TIMES COMPUTER DICTIONARY

a guide to computer jargon

JON WEDGE

JIM BARKER

73-61
392

8963786

TIMES COMPUTER DICTIONARY

a guide to computer jargon

JON WEDGE

Illustrated by

JIM BARKER

FEDERAL PUBLICATIONS
Singapore • Kuala Lumpur • Hong Kong

First published 1984 by
A & C Black (Publishers) Limited, London
with the title *The Computer Dictionary*.

This edition first published 1985 by
Federal Publications (S) Pte Ltd
Times Jurong, 2 Jurong Port Road, Singapore 2261

ISBN 9971 4 0665 9

Illustrations: Jim Barker (mostly from originals by
Christine Wedge)

Printed by Chong Moh Offset Printing Pte Ltd

To Christine, Bill and William

CONTENTS

INTRODUCTION

There was once a Computer Person who found himself struggling to maintain his grip on the English language.

This Computer Person decided to see if he could explain the words that he used all day in such a way that a Reasonable Person could understand them.

He wanted to make computer terminology understandable to someone who did not wish to embark on a full computer education exercise.

He did not like working, so he also tried to ensure that reading his book would not seem like work to others.

When he had finished he had a long think about how successful the methodology had been, functionality-wise. Then he devoted some resource to having a beer or two.

He never did reach a conclusion, but here is the book.

Abort

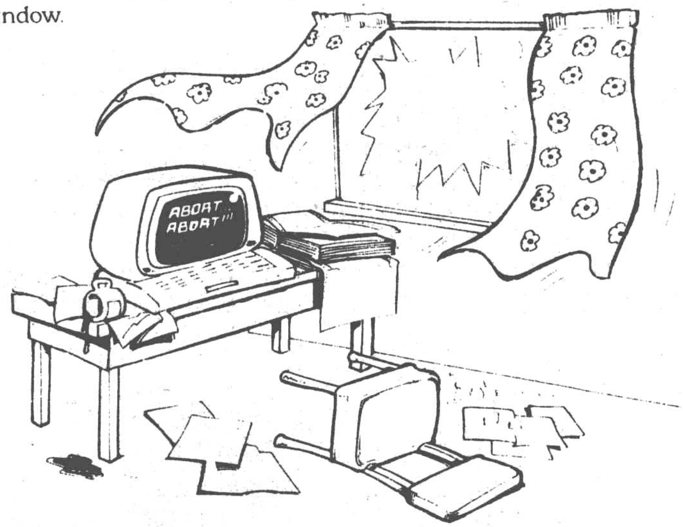
To **abort** a program is to leave it all of a sudden. It is rather like leaving an office block by jumping out of the window.

You would only jump out of the office window if something was seriously wrong, like the building being on fire. Similarly the computer would only abort a program if something drastic had occurred, for example if the program had tried to multiply somebody's date of birth by their surname.

This should never happen of course, but neither should office blocks catch fire.

A variation on an abort is known as an 'Abend', for Aborted, Abnormal or Abandoned End. This is slightly more dignified, in that the program is able to do a bit of tidying up, say closing the **files**, before aborting.

This is broadly equivalent to clearing up your desk before jumping out of the window.



Absolute Addressing

When a program is being run the program instructions and the areas used for data occupy the memory of the computer.

Some types of instruction 'branch' the program to other instructions, while other types access the data areas. Both groups therefore need some means of 'addressing' the appropriate memory locations.

The most common addressing method allows the programmer to use **symbolic** names for the 'addressed' instructions and data areas. The **assembler, compiler** or **interpreter** works out the actual physical byte numbers within memory (see **symbolic addressing**).

Absolute Addressing

Some languages also have **absolute addressing**, whereby the programmer works out and specifies the physical byte number of the 'addressed' data area or instruction for himself.

Doing this is really stupid. If additional instructions are later inserted in the program the original instructions and data areas may be displaced, thus ending up in different memory locations the next time that the program is used. The associated absolute addresses that the programmer originally specified are now wrong, and the program will not work correctly until they have all been changed.

There are only two justifications for absolute addressing. One is for those who are programming just for excitement and find that being permanently on the brink of disaster helps. The second applies to those who are seeking to prove that they cannot program in order to get promoted to **systems analyst**.

