OR operator evaluates to true when any of its inputs are true.
- OR has more than one input.
- OR can be written as the operator \vee .



XOR

- The XOR operator evaluates to true when exactly one of its inputs is true.
- XOR means “Exclusive OR”.
- XOR can be written as the operator \oplus .



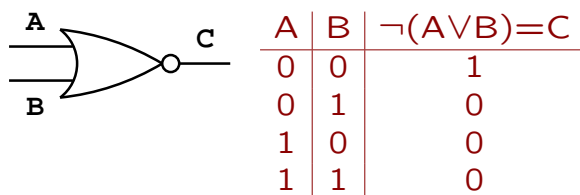
NAND

- The NAND operator evaluates to true when any of its inputs are false.
- NAND has more than one input.
- This is the same as $\text{NOT}(A \text{ AND } B)$.



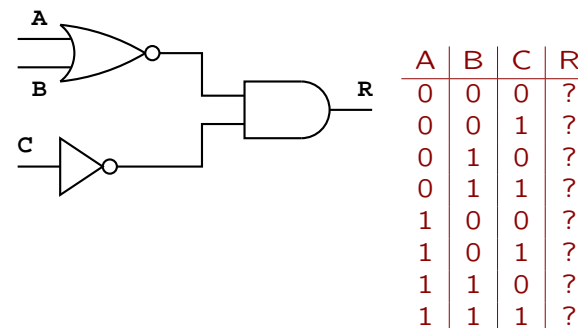
NOR

- The NOR operator evaluates to true when all of its inputs are false.
- NOR has more than one input.
- This is the same as $\text{NOT}(A \text{ OR } B)$.



Design to Truth Table

- By using the truth tables of the basic gates we can always construct a truth table from a circuit design.



Design to Truth Table

A	B	C	$A \vee B$	$\neg(A \vee B)$	$\neg C$	$(\neg(A \vee B)) \wedge \neg C$
0	0	0	0	1	1	1
0	0	1	0	1	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
1	0	0	1	0	1	0
1	0	1	1	0	0	0
1	1	0	1	0	1	0
1	1	1	1	0	0	0

Truth Table to Design

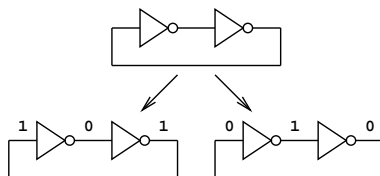
- From any truth table we can also design a set of gates that will evaluate the truth table.
- **This kind of design process goes beyond the scope of this subject.**

A	B	C	R
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

→ ????

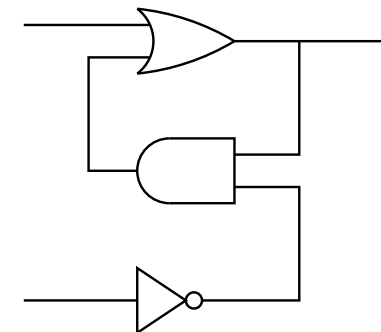
Memory

- Maintaining state information.



- This circuit can store one bit of information.
- There is no way of changing the state of memory within this circuit.

FlipFlop

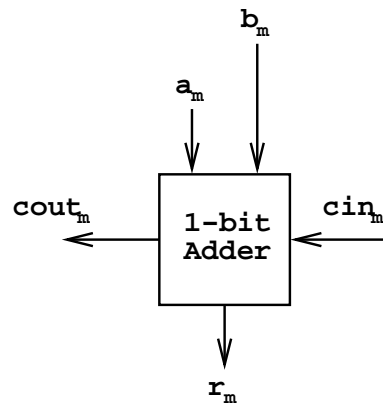


Binary Addition

- Binary addition is an important operation required by most architectures.
- When we add binary numbers by hand we place one number on top of the other.
- Then we move from right to left adding the digits and remembering to include the carry.
- This exact approach can be replicated in digital logic.

