**COMPUTER ARCHITECTURE NOTES**

Contents

[1 Introduction 3](#_Toc91745093)

[2 Data Representation and Number Systems 5](#_Toc91745094)

[2.1 Binary numbers 5](#_Toc91745095)

[2.2 Octal and Hexadecimal 7](#_Toc91745096)

[2.3 Negative integers and two's complement. 8](#_Toc91745097)

[3 Digital circuits 10](#_Toc91745098)

[3.1 Truth tables 10](#_Toc91745099)

[3.2 Circuit design & Boolean Algebra 12](#_Toc91745100)

[3.3 Karnaugh Maps 14](#_Toc91745101)

[4 Combinational Circuits 16](#_Toc91745102)

[5 Sequential Circuits 22](#_Toc91745103)

[5.1 Flip-flops 22](#_Toc91745104)

[5.2 D Flip Flop 24](#_Toc91745105)

[5.3 Clocked RS flip-flop 25](#_Toc91745106)

[5.4 J-K Flip-flop 26](#_Toc91745107)

[6 Introduction memories & CPU 27](#_Toc91745108)

# 1 Introduction

Lecturer - Andrew Wright

Assessment

Final Exam at Summer – 50% (of total marks)

Course Assessment:

Practicals/Tutorials – 50% (of total marks)

So what is Computer Architecture? –

Many different approaches have been taken.

The course is not standard.

Generally cover the workings of the modern computer.

Learn how the interaction between software and hardware takes place.

The course will be divided into two parts:

Practicals/Tutorials

Covers the underlying theory behind how the computer works i.e. A study of number representation, Boolean algebra; Introduction to simple digital electronics; Assemble language

Lectures

Will introduce the components of the modern computer and how they relate to one another.

The computer

Central processor unit; Memories; Storage devices; Magnetic – hard drive, floppy drives; Optical – CD-ROMs, DVDs.

Computers

Are everywhere in our daily lives. Most everyday electronic devices e.g. cars, televisions, washing machines etc. contain a computer in some form or other. The computer that everyone thinks of first is typically the personal computer or laptop etc. All computers basically work the same way.

Computer uses

Are in use everywhere e.g. type letters, write reports, mathematical calculations, play games, send e-mails, browse the web, data storage etc.

Two basic components termed.

Software

Hardware

Software

A set of instructions - called a program. Controls the computer to do useful work. Without software the computer is useless – like a car without a driver.

Software layering

In modern computers there is a number of different program that work the computer:

User program (e.g. microsoft word)

Operating system (e.g. windows/UNIX/LINUX)

The computer systems program (e.g. BIOS)

Hardware

Hardware

The physical components of the computer i.e. the electronic circuit to carry out the instructions. The computer itself and the devices that are attached to it – the peripherals

Input/Output

The data on which the instructions are to be carried out is – called the input. The result is called the output Example:

The addition of 2 + 2 = 4

How do computers work?

Have lots of different parts. Built around a central ‘brain’ a microprocessor called the central processing unit or CPU.

Central Processing Unit

Complex set of electronic circuitry. Executes stored program instructions

Contains: Registers, control unit, ALU, cache memory.

Motherboard

CPU is to be found on the ‘Motherboard’. Main circuit board of the computer. Contains all the important chips needed for the computer to work. Attached to the CPU on the motherboard are various components that feed and take information.

Memories

Fast access times for data used to hold data. Holds data only while computer is switched on. Has a hierarchy of devices from the fastest to the slowest. Two basic types of memories: RAM – random access memory (ever memory location access in the same amount of time). ROM – read only memory.

Computer components

As well as the motherboard the computer contains: Circuits boards for pictures, sound, internet connections etc.; Power source; Cooling devices; Connecting leads; Slots for drives.

Storage devices

Use for permanent storing data. Three main types:

Magnetic - Disk (hard drive and floppy drive) and tape

Optical – CD-ROM, DVD.

Flash - USB, BIOS

Peripherals

Attached to the computer. Used for input and output. Input – keyboard, mouse, scanner. Output – monitor, printer. These will be covered next semester in Computer Systems.

Summery

*"A PC is a general-purpose information processing device. It can take information from a person (through the keyboard and mouse), from a device (like a floppy disk or CD) or from the network (through a modem or a network card) and process it. Once processed, the information is shown to the user (on the monitor), stored on a device (like a hard disk) or sent somewhere else on the network (back through the modem or network card)."*

# 2 Data Representation and Number Systems

In all computers information is represented inside a computer in binary form i.e. using the numbers 1 and 0 i.e. on/off - one state is represented by a voltage and the other state is no voltage. The hardware of a computer system has no other way of representing information.

## 2.1 Binary numbers

The normal denary number system uses ten symbols: 0, 1, 2, 3, 4, etc. Thus 397 means 3 hundreds with 9 tens and 7 ones, or

397 = (3 x 102) + (9 x 101) + (7 x 100)

The significance of the digits is systematically associated with powers of 10.

Binary numbers use only two symbols 0 and 1. The natural binary number system uses the same principle of changing the significance of the digits but in this case each digit corresponds to an increasing power of two. The interpretation of 101 in binary is 1 four, with no twos and 1 one, or

101(natural binary) = (1 x 22) + (0 x 21) + (1 x 20)

= 5 (denary)

The bit at the extreme right (representing units 20) is known as the ***least significant bit*** (LSb) and that at the extreme left (representing 2n-1 for n bits) the ***most significant bit*** (MSb). To avoid confusion subscripts are introduced to indicate whether a number is denary or binary e.g. 1012 and 39710. A natural binary number can be converted into denary simply by multiplying each bit by the appropriate power of 2 and adding them together to get the result e.g.

