# Microprocessor Sourcebook

George Loveday

# Microprocessor Sourcebook

**George Loveday**

CEng, MIERE
Senior Lecturer/Bromley College of Technology


**Pitman**

PITMAN PUBLISHING LIMITED
128 Long Acre, London WC2E 9AN

A Longman Group Company



First published in Great Britain 1986



ISBN 0 273 02154 0

# Preface

This book, arranged as a series of interrelated topics in A to Z order, is intended to be used as a basic reference manual for anyone wishing to understand the complexities of microprocessors and microprocessor-based systems. My aim has been to put in the sort of information, in as concise a form as possible, that is required particularly by personnel and students working in the service and test areas of the microelectronic and allied industries. More advanced data—what might be called "state of the art" material—has been deliberately left out. Thus the central core of the book is the section giving operating details on the industry-standard 8-bit processors. For students, whether technician or undergraduate, the book can variously provide easily-accessible learning material and a ready reference with design and project work.

G. L. 1986

# Contents

## Absolute Address

An address that is permanently assigned to a particular storage location. This assigning will be done at the design stage of the machine. Also called specific address, machine address, actual address, and real address.

Another way of looking at this is to say that an absolute address is a pattern of bits on the address bus (in machine code) that identifies, without any further modification, a unique storage Location.

● ADDRESS ● ADDRESS DECODING ● MEMORY MAP

## Absolute Addressing Mode

An addressing mode where the op-code is followed by a 2-byte address (for 8-bit micros); in other words, the operand of the instruction will be an absolute address.

*Example*

STX $31A0 Store content of X reg. at address $31A0

This mode of addressing is usually called **direct** or **extended** in most processors. An exception is the 6502 which does have an *absolute addressing mode*. Note that with this processor the second byte of the operand in machine code provides the most significant half of the address. In the example given above, using the 6502, the instruction would read:

STX $31A0 (Machine code = 8E A031)

In machine code, the low byte of the address is specified first.

● ADDRESSING MODE ● MICROPROCESSOR [6502]

## Access Time

The word "access" is used as a verb in microelectronics to describe the operation of obtaining data from any memory location, either RAM or ROM, or from a peripheral; a typical statement is

"The data at memory address XXXX was accessed"

The *access time* refers to the speed with which the content of any location within a memory can be made available. It is the time interval between the instant that an address is sent to the memory and the instant that the data stored at that memory address is presented at the output.

*Random access*, where any location in the store can be reached in the same time as any other, is the fastest method. Both RAM and ROM chips are random access devices. Backing stores, such as magnetic disk, drum and tape, use accessing methods which are cyclic (disk, drum) or serial (tape). The access time, then, varies for different store locations, and an average time for access is then quoted; this would be the time for one half revolution in a cyclic system

Access times for semiconductor i.c. memory chips are typically:

| | |
|---|---|
| NMOS ROM | 450 nsec |
| NMOS static RAM | 300 nsec |
| CMOS static RAM | 200 nsec |

● MACHINE CYCLE ● MEMORIES

## Accumulator

There will be at least one, but usually two or more accumulators within a microprocessor unit. An *accumulator*, often referred to as simply a **register**, acts as a temporary storage location inside the processor and is very important for arithmetic, logic and data manipulation operations. The accumulator will hold the result of arithmetic and logic operations carried out by the ALU.

*Example*

ADD A $1000 Add the content of address $1000 to Accumulator A

The result of the addition will then be held in accumulator A.

The movement of data within a microcomputer system to or from memory locations and the microprocessor will probably use an accumulator.

*Example*

LDA $3200 Load Accumulator from memory location $3200

or

STA $0F20 Store content of Acc at address $0F20

Apart from these examples, other instructions that will operate on the accumulator include:

INCREMENT DECREMENT AND OR EXCLUSIVE-OR
TRANSFER (between accumulators) ROTATE SHIFT
COMPLEMENT

● MICROPROCESSOR ● REGISTER

**ACIA**

Abbreviation for ● Asynchronous Communications Interface Adaptor.

