ILLUSTRATED DICTIONARY OF MICROELECTRONICS AND MICROCOMPUTERS

R. C. HOLLAND B. Sc., M. Sc.

ILLUSTRATED DICTIONARY OF MICROELECTRONICS AND MICROCOMPUTERS

R. C. HOLLAND B.Sc., M.Sc.
West Glamorgan Institute of Higher Education, Swansea, Wales

PERGAMON PRESS
OXFORD · NEW YORK · TORONTO · SYDNEY · PARIS · FRANKFURT

Preface

Throughout the 1970s and 1980s the expanding technology of microelectronics has brought with it a new vocabulary. The introduction of new electronic devices and systems, particularly the microcomputer, has been so rapid that a large number of new names, definitions and expressions have evolved into common use by workers in the field. This book is an attempt to present a coherent explanation of this new technology. Terms are presented alphabetically, with cross-referencing where necessary, and illustrations are included when a diagrammatic approach assists the definition. In this way the book is more than simply a glossary of terms — it presents detailed explanations of this new technology.

The book should prove to be a useful source of definitions and descriptions to enable a reader with a rudimentary knowledge of electronic or computer principles to understand this new vocabulary. All recent circuits, systems and applications are described. Although primarily aimed at the electronics engineer and student, the book should act as a useful reference guide for the computer hobbyist, computer science student and even the business computer user.

The author wishes to thank his family and colleagues for their support during the preparation of this book.

A

Abort Discontinue operation of the program which is currently being executed within the *computer*. Control is returned to the master program (*operating system* or *monitor*).

Absolute addressing This is an *addressing mode* which is used with *jump instructions*, i.e. *program* instructions which transfer control to a different part of the program. The absolute *memory* address is specified as follows:

```
JMP 1000H     ;Jump to the memory
              address hexadecimal 1000
```

or

```
JZ 0400H      ;Jump, if zero, to memory
              address hexadecimal 0400
```

The full address is included in the instruction, e.g.

First word	JMP
Second word	1000

This addressing mode must be distinguished from the alternative mode which can be used with jump instructions — *relative addressing*. This specifies the relative position, compared with the jump instruction, of the instruction to which program control is to be transferred.

Access time The time interval between a *memory* device (*semiconductor memory* or *backing store*) receiving the address of an item of information and presenting that item in a usable form.

Accumulator A specialised *register* within a *microprocessor* which receives the result of *ALU* operations. Microprocessors possess one or more accumulators which can be used when arithmetical, logical and *shift* operations are required in an *instruction*.

Accuracy A measure of the validity of a measurement. The accuracy of *binary* numbers should not be confused with the resolution of such numbers. For example, a 10-*bit* binary representation of a plant measurement offers more resolution (1 in 1024) than an 8-bit representation (1 in 256), but less accuracy if it is incorrectly generated or processed.

Acoustic coupler A device that allows a *computer* to connect through the telephone network to a remote *peripheral* or another computer. The telephone handset is placed in the acoustic coupler, which is connected to the computer. The coupler contains a *modem*, which converts the *digital* signals into audio acoustic signals.

Acquire See *Capture*.

A/D See *Analogue to digital converter*.

ADC See *Analogue to digital converter*.

Add To generate the sum of two or more numbers. The addition of two single-*bit* numbers is described as follows:

Augend		Addend	Carry	Sum
0	+	0	0	0
0	+	1	0	1
1	+	0	0	1
1	+	1	1	0

Truth table

The addition of two multi-bit numbers is demonstrated as follows:

Carry	00001110		
Augend	00101110	46_+	
Addend	10001011	139^+	Decimal
Sum	10111001	185	equivalent

If the final carry bit of this 8-bit addition process is 1, then the 8-bit sum is not the complete answer — a ninth bit, which represents arithmetic overflow, is required. For this reason, when single *byte* addition is carried out in an 8-bit *microprocessor* the ninth bit is held in the *carry bit*, which forms part of the *status register*. A *program* that performs addition should therefore check this bit if an overflow condition can be predicted.

In addition to a straightforward byte addition instruction, most 8-bit microprocessors offer an *instruction* which adds bytes and also adds the value of the carry from a previous operation, i.e. adds with carry.

Care must be taken when adding *two's complement* numbers, i.e. numbers in which the left-hand bit is reserved as a sign bit. For example:

$1010\ 1101_+$	-83_+
$1011\cdot1000$	-72
$1\ 0110\ 0101$	$+101$
↑	
Carry ignored	

A positive answer (+101) is erroneously achieved unless the carry bit is checked.

See *Arithmetic and logic unit.*

Adder A circuit that performs 1-*bit* addition, as shown in Fig. 1.

