Course Handout - Short-Term Memory Subtypes in Computing and Artificial Intelligence
Part 4 - A Brief History of Computing Technology, 1951 to 1958

First published online 10:30 BST 17th July 2003. **This version [refresh hyperlinks after 20 years and remount as a .pdf] dated 10:00 GMT 4th March 2024**

This is the fourth part of a seven-part review of how successfully the psychological study of biological short-term memory (STM) has incorporated the full range of concepts and metaphors available to it from the computing industry. The seven parts are as follows:

Part 1: An optional introductory and reference resource on the history of computing technology to 1924. This introduced some of the vocabulary necessary for Parts 6 and 7. To go back to Part 1, click here.

Part 2: An optional introductory and reference resource on the history of computing technology from 1925 to 1942. This further introduced the vocabulary necessary for Parts 6 and 7. To go back to Part 2, click here.

Part 3: An optional introductory and reference resource on the history of computing technology from 1943 to 1950. This further introduced the vocabulary necessary for Parts 6 and 7. In so doing, it referred out to three large subfiles reviewing the history of codes and ciphers, and another subfile giving the detailed layout of a typical computer of 1950 vintage. To go back to Part 3, click here.

Part 4: An optional introductory and reference resource on the history of computing technology from 1951 to 1958. This material follows below and further introduces the vocabulary necessary for Parts 6 and 7. The main sections are:

- **1 - Computing 1951-1958 - First Generation Hardware (US)**
- **2 - Computing 1951-1958 - First Generation Hardware (UK)**
- **3 - First Generation Software**
- **4 - First Generation Artificial Intelligence**

Part 5: An optional introductory and reference resource on the history of computing technology from 1959 to date. This material will further introduce the vocabulary necessary for Parts 6 and 7. To go directly to Part 5, click here.

Part 6: A review of the memory subtypes used in computing. To go directly to Part 6, click here.

Part 7: A comparative review of the penetration (or lack thereof) of those memory subtypes into psychological theory. UNDER CONSTRUCTION.

1 - Computing 1951-1958 - First Generation Hardware (US)

After the top secret development effort of the 1940s [see Part 3], a number of civilian General Purpose Computers (GPCs) were delivered more or less simultaneously in the early 1950s, led by the Ferranti Mark I and the LEO I in February 1951, and followed closely by the US Census Bureau's UNIVAC in March 1951. The design principles were those of the 1950 Eckert-von Neumann machine, and the technology was basically chassis-mounted valve-based circuitry - hot, heavy, and unreliable. The stage was then set for a relentless expansion in commercial data processing capacity as large corporations started to install Eckert-von Neumann derivatives - known nowadays as **"First Generation" Computers** - alongside their existing data tabulating machinery. As the decade progressed, a number of other GPCs emerged from the established centres of excellence, and the final list includes the IBM 701, the Harvard Mark IV, the Bell Labs Models V and VI, the NPL's DEUCE, Cambridge's EDSAC II, Manchester's Marks II through V, and the LEOs II and III; each faster, more reliable, and offering more memory than its series predecessor. But because the centres of excellence were themselves still learning, some surprisingly fundamental innovations were still being made, and in due course these would usher in a newer generation of systems. The transistor was one such innovation, and, having invented the technology in the first place, Bell Labs were one of the first to experiment with transistorised digital circuitry, and their 1954 TRADIC contained 700 point-contact germanium transistors. Manchester University did likewise with two small research machines, which they had running in November 1953 and April 1955 respectively, and so, too, did the UK Atomic Energy Research Establishment, Harwell, with their 1955 CADET. Among the lesser innovations in this period was the development of magnetic tape storage .....

> **Key Innovation - Magnetic Tape:** The first coated tape method of audio recording was developed in 1927 by Fritz Pfleumer in Germany. The commercial development rights to Pfleumer's ideas were then acquired by AEG in 1932, and with assistance in plastic film technology from BASF a modern format reel-to-reel tape recorder was ready for demonstration at the Berlin Radio Fair in 1935. This breakthrough for the broadcasting and home entertainment industries was not lost on the emerging data processing industry, and magnetic tape readers and writers were included on the Eckert-Mauchly Computer Corporation's BINAC development in around 1949, and then on the 1951 UNIVAC, proving faster and more reliable than the earlier paper tape systems. << **AUTHOR'S NOTE: Given that the overall purpose of this seven part paper is to study the penetration of computing concepts into cognitive theory, it is worth noting at this juncture that magnetic tape storage has little explanatory or metaphoric value in this respect. The essence of a tape as a tape is that it is dismountable, which biological memory most certainly is not, and the essence of the data stored upon it is that it is stored "serially", that is to say, there is no overriding logical sequence to the individual items stored nor is there any quick method of retrieving a particular record from mid-file - you just have to start searching at the beginning and keep looking until you chance upon it, which takes time. Biological memory, on the other hand, remains permanently online, and seems to rely on two considerably more flexible storage techniques, namely (a) "sequential" storage, where the individual records either arise naturally in, or have been sorted into, some clear logical sequence (time-stamped order, say), and (b) "random access" storage, where either an "index" or a "hashing algorithm"** [Wikipedia briefing] **allows a target record to be retrieved from mid-file. We therefore see little mileage in the tape metaphor, although our final evaluation will not be available until 2004. >>**

1.1 The MIT Systems

Two of the very biggest innovations - ferrite core memory and the concept of real-time computing for command and control - came from MIT's Whirlwind team. Whirlwind's success for the US Navy in the 1940s, plus deteriorating relations with the then Soviet Union, prompted the USAF to commission an experimental air early-warning system from them. Whirlwind showed that such a complex linking of radar

and computing systems was feasible on 20th April 1951, when it simultaneously tracked three separate aircraft from radar and computed their interception parameters (Waldrop, 2001) [the hard part, apparently, lay in getting the interceptors round behind their approaching targets]. This demonstration resulted in massive government funding for air defence technology, and the result (although it was not fully operationally deployed until 1963) was SAGE (= Semi-Automatic Ground Environment), "a continent-spanning system of 23 Direction Centres, each housing up to 50 human operators, plus two redundant real-time computers capable of tracking up to 400 airplanes at once" (*ibid.*, p75). The machines themselves were massive Whirlwind clones, code-numbered AN/FSQ-7, and each contained more than 50,000 valves and ran "the largest computer program written up to that time".

> **BIOGRAPHICAL ASIDE - J.C.R. LICKLIDER: J.C.R. ("Lick") Licklider** (1915-1990) was a psychologist-in-computing who made his name on the SAGE project. He had previously been with the Harvard Psychoacoustics Laboratory, where he had worked on measuring and improving the intelligibility of the voice communication systems available to bomber crews. He was thus one of the founding fathers of the signal-to-noise-ratio approach to perception which burst upon the psychological scene in the 1950s. He joined MIT in 1950 to work on the human factors aspects of multiple mainframe computing, and soon became convinced that it was only a matter of time and good design before properly trained operators would be able more or less to hold a conversation with their CPU. This was the birth of the **"on-line transaction processing"** type of modern computing, as already profiled. In 1960, Licklider summarised these ideas in what is now seen as a classic paper, entitled "Man-Computer Symbiosis" (Licklider, 1960), in which he wrote "..... the resulting [man-computer] partnership will think as no brain has thought and process data in a way not approached by the information-handling machines we know today". In 1962, Licklider moved to the Pentagon's Advanced Research Projects Agency (ARPA), to continue his work on distributed command and control systems, where he put forward detailed proposals to raise the connectivity of the various systems then in use, so that they could exchange data more easily. He left ARPA in 1963, leaving these concepts to be fleshed out and made to work by his successor, Larry Roberts, under the name ARPANET [more on which in Part 5]. Licklider also contributed towards MIT's 1963 "Man and Computer" (or "MAC") project [more on which in Part 5], which eventually evolved into the world's first "online community" (Waldrop, 2001), and wrote a number of influential papers profiling the sort of knowledge interrogation software which Internet search engines have since made commonplace.

1.2 The IBM Systems

As we have already seen, IBM had been *the* data processing corporation half a century before binary electronics was invented. By 1945, its punched card business had corporate customers worldwide, with storerooms full of data and processing suites full of card sorters and tabulator-adders. Yet these punched card archives largely defied complex data analysis, because the tabulator-adders were basically little more than large desk calculators fitted with card readers - they were like fish out of water when given multiplications and divisions to carry out. IBM's strategy after the war was therefore to see to its customers first and to do pure R & D second. This, after all, had been the strategy which had made them number one in the first place, and their CEO, Thomas J. Watson, saw no reason why it should not keep them there. So they took the new-fangled digital circuitry and installed it in their tabulator-adders, thus turning them into tabulator-multipliers such as the IBM 603 (1946) and IBM 604 (1948).

As for the pure R & D, IBM had been there with its chequebook for the 1937-1943 Harvard Mark I development, and had pushed through the 1947 SSEC with no little urgency once Watson had decided that digital computing skills were too strategically important not to have in house. This was an inspired decision, and the SSEC experience immediately became the basis for the 1952 IBM 701 "Defence Calculator". This was a typical first generation computer, initially designed for the US defence market, and around 25 times faster than the SSEC it replaced. What made this machine historically significant is the fact that it also span off to become IBM's first top-of-the-range production machine for the civilian market, selling to the US Weather Bureau, aerospace companies, and the like. The project was directed by Nathaniel Rochester and Jerrier Haddad, and when the machine was formally announced on 29th April 1952 it came with enough CRT Main Memory for 2048 36-bit words, plus drum and tape backing store. Nineteen machines had been

sold by the time it was replaced by the IBM 704 in 1955. The 704 was essentially a 701 upgraded to take the new and far more cost-effective ferrite core memory, and 123 of these were sold by the time it was replaced in turn by the IBM 7090 in 1960.

The 701 was a large system, both physically and functionally, and with a price tag to match. However, thanks to the vision of one of its then junior engineers, Cuthbert C. Hurd (1911-1996), IBM management were persuaded to include a scaled down version - the IBM 650 - in their sales portfolio. This was a smaller mass-market machine, lacking the expensive CRT RAM, but deliberately intended to be compatible with their tabulating machine range. The 650s retailed at between $200,000 and $400,000 according to the precise configuration, and around 1800 of them were sold (Gray, 1999).

The IBM 650 was also honoured to be the first production machine to offer a random (or "direct") access disk memory option. To understand the true significance of this particular innovation, we must remind ourselves that our previous mentions of random access have related to main memory (where the words had even stamped the initials "RA" into the acronym RAM). When it came to backing store, however, the situation remained that files of any size would have to be stored as a succession of contiguous records - perhaps key-sorted, perhaps not - on magnetic tape; main memory could not (and, fifty years later, still cannot) hold more than a fraction of the total data available to the system. And the problem with magnetic tape, of course, was that as often as not several thousand feet of very delicate data and several minutes of waiting would be between you and the record you were after.

Now we have already seen how magnetic drum storage had been successfully implemented on the Harvard Mark 3 and the Manchester Mark I systems, and designers had already recognised that a spinning disk would have advantages over a rolling drum, if only the problems of size, cost, capacity, and reliability could be overcome .....