**Active State**

This refers to the logic level or a logic change of state (called *transition*) on an input pin to a microprocessor or other microelectronic chip, which activates, or triggers on, a required function.

There are four possible conditions:

ACTIVE LOW A logic 0 state initiates action. This is usually indicated by a bar over the function, i.e. $\overline{\text{RESET}}$.

ACTIVE HIGH A logic 1 state initiates action.

ACTIVE TRANSITION Low to High: the positive-going edge triggers the required function.

ACTIVE TRANSITION High to Low: the negative-going edge triggers the required function.

● ENABLE ● INTERRUPT

**Adder**

A microprocessor has to have circuits to perform the arithmetic operations of addition, subtraction, multiplication and division. An adder, a circuit that adds digital signals, is therefore an essential part of the ALU (*fig. A1*)

In computing, the basic arithmetic operation is addition, since subtraction can be performed by taking the twos-complement of the number to be subtracted and then adding it to the other number. Similarly, multiplication and division are carried out by shifting and adding.

The *half-adder* is the basic circuit. It adds two bits together and produces a sum and carry. A *full adder* is an extension of this circuit so that a carry bit from a previous addition can also be considered.

For an adder the inputs are

Addend An
Augend Bn
Carry Cn-1 (from next lower bit)

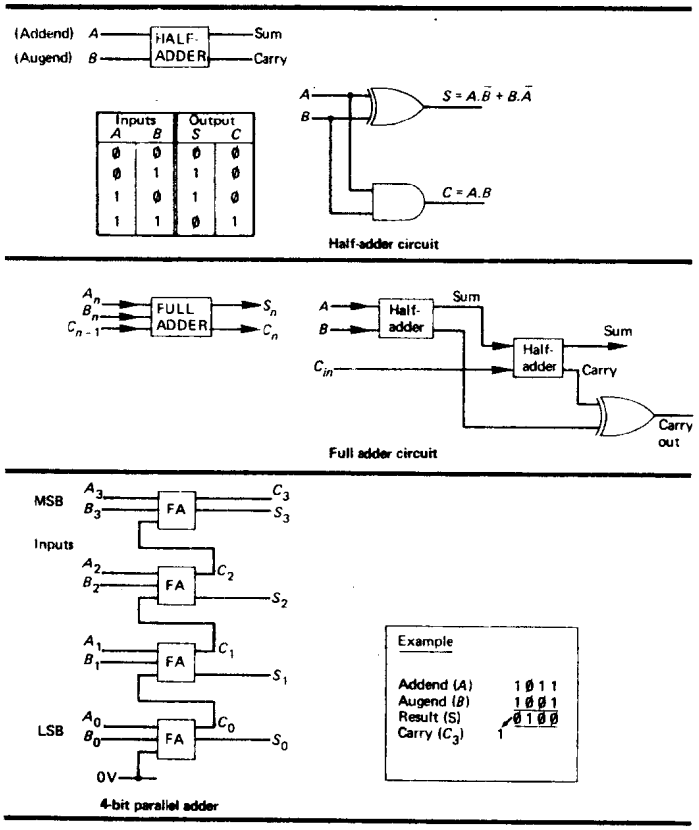
and the outputs are

Sum Sn
Output carry Cn

Full adders are linked together as shown to make a parallel adder (4 bits in diagram). Note that addition can be carried out serially but this is obviously a much slower process.

● ARITHMETIC AND LOGIC UNIT (ALU)

**Fig. A1** Adders




## ADD Instruction

These form part of the arithmetic instruction set for a microprocessor. They are

ADD Add without carry (memory to accumulator)
ADC Add with carry (memory to accumulator)
ABA Add accumulators.

The carry bit will be a flag within the status (condition code) register. The C flag will normally be set if there is a carry from the most significant bit of the result; it will be cleared otherwise. (See *fig. A2.*)

For most arithmetic operations, ADD WITH CARRY is the instruction that should be used.