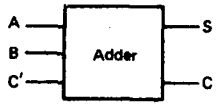
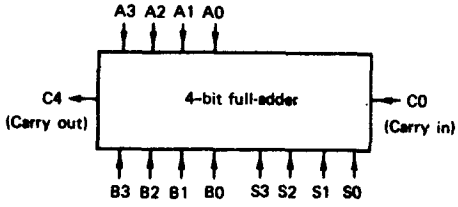


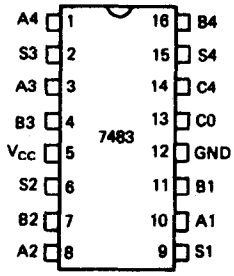
Fig. 1. 1-bit adder.

The C' signal represents the carry from a previous circuit. The adder circuit is in fact constructed using two *half-adders*, and is sometimes referred to as a *full-adder*. The *truth table* for the circuit is shown under *Add*.

A multi-bit adder is available in *integrated circuit* form, e.g. the SN7483, which is shown in Fig. 2.



(a) Functional diagram



(b) IC pin layout

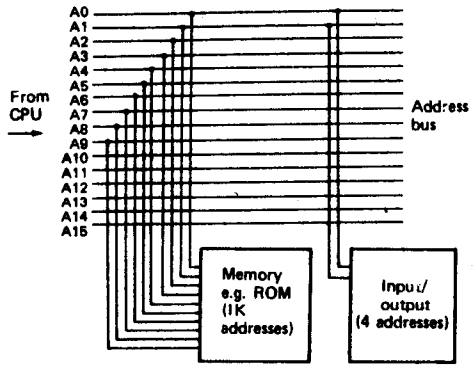
Fig. 2. The SN7483 4-bit full-adder.

The *arithmetic and logic unit* within a *microprocessor* contains an adder circuit.

Address A number that indicates a specific location in *memory* (*semiconductor* or *backing store*) or *input/output*. Normally addresses are 16 *bits* and therefore have a range of 0 to 64K.

Address bus A set of *parallel* connections (normally 16) which are generated by the *CPU* (or *microprocessor*) and pass to *memory* and *input/output* circuits.

Each memory and input/output device connects to as many of the address bus lines as are necessary to select every address on that device. For example, the circuit of Fig. 3 shows the address bus connections to a memory *IC* (1K locations) and to an input/output *IC* (4 locations or addresses).



(*Data bus* connections are not shown)

FIG. 3. Address bus connections to memory and input/output.

The 1K (1024) memory IC requires 10 address lines (2^{10} = 1024 combinations) in order to select each location. The input/output IC requires 2 address lines to select each of its 4 addresses.

Address decoding The technique of selecting a specific *memory* location or *input/output* device. Decoding circuits in *microcomputers* use the *address bus* to select a specific memory or input/output device; the device itself then performs any further address decoding, e.g. to select a specific memory location.

The principal element in an address decoding circuit is a *decoder*, which normally performs 2 to 4 or 3 to 8 decoding. The circuit shown in Fig. 4 demonstrates address decoding to select one of two *ROM*s.

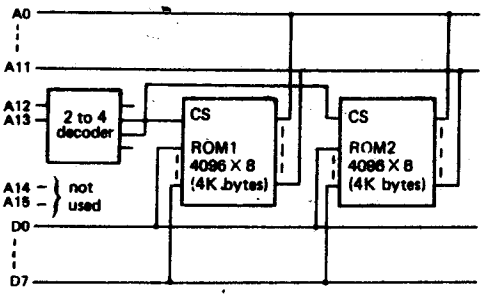


FIG. 4: Address decoding circuit (for 2 memory chips).

Address lines A0 to A11 are required to select each of the 4K addresses on each *chip*, i.e. the chips perform internal address decoding to select the required *byte*. The higher-order address lines A12 and A13 are connected to a 2 to 4 decoder, which generates four *chip select* signals. Each of these signals can be used to select one discrete ROM chip; only two such memory devices are shown connected here. Thus the decoded start address which selects the first byte in ROM1 is:

A15	A14	A13	A12	A11	A10	A9	A8
Not used		0	1	0	0	0	0

2 to 4 decoder

A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	0	0	0	0

This is *hexadecimal* 1000.

Refer to *decoder* to confirm the operation of a 2 to 4 decoder as summarised in its *truth table*.

The decoded start address of ROM2 is:

A15	A14	A13	A12	A11	A10	A9	A8
Not used		1	0	0	0	0	0

2 to 4 decoder

A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	0	0	0	0

This is hexadecimal 2000.

The same address decoding techniques are used to select input/output chips.

Address format The arrangement of the parts of an *address* for a *floppy* or *hard disk*, e.g. drive number, *track* and *sector*.

Addressing modes Different methods of specifying the location of a *data* item which is to be accessed in an *instruction*. For example, a data item may be held:

(a) in a *register*;
(b) in a *memory* location;
(c) in the second *word*/words of the instruction; etc.