Acoustic Coupler

An **acoustic coupler** provides a slow but cheap means of transferring data between computers and 'intelligent' devices in a remote location. It does this by using ordinary telephone handsets.

The device could be a keyboard and printer, or a calculator-like box which has had information keyed into it specifically for transfer to the main computer.

The acoustic coupler consists of a box of electronic tricks – a kind of **modem** – with a couple of wobbly rubber coffee cups sitting on the top.

The wobbly rubber coffee cups are designed to hold the mouth- and ear-

pieces of a telephone handset. The acoustic coupler is also plugged into the computer or other device.

You have an acoustic coupler at each end of an ordinary telephone line. The data then goes between the device and the computer via the line, having been converted into an acoustic signal for the telephone-to-telephone part of its journey.

Of course acoustic signals and wobbly rubber coffee cups form a fair old obstacle course for even the fittest piece of data, which is why transfer speeds are relatively slow.

ADA

ADA is a high-level programming **language**. It was created to reduce the enormous amount of time spent on the development and maintenance of programs used by Western government defence departments.

Speculating on what the acronym ADA stands for is interesting, and in the right company can be worth a few quid in wagers.

There is plenty of scope because it is not an acronym at all. ADA was in fact named after the only child of Lord and Lady Byron, Ada Lovelace. In the 1840s Lady Lovelace consolidated and refined Babbage's work on computing engines and their use. She is sometimes called the first programmer.

ALGOL

ALGOL is a high-level **language** for use in scientific applications. The major design objective was to ensure that the language would be usable on a wide range of different computers.

ALGOL stands for ALGOrithmic Language, which is enough to put anyone off. This it has largely succeeded in doing.

Algorithm

An **algorithm** is the calculation required to resolve a complicated formula. In a computer an algorithm is worked out by a program.

Opinion varies as to how complicated a formula has to be to need an algorithm rather than a mere calculation to resolve it. Some would argue that nothing short of a program to work out which direction you should follow to arrive at Uranus in two years' time deserves the accolade of an algorithm. Others would use an algorithm to work out their winnings from £2 on the nose at 6-4 on.

Analyst/Programmer

An **analyst/programmer** performs the functions of both the **systems analyst** and the programmer. The two jobs involve specifying what a set of **application programs** should do, and actually writing the programs.

The problem is that the two jobs attract entirely different types of people. The analyst normally likes to deal with the outside world and would prefer not to get involved in the detail of what goes on inside the computer. The programmer, on the other hand, likes playing around on the computer and has a strong preference for being left well alone by the outside world.

An analyst/programmer is therefore usually either an analyst who has been forced to program or a programmer who has been forced to analyse.

It is easy to tell which is which.

The analyst prefers writing **specifications,** does it first, and then gradually simplifies it because trying to get the program to do all the things he originally specified is driving him crazy.

The programmer writes the program first because the only way he can get the specification correct is to finish the programming first.

Neither approach is ideal.

The other difficulty facing the analyst/programmer is that analysts habitually blame programmers if something goes wrong, and programmers habitually blame analysts. Things still go wrong for analyst/programmers. This is why so many of them drink too much.

Application Program

The **application program** is important in a computer system in the same way that speakers of some kind are important in a stereo system. Without it the computer is no use to anyone.

It is the application program that dictates what appears on the screen, what is printed, what is recorded on disk or cassette, what keyboard entries are required, what they need to be checked against, what they must be added to and so on and so on and so on and on, and it even decides when to stop and let another program have a go.

A Star Wars game on a home computer could be considered an application program, as could a payroll program running on a microcomputer, or a data-entry program running 40 screens with 40 operators on a **mainframe** system.

An application program can be written in any language. However, it will probably rely on **systems programs** to do the dirty work. For example, the application program might issue an instruction to write someone's payroll record to the payroll file on disk. It is the systems program's job to work out exactly where on the disk it should actually be written to.

The working out is quite important. It is no good burying a bone if you cannot find it again.

The application program, then, is what you see and use for a particular job. This makes it the computer's contact point with the outside world.

Array

See **DIM**.