10110112 = (1 x 26) + (0 x 25) + (1 x 24) + (1x 23) + (0 x 22) + (1 x 21) + (1 x 20)

= 64 + 0 + 16 + 8 + 0 + 2 + 1 = 9110

A denary number can be converted into natural binary by looking for powers of 2:

39710 = 256 + 128 + 0 + 0 + 0 + 8 + 4 + 0 + 1

= (1x28)+(1x27)+(0 x 26)+(0 x 25)+(0 x 24)+(1 x 23)+(1 x 22)+(0 x 21)+(1 x 20)

= 1100011012

Another way of arriving at this is to keep dividing by 2 and list the remainders, r:

397/2 = 198 remainder r = 1 (LSb)

198/2 = 99 remainder r = 0

99/2 = 49 remainder r = 1

49/2 = 24 remainder r = 1

24/2 = 12 remainder r = 0

12/2 = 6 remainder r = 0

6/2 = 3 remainder r = 0

3/2 = 1 remainder r = 1

1/2 = 0 remainder r = 1 (MSb)

therefore

39710 = 1100011012

## 2.2 Octal and Hexadecimal

Because working with binary numbers alone involves writing long cumbersome numbers, base 8 and (more often) base 16 numbers are used. These are referred to as octal and hexadecimal respectively.

In octal and hexadecimal positional weighting of 8 and 16 respectively is used. These representations provide two ways of quickly reducing a long string of binary digits to a more manageable form.

***Octal***

In octal this is done by dividing the binary number into groups of three counting from the LSB. For example 110101102  can be divided into three groups 11, 010 and 110. The natural binary code is then used to replace each group by the corresponding single digit, 0 to 7. That is for the above example 3268.

It can be translated into denary using powers of 8:

3268 = (3 x 82) + (2 x 81) + (6 x 80)

= 192 + 16 + 6

= 21410

***Hexadecimal***

Hexadecimal (abbreviated to 'hex') splits the binary number up into groups of four bits (1101 and 0110 in the present example). Hexadecimal is used extensively as most registers are in-groups of four. Each group of four bits is replaced by a single symbol. From 0000 to 1001 the numbers 0 to 9 are used the extra symbols to make up the 16 are the letters A to F. Hexadecimal numbers are indicated by adding a h on the end e.g. AEh.



An example is 21410 which is equivalent to 110101102 can be written as D6h:

D6h = (13 x 161) + (6 x 160)

= 208 + 6

= 21410

## 2.3 Negative integers and two's complement.

Negative numbers within the computer's hardware can be represented in various ways. The simplest way is by using the MSB as the sign (i.e. 0 for positive numbers and 1 for negative numbers). This number is called a signed integer. The opposite is an unsigned integer which only represents positive integers.

In computers negative numbers for signed integers are almost always represented by so called two's compliment numbers. The two's compliment number is obtained by replacing all 0s by 1s and all 1s by 0s in the corresponding positive number (forming the complement), and adding the value 1 to the complement's result.

***Example one***

Positive integer 1610 0001 00002



negative integer -1610

complement 1110 11112

add one + 1

result 1111 0000

***Example two***

Binary number 1111 11112

interpretation as an unsigned integer:

(1x27) + (1x26) + .... + (1x 21) + (1 x 20) = 25510

interpretation as a signed integer:

MSB = 1 therefore number is negative, thus:

binary 1111 11112

subtract one 1111 11102

complement 0000 00012 i.e. -110

The reason for using the two's compliment representation for negative numbers is that the subtraction of a number from another number can be accomplished by the addition of the relevant two's compliment numbers. A possible carry is ignored.

***Example three***

Calculate 15 - 1

1510 = 0000 11112

110 = 0000 00012

2' compliment = 1111 11102

+1 = 1111 11112

therefore (15 - 1) = 0000 11112

+ 1111 11112

1 0000 11102

Note: the leading 1 (carry) is ignored, thus the result is 0000 11102 or 1410.

# 3 Digital circuits

## 3.1 Truth tables

**List** *all the possible* **instructions with a corresponding output depending on the function of the circuit/computer.** I.e. two inputs, each of which can be 0 or 1, can be combined four ways, namely 00, 01, 10 and 11. Three inputs, can be combined in eight different ways. Four inputs sixteen different ways etc.

An example for three inputs would be.

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | P |
| 0 | 0 | 0 | 0 |
| 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 | 1 |

Basic logic gates

Once a circuit inputs and outputs have been quantified in the Truth Table it can be design using a basic logic gate or a combination of basic logic gates. There are four basic gates with an extra four gates that have outputs which are inversion of the four basic gates;

The gates are:

|  |  |
| --- | --- |
| **GATE** | **INVERSION** |
| BUFFER | INVERTER or NOT |
| AND | NAND |
| OR | NOR |
| EXCLUSIVE-OR (XOR) | EXCLUSIVE-NOR (XNOR) |

The Buffer and NOT gate only have one input. The other gates can have two or more inputs.

Note:

* AND gate - **ALL** **inputs** have to be **one** (high) for the output to be **one** (high).
* OR gate **ANY input** (or more) has to be one (high) for the output to be **one** (high).
* XOR gates an **ODD number of ones** (highs) on the inputs for the output to be **one** (high).