> **Key Innovation - Magnetic Disk:** Inspired by the shape and ease of manufacture of the gramophone record, the magnetic computer disk was a platter (or stack of platters) of non-ferrous material, thinly coated with a magnetic oxide emulsion and accessed by a read-write head (or heads) mounted on the radius line. When installed as a **"disk drive"** computer peripheral, this technology allowed data to be stored in concentric circular **"tracks"**, one beneath each position of the head(s). If you now took your giant tape file and wrote the records *n*-at-a-time onto as many tracks as it took, you could - providing you knew (or could in some way compute) the platter-track address - get hold of any one of them again in only a few tens of milliseconds, **even if it were situated right at the end of the file**.

The search for workable disk technology began in earnest at IBM in 1952, under Arthur J. Critchlow, and the story has recently been re-told by Rostky (1998). IBM announced the new technology on May 1955, and named it RAMAC. Each 50-platter drive stored what would today fit onto three standard floppy disks. It remains only to note that the direct access facility revolutionised both operating system design *and* application programming. To systems programmers direct access technology offered the sort of "paging" which would soon be used to deliver "virtual memory" systems [see Part 5; Section 1.2], and to applications programmers it offered not just disk-based files permanently online, but also "indexed sequential" organisation thereof capable of converting record key into a disk address. It also, in 1961, assisted at the birth of the modern database [see separate story].

1.3 The Remington/Sperry Systems

IBM did not have things entirely their own way, however, for they were facing increasing commercial competition in the early 1950s from the Remington Corporation. In February 1950 and December 1951, respectively, Remington acquired the controlling interests in the Eckert-Mauchly Computer Corporation (EMCC), and EMCC, it will be recalled, was the company which the Johns Eckert and Mauchly had set up

in 1946-7 to build BINAC and UNIVAC, and ERA was the company set up in 1946 by Howard T. Engstrom (1902-1962) and William C. Norris to build the US Navy's 1947 "Task 13" machine (the machine which evolved into the US National Security Agency's October 1950 Atlas 1101 computer). And despite the fact that Remington's own computing department, the EMCC people, and the ERA people were geographically separate units, and retained a "high degree of autonomy, indeed rivalry" (Gray, 2002), Remington had a minor star in EMCC's UNIVAC 1, which they duly capitalised upon by rebranding a demilitarised variant of the Atlas 1101 as the UNIVAC 1101, complete with a parallel organised arithmetic unit. They also won a second National Security Agency contract for the Atlas II (1950-1953), an Air Force contract for the (three-off) UNIVAC 1102 (1952-1956), and got government permission to market a civilian version of the Atlas II, to be called UNIVAC 1103 (Gray, 2002).

This was all cutting edge stuff, of course, and it was projects like these which attracted the likes of Seymour Cray (1925-1996) into computing. Cray joined ERA in 1951, and impressed management so much that in 1953 they appointed him Project Engineer for the 1103 (Breckenridge, 1996). He rewarded their faith in him by turning the 1103A into one of the most innovative machines of its day. One development in particular - a "program interrupt" facility - was developed on an 1103A at the Lewis Research Centre, Cleveland (Gray, 2002), an outstation of the US National Advisory Committee on Aeronautics (NACA).

> **Key Innovation - "Program Interrupts" (First Mention):** According to Gray (2002), the 1103A interrupt facility "permitted the 1103A to interrupt the processing of one program to handle another". This might be useful, for example, if an urgent job arose part-way through the execution of a much longer one, but where too much processing had already been done just to abandon it and start again later. To do this switchover safely, requires taking careful note of the position of the executing program, and then being able to reinstate all the registers and Main Memory settings to their pre-interrupt state afterwards, and this in turn requires sophisticated systems software and additional memory resources. We return to this topic in Part 5. << **AUTHOR'S NOTE: The idea of one program interrupting another is quite popular in modern theories of high level motor control, especially those of the Norman-Shallice type. For specific examples, click here or here. The role of interrupts, however, is nevertheless poorly dealt with. Our full evaluation will not be available until 2004.** >>

Further enhancements to the 1103A followed in the form of the 1958 1103AF, the first production machine to provide a reliable floating point arithmetic facility (Gray, 2002) .....

> **Basic Terminology - Decimal Fraction:** A decimal fraction is the sort of number we see all around us every day, that is to say, numbers such as 127.30 or 3.142. Each such number consists of two theoretically infinite strings of digits, either side of a **"decimal point"**. The digits to the left of the decimal point are known as the **"integer"** part of the fraction, and the digits to the right are known as the **"decimal"**. Taken together, these three elements are sometimes referred to as a **"mantissa"**.

> **Key Innovation - Floating Point Arithmetic:** Ever since the days of Schickard and Pascal, calculating machinery manufacturers have had to trade off complexity and cost against accuracy. If you decided to offer your customers ten-digit accuracy, for example, then someone wanting to multiply two such numbers together would suddenly need to display up to 20 digits, yet it would be wasteful to provide a register that big if it were only going to be used once in a blue moon. This is why Babbage's 1822 Difference Engine went for 31 digits and turned out to be an engineering nightmare, whilst the 1642 Pascaline made do with eight and was well received. An added problem was that the number to be displayed might be all integer, all decimal, or part and part, so calculator designers also had to keep track of where the decimal point should go, which they did by "fixing" it, and by leaving it to their customers to do **"fixed point arithmetic"**. Things were no less troublesome for the computer designers of the 1940s, especially those working in binary (because base 2 numbers are roughly four times as long as their base 10 counterparts), and the eventual solution was to convert all numbers into what is called **"scientific notation"**, a numbering convention where only one significant digit is allowed in the integer part of the mantissa, and in which the true scale of the resulting number is then obtained by multiplying it by an associated power term. Each power term consists of a **"base"** - typically 2 in binary or 10 in base 10 - and an **"exponent"**, the power to which the base must be raised. Thus .....

0.147 is the same thing as 1.47 multiplied by 0.1, and can be expressed as **1.47×10^{-1}**
147 is the same thing as 1.47 multiplied by 100, and can be expressed as **1.47×10^{2}**
14700 is the same thing as 1.47 multiplied by 10000, and can be expressed as **1.47×10^{4}**

..... and so on. The decimal point, in other words, is now free to **"float"** from wherever it started off until it arrives at its standardised new position, leaving the power term to remember where it had been. The **"precision"** of a given system is the number of significant digits catered for in the mantissa. Thus the number 14741 in a precision-3 system (as shown above) loses the fourth and fifth significant digits and will be treated as 14700, and this means that we risk introducing **"rounding error"** into our arithmetic if we throw lots of significant digits at a machine which has not been set up with the precision necessary to handle that number of digits safely. However, the advantage of scientific notation is that it greatly simplifies the task of computerising the multiplication process, because your two multiplicands will now always lie between 1 and 9 (and can therefore easily be fixed point multiplied), and the power terms can be multiplied by adding the two exponents. Thus a precision-2 multiplication of 0.0201 by 35192 will proceed as follows .....

(1) Convert each multiplicand to scientific notation, thus .....

$(2.0 \times 10^{-2}) \times (3.5 \times 10^{4})$

(2) Do the multiplication as described above, giving .....

$7.0 \times 10^{-2} \times 10^{4}$

(3) Calculate the new exponent as described above, giving .....

7.0×10^{2}

(4) Convert the answer back into everyday form, giving .....

700

As it happens, the arithmetically correct answer is 707.3592, so we are about 1% out, the difference arising from the aforementioned rounding error. It would therefore be wise to increase our precision, and a precision-4 multiplication of the same two numbers duly gives a better approximation, namely 707.3 [you actually need a precision-7 multiplication to remove the error altogether]. Unfortunately, rounding error is endemic in floating point arithmetic, so for scientific applications the precision is nowadays set high at an IEEE-recommended 32 bits for **"single precision"** working, or 64 bits for **"double precision"** working; and even then things can go wrong, as when an unexpected rounding error of this sort contributed to a failure in a Patriot missile launch during the First Gulf War, resulting in a Scud strike on a US barracks, killing 28 servicemen. As Goldberg (1991) puts it, "squeezing infinitely many real numbers into a finite number of bits requires an approximate representation", thus presenting the computing industry with the intriguing paradox that the faster you can do your calculations the more you need to worry whether one of these inherent inaccuracies has taken place. << **AUTHOR'S NOTE: The floating point concept - perhaps surprisingly - is not totally incompatible (*mutatis mutandis*) with what is known about modular mathematical cognition. As we have explained elsewhere (Smith, 1998), mathematical cognition consists of at least two basic "left hemisphere" processes and at least one "right hemisphere" process, all of which seem capable of simultaneously working on a given mathematical task. The left hemisphere processes are (a) to establish the conceptual meaning of the numerals and arithmetical symbols used, and (b) to activate and put into effect the rule sequence by which the solution might be derived. The right hemisphere process is to predict the result of the overall calculation by roughly estimating what the answer should be. In addition, the ability to use the "number line" may provide both conceptual (left) and visuo-spatial (right) support.**

In 1955, Remington merged with Sperry Gyroscopes to form the Sperry-Rand Corporation, and in 1956 the 1103 was supplemented by the markedly smaller UNIVAC "File Computer". This machine got its name from another important innovation, namely the long-term storage of large commercial data files on magnetic drum, rather than on punched card - thus mounting a direct assault on IBM's hitherto captive market. Not only could large bodies of commercial data now be rolled forward on a daily, weekly, monthly, or yearly

basis, but it would henceforth no longer be necessary to trolley the files in and out of the machine each time they were needed.