ASCII

ASCII stands for the American Standard Code for Information Interchange. It is pronounced 'Askey' in tribute to that great English comedian, Arthur.

ASCII is one of two dominant coding systems (the other is **EBCDIC**) defining the combination of bits which represents each character. It is like a souped-up Morse code, using the on/off condition of bits instead of dots and dashes.

Each character normally has a byte devoted to it, though ASCII in fact only uses seven of the eight bits in a byte. (If you are not entirely sure what bits and bytes are, then be practical. Look them up now. This is going to get worse before it gets better.) The eighth bit is set to 'O'.

Seven bits provide 128 unique combinations, which allows for unique definitions of each letter, both lower and upper case, plus numbers, punctuation marks and symbols. This leaves a few for the computer's internal use (see **control characters**).

To illustrate the use of ASCII here's an example of what would be sent to the printer if the word 'Poppy' is to be printed. The **hexadecimal** notation is also given for completeness. You can ignore it for now if you like. However, if you do not know what hexadecimal is and you are not even curious, what on earth do you think you are doing reading up on ASCII?

CHARACTER	P	o	p	p	y
BITS SENT	01010000	01101111	01110000	01110000	01111001
HEXADECIMAL	50	6F	70	70	59

Incidentally, some weirdos know the entire ASCII code set by heart. I know I do. I like to brag about it, but it never seems to do me any good.

Assembler

See **assembly languages**.

Assembly Languages

Back in the good old days when men were men and programmers were programmers the programs were written and executed in the form of a glorified string of bits called **machine code.**

This made life easy for the computer and very difficult for the programmer.

As computers became more powerful, **assemblers, compilers** and **interpreters** were developed. These converted programs written in a language (see **source code**) into a form which could be run by the computer (see **object code**).

This made life a lot easier for the programmers, who could now write programs using symbols and real words rather than bit strings.

The many **assembly languages** (also referred to as 'assemblers') convert direct into machine code through the good offices of an assembler **utility program**. This proximity to the machine code makes them 'low-level' languages. If you need convincing, look at this sequence of Intel 8080 instructions, which come with equally incomprehensible comments:

```
MOV A,B       *Move register B to register A
ORA A         *Logical 'OR' on A using Accumulator contents
JNZ AGAIN     *If no bits are at zero jump to 'AGAIN'
```

See **languages** for a comparison of this kind of thing with boring high-level languages. The important thing here is that this is definitely one step up from a string of bits. It also allows use of **symbolic addressing**, which is a great step forward from raw machine code.

Nowadays virtually all machine code is first written in an assembly language. They certainly have a little more zap than your average high-level language.

'PRINT RESULT'. Call that programming?

Asynchronous Communication

Asynchronous communication is a method of sending data from one device to another.

The devices could be two computers, two telex machines, or a computer and a **peripheral.**

Asynchronous communication means that characters are sent individually, as and when they are ready. The devices at either end do not need to 'synchronise' on a consolidated block of characters such as is sent when using **synchronous communication.**

Characters sent asynchronously are like drips from a tap. Each drip has its own start and stop bits and because of this overhead it is not possible to achieve the data transfer speeds attainable with synchronous blocks. However, transfer speed is not exactly vital when the receiving device cannot handle data quickly, as is the case with most printers or telex machines.

ATM

An **ATM** is an Automatic Teller Machine. It is a machine which provides some of the services usually provided by a bank teller, or cashier.

The ATM lets you identify your account by the insertion of a plastic card. You verify that you are the authorised card user by entry of a number. This is called the PIN, or personal identification number.

The ATM is then able to look up your account record on the computer, check that the PIN is correct and, unfortunately, check how much money you have in the account.

Backing Storage

See **storage**.

Bar Codes

Bar codes are the curious black and white stripes that you see on cans of baked beans, or library tickets. You can also see one on the back of this book. If it is not there, somebody has changed the cover.

A bar code is simply another way of representing numbers and characters, each of which has a unique combination of stripes, defined by a coding system.

There are several different systems in common use; in most of these the stripes come in four varieties, thick and thin black, and thick and thin white.