**GATE SUMMARY SHEET**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GATE** | **BUFFER** | **AND GATE** | **OR GATE** | **exclusive-OR (XOR)** |
| **BOOLEAN EQUATION** | *P = A* | *P = A . B* | *P = A + B* | *P = A ⊕ B* |
| **TRUTH TABLE** | *A P*  0 0  1 1 | *A B P*  0 0 0  0 1 0  1 0 0  1 1 1 | *A B P*  0 0 0  0 1 1  1 0 1  1 1 1 | *A B P*  0 0 0  0 1 1  1 0 1  1 1 0 |
| **SYMBOL** |  |  |  |  |
| **SYMBOL** |  |  |  |  |
| **TRUTH TABLE** | *A P*  0 1  1 0 | *A B P*  0 0 1  0 1 1  1 0 1  1 1 0 | *A B P*  0 0 1  0 1 0  1 0 0  1 1 0 | *A B P*  0 0 1  0 1 0  1 0 0  1 1 1 |
| **BOOLEAN EQUATION** | *P = A* | *P = A . B* | *P = A + B* | *P = A ⊕ B* |
| **GATE** | INVERTER  or NOT Gate | NAND | NOR | Exclusive-NOR  (XNOR) |

## 3.2 Circuit design & Boolean Algebra

Digital circuits are design in a number of stages:

* Compile a Truth Table that will list the outcomes to all the possible combinations of inputs. Outcomes that are not used are called "**don't care"** and marked with an "**X"**.
* Two Boolean equations (called sum-of-products ) can be compiled from the truth table: one for an output of 1 and the other an output of 0.

***Example -*** Given the truth table below:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | P |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

The two sum-of-products equations for the above truth table would be:

The sum-of-products equation can be used to design a circuit in terms of an AND gate for each minterm and one OR gate to connect them together. However, in most cases the ***sum-of-products*** equation can be simplified by using Boolean Algebr**a.**

***Boolean Algebra***

Summary of theorems, rules and laws

|  |  |  |
| --- | --- | --- |
| **Name of Law** | **OR version** | **AND version** |
| **Identity** | ***A + 0 = A*** | ***A.1 = A*** |
| **One/Zero** | ***A + 1 = 1*** | ***A.0 = 0*** |
| **Idempotent** | ***A + A = A*** | ***A.A = A*** |
| **Inverse** | ***A+ A’ = 1*** | ***A.A’= 0*** |
| **Commutative** | ***A + B = B + A*** | ***A.B = B.A*** |
| **Associative** | ***(A + B) + C = A + (B + C)*** | ***(A.B).C = A.(B.C)*** |
| **Distributive** | ***A + (B.C) = (A + B).(A + C)*** | ***A.(B + C) = (A.B) + (A.C)*** |
| **DeMorgan’s** | ***(A + B)’ = A’B’*** | ***(A.B)’ = A’ + B’*** |

Note: The AND dot (.) is almost always omitted i.e. *A.B = AB*.

**Example**

Design a circuit for a factory given the following conditions: A room in the factory has four doors (A, B, C or D) given that shut=0 & open=1. Design a circuit that gives a green light (P=1) when doors A and C are shut or doors B and D are open. The truth table for the circuit is presented below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | P |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

The truth table has seven minterms and gives the following Boolean equation:

**Simplifying using Boolean Algebra**:

When a Boolean expression contains minterms which have variables in common, it is possible to use brackets by using the distributive law. If the expression inside the brackets contains only a single variable in inverted and non-inverted form, it is possible to simplify the above expression as follows:

Substituting Gives:

Using Gives:

This equation can be further simplified:

Which can be simplified to:

## 3.3 Karnaugh Maps

Karnaugh maps are a pictorial way of simplifying a design for a combinational circuit. They are named after Maurice Karnaugh, an American engineer who first suggested their use in 1953. They are an alternative way of writing down the information that is contained in a truth table. In this form it is easy to identify groups of minterms that can be reduced to smaller and fewer product terms. The values inside the squares are copied from the output column of the truth table, so there is one square in the map for every row in the truth table. Around the edge of the Karnaugh map are the values of the input variables.

An example of a truth table and its Karnaugh map for 3 variables is given below:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | P |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |

1

1

0

1

10

0

11

01

1

0

1

0

00

***BC***

***A***

0

Simplification by Karnaugh map consists of forming groups that are as large as possible.

**Rules of simplification are:**

* Include all horizontally and vertically adjacent 1s in a group. However, groups must be square or rectangular.
* Make groups as large as possible. However, groups must be of size 2 to the power of n, where n is less than or equal to the number of inputs. Where n = 4, the group size must be 1, 2, 4, 8, or 16.
* Groups may overlap.
* Use the simplest of the possible combinations of terms.
* Terms at the edge may be regarded as adjacent to the term symmetrically opposite. i.e. visualise the opposite edges as touching.
* Don't care options are marked as an X on the Karnaugh map and can be used in the grouping where necessary.

***Example simplification using a Karnaugh Map***:

Using the same example presented above for the following equation:

Fill in the 1’s in the table and group them:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CD | 00 | 01 | 11 | 10 |
| AB |
| 00 | 1 | 1 | 0 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 1 | 1 | 0 |
| 01 | 0 | 0 | 0 | 0 |

This gives the follow equation (two groups – two terms):

# 4 Combinational Circuits

The output of a combinational logic circuit changes in response to the combination of binary digits that are put on its inputs and is not influenced by conditions that have occurred before. No memory is required – as opposed to sequential circuits that have a “memory”.

Combinational Circuit

Sequential Circuit

Input

Input

Output

Output

Feedback lines

Adders, multiplexers are examples of combinational circuits. Flip-plops are an example of a Sequential circuit. Adders are found in the CPU. Multiplexers are found in memory systems and serial/parallel connections; flip-flops are found in the registers and cache memory (SRAM).