> **Key Innovation - Periodic Batch Processing:** Accountants and bankers have had to "carry forward" balances from the end of one accounting period to the beginning of the next ever since money was first invented, and the basic principles of book-keeping within such an accounting period are laid out in Luca Pacioli's "Summa de Arithmetica" (1493). [William Goetzmann, of the Yale School of Management, places the first recognisable bank in Uruk in Mesopotamia (modern Iraq) around 5,000 years ago] The calculations, however, were done by hand until the advent of mechanical calculators towards the end of the 19th century, and - if you were lucky enough to be able to afford it - were then semi-automated by the Hollerith style punched card industry of the early 20th century. Now the point about punched card processing is that data flowed from storage point to storage point via machine after machine and through tray after tray, being merged, sorted, updated, copied, collated, resorted, etc. Finance Departments suddenly had to be totally precise (a) in how they allocated their records - the cards - into files, and (b) how the files then metamorphosed as they passed through the various processing stages. Fortunately, although the nature of the data varied widely according to the nature of the business involved, there were some clear basic patterns. By far the greatest volume of data was **"transaction data"**, that is to say, individual records of individual happenings during the period in question - individual sales, individual deliveries, individual money transfers, etc. However, this type of data only really meant anything when seen in the context (a) of the situation at the end of the preceding period (as set out in the opening balances contained in some sort of permanently maintained **"master file"**), and (b) of supporting detail such as rate books (**"reference data"**) and customer details (as set out in some sort of **"customer file"**). Thus the sequence of events in a typical 1920s stock system might be to open up a master file of opening stock levels (containing one record card per item per location), and then to "tally forward" each stock level by the receipts, issues, and adjustments of that item contained in one or more **"transaction files"**. At period end, this tally forward would generate the theoretical carried forward stock levels, which could then be audited against a physical stockcheck, if necessary. The experience of the Lyons Company's office systems guru, John Simmons, is probably indicative. He had been appointed in 1923, remember, in order to bring scientific rigour into the design and operation of Lyons' corporate processes [to streamline, first and foremost; to automate if that helped streamline; and to computerise if that helped automate], and their 1949 LEO project [reminder] succeeded in automating the batch processing routine for payroll applications in December 1953 [more on this in Section 2.4].
> **<< AUTHOR'S NOTE: The standard tool for the specification of these large batch processes was the dataflow diagram (DFD). DFDs are now the diagram of choice for tracking the movement of information (a) through an extended system, and (b) through the processing stages carried out at any one node within that system. As for the use of flow diagrams to help explain cognition, they are already standard practice (albeit they are not always very well done technically). We have written at length on this subject elsewhere, and the subject can safely be approached from any of three directions, for the material constantly cross-refers. To begin with a specific example of a modular cognitive system, click here, or to begin with an introductory commentary, click here, or to begin with some back-to-basics practical diagramming, click here. >>**

Lyons' and Sperry's pioneering work led to a growing confidence that large corporate files could be safely committed to a medium which was inherently unreadable by humans, and this was commercially highly significant because it enabled the new electronic systems to threaten the data tabulating industry in its own back yard; to offer an upgrade pathway for the literally thousands of tons of legacy data still held on punched cards. The design spotlight therefore fell on such factors as storage capacity, processing speed, and system reliability, and it soon became standard design practice to assess whether the file handling and the number crunching aspects of a given process presented broadly balanced demands. If they did, then all was well, but if either was slow compared to the other, then you had to suspect a bottleneck of some sort .....

> **Key Innovation - I/O vs CPU Bottlenecks:** New terminology soon emerged to describe the various problem scenarios. If a process struggled when reading data in or when writing data out it was referred to as **"peripheral bound"**, but if it struggled during the number-crunching phase it was referred to as **"processor bound"**. When the present author joined British Telecom in 1980, he learned of "the Great ISOCC Tea Trolley Disaster" [English humour]. ISOCC was the BT computer system then responsible for charging for operator-connected calls. For a time, however, "telephone tickets" [picture] were being produced by the nation's telephone operators faster than the billing centres could process them; specifically, it took about 25 hours to process every day's input slips, so the processing backlog simply got longer by the day. We can no longer recall with certainty whether the problem lay in getting

the data punched into the machine (in which case the process would have been peripheral bound), or in carrying out the necessary calculations once it had been (in which case it would have been processor bound), but it was probably the former. << **AUTHOR'S NOTE: Neuroscience does not know enough about biological processing architectures to judge whether the unit of binary decision making - that is to say, the biological equivalent of a logic gate - is a single neuron, a small neuron cluster, a large neuron cluster, an entire gyrus or ganglion, or an entire cortical layer. It is conceivable, indeed, that structures traditionally ignored by neuropsychologists, specifically the brain's "white" as opposed to "grey" matter, or even the glial supporting cells, may also be able to act in some way to modulate what is going on in their electrically noisier neighbours. We raise this as an issue, because until neuroscience can identify the logic gates it will remain unable reliably to identify the larger logic or control units either, so it is difficult to agree on which brain areas count as peripherals and which as central systems, and therefore which is getting overloaded. We extend the argument when discussing serial and parallel computing in Section 2.2.** >>

Hollingdale and Tootill (1965) explain the problem, and its conventional solution, this way .....

> "Before 1958 most computers were engaged on scientific or technical calculations of one kind or another; nowadays most of them are employed on clerical tasks - on payroll or stock control calculations, for example. In this kind of work the essential role of the computer is to manipulate large quantities of digitally coded material - usually referred to as *data* - in a prescribed manner [.....] The amount of computation that has to be done on each unit of data is usually quite small [..... //] The organisation of the rapid input and output of streams of coded data is, in fact, one of the major problems of [commercial computing]. [.....] The basic idea is to carry out all the slower input and output processes away from the main computer, whose precious time must not be squandered on sluggish menial tasks. These are delegated to a subsidiary computer, the essential function of which is to transcribe information from one physical medium to another ....." (p239-240.)

The upshot of all these problems was that Lyons and Sperry perfected some historically important methods of coping more effectively with peripheral bound processes by strategically placing additional memory resources about the system .....

> **Key Innovation - "Buffers" and "Buffering":** **"Buffers"** in general are simply areas of temporary storage somewhere in size between a register and a decent slice of Main Memory, and they are important because they allow **"buffering"**, that is to say, "the temporary holding of information in transit" (Purser, 1987, p312). Like any other form of digital storage, a buffer consists ultimately of a contiguous series of flip-flops, and the underlying technology may vary from machine to machine. To understand how buffering works, we need to remind ourselves (a) that Eckert-von Neumann GPCs [reminder] habitually process their input and output files through reserved record areas in Main Memory, (b) that file reading and writing are comparatively slow processes, and (c) that Lyons' and Sperry's customers were trying to funnel enormous volumes of data through card or magnetic tape readers. **Bit String Buffering:** The Lyons and Sperry designers therefore added what was effectively a small free-standing register between the relatively fast data bus and the relatively slow peripheral device controllers. As data arrived in the form of a slow stream of bits from a peripheral device, it was loaded into an **"input buffer"**, and only when that buffer was full was the accumulated stream released - **either at a higher serial rate, or in parallel** - onto the bus to complete its journey to its destination in CPU or Main Memory. **This short delay left the CPU free to be getting on with something else**. A similarly arranged **"output buffer"** could buffer data *en route* from the output record area to an output device. Pinkerton (1990) profiles the buffering system on the LEO as follows: "The buffers in each of the input channels were all valve circuits and mercury delay lines. They had little short tubes which assembled individual numbers and put them into position, one after the other until they filled it up. The same thing on the output side; some of the buffers were doubled on the input though not on the output. The logic allowed four input channels and four output channels ....." **Main Memory Buffering:** It is also possible to use Main Memory for buffering. This can be done by doubling (etc.) the number of Main Memory record blocks allocated to each file, and by then routinely loading these **one record ahead of the ongoing processing**. This sort of **"double buffering"** introduced a degree of record-level overlap into the system, such that the input delay for record <n+1> coincided with the processing time for record <n>, which, in turn, coincided with the output delay for record <n-1>. For very fast processes, "treble buffering" (etc.) could be resorted to, thus creating even more overlap. It follows that a given input character was likely to have been buffered twice on its way to the CPU, once at the device stage, and then again at the Main Memory stage. **Instruction Buffering:** We deal with another type of buffer - the **"instruction buffer"** - in Section 2.3, under the heading *Pipelining*. **"Store and Forward" Buffering:** This is a form of buffering which occurs (in addition to all the others) at the subordinate nodes in a

communications network. It is needed whenever three or more computers have been interconnected, and it allows transmissions to be stored temporarily at node #2 en route for node #3, should the line to node #3 be down or otherwise engaged, or should the CPU at node #3 be too busy to accept the input immediately. Modern telecommunications - including the technology currently bringing you this page - devotes much of its attention to such problems, and network designers need to carry out a complex traffic queuing analysis before deciding how much buffering store to provide at each intermediate node. << **AUTHOR'S NOTE: The concept of output buffering is heavily used in modern theories of motor behaviour - for specific examples, click here, or here, or here. The concept is less well handled in theories of perception, although the phenomena (a) of perceptual memory [see Sperling (1960)] and (b) of saccade-based text processing [see Gough (1972)] are both prime candidates for it. It is also poorly handled in matters of human communications, where the "store and forward" version of buffering is not catered for in any of the common forms of the "sociogram". This, as we have repeatedly argued elsewhere [e.g., Smith (1997; Chapter 1)], seriously erodes the explanatory value of most modern cognitive models [for our most recent offering on this topic, see our e-paper on "peer-to-peer" telecommunication].** >>

2 - Computing 1951-1958 - First Generation Hardware (UK)

The 1945 demobilisation took all but a select few of the British wartime engineers from their top secret work and returned them - tightly sworn to silence by the Official Secrets Act - to peacetime employment. We begin this phase of our story with one of these demobilised boffins, Andrew D. Booth, one of the pioneers of magnetic drum memory .....

2.1 The Birkbeck and "British Hollerith" Machines

Andrew D. Booth [Wikipedia biography] was a Birkbeck College physicist who, as an employee of the British Rubber Producers Research Association, had been involved in X-ray crystallography during the war .....

> **ASIDE:** The British Rubber Producers Research Association (BRPRA) was founded in 1938 to promote formal scientific knowledge of said product. It set up research centres in both Malaya and Welwyn Garden City, Hertfordshire, and one of its first researchers was Cardiff-born George Alan Jeffrey (1915-2000), later Professor of Crystallography at the University of Pittsburgh. Now the point about rubber is that it is a highly strategic commodity, perfectly capable of winning or losing wars. Without it, for example, you have no tyres for your vehicles or aeroplanes, no engine mountings, no oil seals, no hoses, no rubber dinghies, etc. Strategically, therefore, you either have to own the hot places in the world where the product can be grown naturally, or you have to have the raw materials and industrial base to manufacture it synthetically. Either way, you need a first class team of physicists and chemists trying to make the stuff longer lasting and constantly analysing specimens cut from wrecked enemy equipment to see if they had managed to do the job better than you had (much of Jeffrey's war was spent X-raying slices of rubber cut from wrecked German aircraft). It was this strategic war work which the BRPRA was there to organise, and, humble though the job might sound, the rubber boffins probably contributed every bit as much per capita to the war effort as did the codebreakers at Bletchley Park. It was Jeffrey who recruited Booth, and it was the fact that X-ray crystallography creates raw data like there is no tomorrow that led Booth to develop automatic calculating machinery as a laboratory aid to data analysis in much the same way that Wynn-Williams had scratched together his thyratron circuits at Cambridge in the early 1930s [reminder].

Booth moved to Birkbeck College, University of London, after the war, and in 1946 his automatic calculating work brought him to the attention of the US National Defence Research Committee's chief "talent scout", Warren Weaver [bio follows in Section 4.1]. Weaver was intrigued by what he saw, and put Booth forward for a study scholarship under John von Neumann at Princeton's Institute of Advanced Studies [a cynic might ask in retrospect who was studying whom]. Accompanied by his research assistant Katherine H.V. Britten (whom he subsequently married), Booth was at Princeton from March to September 1947, and upon his return to Britain proceeded to build a small relay computer named the Automatic Relay Computer (ARC) (Lavington 1980). He demonstrated this machine to Weaver on 23rd May 1948 at BRPRA's Welwyn Garden City centre (Hutchins, 1986), and we shall be finding out what Weaver did next in Section 4.1. But what made the ARC more than just a British version of, say, the Bell Mark II relay computer [reminder] was the fact that Booth had cleverly linked it to a simple magnetic drum memory .....