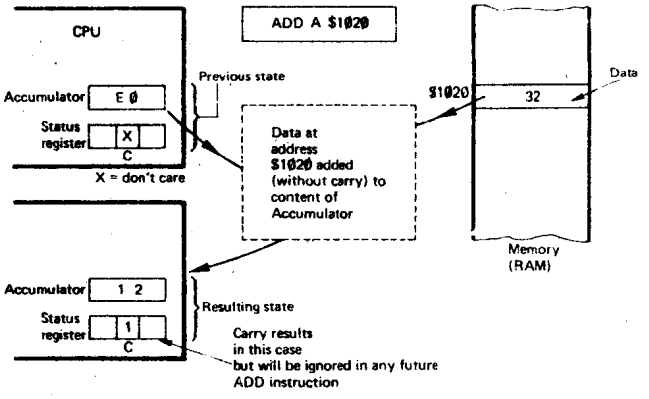
*Typical add instructions for some microprocessors are:*

| OPERATION | SYMBOLIC NOTATION | MNEMONIC 6800 | 6502 | Z80 |
|---|---|---|---|---|
| ADD | A + M → A,C | ADD A | | ADD (A) |
| | | ADD B | | ADD (HL) |
| ADD WITH CARRY | A + M + C → A,C | ADC A | ADC | ADC (A) |
| | | ADC B | | ADC (HL) |
| ADD ACCUMULATORS | A + B → A | ABA | | |

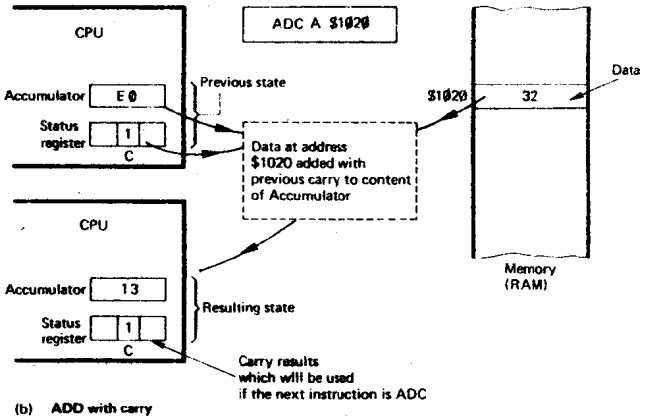
The full range of addressing modes can usually be used with these instructions.

● CARRY AND CARRY BIT ● INSTRUCTION

**Fig. A2 ADD instructions**




(a) ADD without carry from previous result




(b) ADD with carry

## Address

A binary coded number or word that is used to specify a location within memory or input/output ports.

A unique location has to be provided for all the data and each of the instructions used in a program. These memory locations, which are in ROM, RAM and I/O, are each identified by an *address* which is simply the pattern of binary bits sent out by the microprocessor over the address bus. An 8-bit microprocessor will have 16 address lines ($A_0$ to $A_{15}$) so that it can address $2^{16}$ possible locations, with each location containing one byte of information.

$2^{16} = 65\,536$, which is normally referred to as 64 K.

As an example, consider the simple arrangement of a 256 × 4-bit semiconductor RAM chip. The 256 store has 16 × 16 locations arranged as 16 rows and 16 columns; this is shown in block form in *fig. A3*. To access any location within this RAM chip, an 8-bit binary coded address can be used, 4 lines for X and 4 for Y. For example if the address is

$X = Ø11Ø \quad (=6_{10})$

$Y = 1Ø1Ø \quad (=10_{10})$

then only the memory location (holding 4 bits of data in this case) at $X_5$ and $Y_9$ will be addressed. Note that X = ØØØØ and Y = ØØØØ is the first location at row zero and column zero.

In a practical system, the memory i.c.s have chip select or chip enable signals which are decoded from the upper address line signals. In this way, memory i.c.s can be allocated certain memory areas without overwrite.