**ADDERS**

Type of combinational circuit.

**Half adder**

The function of a half adder is to add two binary digits, producing a SUM and a CARRY. The truth table for a half adder is:-

|  |  |  |  |
| --- | --- | --- | --- |
| X | Y | SUM | CARRY |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

The circuit can be drawn as:



The detailed circuit for the half adder is an Exclusive-OR gate for the Sum and an AND gate for the Carry:

X

Y

Carry

Sum

**Full Adder**

When more than two binary digits are to be added several half adders will not be adequate because the half adder has no input to handle carries from other digits. For example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X= | 1 | 0 | 1 | 1 |  |
| Y= | 1 | 1 | 1 | 0 |  |
|  | 0 | 1 | 0 | 1 | = Partial sum |
|  | 1 |  | 1 |  | = Carry bits |
| 1 | 1 | 0 | 0 | 1 | = Complete sum |

We must consider carries generated in each column during the addition process as an extra input i.e. the “Full Adder has three inputs – *X*, *Y* and *Carry-in* and two outputs *Sum* and *Carry-out*. The truth table for a Full adder is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INPUTS | | | OUTPUTS | |
| X | Y | Cin | S | Cout |
| 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 |

Full

Adder

X

Y

Cin

S

Cout

Using a K-Map we can produce a produce a product sum for S

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | YCi |  |  |  |  |
| X |  | 00 | 01 | 11 | 10 |
|  | 0 | 0 | 1 | 0 | 1 |
|  | 1 | 1 | 0 | 1 | 0 |

A good short cut is to note that a chequer board pattern of 0’s and 1’s means that all inputs are XOR’d together. So from the above Karnaugh map we can say that:

Equation 1

*S = X ⊕ Ci ⊕ Y*

For practice we won’t use the K-Map to produce the product sum for Co but will instead just use the Boolean laws to simplify. From the above truth table:

Simplifying the Boolean equation for the carry out gives:

Equation 2

However, to help fully understand the full adder circuit below we will use a Karnaugh map to get the Boolean expression for Co.



Karnaugh Map 1

In the above Karnaugh map we could have also grouped the vertical 1’s to further simplify the expression but it suits us not to do so in order to explain the full adder circuit.

It is also useful to observe the truth table for the XOR function below.



From the XOR truth table above it can be seen that the output is 1 when Y + X

Therefore, we can say that

Equation 3

X⊕Y = Y + X

A 1-bit full adder can be constructed using half adders as components.

*S*

*Cout*

Half-Adder

Half-Adder

*X*

*Y*

*Cin*

FULL ADDER

X⊕Y

XY

X⊕Y⊕C*in*

(X⊕Y)C*in*

XY

Notice that output ‘S’ is exactly as we had expected from the Karnaugh map in *Equation 1* on page 18. However, the output Cout needs a bit more explanation.