Key Innovation - Magnetic Drum Storage: The advantage of magnetic drum storage over tape lies in the fact that the storage surface rotates continuously against a linear array of read-write heads. This means that your data is re-available to you once per revolution, unlike tape-borne data, where there is a lengthy re-wind time between passes. The sequence of magnetically coded pulses recorded by any one head is known as a **"track"**, and each stored bit offers itself back to the head for reading and/or overwriting every few milliseconds [this rotational delay being known as its **"latency"**]. The system has quite a history, in fact, for the idea of "rolling up" your data storage in order to save space is as old as the scrolls of ancient times, and the idea of winding data around a cylinder, so that it spirals gradually from one end to the other, is clearly seen in both the Greek cryptographical system known as the *scytale* and Edison's famous cylinder phonograph. It is therefore not surprising that when, in 1898, the Danish inventor Valdemar Poulsen was looking for somewhere to keep his magnetised wire recordings, he chose a drum; nor that the computer pioneer John Atanasoff toyed with the drum concept in the late 1930s on the Atanasoff-Berry Computer [reminder], albeit he ended up using surface capacitors as his storage substrate rather than a magnetic coating (Gray, 1999). What is not so clear is who first applied Pfleumer's ferrite emulsion as a magnetic lacquer coating to a drum, and the first clear suggestion to this effect seems to have been in an MIT master's thesis by one Perry Crawford.

> **BIOGRAPHICAL ASIDE [INCOMPLETE] - PERRY O. CRAWFORD:** According to Jay Forrester, the Whirlwind Project Director, it was Crawford who first called his attention to the possibility of digital computation (Forrester, 1988), this having been the subject of his MIT master's thesis during the period 1939? to 1942?. Mindell (2002) mentions Crawford as having succeeding Claude Shannon as postgraduate student at MIT, sometime around 1940, and it was in the period 1940 to 1942 that Crawford developed his ideas on the use of digital methods to fire control problems. During 1942 to 1945 Crawford served (as an attached civilian) with the navy's Special Devices Centre, later at Sands Point, Long Island, building flight simulators and training devices and coming up with the name "Whirlwind" into the bargain. In October 1945 he went to work for the Special Services Division of the Office of Naval Research. Crawford left the navy in 1952, and moved to IBM, where he became involved in the SABRE project [see Part 5]. He was still around to be interviewed in 1995 (by Mindell), but seems to have published little.

Booth experimented with the drum concept in his Birkbeck College laboratory in 1946-1947, and passed on the details to the Manchester University team who added it to their Baby [reminder] (Booth, 1995). On balance, however, it is probably fairer to credit the first large scale investment in drum technology neither to MIT nor Birkbeck, but to ERA's John Coombs, as part of the US Navy's Project Goldberg, also in around 1946. ERA used a 5" drum revolving at 3000 rpm, and set their packing density at 230 bits per inch. The literature also mentions an 8" drum developed in 1948 by the University of California at Berkeley (Spice, 2001) and recording at 800 bits per inch, and other units on the Harvard Mark III and the SEAC. << **AUTHOR'S NOTE: The notion of a constantly recycling memory is superficially similar (a) to the psychological concept of the "rehearsal loop", and (b) to the physiological concept of the self-sustaining neural "reverberatory circuit". Rehearsal has been suggested by many authors as a mechanism whereby a small number of auditory items can be maintained in memory by constant covert repetition. Our full evaluation of the literature here will not be available until 2004, but to see rehearsal loops in a cognitive model click here.** >>

After making his name with the ARC, Booth moved up to thermionic valves, trying out his circuitry in a 1949 prototype machine called the Simple Electronic Computer (SEC). When he was happy that he knew what he was doing, he followed this up with a number of larger, but still drum-equipped, machines in his 415-valved APE*C Series, where the APEC stands for "All Purpose Electronic Computer" and the asterisk indicates the particular commercial sponsor at the time. Thus the APERC] was delivered to the British Rayon Research Association, the APEHC was delivered to the British Tabulating Machine Company ("H" because BTM were the holders of the Hollerith licence in Britain), the APENC was delivered to a Norwegian sponsor, and the APEXC was his own Birkbeck machine. The APEHC was BTM's first excursion into electronic computation, and they quickly reworked it in house as the HEC. They then - goaded no doubt by IBM's success with its electronic multipliers [see Section 1.2] - used the HEC experience to build punched card processors of their own, such as the BTM 541 (1953) and BTM 555 (1957). The HEC was also the inspiration for the BTM 1200 (1954) and BTM 1201 (1957) computers, and thus indirectly for the ICT 1300-series and ICL 1900-series mainframes which followed them.

2.2 The Last of the Boffins

So what of the boffins who had been kept on after the war in the secretive government and military research stations we heard so much about in Part 3. Well basically we spoke of four such sites - Bletchley Park, the National Physical Laboratory (NPL), Dollis Hill, and the Research Establishment at Malvern, Worcestershire, so there are only four stories to tell, and we shall take them in order of mention.

After the war, the GCCS at Bletchley Park continued to develop its Colossus machines, sold an unspecified number to friendly foreign governments [which is one of the reasons their existence was not acknowledged for such a long time - Ed.], and remained active in the security business for many years. In the end, however, the unit was relocated and the site itself has recently become a museum. But GCCS made no further computers - or, if it did, they are still classified.

We left the NPL in 1949, struggling to get their cool but decidedly complicated Pilot ACE up and running [reminder]. In 1948, Turing had been transferred to Max Newman's laboratory at Manchester, and the Pilot ACE continued in development for two more years under James H. Wilkinson and E.A. ("Ted") Newman. It executed its first program on 10th May 1950 (making the NPL, they claim, the fourth group in the world to deliver a stored program electronic digital computer), and was publicly demonstrated in December that same year. The NPL then transferred the production rights in Pilot ACE to the English Electric Company, the then owners of the Marconi interests, who tidied it up a bit before putting it to market as the **Digital Electronic Universal Computing Engine (DEUCE)**, eventually delivering more than 30 machines at around £50,000 apiece. The NPL's **(Full) ACE** was ready in 1958, still with delay-line storage, whereupon it was decided that there was no longer any role for government owned mainframe development. Full ACE thus marked the end of the NPL's involvement in mainstream computer development.

> **ASIDE:** Rumour - probably not totally without foundation - has it that the Royal Aircraft Establishment actually bought two DEUCEs, so that they could run their really important programs on both of them, only trusting the results if both machines came up with the same answer (Kershaw, 1994)! For more on Pilot ACE, see Newman (1994), and for DEUCE, see Fisher (2003).

As for the Post Office boffins at Dollis Hill - that is to say, Tommy Flowers, A.W.M. Coombs, and their respective teams [reminder] - their fate is easy to relate because as telecommunications engineers in the first place they simply reverted to their former existences. Nevertheless there were occasional opportunities for off-site adventure, as when Coombs was called in to assist at .....

..... the War Department's own high-tech installation at Malvern, home of two of Britain's early defence computers. The first of these was a derivative of Pilot ACE called MOSAIC, sponsored by the Ministry of Supply for missile tracking applications. This was a 6000-valve, 2000-transistor, serial processing machine built between 1947 and 1954 by the Post Office's A.W.M. Coombs and team (Lavington, 1980), and its Main Memory comprised 1024 40-bit words in mercury delay lines ("nearly a ton" of the stuff, according to Lavington). The second British defence computer was a more technically adventurous parallel processing machine called TREAC. This was designed in 1947 by Albert M. Uttley, went operational in 1953, and used a 512-word Williams-Kilburn system for fast store. The Australian Computer Museum also remind us of the work of Trevor Pearcey, a TRE veteran who in 1946 emigrated to Australia, where he designed the 1949 Australian CSIRAC computer, since claimed as the fifth stored program computer to be completed.

> **Key Innovation - Serial vs Parallel Processing:** To do things "in series" is to do them one after another. To do things "in parallel" is to do them simultaneously. Serial computers therefore execute their data moves and their arithmetical operations one after another, whilst parallel computers do them simultaneously. As originally conceived, parallel processing computers used one wire per bit when moving whole words of data at a time between their registers and their logic gates, whilst serial computers used one wire only, and moved one bit at a time. The

parallel machines were accordingly much faster, but much more electrically complicated. EDSAC's Maurice Wilkes explained the differences this way .....

> "The most obvious division of computing machines is between those which are serial in operation and those which are parallel [but] the subject is an endless one in that many different variants of each kind of machine are possible. I do not consider that the question of whether it is better to build a serial or a parallel machine will ever be finally answered; each type has its advantages, and the final decision will always be dependent on the designer's personal preferences, and to some extent on his terms of reference. Also, as time goes on, the balance of advantage and disadvantage will swing as improvements are made in this or that component, or as entirely new components become available. Certain fundamental comparisons between serial and parallel machines can, however, be drawn readily enough. [//] The arithmetic unit of a parallel machine tends to be much larger than the arithmetic unit of a serial machine, since a separate flip-flop or equivalent circuit element must be provided to accommodate each digit in the various registers. Moreover, since addition of the digits takes place simultaneously, or nearly so, a separate adder must be provided for each digit. The disparity [.....] increases as the fundamental number length is increased. [...../] The control, on the other hand, is much simpler in a parallel machine. In the first place the necessity for the generation of a set of digit pulses does not arise, and secondly, the timing is much simpler, since a number can be moved [.....], or an addition performed, by the application of a single pulse to a set of gates ....." (Wilkes, 1956, p66)

What we are looking at, therefore, are alternative ways of organising a single CPU - serial for cheap but slow, or parallel for complicated and expensive but fast; it is just another design decision to be made in the early stages of a machine development project. << **AUTHOR'S NOTE: The parallel processing concept used by TREAC and the US parallel machines [Whirlwind and the 1101] is NOT THE SAME as the "parallel distributed processing" talked about by the PDP school of cognitive scientists [see our e-paper on Connectionism]. A parallel wired CPU moves whole data words at a time WITHIN A SINGLE CPU, but PDP as an incarnation of modular cognition REQUIRES MORE THAN ONE CPU. Or to put it another way, the essence of "parallel" in a parallel CPU is that many bits are moved simultaneously under the control of a single Control Unit, whilst its essence in a PDP configuration is that more than one Control Unit is at work. Now when we look at the brain, there is certainly no doubt that data moves simultaneously within even the smallest aggregate of neurons, but as we do not yet know the unit of binary decision making it would be unsafe to classify this as parallel logic. There is also certainly no doubt that different brain areas are simultaneously active during different types of cognitive task, but until you know which modules count as Control Units [and that is a philosophical problem as much as an engineering one] you cannot be sure what these resources are each contributing, or, indeed, to what extent they are in fact co-operating. Moreover, because each component CPU can be serial or parallel within itself, you can theoretically have PDP in a network of simultaneously active serial CPUs, albeit this would be slower than PDP in a network of simultaneously active parallel machines. You could even have PDP in a part-serial, part parallel, processing network. Our full evaluation of the literature pertaining to this issue will not be available until 2004, but there is much to learn in the meantime from the literature on brain injury. See, for example, our timeline notes on the nineteenth century neurologists Dax and Lordat, or the debate between the localisers and the globalists [timeline]. Note especially that both the hierarchical block diagram of Kussmaul (1878) and the transcoding block diagram of Charcot (1883) both derive from this type of evidence, and therefore so, too, do modern derivatives thereof, namely Norman (1990) and Ellis and Young (1988) respectively. >>**

2.3 The NRDC Machines

Another cluster of first generation computers evolved out of the tripartite collaboration of Manchester University, the Ferranti Company, and Elliott Brothers, and the coordinating force here was the Government's scientific funding office, the National Research and Development Corporation (NRDC) [not to be confused with the American NDRC mentioned in Section 2.1 above and seen again in Section 4.1]. This organisation was set up in 1949 by the then Trade Minister Harold Wilson (later Prime Minister between 1964 and 1970, and 1974 to 1976), under the direction of John Anthony Hardinge Giffard (1908-2000), Third Earl of Halsbury, in an entirely laudable attempt to stretch the British taxpayer's return on state-funded research. This meant co-ordinating the work of the remaining defence establishments with interested parties throughout industry and academia, and the newly born computing industry was high on the NRDC's list of strategic priorities.