See *Direct addressing*, *Absolute addressing*, *Immediate addressing*, *Indexed addressing*, *Relative addressing*, *Paged addressing* and *Autodecrement/autoincrement*.

Algorithm A set of procedures which are required to achieve a desired result. The term is applied to programming, and it is the name given to a description of the steps which a *program* must perform, e.g.

(a) Read a set of switches.
(b) Display a number if any switch is pressed.
(c) Sound an alarm if a particular switch is pressed.
(d) Call a program which reads and processes an instrumentation signal.

Allophone A sound that is generated by a *speech synthesiser* system. Words can be constructed using several allophones. The alternative method of speech synthesiser generates individual complete words. The allophonic technique enables a wider range of words to be generated, but quality of word reproduction is often poor.

Alphanumeric The normal range of numbers (0 to 9) and letters (A to Z). An alphanumeric code sometimes includes additional special control codes. An example of the use of alphanumeric characters is in the use of an alphanumeric display, which may display *characters* in the form of a *dot matrix* or a *segment display* pattern.

ALU See *Arithmetic and logic unit*.

Analogue A signal which is continuous, i.e. it can take any value over its range. For instance, an analogue voltage may take any value over a range of 0 to 10 V, and may represent a plant measurement, e.g. temperature. An analogue signal cannot be handled by a *computer*, and it must be converted to a *digital* form before processing can occur.

Analogue to digital converter Converts an *analogue* voltage into a *digital* representation for use by a *computer* system. A typical system of connection for an analogue to digital converter (or A/D or ADC) is shown in Fig. 5.

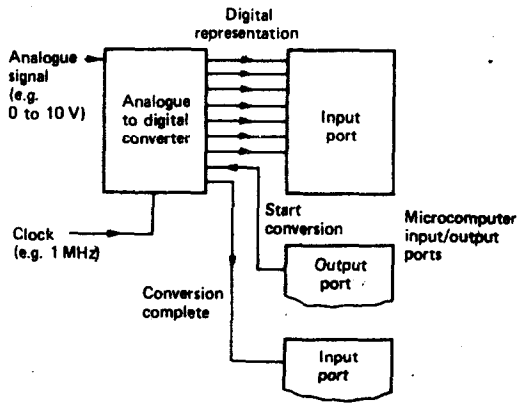


FIG. 5. Connection of A/D to microcomputer.

The A/D converter is a single IC (*integrated circuit*). The digital representation may be 8, 10, 12 or (rarely) 14 *bits* — an 8-bit device is shown in the diagram. Typical analogue voltage ranges are 0 to 2.5 V, 0 to 5 V and 0 to 10 V. The conversion process is initiated by the setting of the Start Conversion signal, and it is timed by the fast Clock pulses. The A/D generates a Conversion Complete signal, which should be checked by the *microcomputer*. The microcomputer *program* should only read the digital representation of the analogue signal when conversion is complete.

The Start Conversion and Conversion Complete signals perform a *handshake* function between microcomputer and A/D.

There are two common techniques employed in the A/D conversion process — see *Successive approximation* and *Integrating A/D*.

AND The *logic* AND function operates on two *bits* as shown in Table 1.

A	B	A.B
0	0	0
1	0	0
0	1	0
1	1	1

Table 1. Truth table for AND function.

The function "A and B" is represented by A.B, where the dot (called "period") represents the logic AND operation. Therefore the result of an AND operation is only set to 1 if both source bits are set to 1. The *truth table* is a convenient method of representing every possible bit combination.

The AND function can be performed by *hardware* (electronic circuitry) or *software* (*computer program*). The hardware AND *gate* can be represented by the circuit symbols shown in Fig. 6.

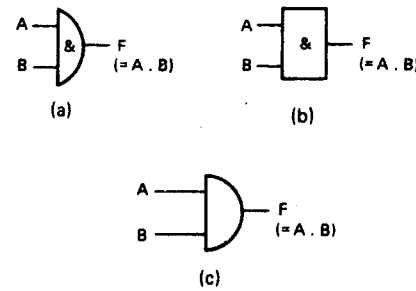


Fig. 6. Circuit symbols for AND gate.

A useful *TTL integrated circuit* which offers three AND gates is the SN7411, which is illustrated in Fig. 7.

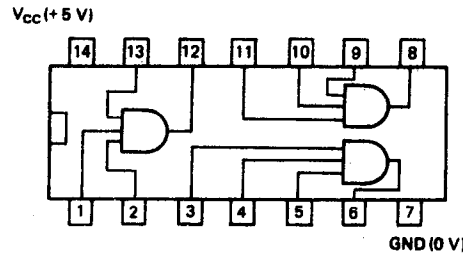


Fig. 7. Triple 3-input AND gate (SN7411).

Notice that these gates have three inputs. Clearly each input must be set to 1 to cause the gate output to be set to 1. Two- and four-input gates are also commonly available.