**Fig. A3** Principle of addressing

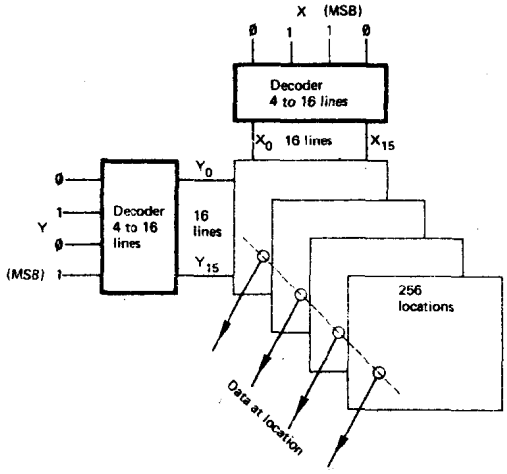
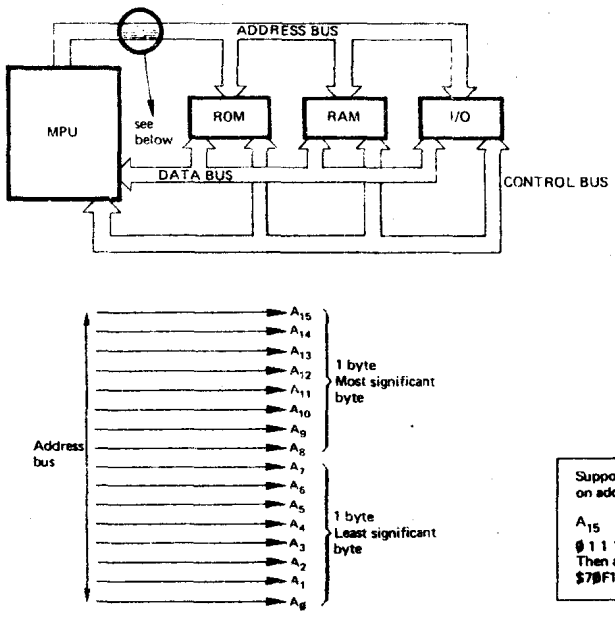



**Fig. A4a** Address bus




As stated, an address is a binary number and this is what is required by the machine. In machine code and mnemonic (assembly) language, addresses will be specified in hex. For a 16-bit address the range in hex. is from $0000 up to $FFFF.

● ABSOLUTE ADDRESS ● ADDRESS DECODING ● MEMORY MAP

## Address Bus

A bus is a major set of parallel conductors used within a microelectronic system to minimise the amount of interconnecting. The address bus in a microcomputer is unidirectional and typically 16 or more bits wide. It conveys address information from the microprocessor unit to memory as a binary pattern. (*Fig. A4a.*)

● ADDRESS ● ADDRESS DECODING ● BUS SYSTEMS

## Address Decoding

Each of the individual memory i.c.s used within a system must be allocated particular memory areas. This is essential to prevent any overlap. Address decoding, usually of the higher-order address lines, is used for this purpose. An *address decoder* is a logic circuit arranged so that for $n$ inputs there are $2^n$ output lines (*fig. A4b,c*). Only one of these output lines is high (or low) for each of the possible binary input combinations. The types used in microprocessor systems are 2-to-4, 3-to-8, or 4-to-16 line. A truth table for a 2-to-4 line decoder where the inputs are considered to be address lines $A_{14}$ and $A_{15}$ illustrates the principle:

| INPUTS | | OUTPUTS | | | |
|---|---|---|---|---|---|
| $A_{15}$ | $A_{14}$ | 1 | 2 | 3 | 4 |
| Ø | Ø | 1 | Ø | Ø | Ø |
| Ø | 1 | Ø | 1 | Ø | Ø |
| 1 | Ø | Ø | Ø | 1 | Ø |
| 1 | 1 | Ø | Ø | Ø | 1 |

In many systems the decoder output is the complement of this table.