ASIDE: In a blistering appraisal of the whole NRDC experience, Hendry (1989) explains how most of the companies the government helped had gone out of business by the mid-1960s [and the survivors - as we shall be seeing in Part 5 - have either gone since, or are pretty sick]. For a number of reasons, some screamingly obvious but others quite subtle, the NRDC tended to "help to no advantage". It transpired that neither government gifts, nor advantageous development loans (nor, indeed, roving facilitators) were particularly effective instruments of public policy. "At no time," Hendry concludes, "did the aid given or offered to any firm make any perceptible difference to that firm's business strategy" (p168). In the ten years to 1959, the NRDC distributed just short of £1 million in cash, and underwrote loans for a further £1.25 million. This compares with US military patronage for the US computer industry during the same period of "over $1 billion" (p161). For his part, Lord Halsbury described the whole experience as rather like "pushing mules uphill" (Halsbury, 1991, p272), and had little positive to say about the quality of the decision making in the companies he had been trying to assist. The Ferranti company was founded by Sebastian Ziani de Ferranti in 1889 in London, to manufacture electrical generating equipment. During the First World War they produced munitions, and in the 1920s moved into radio and consumer electricals. The radio experience then stood them in good stead come the 1930s, when the RAF needed re-equipping, and again when the radar contracts started to come through shortly afterwards.

So let us start with the collaboration between Manchester University - whose Mark 1 we have already spoken about in detail [reminder] - and the local Ferranti works at Moston (later West Gorton), Manchester. The strategy here had already been hammered out at an NRDC meeting on 14th December 1949, and involved Ferranti preparing the Manchester Mark I for market as the Ferranti Mark I, thus freeing up the academics to push ahead with further technical developments. Manchester University therefore began the decade by getting on with the Manchester Mark II, alias "MEG". This incorporated a number of relatively minor developments, but when it first operated in May 1954 was nevertheless 20 times faster than the Mark I, thanks primarily to having changed from serial to parallel logic (Lavington, 1978). This, too, was eventually marketed by Ferranti, as the "Mercury", and sold fairly successfully in competition with the IBM 704, because despite being a lot slower it was also a lot cheaper (*ibid*.). The university then moved on to the Manchester Mark III, an experimental transistorised machine, which contributed greatly to the 1956 (six-off) Metropolitan Vickers MV950, which has been claimed as the first operational transistorised computer. [We shall be dealing with Manchester's later machines, the second generation MUSE and Atlas, in Part 5].

Other work, meanwhile, had been going on in London and the Home Counties, the industrial player here being Elliott Brothers .....

HISTORICAL ASIDE**:** Elliott Brothers were founded by William Elliott in 1795 to supply Royal Navy officers with precision technical instruments, and had then simply grown with the technology to provide many of the Royal Navy's fire control systems during the analog computing era [see Part 2], not to mention supplying some of the parts for Babbage's original Difference Engine (Clarke, 1975). After World War Two, they were keen not to lose this marketing foothold, and in 1946 established a digital computing research unit of their own at Borehamwood, Hertfordshire. They selected John F. Coales, who had just spent the entire war on Elliott's gunnery radars, to run this unit, and their first major project was to develop a state-of-the-art computerised naval anti-aircraft fire control system, codenamed MRS. The broad specification here was for "ship-borne radar [Elliott's specialism, remember] [to] lock on to a target at about eight miles and then track it and supply aiming information to a ship's guns" (Lavington, 2000, pp8-9). A large number of R & D staff were duly taken on, and the computer which resulted was the Elliott 152, with logic circuits designed by Charles E. Owen (details in Clarke, 1975). The 152 "executed several instruction streams in parallel from an electrostatic read-only memory" (Lavington, 2000, p11). Government contracts being what they are, however, minds soon changed, and MRS was cancelled in 1949, threatening to put over half of Elliott's researchers out of a job (Hendry, 1989) .....

Now it so happened that the collapse of the MRS project coincided roughly with the inaugural meeting of the NRDC, so it made good political sense for the government to try to keep Coales' team together. The NRDC therefore chose Elliott to turn its now idle 152 into a medium-sized computer to complement the larger Manchester machines, issuing the first of a number of contracts in September 1950. The result, in 1953, was the Elliott 401, the prototype of a series of five increasingly sophisticated "400-Series" machines. The 401 was designed by Andrew St. Johnston as a valve-based computer with delay line and magnetic disc

memory, and ran its first program in March 1953. It had a front-loading modular design for ease of maintenance, achieved high availability figures, and actually remained in use until 1965. For more on the 401, see Lavington (2000). The machine itself is now preserved at the Science Museum, London.

> **Key Innovation - Magnetic Disk Storage:** One of the 401 innovations was the first magnetic disk. This was designed by Chris Phillips as a more compact and easier to engineer form of magnetic medium than the magnetic drum. However, the functional principles - track-based storage, access latency, etc - were the same as for the drum. **<< AUTHOR'S NOTE: Comments as for the magnetic drum [see Section 2.1]. >>**

The 401 was followed in 1955 by the 402 and 403 production machines, the 403 being historically noteworthy because it fielded an early version of **"pipelining"** .....

> **Key Innovation - "Pipelining" and Related Concepts (First Mention): "Pipelining"** is a technique "whereby multiple instructions are overlapped in execution" (Patterson and Hennessy, 1996, p125). We have already seen how the buffer concept allowed a temporary memory resource to be used to speed up I/O [see Section 1.3]. The Elliott 403 designers simply applied a similar concept to the instruction stream as well, arranging for the FETCH part of the basic FETCH-and-EXECUTE cycle [reminder] to be a step ahead of the EXECUTEs. This was accomplished by providing an **"Instruction Buffer"**, to contain instruction <n + 1> in FETCH mode, while the Instruction Register [reminder] contained instruction <n> in EXECUTE mode. Needless to say, when instruction <n> had been successfully executed, an important decision needed to be made - had that instruction resulted in any form of program jump? If it **had not**, then the operands corresponding to the presumed next instruction, that is to say, instruction <n + 1> in the Instruction Buffer, could safely be used, and were duly copied to the Instruction Register in a ready-to-execute state. If, on the other hand, a program jump **had** been requested, then the presumption had been wrong, and the contents of the Instruction Buffer were unsafe, had to be discarded, and a new FETCH operation done to initiate the program branch. Since the safes were generally more frequent in practice than the unsafes, the technique delivered real savings in speed of processing. [We return to these concepts in Part 5, when dealing with the IBM 7030 and CDC 6600.] **<< AUTHOR'S NOTE: Neuroscience does not know enough about biological processing architectures to judge whether the brain has devised some sort of op code / operand system for itself, and accordingly to judge whether it does pipelining or not. We suspect that it does, allowed to by its own internal modularity and forced to by its very slow clock speed, and shall be returning to the question in Part 7. >>**

The last of the Elliott 400-Series was the Elliott 405. This sold well both at home and internationally [possibly under the name NCR 304?], thanks to a licensing agreement with the American NCR Corporation. For more on the early Elliott range see Gabriel (1997) or Hoare (1981). Jack Smith (2001) provides a handy memoir of how an Elliott 403 contributed to the Australian weapons research programme. For the later "800-Series" machines, see Part 5.]

> **Key Innovation - Direct Memory Access:** The architecture devised for the Elliott 405 allowed its programmers to maximise CPU performance by re-routing much of its non-essential traffic around it rather than through it. Using an electronic equivalent of a city orbital [US = beltway], data *en route* from peripheral to Main Memory was given a direct access route, **without going via the CPU** (hence the epithet "direct"). This was implemented by giving each peripheral its own controller, with its own instruction set. This way of doing things immediately became an industry standard, and is nowadays known as **"Direct Memory Access" (DMA)**. **<< AUTHOR'S NOTE: The DMA set-up is not in essence dissimilar to the psychological concept of "implicit learning", that is to say learning which gets stored without ever having been consciously attended to. >>**

The success of the 400-series kept Elliott so busy during the early 1950s that they even had to give some of their good ideas away. We refer here to the 1952 **"Nicholas"** - another Charles Owen design, with system software by George Felton - a machine which Elliott put together in a hurry to carry out in-house data processing. So successful did this prove, that the NRDC arranged for Ferranti's London works to turn it into a marketable package, and the result - the **"Pegasus"** - ran its first program in October 1955 and entered commercial service six months later (Lavington, 2000). It proved a highly reliable general purpose serial computer, and despite having to make do with thermionic valves and magnetic drum storage, its designers had devised a "powerful, straightforward" (p16) 49-item three-operand instruction set which made for very

concise programming, although in achieving this they had been assisted by one of the NRDC engineers, Christopher Strachey .....

> **BIOGRAPHICAL ASIDE - CHRISTOPHER STRACHEY:** [This from Campbell-Kelly's (1985) technical biography.] Christopher Strachey (1916-1975) was the son of a First World War cryptologist. He graduated in mathematics and physics at Cambridge, and ended up as a wartime radar engineer. Employed in 1939 by the Standard Telephone and Cable Company on the development of thermionic valve technology, he had occasion to run complex calculations through a small differential analyser, and this aroused his interest in computing. Upon his demobilisation in 1945, he obtained a position as a science teacher, firstly at St. Edmund's School, Canterbury, and later at the prestigious Harrow School, where a chance encounter with Norbert Wiener's recently published "Cybernetics" re-awakened his interest in computing. He therefore approached the National Physical Laboratory's Pilot ACE development team a few miles away across Middlesex, and by January 1951 was attempting to write programs in his spare time. Inspired by an article on computerised board games by the NPL's Donald Davies (Davies, 1950), he chose to attack the problem of getting a computing machine to play draughts. He did not quite succeed, but his program was promising enough to earn him a second attempt, this time on the Ferranti Mark I at Manchester University, and over the ensuing months he improved the code so much that the university promised him a job as soon as finances allowed. In the event, however, he was poached by Lord Halsbury's newly created NRDC before this could happen, and joined them in June 1952. He presented his draughts program at the Second ACM Conference in Toronto in September 1952, and the consequent boost to his reputation then earned him a succession of roving placements in both North America and Britain. This included time on site in 1953 evaluating progress of the Elliott 401, in 1954 improving the Ferranti Pegasus, and in 1958 helping to develop the MUSE [more on which in Part 5]. He left the NRDC in 1959 to set up in private consultancy, and for six years toured the country, contributing to the development of third generation programming languages [see Part 6]. Finally, he was headhunted in 1965 by Oxford University to direct their recently established Programming Research Group, and worked there until his death in 1975.