The software version of the AND function operates as follows. The *instruction set* of every *microprocessor* contains an instruction which performs the AND operation. For example, the AND instruction for an 8-bit microprocessor may be:

```
ANA B;AND the contents of registers
      A and B
```

The effect of this instruction is, for example:

```
A register = 0101 0101
B register = 0000 1111
-----------------------
Result     = 0000 0101
```

Therefore, the instruction sets a 1 in each bit of the 8-bit result only if both corresponding bits in the source data items are 1. This function is of value when it is required to *mask* (i.e. set to zero) certain bits in a data item, e.g. the top 4 bits in the A register in the example above.

ANSI (American National Standards Institute) An organisation responsible for setting standards, e.g. for *computer* systems.

Application package/software A *program* or set of programs that performs a particular function, e.g. *stock control* or *payroll*, and which may have to be tailored to satisfy the requirements of a particular user.

Architecture The name given to the generalised *hardware* configuration of a *microcomputer*.

Argument The name given to a number which is passed from one part of a *program* to another. For example, a section of *high-level language* program may call a *subroutine* and wish to pass a number to that subroutine. Alternatively it may wish to enter a section of *machine code* program and require to pass a *data* item.

Arithmetic and logic unit The heart of a *microprocessor* which performs arithmetic, *logic* and other functions. It is commonly called the ALU. Refer to *Microprocessor* to see its overall role within the microprocessor.

The functions which are performed by the ALU are summarised in Fig. 8 for an 8-*bit* microprocessor.

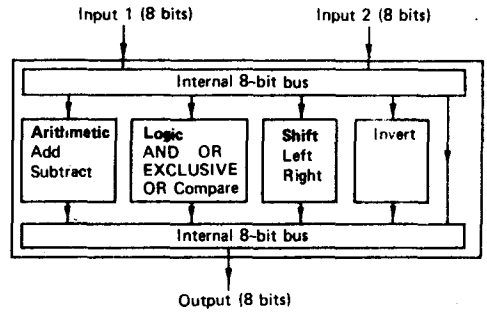


FIG. 8. Internal organisation of the ALU.

There are two input channels, which are normally fed from microprocessor *registers*, and one output channel, which normally feeds the *accumulator*.

Arithmetic operations are normal add and subtract. Multiply and divide are only available in 16-bit microprocessors.

Logic functions are *AND*, *OR* and *EXCLUSIVE OR*. Additionally two numbers can be compared, i.e. checked if they are identical or if one is greater than the other.

Shift left and shift right functions can be performed using one 8-bit input. These operations are useful if it is required to remove bits in a *data* value (e.g. shift a number one place to the left so that the most significant bit is lost), or to multiply or divide numbers by powers of 2 (e.g. shift left one place multiplies the number by 2, shift left one more place multiplies it by 4, etc.).

The invert, or "complement", function changes each bit in a single 8-bit data value. Thus each 0 is changed to a 1, and each 1 is changed to 0.

Additionally a direct path through the ALU exists, i.e. no processing is performed on an input data value or number. Thus an input from one of the microprocessor's registers can be passed directly through the ALU to transfer into another register.

Arithmetic shift A shift that retains the setting of the sign *bit*. Additionally it is a shift operation which effectively multiplies or divides a signed number by powers of 2. For example:

$+3_{10}$ = 0000 0011

↑

sign bit (0 = positive
1 = negative)

Shifting left 1 place gives 0000 0110 = $+6_{10}$.

Therefore multiplication by 2 has occurred.

Similarly:

-6_{10} = 1111 1010 (see *Two's complement* for negative number representation)

↑

sign bit

Shifting right 1 place gives:

1111 1101 = -3_{10}

↑

1 shifted in

Therefore division by 2 has occurred.

Thus arithmetic right shift *instructions* ensure that a 0 (for positive number) and a 1 (for negative number) is shifted into the sign bit.

Notice that errors can occur if the penultimate bit (the bit after the sign bit) is different from the sign bit for a shift left instruction. For example:

$+64_{10}$ = 0100 0000

Shifting left 1 place gives 1000 000 = -128_{10} not the correct $+128_{10}$.

Therefore the programmer should take great care with this particular instruction.

Array A list of numbers or *variables* which is accessed in a *high-level language program* using a two-dimensional reference (occasionally three-dimensional). For example, if six numbers are stored by the program, and it is required to reference them using two-dimensional co-ordinates as follows:

TOM(1,1)=100 TOM(1,2)=150 TOM(1,3)=360
TOM(2,1)=400 TOM(2,2)=135 TOM(2,3)=270

then any number from this list of 100, 150, 360, 400, 135 and 270 can be accessed and used within the program, e.g.

```
50 CHARLIE=TOM(2,1)+500
```

So CHARLIE takes the value 900.