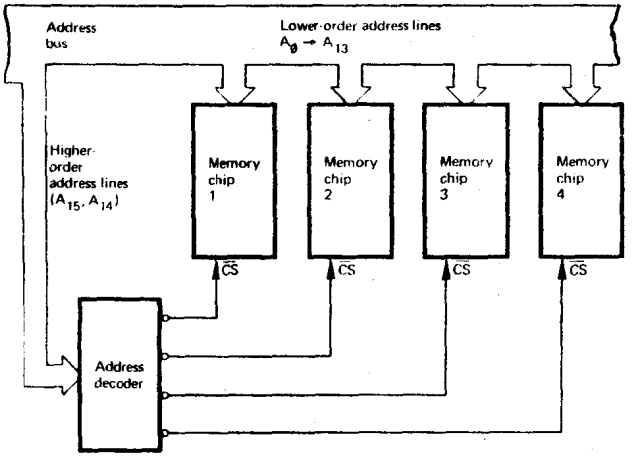



**Fig. A4b** Principle of address decoder chip

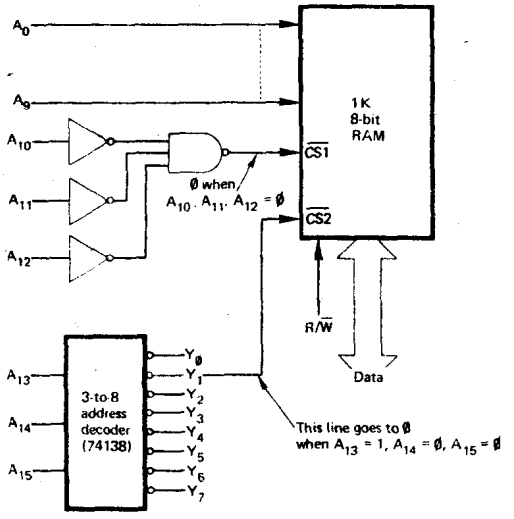



**Fig. A4c** Full address decoding for a memory chip

The memory i.c.s used within the system are provided with one or more chip select or chip enable lines (CS, CE) and the outputs of the address decoder are connected to these enable pins on the various i.c.s as shown.

In some systems where the memory required is small, for example a dedicated controller, not all the address lines need be decoded, but this will result in overwrite (one memory i.c. occupying a larger area of memory than it actually requires). In other cases, all address lines must be fully decoded. Take the example of a 1 K × 8-bit RAM i.c. with two chip select lines. The lower-order address lines $A_0$ to $A_9$ ($2^{10} = 1024$) are used to select a particular location within the RAM. The binary pattern on these lines is decoded by circuits inside the chip. The higher-order address lines are decoded fully using a logic circuit for $A_{10}$, $A_{11}$ and $A_{12}$, while $A_{13}$, $A_{14}$ and $A_{15}$ give a signal to $\overline{CS2}$ via the address decoder. The address is then from $2000 to $23FF, and there will be no overwrite.

● ADDRESS ● MEMORY MAP ● PAGE

## Addressing Mode

Consider the process of loading a register within a microprocessor with data from a memory location. There are several ways of specifying the location of this data: it could be at an absolute address, or held in the memory location immediately following the load instruction, or within an area of memory pointed to by an index register. The method of specifying the exact location is termed the *addressing mode*. In the above example we have, in order: Absolute (or Extended), Immediate, and Indexed addressing (*fig. A5*).

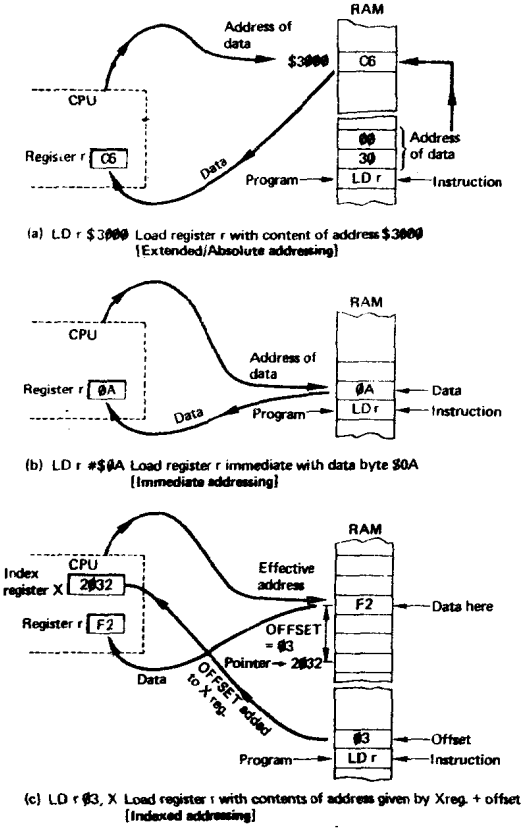
The variety or richness of the addressing modes employed by a microprocessor enhances its processing capability. Microprocessors do not all possess identical addressing modes and some manufacturers have different names for a particular mode. The most common are the following:

*Addressing modes*
Implied (or Inherent)
Immediate
Direct (or Zero Page)
Extended (or Absolute)
Indexed
Relative

Each of these is covered in a separate section and further examples are given under the section on microprocessors.

● ABSOLUTE ADDRESS ● ABSOLUTE ADDRESSING MODE ● IMMEDIATE ADDRESSING ● INDEX REGISTER AND INDEXED ADDRESSING ● MICROPROCESSOR

**Fig. A5** Addressing modes



**Address Register**

Sometimes known as Memory Address Register (MAR), this is a register, usually in the central processor, that stores an address.

● ADDRESS

**Algorithm**

A complex task can usually be divided up into several small, easily followed steps. The set of well-defined steps or processes for the solution of the task is called an *algorithm*. In other words, an algorithm describes a way in which a rather complicated task can be solved, or demonstrates that it cannot be solved.

Take the example of clearing a block of memory that consists of $n$ addresses. The algorithm for this could be:

1) Load a counter with a number equal to $n$.
2) Set the index register to point to the first address of the block of memory to be cleared.
3) Clear the address pointed to by the index register.
4) Increment (add 1) to the index register.
5) Decrement (subtract 1) from the counter.
6) Check to see if the counter has reached zero. If it has not, then return to repeat 3); otherwise end.

The structure of the program is now defined, which should make the writing of the program easier.

The word "algorithm" or "algorism" is derived from the surname of the Arab mathematician 'Abu Ja'far Muhammed ibn Musa al-Kuwarizmi (the man of Kuwarizm).

● FLOWCHART

**Alphanumeric Code**

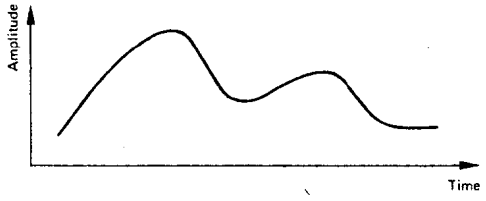
A set of characters consisting of the numbers 0 to 9 and the letters A to Z.

● ASCII

**Analog**

This describes those types of signal which are of a continuous nature, and not broken up into discrete steps as are digital signals (*fig. A6*).

**Fig. A6** Analog signal




Most changes in physical conditions, such as temperature, light level, pressure, and movement, when sensed by an electrical transducer result in analog signals.. The voltage or current varies in a way that is analogous to the change in input quantity. Similarly, circuits and components that process these signals are also termed analog or linear type devices.

● ANALOG-TO-DIGITAL CONVERTOR ● DIGITAL-TO-ANALOG CONVERTOR
● SENSOR

**Analog-to-Digital Convertor (ADC)**

To be acceptable to a microprocessor or other digital logic system, any varying input signal must first be converted into a suitably coded digital word. This conversion task is performed by an ADC: a circuit which accepts an analog input signal, samples it, and then produces at its output a digital word with a weight that corresponds to the level of the analog input (*fig. A7*).