One of the things Strachey did for Pegasus was to help design an enlarged set of CPU registers, in what is now known as a **"general register"** arrangement (Lavington, 2000). He began by looking at the inefficiencies and inadequacies of earlier instruction sets and found that the chief time stealers - the "red tape" instructions - were control branches, data transfers, and handling large data arrays. He therefore arranged a simultaneous upgrade of the logic circuitry, the register set, and the instruction set. A 39-bit word size was chosen, within which was a six-bit op code field. This would theoretically have allowed up to 64 separate operations, but only 49 were initially used. The operations were presented as eight blocks of eight, by sub-dividing the op code into two three-bit triplets, and transliterating each of these into octal. Thus 010011 was treated as 010 (octal/decimal = 2) followed by 011 (octal/decimal = 3), and shown as 23 (decimal 19).

> **TECHNICAL ASIDE - OCTAL:** Octal is the base eight numbering system, which means that each left order integer represents a power of eight higher than the one to its right. Only the digits 0 to 7 inclusive are used, because the decimal 8 will always "carry one" - thus octal 7 plus octal 1 gives octal 10 (decimal 8), and octal 777 (decimal 511) plus octal 1 gives octal 1000 (decimal 512).

Programmers soon began to remember these numeric codes as **"mnemonics"** for the operations they controlled [more on which in Section 3.2]. Eight accumulator registers were built into the CPU, numbered X0 to X7, with X0 always set to binary zero. These could be used for any relevant purpose, specifically indexing large data arrays, loop counting, and holding intermediate values during complex calculations. This is how array handling worked .....

> **Key Innovation - Handling Data Arrays:** When allocating Main Memory for data storage, it frequently happens that a data item naturally occurs a number of times. Suppose, for example, that you needed to store <GROSS-WEEKLY-PAY> for every one of a thousand employees. The trick here is to set up a contiguous **"data array"**, that is to say, one thousand consecutive word addresses in Main Memory. You can then select the particular occurrence you are momentarily interested in by **"indexing"** that array with a bracketed suffix. Thus <GROSS-WEEKLY-PAY(INDEX)> will get you record #8 when INDEX reads 7 (all computer counts start from zero,

remember), record #652 when it reads 651, and so on (it will also blue screen on you if you are careless with your indexing and try to access a value less than zero or greater than 999). << **AUTHOR'S NOTE: Neuroscience does not know enough about biological processing architectures to judge whether the brain has devised some sort of array indexing system for itself. We suspect that it does.** >>

About 40 Pegasus machines were sold before the product line was discontinued in 1962: the sixth off the production line helped Vickers Armstrong design the *Concorde* supersonic airliner, and the 25th has been restored to working order and is on display in the Science Museum, London. "On the international stage," writes Lavington (2000, p57), "the huge number of modern commodity computers with a general register-set architecture is testimony to the usefulness of Christopher Strachey's ideas for Pegasus". For background detail on Pegasus see Cooper (1990) or Lavington (2000).

2.4 EDSAC and EDSAC Clones

We have already explained in Part 3 how Cambridge University's EDSAC had come from behind in the years 1946 to 1949 to put together the world first stored program computer. The machine continued to be improved during the early 1950s, and brought forth many innovations, not least microprogramming [more on which in Part 5] and assembler programming [see Section 3.2 below].

The EDSAC development which most concerns us at this juncture is Maurice Wilkes', David Wheeler's, and Stan Gill's work on **"subroutines"**, ca 1947. Smotherman (2002) reviews the early history of the subroutine concept, and suggests that Ada, Countess of Lovelace, deserves first mention in that in 1842 she proposed reusable routines for Babbage's Analytical Engine. Alan Turing then proposed much the same for the Pilot ACE in 1946, followed by the EDSAC team in 1947. Konrad Zuse also deserves mention in that his Z4 was upgraded ca. 1950 to allow a single subroutine to be executed by mounting it on a second tape reader. Here are the some of the technicalities .....

Key Innovation - Subroutines: Subroutines are "self-contained sequences of orders each of which performs a distinct part of the calculation in hand" (Wilkes, 1956, p86). More importantly, they allow a discrete operation to be developed once and then re-used as often as wanted, and as such they represent probably the greatest single aid to programmer productivity ever devised. In practice, however, there are several ways of benefiting from pre-written code. The first is to copy a trusted pre-existing program in its entirety and then edit it where necessary, the second is to edit in a block of instructions from such a program, and the third is to set a block of instructions up as a program-in-miniature - the **"subroutine"** - and then formally transfer control to it. All three ways work, but only the subroutine approach creates genuinely reusable code. The transfer of control is known as making a **"call"**, and the eventual transfer back is known as a **"return"**. Wilkes explains that it is frequently an engineering choice whether to perform a given operation via a subroutine, or else to hardwire in some additional logic circuitry. As is often the case in such matters, there is a trade off between complexity (circuitry costs money to design and test) and speed (subroutines add execution time), and it is the relative frequency of the task which often swings the decision one way or the other. The EDSAC, for example, deliberately chose a slow subroutine to perform division on the grounds that divisions were comparatively infrequent. Other "obvious examples of subroutines suitable for including in a library are ones for calculating sines, cosines, exponentials, and similar functions" (*ibid*.). << **AUTHOR'S NOTE: Biological cognition probably relies heavily on subroutine calls. Taking speech production as an example, and adopting the hierarchical view proposed by psycholinguists such as the University of Arizona's Merrill Garrett (see, for example, Garrett, 1990), we find that there are several points at which the primary justification for the existence of a high level process can only be to control the rapid concatenation of a sequence of lower level ones. Thus with the allocation of phonemes to words at the phonological build stage, or with the allocation of muscle contraction profiles to phonemes at the subsequent muscle activation stage. Another (very significant) property of subroutines is covered when we get to the work of John McCarthy in Section 4.7.** >>

Key Innovation - "Return Codes": One of the complications introduced by subroutines is that the calling program must be sensitive to the possibility that the called subroutine might fail in some way. Such a failure will often "crash" or "loop" the entire machine, but if it does not, then it falls to the calling program NOT to carry on in what is then probably a corrupt state. The way around this is (a) for the calling program to precede the call with the

setting of a **"return code"** data field to a non-zero value (to be interpreted as "failure"), (b) for the last instruction in the called program to set that same field to zero (to be interpreted as "success"), and (c) for the first instruction in the calling program after the return to test for zero or non-zero. If the subroutine has worked, then the return code will have been set to zero, and the calling program can proceed about its business. If the subroutine has not worked, then the calling program can branch to a pre-written soft abort control sequence. So successful did the return code idea work in practice, that designers soon introduced a whole range of non-zero return codes (alternatively, **"error codes"** or **"diagnostics")**, each enabling a different type of failure to be accurately identified. Internet users see these every day, of course - click here for a non-existent Internet link, and note the resulting error message. << **AUTHOR'S NOTE: Neuroscience does not know enough about biological processing architectures to judge whether nature has invented for itself some sort of cognitive return code. For our part, we suspect not only that it has, but also that the mechanism works closely with the biological interrupt system [see Section 1.3] to support hierarchical control in general terms. The issue will be reviewed in greater detail in Part 7.** >>

EDSAC was also the inspiration for the Lyons Company's **LEO I** [reminder], and here again the main player was a demobilised wartime boffin, an ex-TRE electronics engineer named John M.M. Pinkerton (1919-1997). Pinkerton took charge of the project in January 1949, and the resulting EDSAC clone ran its first test program on 5th September 1951 (Bird, 1994), and its first business application - a bakeries valuation - on 29th November 1951 (Land, 1999). It then parallel ran its first live payroll on 24th December 1953, at which time its designers claimed that what took an experienced clerk eight minutes could be done by the machine in 1.5 seconds (Bird, 2002). After careful cross-checking of manual and computer output, the manual system was withdrawn, and LEO ran unsupported on 12th February 1954 (Caminer and Land, 2003). Apart from the immediate benefit to their Accounts Department, Caminer and Land (2003) describe that date as marking "a world first". A more powerful machine, the LEO II was begun in 1954 and entered service in May 1957. Eleven of these machines were built, but only the last four were equipped with the more expensive ferrite core memory (Ferry, 2003). A third generation machine, the LEO III is mentioned in Part 5.

3 - First Generation Software

Alongside the electronics and engineering developments, the early 1950s also found a growing army of computer programmers struggling to keep their CPUs gainfully employed. The problem here was essentially one of too good a press, for now that potential system sponsors knew how versatile the new digital machinery could be, they were raising requests for computer solutions faster than the necessary programs could be written. Unfortunately, the penny was also beginning to drop as to how much time these programs could take to write; and worse, how difficult they then were to amend; and worse still, how often customers would ask for such amendments. Thus .....

> "In June 1949 I realised that a good part of the rest of my life would be spent finding errors in my own programs." (EDSAC's Maurice Wilkes, in Sabbagh, 1999.) [We believe (May 2003) that Wilkes is still alive, so it is clearly good to make lots of errors!]

3.1 Application vs Operating Software

To cope with this early **"software backlog"**, the computing industry was forced to specialise on the programming side of things the way it had specialised during its formative years in the physics and the engineering, when valve men with soldering irons produced the hardware and mathematicians decided how it should be used. It began holding summer schools (for example at Cambridge in 1950) and providing training courses, and at first two, and then three, and then four, areas of specific expertise emerged. The first distinction to emerge was that between **"computer hardware"** - the physical machine - and **"computer software"** - the programs which had to be written to make the hardware productive [the word "software" itself was coined by John Tukey at Bell Labs]. The second distinction was then between **"application software"**, programs which specifically addressed this or that real world problem, and **"operating**

software" (sometimes **"systems software"**), programs which helped interface these applications with the hardware in some standardised and user-friendly way. A computer's **"Operating System"**, or "OS", became the software which was actually "closest to" the hardware, and what was happening in essence was that one program - the OS - ended up (thanks to yet more call-and-return branching) executing another - the application .....