ASCII (American (National) Standard Code for Information Interchange) This is a world-wide standard code for 7-bit coded *characters* (plus 1 bit for *parity* check) which is used for information interchange between *computers* and external *peripherals* (*VDUs*, *printers*), or other computers. The full *alphanumeric* set of numerals, letters, punctuation symbols and special control characters is included, as shown in Table 2.

See *RS 232-C*.

Assembler A program that translates *assembly language* statements (in *mnemonic* form) into *machine code*.

There are two types of assembler:

(a) Full assembler, which waits until the complete program is entered, and then generates the machine code version — comments, which describe program operation, can normally be entered and memorised with this type (see also *Macro-assembler* and *Two-pass assembler*).

(b) Line-by-line assembler, which converts each assembly language *instruction* into machine code as each instruction is entered.

Assembly language A programming *language* that is line-for-line convertible to *machine code*, but which uses *mnemonics* (that help to describe *instruction* action) and *labels* (words in place of absolute memory addresses).

Character	Hex.
NUL	00
SOH	01
STX	02
ETX	03
EOT	04
ENQ	05
ACK	06
BEL	07
BS	08
HT	09
LF	0A
VT	0B
FF	0C
CR	0D
S0	0E
S1	0F
DLE	10
DC1	11
DC2	12
DC3	13
DC4	14
NAK	15
SYN	16
ETB	17
CAN	18
EM	19
SUB	1A
ESC	1B
FS	1C
GS	1D
RS	1E
US	1F
SP	20
!	21
"	22
#	23
$	24
%	25
&	26
'	27
(	28
)	29
*	2A
+	2B
,	2C
-	2D
.	2E
/	2F

Character	Hex.
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39
:	3A
;	3B
<	3C
=	3D
>	3E
?	3F
@	40
A	41
B	42
C	43
D	44
E	45
F	46
G	47
H	48
I	49
J	4A
K	4B
L	4C
M	4D
N	4E
O	4F
P	50
Q	51
R	52
S	53
T	54
U	55
V	56
W	57
X	58
Y	59
Z	5A
[	5B
\	5C
]	5D
^	5E
_	5F

Character	Hex.
`	60
a	61
b	62
c	63
d	64
e	65
f	66
g	67
h	68
i	69
j	6A
k	6B
l	6C
m	6D
n	6E
o	6F
p	70
q	71
r	72
s	73
t	74
u	75
v	76
w	77
x	78
y	79
z	7A
{	7B
|	7C
}	7D
~	7E
DEL	7F

Note Characters hex. 00 to 1F are control characters. Character hex. 7F is delete, or rub-out.

Table 2. ASCII Code

Therefore the programmer must understand machine operation, but he does not need to generate the *bit* pattern (or *hexadecimal* equivalent) of each instruction. He calls an *assembler* to perform the conversion for him. The assembly language version of the program is often called the "source code", and the generated machine code version is called the "object code".

An example of an assembly language program is:

```
        MVI A,9     ;Move 9 into A register
LOOP:   OUT 40H     ;Output A to port address 40
        DCR A       ;Decrement A
        JNZ LOOP    ;Jump, if A is not zero, to
                     LOOP
```

Notice that MVI, OUT, DCR and JNZ are mnemonics which are chosen to meaningfully represent the instruction required. LOOP is a label, which allows the programmer to write the program without having to worry about the memory location of the OUT instruction when he refers to it in the JNZ instruction. See also *Pseudo-instruction* and *Macro-assembler*.

Assign To allocate a name to a *variable* in a *program*.

Astable multivibrator A circuit which generates *pulses*. Strictly it is a multivibrator, or two-state circuit, which has no stable state, i.e. it oscillates from one state to the other continuously. It is used as a pulse generator circuit for applications such as *CPU clock*, *A/D* converter clock, etc.

It can be constructed using two inverting *gates* as in Fig. 9:

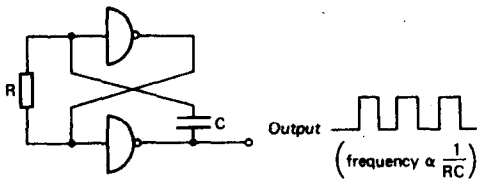


FIG. 9. Astable multivibrator using two inverters.

An alternative circuit which uses a 555 timer chip (effectively two inverting gates connected *back-to-back*) which gives greater control of pulse waveform is shown in Fig. 10.

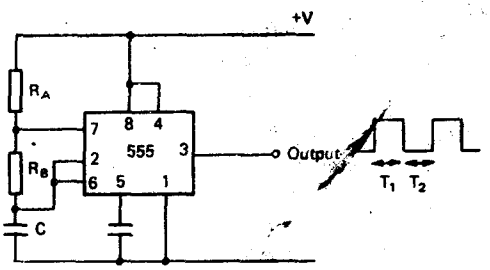


FIG. 10. Astable multivibrator using 555 timer.

$$T_1 = 0.7(R_A + R_B)C$$
$$T_2 = 0.7R_BC$$

Thus the pulse frequency, as well as the *mark/space ratio*, can be selected by suitable choice of R_A, R_B and C.