Looking at the inputs to the OR gate in the above circuit we can see that

Cout = (X⊕Y)C*in* + XY

Making use of X⊕Y = Y +X the above expression can be rewritten as

Cout = (Y + ) C*in* +XY

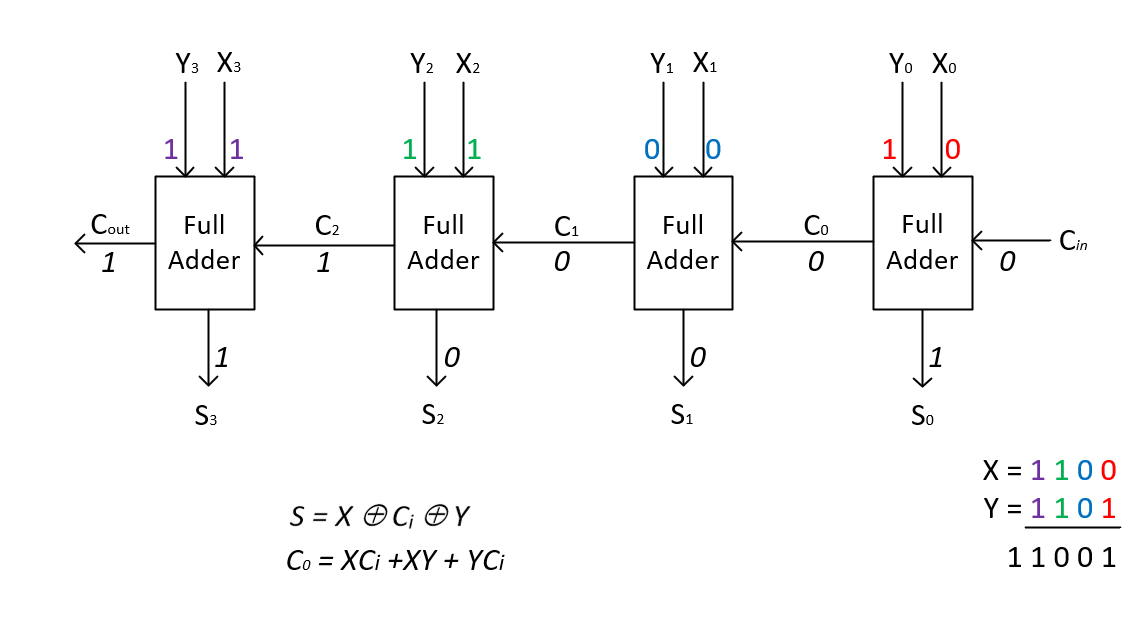
Expanding this expression we get

Cout = YCin + Cin + XY

This is what we had derived from *Karnaugh Map 1*on page 19

**Four-Bit Parallel Adder**

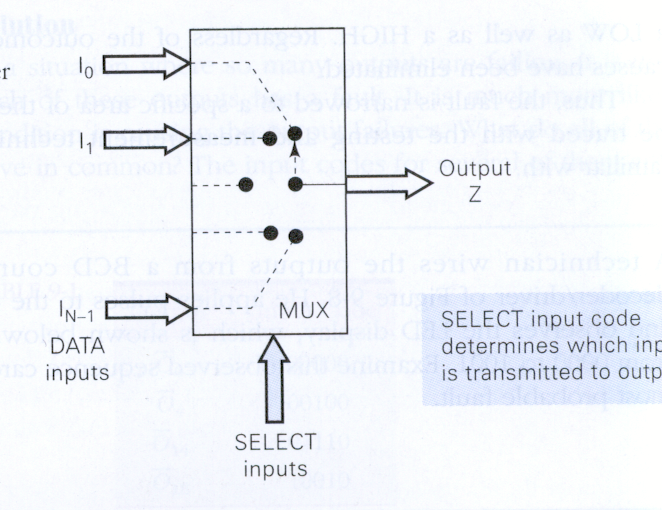
We can combine four 1-bit full-adders in parallel to add 4-bit binary integers. The circuit for this is:



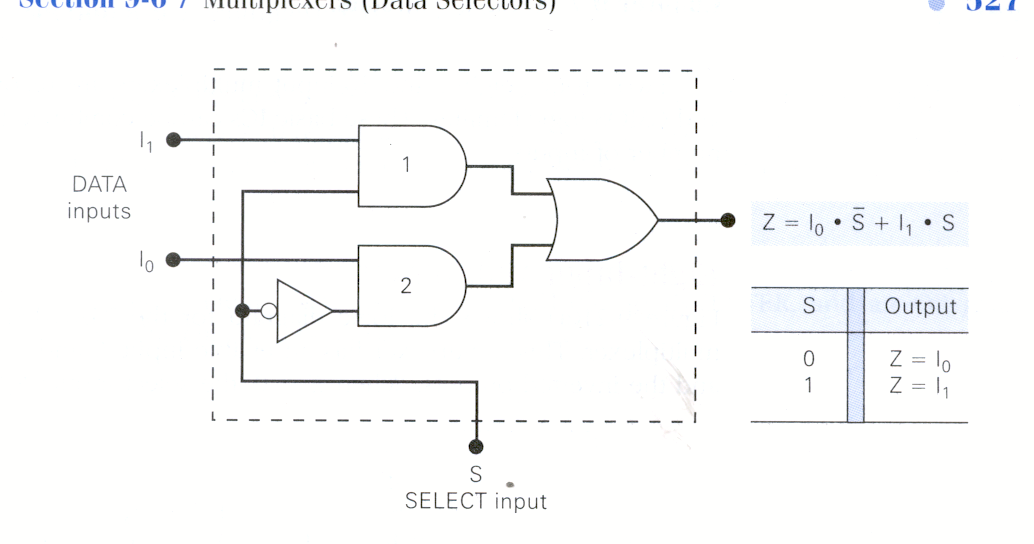
**Digital Multiplexers**

A multiplexer selects one of several input signals and passes it on to the output. The multiplexer acts like a digitally controlled multiposition switch where the digital code applied to the select inputs controls which data inputs will be switched to the output. The number of inputs are usually 2, 4, 8, or 16.

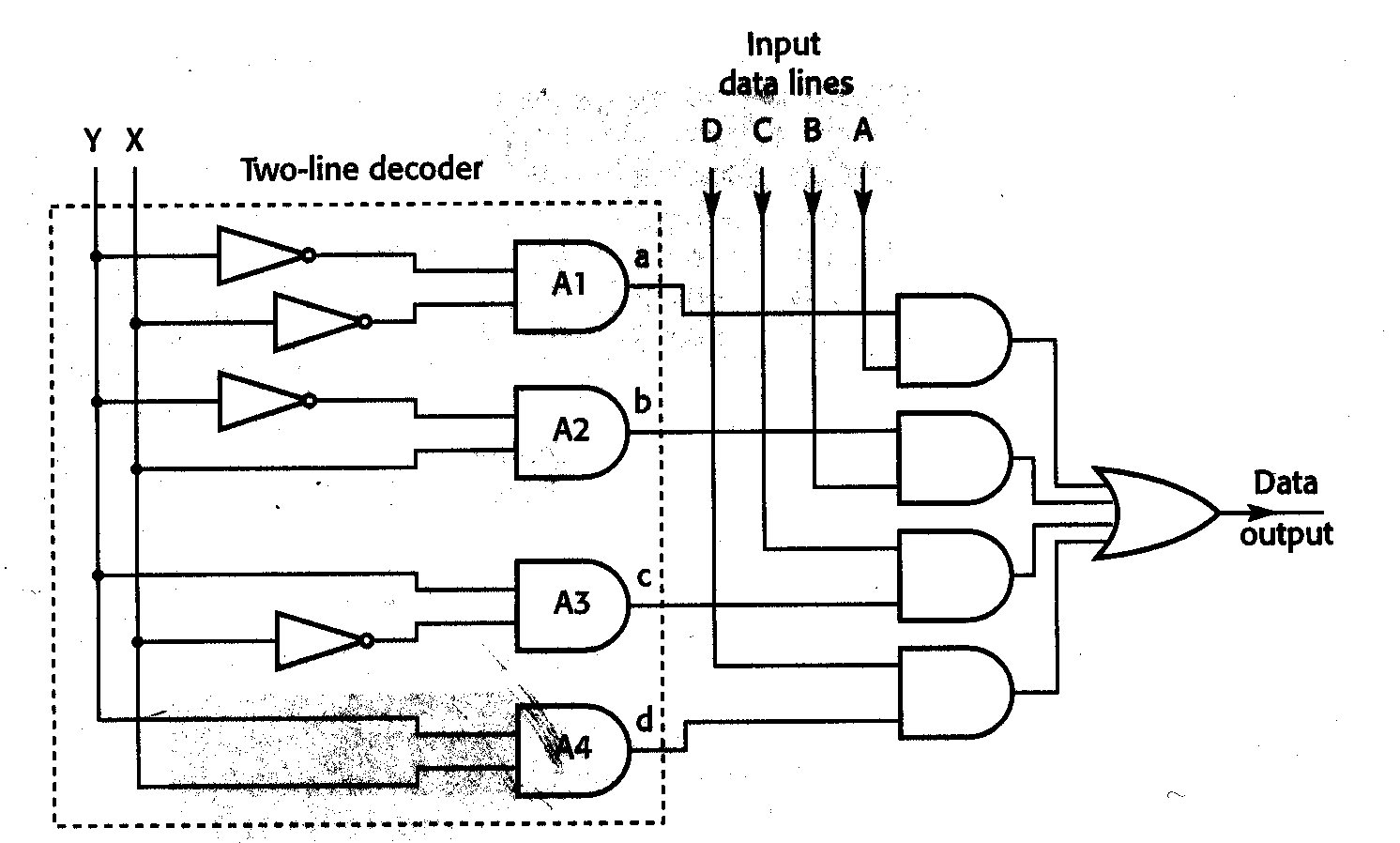
The multiplexer uses a number of AND gates for selecting the relevant input - one for each INPUT. The SELECT inputs make the relevant AND gates inputs high on the gate connected to the required input line and at least one of the inputs low on all the other AND gates in the multiplexer.