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

1-bit Adder



Binary Addition in digital logic

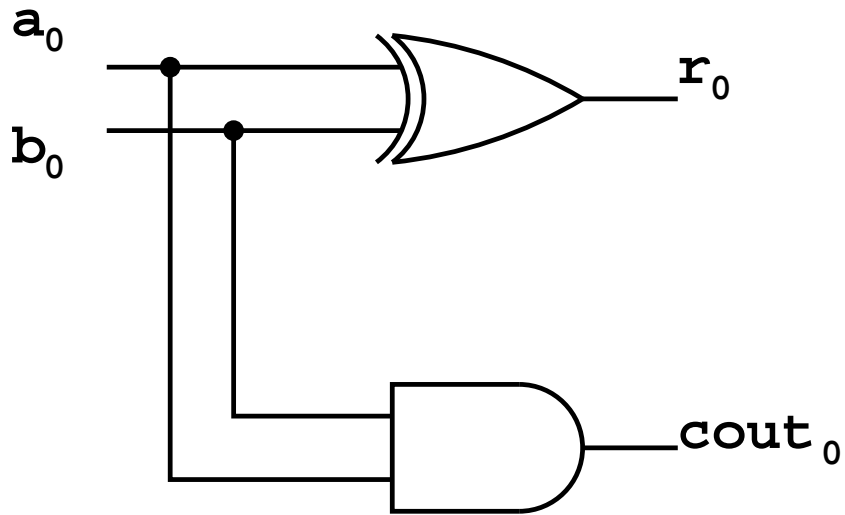
- We first create a component that will evaluate one column of the addition.
- The inputs will be
 - a single bit from each binary number of the column in question a_m and b_m .
 - the carry in from the previous position c_{in_m} .
- the outputs are
 - a single bit result for this column r_m .
 - a carry out to the next position $c_{out_m} = c_{in_{m+1}}$.

1-bit Adder

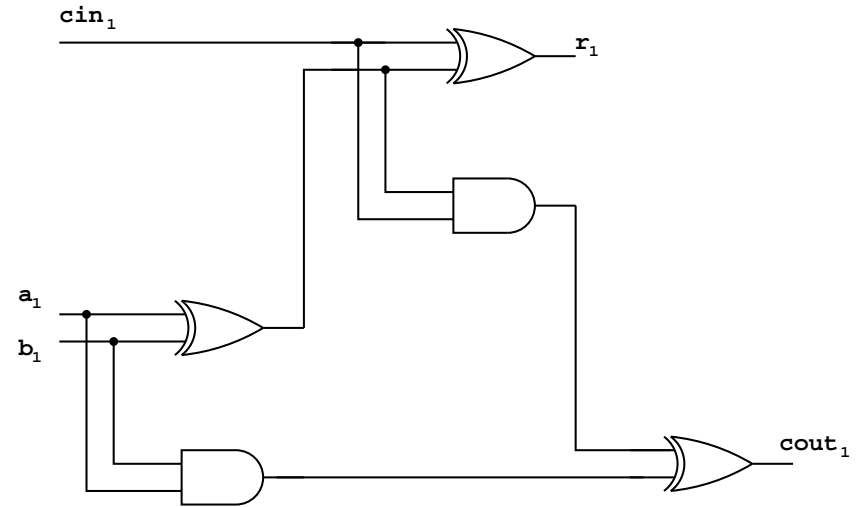
- We can construct a truth table for this one bit adder.
- From this truth table we can then design a circuit.

a_m	b_m	c_{in_m}	r_m	c_{out_m}
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

Half Adder

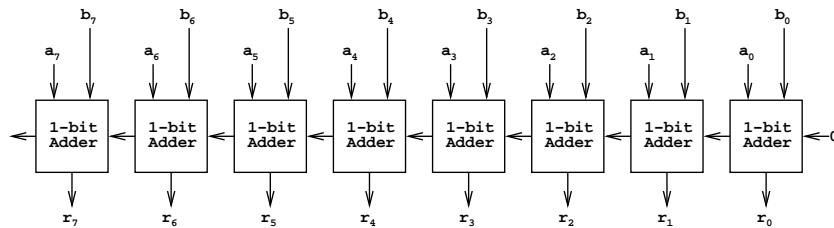


1-bit Adder



8-bit Adder

- We can combine 8 of these 1-bit adders to form a serial 8-bit adder.



Reading and Self Assessment

- Chapter 1 (Required reading)
 - 1.2, 1.4, 1.5, 1.6, 1.7
- Questions/Exercises (You are strongly encouraged to do these.)
 - 1.1 : 2,3,6,7.
 - 1.2 : 1,2,3.
 - 1.4 : 1,2,3,5,6.
 - 1.5 : 1,2,3,5,6.
 - 1.6 : 1,2,3,5,6,9,10.
 - 1.7 : 1,2.