Asynchronous A circuit or a system that is not synchronised by a common *clock*.

In a synchronous *counter* circuit, each stage of the counter is triggered at each occurrence of the common clock pulse. In an asynchronous counter circuit, the stages do not share a common clock pulse signal but trigger one after the other.

In *serial* data transmission systems (see *RS 232-C*) an asynchronous link does not use a clock pulse which is common between, and therefore connected between, transmitting and receiving circuitry. In place of this an identifying start pulse is generated by the transmitting circuit before the *character* code, so that the receiving circuit is primed to receive the character.

ATE (Automatic Test Equipment) *Computer* based equipment is commonly used to test manufactured electronic components and systems. The advantages of this technique are speed

and detail of test procedures, as well as reprogrammable nature of test equipment.

Attribute A property or characteristic assigned to a *data* value in a *program*, e.g. *real* or *integer*, single length or double length number.

Audio cassette A domestic audio cassette recorder can be used to store *programs* and *data* from *microcomputers*. Such recorders are cheap and readily available. See Fig. 11.

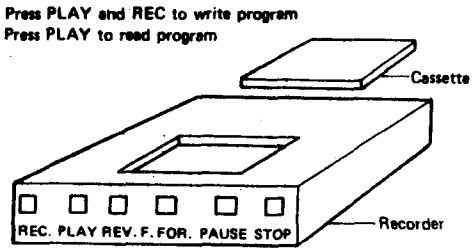


FIG. 11. Audio cassette recorder.

Autodecrement/Autoincrement

A variation of *indirect addressing* in which the *memory* address, which is used to reference a *data* item, is automatically decremented (1 subtracted) or automatically incremented (1 added) when the *instruction* is completed. Few *microprocessors* possess this feature, and the autoincrement is the more common of the two.

For example,

MOV Register 1, Indirect Register 3+

is an instruction *mnemonic* (for demonstration purposes only) which moves the contents of Register 1 to the memory address which is held in Register 3 (indirect addressing), and adds 1 to the contents of Register 3. Therefore, when the instruction is completed Register 3 holds the address of the memory location which follows that used in the instruction.

B

Backing store A *data* storage medium that "backs up" the *main memory* within a *computer*. Normally a backing store is an electromechanical system which provides a large amount of memory (100 *Kbytes* up to several *Mbytes*) but with an *access time* that is far slower than main memory.

See *Floppy disk*, *Hard disk*, *Audio cassette*, *Cartridge disk* and *Bubble memory*.

Backplane A circuit board that supports other boards in an electronic system. Therefore, other boards plug into a backplane, which carries interconnections between boards, in the following manner:

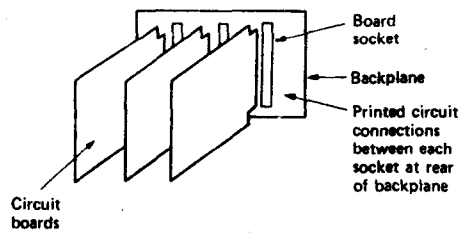


FIG. 12. Backplane supporting circuit boards.

Whilst the boards which plug into the backplane support components, the backplane itself normally supports only interconnections. Invariably these interconnections are *printed circuit*, but they could be wire connections. If the backplane supports circuitry, it is commonly referred to as the *motherboard*.

Back-to-back A circuit in which the output is connected to the input. An example of a back-to-back connection is shown in Fig. 13 and applies to a *microcomputer serial input/output* channel.

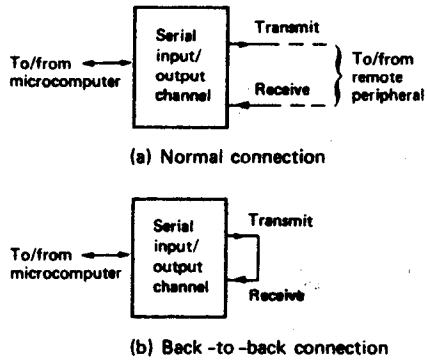


FIG. 13. Serial link in normal and back-to-back connection.

In the normal mode of connection a separate transmit and receive signal connect the microcomputer to the remote *peripheral*, e.g. a *VDU*. If the input/output channel is connected "back-to-back", then the link to the remote peripheral is isolated, and the signal that is transmitted from the channel on the output connection is received simultaneously on the input connection. Such an arrangement assists fault tracing if a failure in the overall microcomputer drive to the peripheral exists because it allows testing of the input/output channel in isolation from the interconnecting cables and the remote peripheral itself.

Back-up A standby facility which can be activated in the event of failure, e.g.