The simplest multiplexer has two INPUT lines and (therefore) one SELECT line (i.e. 0 or 1):



The circuit for a 4-INPUT multiplexer with 2 SELECT lines is as follows:



Circuit diagram for a 4-input multiplexer

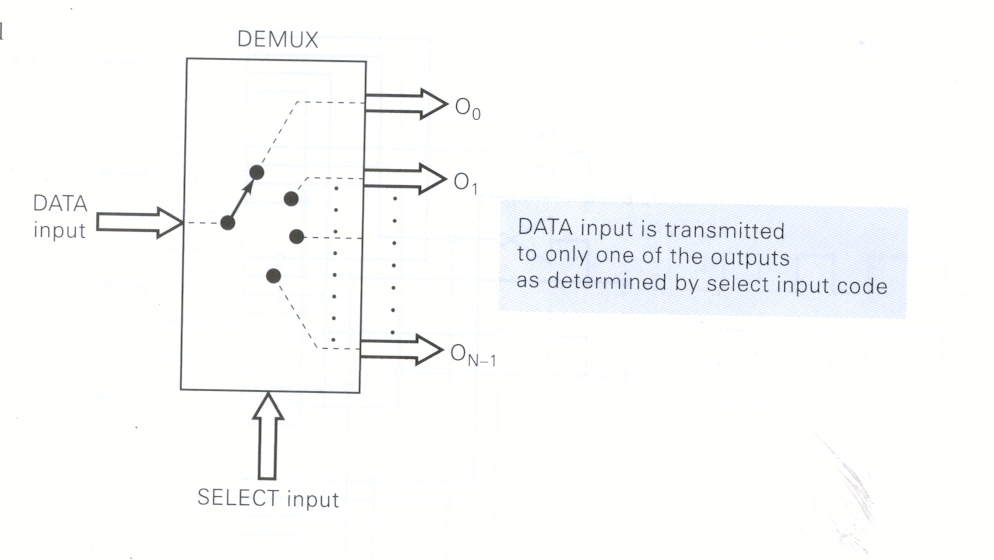
The address inputs (X and Y) select ONE data input to be connected to the output.

The Boolean equation for the output of the 4-INPUT Multiplexer is:

**uses** - Multiplexers applications include data selection, data routing, operation sequenceing, parallel-to-serial conversion, waveform generation and logic function generation.

**Demultiplexers**

Demultiplexers work the other way around to multiplexers. There is one INPUT line and the SELECT lines select one of a number of OUTPUT lines to route the INPUT line to. They are used for serial-parallel conversion.



# 5 Sequential Circuits

Sequential Circuits are circuits where the outputs depend not only on the present inputs but also on what has happened before i.e. they have a sense of time. They carry out this function by the outputs depending not only on the inputs but also on the outputs. The flip-flop is a basic example of this type of circuit.

## 5.1 Flip-flops

Flip-flops are circuits made up of logic gates. There are many different types depending on the application required e.g. RS flip-flop, D-latch, JK flip-flop etc. Flip-flops generally have two inputs e.g. for the RS flip-flop (often called the RS latch) - is one of the simplest known flip-flop. The inputs are labelled Set (S) and Reset (R). The flip-flop has two complementary outputs Q and Q'. Complementary outputs means that Q' is opposite to Q (i.e. when Q = 1 Q' has to equal 0 and when Q = 0 Q' has to equal 1). This arrangement explains why we call these devices flip-flops - if the Q output 'flips' then the Q' output 'flops'.