> **Key Innovation - Application Software:** Applications are what you, the user, want to do with your computer. If you wanted to run an appointments system, for example, you would need a carefully designed and interlocking set of **"applications"**, each addressing one aspect of the overall task of managing appointments. The distinct practical purpose of each program is known as its **"functionality"**, and it is functionality which gives an application its business value. The sum total of individual applications is known as the **"application system"**, and the programmers who specialise in coding applications are known as "**applications programmers**". Applications programmers are assisted in their work by the **"programming languages"** available on their machines [see next section], and also by a deep understanding of the real world they are trying to computerise. In the event, the best programmers often turned out to be the worst at understanding the real world, and so a separate breed of beings called **"systems analysts"** was recruited to do that part of the job for them, and hybrids of the two sorts of skills - those who can manage both real world and programming tasks - are called **"analyst-programmers"**.
>
> **Key Innovation - Operating Software:** The key to understanding operating software lies in grasping a single basic fact, namely that a high proportion of any one program re-uses code already written (often at great cost) elsewhere. It therefore makes sense not just to provide the applications programmers with a library of tried and tested subroutines, but also with a way of accessing these procedures from within a programmer-friendly keying environment. The operating system is the mechanism by which this is all achieved, and its standard functions include interfacing with the operator, managing the library procedures, managing the peripherals, displaying the contents of magnetic media, knowing what filenames have been allocated, knowing where on the available disks it has put each file (and the really clever operating systems are able to store a given file in a number of non-contiguous fragments, so that takes a lot of managing), managing disk I-O, issuing warnings, help, and error messages, keeping track of date and time, and marking same on the filestock so users know how old they are, and last (but by no means least) controlling the execution of end-user programs. Some operating systems are easier to get along with than others. The worst ones are those invented in the days before the general public were allowed to get their hands on computers. The easy to use ones are called **"user friendly"**, and the trend of late has been for them to become more and more user friendly as time goes by. Operating systems are written by boffins and maintained by "**systems programmers**", who (because they are the only ones who understand what they are doing) get paid a lot more than applications programmers.

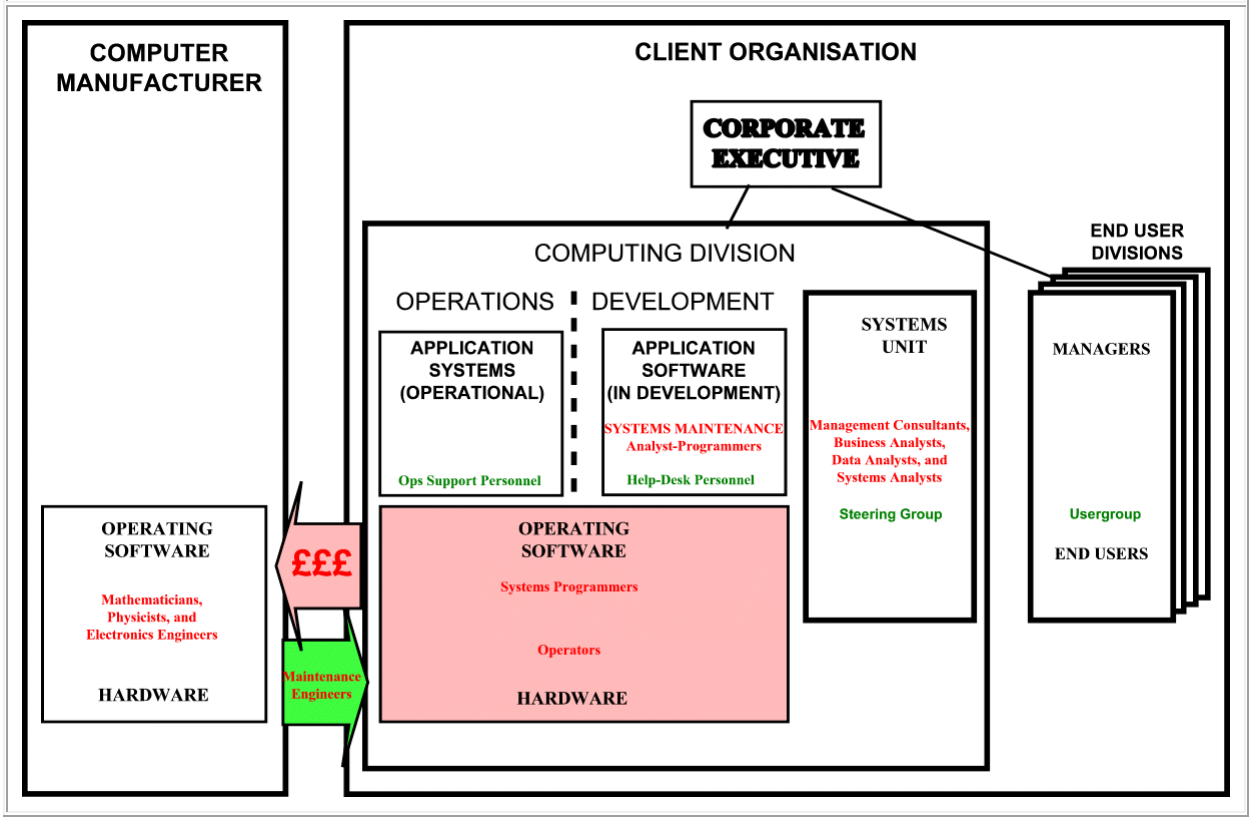
Figure 1 shows where all these (very expensive) people fit within a commercial computing installation .....

Figure 1 - Key Roles in Corporate Computing: Here we see the basic structures and personalities you would find in a typical computerised corporation. Two separate organisations are shown. On the left hand side of the diagram is a computer manufacturing organisation (Ferranti, say). On the right hand side of the diagram is a client organisation, a purchaser-user of computing power. We have subdivided the client organisation so as to show its Computing Division (large panel, left, drawn over scale) separate from its end-user divisions (stacked panels, right, drawn under scale), and both subordinate to the Corporate Executive (top centre). Within the Computing Division, we show Operations and Systems Development activities separately, albeit both share a common physical system (pink panel). Within the Systems Development Department, the principal activity is the writing and testing of application software **for the future**, whilst within the Operations Department it is running application software **already formally authorised for live use**. This separation of duties is vitally important. Indeed in most large organisations, the operations side of things will be located in a purpose-built **"Computer Centre"**, whilst the development side of things remains a headquarters function.

Note the many different specialists (red captions) dotted around the various departments. In the manufacturing organisation, we find the boffins who build the hardware and develop the operating software. Hardware boffins improve circuitry and invent memory mechanisms, etc. Software boffins improve instruction sets, implement ever better pipelining, etc. **"Maintenance engineers"** provide an agreed level of after-sales service (green arrow) in return for a consideration (pink arrow). In the Operations Department, there are **"operators"** to tend the machine itself and **"systems programmers"** to keep it in tune. In the Systems Development Department, it is usual to find

a number of **"systems maintenance teams"**, each consisting of the analysts and programmers familiar with a particular application system. There will usually also be an overarching **"Systems Unit"** [names vary] carrying out more strategic investigation and design.

Given the many specialisms, the need also arises for help desks and liaison officers (green captions). These include **"operations support"** people to liaise with the systems maintenance people in order (a) to schedule work on a day-to-day basis, and (b) to report and prioritise system errors. The maintenance team also have to liaise with the Systems Unit over the commissioning of broader system enhancements. For the sort of systems disasters which result when people do not communicate effectively, see our IT disasters database.



3.2 Interpreters, Assemblers, Loaders and Linkers, and Compilers

As far as software productivity is concerned, there has only ever been one problem and there has only ever been one solution to it. The problem is the unfriendliness of machine code, and the solution has been to distance programmers from its complexity as far as possible. Put simply, writing efficient machine code from scratch is onerous in the extreme, and trying to work out how you did what you did at a later date is nigh on impossible. As a result, generation zero development teams devoted a lot of effort to simplifying the process, and their answer was to write utility programs to stand between the programmers and the machine code. In their simplest form, such programs offered to translate from a relatively human-friendly vocabulary into the decidedly human-unfriendly vocabulary of the machine's instruction set. These utilities were the first **"programming languages"**, and they brought with them some important new terminology .....

> **Key Innovation - Programming Languages, "Source Code", and "Object Code":** A programming language is a computer program capable of recognising selected human-meaningful input commands, and generating as output the logically corresponding machine code instructions. The input commands are known as the **"source code"**, and source code - of essence - is far easier to read for its logical intention than is machine code, both by the original programmer during development, and then by a maintenance programmer at a later date. The output is known as the **"object code"**, and if the developers have done their job properly this will have been optimised for

technical efficiency, as well as being standardised and bug-free. Special dummy source code instructions allow explanatory annotation to be added to the source program as *aides memoires*, but withheld from the machine code.

Ideally, you want to be able to tell the machine something you understand, like <MULTIPLY TAXABLE-PAY-THIS-WEEK BY TAX-RATE GIVING TAX-THIS-WEEK> and have it turn that instruction into something the machine's arithmetic unit will understand, like <00101111001011011000101101001110110111 etc.>. Unfortunately, the source instructions had to be kept short to save memory, and so were usually abbreviated to two- or three-character **"mnemonics"** such as M for MULTIPLY .....

Key Innovation - Mnemonics: In everyday English, a mnemonic is a memory aid of some sort (such as "ROY G BIV" for the colours of the rainbow, etc.). Because there is no point in having source code at all unless it is distinctly memorable, mnemonic systems usually consist of a set of handy verbal tags .....

Example: We have already seen how Pegasus programmers took to identifying their binary op codes by their octal couplets. Thus Pegasus op code mnemonic "four-one" [see Section 2.3] instructs the machine to add a particular given number to a particular accumulator. Not only is "four-one" easier to remember than the logically identical binary code "one-zero-zero-zero-zero-one", but it is also easier to convert back to binary than the decimal equivalent. Thus octal 4 gives you binary 100 and octal 1 gives you binary 001, so octal four-one gives you 100001.

The simplest form of translation utility works on a one-to-one basis, and would nowadays be termed an **"Interpreter"**. However, this level of assistance was not a lot of use to generation zero machine code experts, and so was little used. Instead, the boffins concentrated on a more powerful class of utilities known as **"assembly languages"**, or **"assemblers"** for short .....

Key Innovation - Assembly, Loading, and Linking: "An assembler is a translator that translates source instructions (in symbolic language) into target instructions (in machine language), on a one to one basis" (Salomon, 1993). Jack Gilmore wrote such a program for the Whirlwind, using mnemonic codes, and allowing decimal input. The Assembly process then substituted one or more machine instructions for the mnemonic and turned the decimals into binary. This was followed by SHORT-CODE, developed by John Mauchley, and A-0 developed by Grace Hopper. Wilkes (1956) claims "an assembly subroutine" for the EDSAC in 1950. << **AUTHOR'S NOTE: Neuroscience does not know enough about biological processing architectures to judge whether the brain processes some sort of machine code in some locations and some sort of assembler mnemonics in others. What can be said with certainty is that there are repeated major shifts in data coding as information flows end-to-end through the cognitive system. The most thorough illustration of this we have come across is Ellis's (1982) transcoding diagram, in which each separate information flow is carefully tagged with the nature of the coding method used within it. >>**

One point about simple assemblers was that the object code they produced was actually incomplete. Specifically, it did not include the object code for library-held subroutines. An additional processing stage was therefore needed, called **"loading and linking"**. This was/is an additional utility program which located the various object code modules and - after some careful address cross-calculations - concatenated them into a single executable object code program. When done properly, all subroutine calls will branch and return to valid addresses.