(a) Back-up of *hardware* — spare modules of electronic circuitry (or electromechanical *peripherals*) which can replace faulty modules either by manually changing the module or even by automatic changeover to the standby unit, e.g. dual *floppy disk* drives on a large *microcomputer* system.

(b) Back-up of *software* — second copy of a computer *program* or *data* file which can be used to overwrite corrupted software.

(c) Back-up of technical service — advisory service by a source of hardware or software expertise.

Base The total number of distinct symbols in a numbering system; sometimes called *radix*. The normal *decimal* (or "denary") system uses a base of 10, e.g.

Decimal 6 can be represented as 6_{10} (6 to the base of 10), and also as 6×10^0 (10 to the power 0).

Thus a larger number is:

$$\text{Decimal } 527 = 5 \times 10^2 + 2 \times 10^1 + 7 \times 10^0 = 500 + 20 + 7 = 527_{10}$$

Computers use *binary* numbers (base 2), but we normally refer to these numbers using the *hexadecimal* (base 16) numbering system because it makes the numbers shorter and more manageable.

BASIC BASIC is by far the most common *high-level language* that is used with *microcomputers*. The word BASIC stands for Beginners All-purpose Symbolic Instruction Code.

BASIC was designed to be an easy-to-use language which programmers can apply to write and test *programs* rapidly with a minimum knowledge of microcomputer operation. The extent of standardisation of this language throughout the microcomputer industry is such that a BASIC program which is written for one machine can be transferred normally with a minimum of amendment to another machine.

An example of a simple BASIC program is:

```
10 REM TEST PROGRAM TO
   DEMONSTRATE SIMPLE
   ARITHMETIC
20 FIRST=999
30 SECOND=123
40 REM DISPLAY SUM
50 PRINT FIRST+SECOND
60 REM DISPLAY PRODUCT
70 PRINT FIRST*SECOND
80 END
```

Each line, or "statement", is numbered. It is normal to increment line numbers by 10 so that additional statements can be inserted later if the program is amended. The statements at lines 10, 40 and 60 are REM (or remark) statements, and are not implemented when the program is run — they simply allow a method of program documentation to be built into the program listing. The two numbers are given the names of *variables*, e.g. FIRST and SECOND, which can be virtually any collection of letters, but are normally chosen by the programmer to be as meaningful as possible. The PRINT commands cause the value of the variable, or a combination (sum and product in the example above) of variables, to be displayed on the computer's *CRT* screen.

After a BASIC program is entered into a microcomputer, it is run at a later time by an *interpreter* or *compiler* which converts it into *machine code* before execution.

Different versions of variable complexity and ease of application, of BASIC are available, e.g. basic BASIC, extended BASIC, structured BASIC, and versions which are tailored to a specific microcomputer (perhaps to include commands which activate colour *graphics*).

An example of a more complicated BASIC program which demonstrates more of the standard commands is:

```
10 REM THIS PROGRAM
   DISPLAYS THE SQUARE
   ROOTS OF SEVERAL
   NUMBERS
20 DATA 49,184,26,403,72
30 FOR I=1 TO 5
40 READ CHARLIE
50 PRINT "SQUARE ROOT OF";
   CHARLIE; "IS"; SQR(CHAR
   LIE)
60 NEXT
70 END
```

In this example the DATA and READ commands are used in conjunction with each other. The DATA command specifies a list of data items, and whenever the READ command is implemented in the program each succeeding data item is extracted and given the variable name CHARLIE. The FOR and NEXT commands similarly work together, because the section of program which is enclosed by these commands (lines 40 and 50 in this example) is executed the number of times specified in the FOR statement (5 times in this example). Notice that the PRINT command demonstrates the use of two print options — text, in inverted commas, and variables, e.g. SQR(CHARLIE). Thus when the program is executed, by typing in RUN, the following is displayed:

```
SQUARE ROOT OF  49 IS  7.0000
SQUARE ROOT OF 184 IS 13.564
SQUARE ROOT OF  26 IS  5.0090
SQUARE ROOT OF 403 IS 20.075
SQUARE ROOT OF  72 IS  8.4853
```

Baud rate The speed of *data* transmission, expressed in signal elements per second. Normally the term is applied to *serial* data transmission systems in which one signal element is one *bit*, so that baud rate = bits per second.

Examples of application of the term are:

(a) Serial transmission using the *RS 232-C* interface between *computers* and *peripherals* (or other computers). In this case normally 8 data bits are transmitted one after the other (together with a start bit and a

stop bit) along a single conductor, so that one *character* requires 10 bits, as follows:

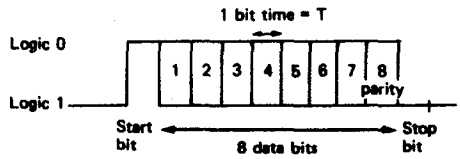


FIG. 14. Serial data transmission waveform.