The RS flip-flop is constructed out of two gates, these can be two NOR gates or two NAND gates.. The diagram below shows the layout of the RS flip-flop built with NOR gates.

Diagram

Description automatically generated

Notice that the outputs are the complement of each other (Qn and Q̅n). It is also important to understand that the next output Qn+1 is determined not just by R and S but also by the previous output Qn since it is fed back to the input.

Observing the truth table above you can see that there are four conditions, which are separated by the grey line. The first condition is where R=0, S=0, and Qn=0 and 1. This way all conditions are considered.

Before we look at each of these conditions, we will apply a few simple rules. We will always consider the next stage as Qn+1. You can consider the complement output either, the choice is yours. If the output produced is NOT 0, or 1 then you must consider the complement output. So let’s start with the first condition.

(i)



(ii)



(iii)



(iv)



The main things to note about the RS flip flop is that when both inputs are zero the output doesn’t change. This is memory. When S=1 and R=0 the output is ‘set’ and when R=1 and S=0 the output is reset. In the RS flip flop we don’t use S=1 R=1 since it is of no use to us.

## 5.2 D Flip Flop

The RS flip flop can cause uncertainty when S = R = 1. The D Type flip-flop circumvents this issue with the use of using a single input and an inverter. The circuit below illustrates the D type flip-flop.

Diagram

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Clk** | **D** | n+1 | n+1 |
| **0** | **0** |  |  |
| **0** | **1** |  |  |
| **1** | **0** |  |  |
| **1** | **1** |  |  |

By looking at the circuit we can see that when the clock (CLK) input is lo the output from both AND gates will be zero. Therefore R=0, S=0, so as before the outputs will remain unchanged. The CLK input must be high for the D input to propagate through to R & S. The inverter ensures that the illegal condition of R=1, S=1 is avoided.

Complete the truth table for yourself.

## 5.3 Clocked RS flip-flop

A clock is a device that generates a precise pattern of periodically occurring pulses. One such pattern is shown in below:

1

0

The basic RS flip-flop is an asynchronous device. It does not need or use a clock signal. When the R or S inputs change, the Q and Q’ outputs respond immediately. The clocked RS flip-flop operates in step with a clock wave form. Such a flip-flop is said to operate in synchronous mode. The circuit and truth table is given below. The basic flip-flop operates much as before, with the proviso that the Q and Q’ outputs will only change state when the clock signal goes high. We can implement a clocked RS flip-flop using NOR and AND gates as shown below.

R

Q'

Q

S

CLOCK

This circuit works as follows: if CLK is low then both AND gates are disabled and have low outputs. This means that R = S = 0 and so the Q and Q’ outputs remain in their previous state. When the clock input goes high, both AND gates are enabled. This means that the R and S signals can reach the R and S inputs. In this way the flip-flop is either set or reset depending on the value on the value of R or S. Therefore the clocked RS flip-flop cannot change state until the clock signal is high. Clocking a flip-flop is important in large digital systems where hundreds of flip-flops may be interconnected. The clock signal is applied to all flip-flops simultaneously. This ensures that they change state together.

The truth table for the clocked SR-flip-flop with NOR gates is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLK | S | R | Q | Q’ |
| ̅0 | X | X | No change |  |
| 1 | 0 | 0 | Hold |  |
| 1 | 0 | 1 | 0 | 1 (reset) |
| 1 | 1 | 0 | 1 | 0 (set) |
| 1 | 1 | 1 | Prohibited |  |

## 5.4 J-K Flip-flop

The flip-flop on the 'logic trainer boards' used in the practical is called the J-K flip-flop. It is more complicated than the S-R flip-flip and has more functionality as: The outputs Q and Q' only change on a negative clock pulse i.e. when (and only when) the clock changes from 1 to 0. The inputs J=1 and K=1 results in Q and Q' changing state to the opposite of what they where i.e. if Q=0 and Q'=1 then it would change to Q=1 and Q'=0. This is called the toggle mode of operation. There are two inputs that can feed data into the flip-flop asynchronously i.e. the flip-flop will respond immediately (asynchronously) rather than waiting for a clock cycle. These are called the PRESET and CLEAR lines.

The truth table for the J-K flip-flop is given below:

|  |  |  |  |
| --- | --- | --- | --- |
| J | K | CLK | Q |
|  |  | ↑ | Q0 - no change |
| 0 | 0 | ↓ | Q0 - no change |
| 0 | 1 | ↓ | 0 |
| 1 | 0 | ↓ | 1 |
| 1 | 1 | ↓ | Q'0 - toggles |

Truth table for the PRESET and CLEAR line for the J-K flip-flop:

|  |  |  |
| --- | --- | --- |
| PRESET | CLEAR | flip-flop response |
| 1 | 1 | Clocked operation i.e. flip-flop will respond to J, K and CLK. |
| 0 | 1 | Q = 1 (regardless of CLK) |
| 1 | 0 | Q = 0 (regardless of CLK) |
| 1 | 1 | Not used |

# 

# 6 Introduction memories & CPU

Memory and Storage

Memory

Stores data temporarily. CPU refers to it for both program instructions and data

Storage

Permanent. External e.g. hard drive, floppy disks, memory sticks etc.

Memory

Known as main memory. Holds data and instructions for processing. Needs power.

The CPU and Memory

CPU cannot process data from disk or input device. It must first reside in memory. Data is moved from the Storage to memory by instructions from the BIOS or Operating System. On finishing task - data needing to be saved is moved to storage by instructions again from Bios/OPS.

Memory Addresses

Each memory location has an address. A unique number, much like a mailbox. Contains a binary number of fixed length – usually one byte. All locations are the same size.