Take the example of a 3-bit ADC where the digital output can have a coded value from 000 up to 111. This means that the analog input is split up or

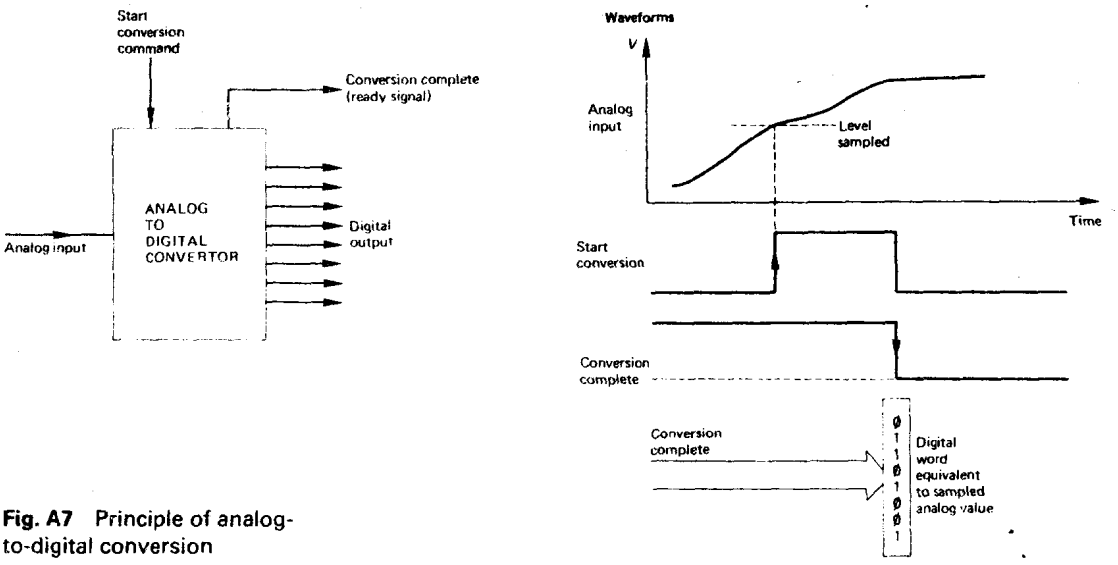



**Fig. A7** Principle of analog-to-digital conversion

"*quantised*" into 8 levels. Suppose that each quantum level is 350 mV. A table showing the values of analog input voltage with corresponding digital codes will be:

| ANALOG INPUT | DIGITAL CODE |
|---|---|
| 0 V | 000 |
| 0.35 V | 001 |
| 0.70 V | 010 |
| 1.05 V | 011 |
| 1.4 V | 100 |
| 1.75 V | 101 |
| 2.1 V | 110 |
| 2.45 V | 111 |

Since the analog input is a continuous signal, there will be some uncertainty over the conversion. This uncertainty is called **quantising error** and will be $\pm\frac{1}{2}$ LSB.

The **useful resolution** of an ADC indicates that no missing codes will be present at the digital output. The table above has no missing codes and therefore the 3-bit ADC can be said to have a useful resolution of 3 bits.

Because of the uncertainty in the conversion, some information detail in the analog input signal will always be lost in the conversion process. This loss can be minimised by increasing the number of bits used. For example, with an 8-bit ADC the analog input will be quantised into 256 levels, and a 12-bit ADC will have 4096 levels.

An important parameter of an ADC is the **conversion time**: the time interval between the command being given to start the conversion and the appearance at the output of the complete digital equivalent of the analog input. The speed of conversion varies with the type of ADC and ranges from the relatively slow (milliseconds) and cheap, to the ultra fast (50 nsec) and relatively expensive. In many applications speed may not be the main consideration and an ADC that is relatively slow can be used.

There are many ways of performing analog-to-digital conversions. The commonly used methods in microelectronic systems are:

Parallel or simultaneous conversion (flash convertor)

Single ramp and counter  Tracking convertor  Successive approximation.

**1 Parallel or Simultaneous ADC** This type, often called a *flash convertor*, is the fastest type available. This is because all the bits for the digital representation of the analog input level are determined simultaneously. The analog input is applied to a parallel bank of voltage comparators, each of which responds to a different discrete level of input voltage.