If T = 0.001667 sec (1.667 msec) then baud rate =

$$\frac{1}{1.667 \times 10^{-3}} = 600$$

Thus a baud rate of 600 gives 60 characters per second. This is a typical transmission speed between computer and *printer*. Faster speeds are typical for *VDUs* and inter-computer links. The standard range of baud rates which can be generated by *UARTs* (generators of this serial waveform) are:

110, 300, 600, 1200, 2400, 4800 and 9600

(b) Serial transmission to and from *floppy disk* (or *hard disk*) read/write *head* and controlling circuit, again expressed in bits per second.

(c) *Bus* transfer speeds, e.g. the speed of *byte* transfer from a microcomputer master board (containing *microprocessor* and *bus drivers*) along a *backplane* to supporting boards using the *address bus* and *data bus*.

BCD (Binary Coded Decimal)

This code uses 4 *bits* to represent the 10 *decimal* numbers 0 to 9, as shown in Table 3.

Decimal number	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table 3. BCD (Binary Coded Decimal)

Notice that the last 6 of the possible 16 codes are not used. Normally two BCD *digits* are packed into one *byte*. For example, the BCD number 8413 can be held in two bytes as follows:

0001 0011	13
1000 0100	84

Normally it is inconvenient to handle BCD numbers within a *microcomputer program* because conversion to pure *binary* numbers may be required.

Benchmark A parameter for comparison between two systems. Benchmarks are commonly applied to compare *microprocessors* as follows:

(a) Time to perform 8-*bit* addition.

(b) Time to perform a more complicated and extensive test exercise, e.g. a *program* which transfers a block of *data*, with manipulation (perhaps shifting and logical modification), from one area of *memory* to another.

Bidirectional Signal flow can pass in either direction. Bidirectional signal flow along the same conductor (or conductors) is uncommon in most electronic systems, but the most familiar example is the bidirectional *data bus* within a *microcomputer*.

Binary A number system which uses a *base* of 2; the normal *decimal* number system uses a base of 10. Only two symbols are used in the binary system — 0 and 1. It is far easier to design electronic circuits which handle only two signal levels, e.g. a voltage and no voltage, and it is for this reason that *computers* handle numbers in binary form.

A "bit" (binary digit) can take the value of 0 or 1, and forms one part of a binary number, as follows:

$$\text{Binary } 1101 = 1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 8 + 4 + 0 + 1 = 13_{10}$$

Thus 1101 is the binary representation of decimal 13. Each bit therefore represents a component, in "powers of two", of the overall number. 8-bit *microprocessors* represent numbers using 8 bits, e.g.

Binary 0100 1100 =

128 64 32 16 8 4 2 1 (powers of 2)

0 1 0 0 1 1 0 0 = 76_{10}

Therefore the largest number that can be represented using 8 bits is 255.

A method for performing conversion in the opposite direction, i.e. from decimal to binary, is as follows:
Convert 12_{10} to binary.

		1	1 (third remainder)
Continually →	2	3	0 (second remainder)
divide by 2	2	6	0 (first remainder)
	2	12	

Answer = 1100_2

Binary dump A transfer of the contents of *memory* to a storage medium, e.g. *magnetic tape* or *printer*, in *binary* (or *hexadecimal*) form.

Bipolar Possessing two poles, i.e. containing electric charges of opposite polarity. The *TTL* family of *integrated circuits* uses bipolar *transistors*, in which both positive and negative charge carriers exist. The recent *MOS* and *CMOS* families use *unipolar* transistors, in which only one type (positive or negative) of charge carrier occurs — these transistors are often called *FETs* (Field Effect Transistors).

Bipolar integrated circuits (*ICs*) are classified by circuit type:

(a) *TTL* (Transistor Transistor Logic);
(b) *ECL* (Emitter Coupled Logic);
(c) I^2L (Integrated Injection Logic).

The conventional circuit symbol for a bipolar transistor is shown in Fig. 15.

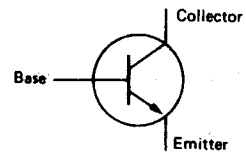


FIG. 15. Bipolar transistor circuit symbol (NPN transistor).

The *planar* construction of a bipolar transistor is illustrated in Fig. 16 for a TTL IC (integrated circuit).

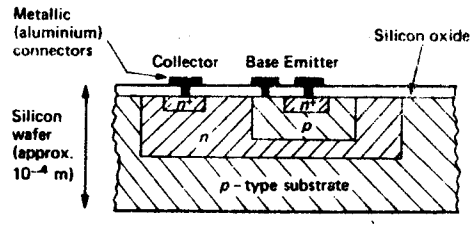


FIG. 16. Bipolar "silicon planar epitaxial" transistor.

The manufacture of this device involves oxidation of the surface of the silicon wafer, followed by a sequence of photomask and diffusion processes in order to transform areas of the silicon into p (majority carriers are positive) and n (majority carriers are negative) type. Metallic interconnections, which are