When data is written back to memory, previous contents of that address are destroyed

Referred to by number – first number is always zero.

How the CPU Executes Instructions

Four steps performed for each instruction

Machine cycle: the amount of time needed to execute an instruction. Very fast - executes millions of instructions/second. Each CPU has its own instruction set.

The Fetch/Execute (Machine) Cycle

The time required to retrieve, execute, and store an operation. Components: Instruction time & Execution time. System clock synchronizes operations

Instruction Time (I-time)

Control unit gets instruction from memory and puts it into a register.

Control unit decodes instruction and determines the memory location of needed data

Execution Time

Control unit moves data from memory to registers in ALU. ALU executes instruction on the data

Control unit stores result of operation in memory or in a register

Registers

High-speed temporary storage areas for data and instructions being carried out. Work under direction of control unit. Accept, hold, and transfer instructions or data. Keep track of where the next instruction to be executed or needed data is stored

Cache Memory

A small block of high-speed memory which speeds up data transfer to the CPU from the main memory as transferred from cache much faster than from memory

Stores most frequently and most recently used data and instructions – the CPU looks for what it needs in cache first. If not in cache, control unit retrieves from memory. The more cache “hits” the faster the system performance

Different levels of Cache memory

* Internal (Level 1) cache built into microprocessor - Fastest access
* Internal/External (Level 2) cache on separate chip or Incorporated into processor on some current microprocessors
* External (Level 3)

Microprocessor

Central processing unit etched on silicon chip

Contain tens of millions of tiny transistors

Transistors

Electronic switches that may or may not allow electric current to pass through

If current passes through, switch is on, representing a 1 bit

Otherwise, switch is off, representing a 0 bit

Types of Chips

There are many different chips on the market - Intel is one of the most well-known. Xilinx produce a chip that can be programed

Memories - RAM and ROM

Terms used to categories memories.

RAM - Random Access Memory & ROM - Read-Only Memory (note ROM is a type of RAM!)

Data can be accessed randomly i.e. ALL MEMORY ADDRESSES CAN BE ACCESSED IN THE SAME TIME i.e. Memory address 10 can be accessed as quickly as memory address 10,000,000

Memory Types

Types of semiconductor memory: Dynamic RAM; Static RAM, CMOS, ROM, Flash memory etc.

Dynamic RAM

Used by most modern computers. Reliable, inexpensive, and compact

Must be continuously refreshed by CPU or it loses its contents – **Volatile** i.e. If the current is interrupted, data is lost. Used in main memory

Types include - Synchronous DRAM (SDRAM); Rambus DRAM (RDRAM).

Packaged on circuit boards e.g. Single in-line memory modules (SIMMS) or Dual in-line memory modules (DIMMS)

Static RAM

Faster and more expensive than DRAM. Used for register and cache memory

Complementary Metal Oxide Semiconductor (CMOS)

Used to store information needed when the computer boots i.e. BIOS. Needs power.

Read-Only Memory

Contains programs and data permanently recorded into memory at the factory - cannot be changed by user. Not volatile: contents do not disappear when power is lost

Programmable ROM (PROM) chips can be changed with a ‘process’ e.g. BIOS chip.

Flash Memory

Nonvolatile RAM. Used in cellular phones, digital cameras, and handheld computers. Chips resemble credit cards. Smaller than disk drive and require less power.

The System Bus

Parallel electrical paths (bundle of wires) that transport data between the components of the motherboard.

Generally divided in three type:

* Data bus
* Address bus – size dictates the amount of available memory
* Control bus e.g. read/write line/clock signal.

Bus Width

The number of electrical paths to carry data - measured in bits. The larger the bus size, the more data can be moved at a time

Bus Speed

Measured in megahertz (MHz). The faster the bus speed, the faster data travels through the system

Expansion Buses

Add peripheral devices to system

Expansion Boards

Connect to expansion slots on motherboard

Used to connect peripheral devices

Ports

External connectors to plug in peripherals such as printers

Two types of ports

Serial: transmit data one bit at a time

Parallel: transmit groups of bits together side-by-side

Examples of Expansion Buses and Ports

* Industry Standard Architecture (ISA) bus - Used for slow devices such as the mouse and modem
* Peripheral Component Interconnect (PSI) bus - Used for faster devices such as hard disks
* Accelerated Graphics Port (AGP) - Provides faster video performance
* Universal Serial Bus (USB) port - Allows you to convert many devices in a series into the USB port
* IEEE 1394 bus - A high-speed bus normally used to connect video equipment
* PC Card bus - Used on laptops to plug in a credit-card sized device

**Definitions**

RISC (Reduced Instruction Set Computing)

Uses a small subset of instructions all of the same length (i.e. same no. of bits). Fewer instructions increase speed. Simpler hardware.

Drawbacks: programs complex operations have to be broken down into a series of smaller instructions

CISC (Complex Instruction Set Computing)

More instructions. Sometimes of different lengths. Hardware more complex. Programming easier.

Dual core & Parallel Processing.

A variation of traditional serial processing - using multiple processors at once. Control processor divides problem into parts sent to separate processor. Each processor can has its own memory. Control processor assembles results. Dual core has two processors with two program counters etc.

Pipelining

Feeds a new instruction into CPU at each step of the machine cycle i.e. instruction 2 fetched when instruction 1 is being decoded, rather than waiting until cycle is complete. All processors use piplining to some extent.

Further Reading

* Tools for an information age Chapter 1, 2, 3 & 4 Capron& Johnson. Any edition.
* ‘How Stuff works’ website.