*Fig. A8* illustrates the principle for just 3 bits but the available types are usually 6 or 8 bits. A constant current source supplies a chain of resistors $R_1$ to $R_7$. These set up the levels at which the seven comparators switch, i.e. 0.5 V, 1 V, 1.5 V, 2 V up to 3.5 V. If the analog input just exceeds 1.5 V, then the outputs of comparators A, B and C will be the logic ∅, while comparators D, E, F and G will give a logic 1 output. The logic gates then convert the outputs of the comparators into the 3-bit digital output, ∅11 in this case. To give ∅11 at the output, the logic has inputs

$\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D \cdot E \cdot F \cdot G$

For $n$ bits of binary in the conversion, the method requires $(2^n - 1)$ comparators plus a lot of logic. An 8-bit flash convertor requires 255 comparators. Therefore this method is not cheap, but now that LSI circuits are available it is increasingly used. An example of the type is the 3300 which is a 6-bit CMOS logic flash convertor with a conversion speed of 15 MHz ($V_{DD} = 8$ V).

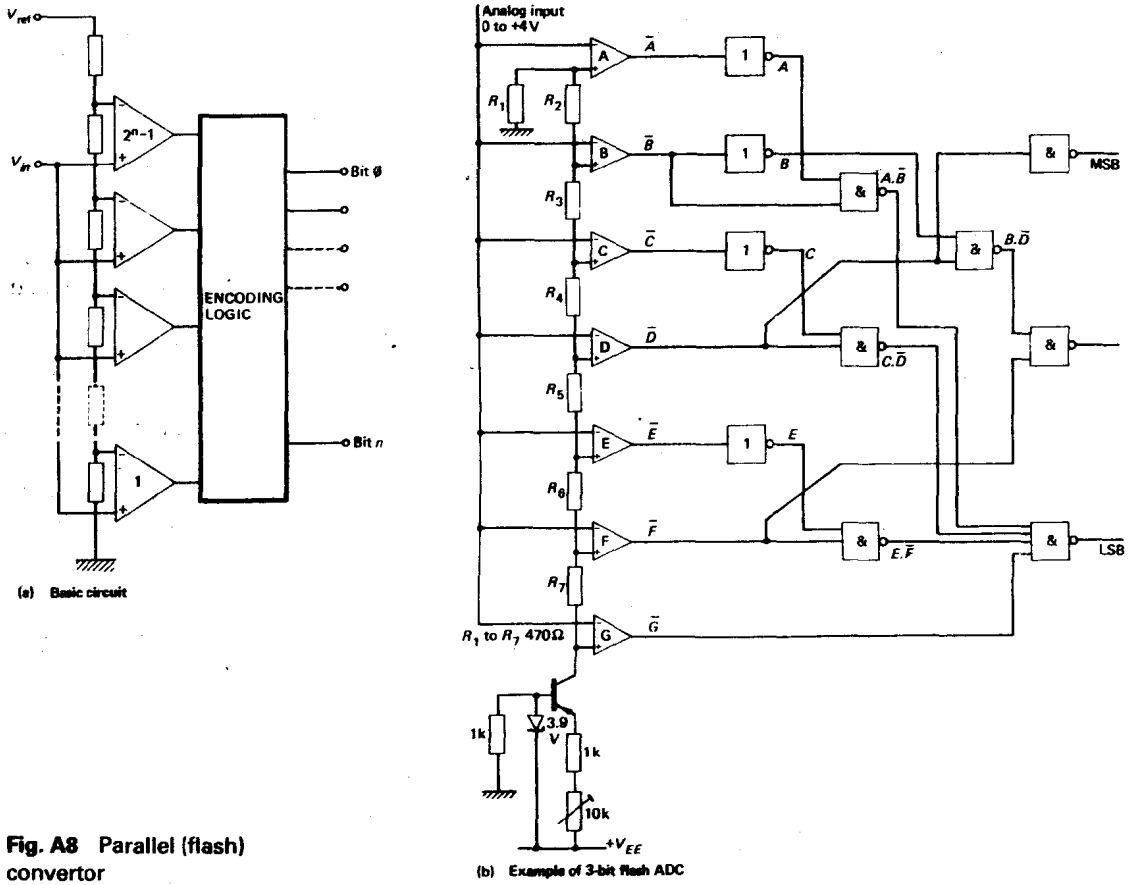



**Fig. A8** Parallel (flash) convertor