Bar codes are used because they are easy to read into a computer using a special 'pen', or 'wand'. The pen works out for itself how fast it is being moved over the code from the first few stripes, which are the same in every code for which the pen is used. It can then convert the time it spends looking at black or white into the stripe pattern, and thus into the characters being represented.

It makes sure that the read was accurate using a **redundancy check** calculation. Most pens emit a beep to tell the operator that the read has worked.

Shortly after a friend of mine accidentally became a **technical programmer** with a **software house** he got involved in trying to get a prototype bar code reader to send the characters it had read into a Point of Sale **(POS)** computer. His role was to sit with someone else and say 'I see' or 'Oh dear' every once in a while.

He said that it was quite relaxing, because each time they blew up a prototype they had to wait ages for another one to arrive. This is all part of being at the forefront of technology.

Barrel Printers

See **printers**.

BASIC

BASIC is a simple high-level programming **language**. It is the most commonly used language for home and personal microcomputers, because it is easy to learn.

BASIC stands for Beginner's All-purpose Symbolic Instruction Code and was originally developed in Dartmouth College, New Hampshire, in 1965. It has since bred uncontrollably, and now there are literally hundreds of slightly different BASICs, having slightly different instruction sets.

You cannot have it both ways of course. BASIC may be easy to learn, but it is quite inadequate for getting many jobs done. Similarly, it is relatively easy to learn to fly a hang-glider, but a hang-glider is not much good for a real flying job like transporting 300 people from London to New York.

However, programming, even in BASIC, comes more naturally to some people than others . . .

I'm a middle-aged commuter
Who has bought a home computer
And has seen his lifestyle turned upon its head;
For since starting on this caper,
I've not seen a daily paper,
Solved a crossword clue, or read a book in bed.

I no longer walk the spaniel –
No! I read my BASIC manual,
If my son can understand it so can I.
Though my program isn't working
It is not because I'm shirking
It's because thus far I haven't found out why.

I've the LET instruction mastered,
But ON/GOTO has me plastered,
IF and THEN I only fight with if I must;
I find integers confusing,
Nested GOSUBs quite bemusing,
While FOR/NEXT loops leave me totally non-plussed.

My son treats me with charity,
He says that perfect clarity
Is difficult for withered brains to f nd.
I'll ignore his tittle-tattle!
I will not forsake this battle!
For good, or ill, I've BASIC on my mind!

Batch Communications

Batch communications are used when it is appropriate to gather together a batch of data in one location for transfer to another location.

Take as an example a chain of shops where individual stores submit sales information and stock replenishment requests to a central computer.

It would be possible to send details of each individual sale as it occurs, but it may not be necessary. All sales information can be collected into a consolidated batch for transmission at the end of the day, minimising costs and simplifying the job of the central computer.

This may sound like putting your entire batch of eggs into one basket. It is.

What do you do if the batch communications go wrong? Go home. The **operations department** is paid to worry about things like that.

Batch Processing

Some computer processes handle things individually, others handle things in batches.

An enquiry on a bank customer's current balance would be handled on an individual basis as and when required, for example because the customer has just walked into the branch and asked for it. The production of bank statements for all the branch's accounts, on the other hand, would be handled by a **batch processing** program or 'suite' of programs.

Batch processing, then, is the handling of data in batches by a suitable **application program,** or suite of programs. 'Suitable' excludes all those programs which are written to handle a batch of data every day but which take 25 hours to run.

Couldn't happen? Huh.

Baud

The **baud** rate is a measure of the speed at which data is transferred down a communications line.

Baud is pronounced as in 'The fact that absolutely no data has arrived down this line for the last seven hours does not **bode** well', not as in 'I'm getting **bored** stupid waiting for some data to come down this darned line'. The word originates from the name of one J. Baudot, a French telegraphic engineer who bit the dust in 1903.

Strictly speaking the baud rate is the number of times that the signal can change in a second, so if each change represents an on or off bit the baud rate is the same as the number of bits which can be communicated per second.

In fact baud rate is normally taken as being the exact equivalent of 'bits per second', or 'bps'. This is wrong. A signal which can be in one of four states after a change can convey the value of a pair of bits at a time, and the number of bits sent per second is then twice the baud rate.