Further major weaknesses gradually became apparent. Firstly, because machine instructions are individually very simple actions, there often needs to be a one-to-many relationship between a mnemonic and its equivalent object code instructions. Secondly, the machine code from one source instruction often needs interleaving with the machine code from adjacent source instructions. For example, mnemonic-by-mnemonic execution may be ideal for interacting with a human operator, but it is totally unable to cope with intentional program loops. And thirdly, assemblers were hardware-specific. You could not write source code once, and have it assembled onto different computers. This is because the one-to-one relationship

between source and object instruction tied the translation program inflexibly to the specific host's specific instruction set. This led to the development of more advanced, machine-independent, programming utilities called **"compilers"** .....

> **Key Innovation - Compilers:** A "compiler" is a package of utility programs, fronted by a powerful conversion routine, supported by all the necessary loaders and linkers, and complete with copious error detection and optimisation routines. As such, they allow programmers to write neatly structured source code, relying heavily on subroutines, and yet leave all the technical details to the compilation process. Where both the amount and the complexity of the object code per source instruction is high, the language is known as a "high level language". In 1951, R.A. ("Tony") Brooker was put in charge of Manchester Mark 1 software and by 1954 had produced the first recognisable compiler, namely the "Mark 1 Autocode". Compilers can conveniently be divided into "scientific", where there the source approximates to formal mathematical representation, or "business-oriented", where the source approximates to everyday English. The standard example of the former is FORTRAN, and of the latter, COBOL. The way things timed out, many first and second generation business systems were written at great cost in Assembler, only to find themselves "up for a re-write" ten years later when COBOL came along. In investment terms, this is not good commercial practice, and British industry (at least) was so eager to squeeze every drop of return out of its capital, that commercial systems written in Assemblers during the early 1960s were still around (as "legacy systems") nearly 30 years later. The present author did a stint on a large Assembler maintenance project as late as 1988.

In fact, the word "compiler" should be treated with some caution, because its meaning drifted somewhat in the early years. Backus (1984) argues that many early "compilers" are better regarded as assemblers, for their most significant aspect is their mnemonic set, rather than any program structuring functionality, while Hopper (1959) emphasises their role in managing subroutines. We include a full (albeit short) Assembly program, and elements of a compiled program, in Part 6, where we shall also be looking at how programming languages give programmers hands-on control over their machine's short term memory resources.

4 - First Generation Artificial Intelligence

Thus far, we have limited our consideration of software to military applications such as codebreaking and ballistics, and to civilian applications such as payroll. However, a number of more adventurous applications had been bubbling under during the late 1940s, and finally hit the headlines in the early 1950s. These were (1) machine translation, (2) semantic networks, (3) mechanised game playing, (4) man-machine conversation, (5) "neurodal" networks, (6) robotics, (7) machine problem solving, and (8) artificial consciousness. Taken together, these make up a domain of enquiry we know today as **"artificial intelligence" (AI)**, and in the remainder of this section we look at each of these AI traditions in turn .....

4.1 Machine Translation (MT)

Hutchins (1997) dates the idea that computers might be used to automate the process of natural language translation to two conversations which took place 20th June 1946 and 6th March 1947 between the aforementioned Andrew D. Booth [see Section 2.1] and Warren Weaver of the **Rockefeller Foundation**, and a letter dated 4th March 1947 from Weaver to the cyberneticist Norbert Wiener .....

> **BIOGRAPHICAL ASIDE - WARREN WEAVER:** Weaver was a University of Wisconsin mathematics professor, turned career philanthropist, turned wartime government advisor, turned "wartime 'Mr Fix-It'". He joined the Rockefeller Foundation in 1932 as Director of the Division of Natural Sciences, did much of the facilitation during the early construction of the giant 200-inch Hale Telescope at the Mount Palomar Observatory [this project actually lasted 1928 to 1948, with a break during World War Two], and by the start of the war had established enough of a reputation as a loyal scientist to be appointed Director of the Control Systems Division of the newly formed US National Defence Research Committee (NDRC) [not to be confused with the British NRDC mentioned in Section 2.2]. Much like Princeton's John von Neumann [reminder], his role within the NDRC made him a roving advisor-facilitator with responsibility for just about

every mathematical aspect of the US war effort [mostly classified, of course], not least the transition from analog to digital computation (see Mindell, 2002). It also involved him in at least one top secret mission across the Atlantic. After the war, Weaver became interested in the concept of MT, and it was on one of his convenient Rockefeller Foundation visits that he met Booth, who, as we are about to see, shared his belief that MT was theoretically possible. He also went official second author on one of the major classics of modern information theory, namely Shannon and Weaver (1949).

Following Weaver's inspired and energetic string-pulling, one of the first computers off the blocks was Booth's own ARC [see Section 2.1], and the formative event here seems to have been a chance professional exchange of minds between Booth and the Cambridge geneticist Richard H. Richens. Richens dates their first meeting to 11th November 1947, and explains that it came about thanks to their shared interest in punched card methods of processing research data. Richens later explained that .....

"..... the idea of using punched cards for automatic translation arose as a spin-off, fuelled by my realisation as editor of [the journal *Plant Breeding Abstracts*] that linguists conversant with the grammar of a foreign language and ignorant of the subject matter provided much worse translations that scientists conversant with the subject matter but hazy about the grammar." (Richens, 1984; private correspondence cited in Hutchins 2000)

The unlikely pairing of the x-ray crystallographer with the drum storage system and the geneticist with punched card experience then set to work trying to mechanise translation using a stored dictionary. In a jointly authored paper published a few years afterwards, they explained what progress they had made so far, and made a number of predictions as to how they saw the new science developing .....

"The most obvious way of using a computing machine for translation is to code each letter of a proposed dictionary word in 5-binary-digit form and to store the translation in that memory location having the same digital value as the aggregate value of the dictionary word. Translation would then consist in coding the message word (a teletyper does this automatically) and then extracting the translation in the same or next lower (digitally valued) storage position, the first giving exact translation and the second the stem translation with a remainder. [.....] "

TECHNICAL ASIDE: This is our old friend the Baudot telegraph code again [reminder], in which the letters of the alphabet are given binary codes in the range 00000 (decimal 0) to 11111 (decimal 31). The letters C-O-W, for example, would be given the following codes .....

C = 01110 (decimal 14)
O = 11000 (decimal 24)
W = 10011 (decimal 19)

What the authors were suggesting was that they should concatenate the three separate five-bit binary codes to give a 15-bit binary code (in this case, 011101100010011 or decimal 15,123). This code would then be automatically generated by typing the word "COW" at any standard teletype keyboard, and all that the computer needed to do was to store the translation of that word at storage address 15,123 in the computer's memory. In practice, however, it was not that easy .....

"Despite its simplicity, this scheme is quite impracticable with any existing storage device because the naive process of placing each dictionary word in a storage position equivalent to its own binary numerical version is grossly redundant."

TECHNICAL ASIDE: The San Jose Scrabble Club helpfully lists 973 three-letter words in the English language, whilst there are 32,768 different 15-bit addresses between 000000000000000 and 111111111111111 inclusive. Storing one in the other would therefore give you an unused memory factor of approximately 97%! This would be an unforgivable waste of resource even in the modern world of cheap memory, but in the 1940s would have been a total non-starter.

"[..... so] to overcome this difficulty, a slightly more sophisticated approach has to be adopted. Dictionary words are stored in alphabetical order and in consecutive memory locations. Possibly these might be broken up into initial letter groups such that each group starts in a storage location indicated by the binary coded form of its initial [.....] The coded word to be translated is sent to an arithmetic register and is there compared, by subtraction, with the dictionary words. In this way,

by means of the normal conditional control transfer order [ie. you only do it when the aforementioned subtraction gives zero, which will only happen when the two items are identical - Ed.], the dictionary stem corresponding to the word can be obtained and its translation printed. Next, the remainder of the message word, left after subtraction of the stem, is shifted to occupy the extreme left-hand register position, and the comparison process is repeated, this time with the words of the ending dictionary." (Richens and Booth, 1955, pp45-46)

> **TECHNICAL ASIDE:** The authors are here addressing the problem of translating morphologically complex words such as "COWS", where we have a singular root noun, referred to here as the "stem", to which is suffixed the pluralising morpheme "-S". This is the problem of **"inflection"** .....
>
> > **BASIC LINGUISTICS - INFLECTION:** "Inflection is variation in the form of a word, typically by means of an affix, that expresses a grammatical contrast which is obligatory for the stem's word class in some given grammatical context". It is the mechanism by which many of the world's natural languages cope with the number (single or plural), gender (masculine, feminine, or neuter), and grammatical role of adjectives and nouns, and the number (single or plural), tense (past, present, or future), and person (first, second, or third) of verbs. Thus the suffix <s> will pluralise many English nouns (e.g., "cats") or put many verbs into the third person singular (e.g., "he eats").
>
> Richens' and Booth's proposal was that such words should be matched in two stages, from two functionally separate dictionaries, namely a stem dictionary and an ending dictionary. Thus COWS would divide as <COW + S>, translate into German as <KUH + E> and be reconstituted (on this occasion correctly) as KUHE. In practice, however, this was not that easy either .....

Other universities quick to get involved were MIT and the Harvard Computation Laboratory. The first MT conference was organised by Yehoshua Bar-Hillel of the University of Jerusalem (fortuitously on a sabbatical at MIT), and took place 17-20th June 1952. Among the delegates were Whirlwind's Jay Forrester, Harry Huskey, Andrew Booth, and the linguist in charge of the simultaneous translation facility at the 1945-1946 Nuremberg Trials, Georgetown University's Leon Dostert. This conference was followed two years later by the "Georgetown-IBM" demonstration of automated translation. This was organised by Dostert, took place on 7th January 1954 at IBM Headquarters, New York, and used software developed by Paul L. Garvin with technical input from IBM's Peter Sheridan and management backing from Cuthbert Hurd [whom we have already met in Section 1, and of whom even more in Part 5]. The demonstration processed a 250-word Russian-English vocabulary, and made front page news the next morning.

> **BIOGRAPHICAL ASIDE - ANTHONY G. OETTINGER:** Anthony G. Oettinger studied for his 1954 doctorate at the Harvard Computation Laboratory on the requirements of an automated Russian-English dictionary (Oettinger, 1955). Given the abysmal state of US-Russian relations in the McCarthy era, there is little doubt that it was the promise of being able to translate intelligence material out of Russian more effectively which made MT a defence issue, and which therefore secured it much of its funding [see, for example, Bar-Hillel (1960)]. We meet Oettinger again in Section 4.5.

Unfortunately, the Richens-Booth approach only works for the relatively simple words in a language, and starts to descend into a morass of ever more complicated logic branches and special rules once it encounters "irregular" words and complex grammatical structures. Erwin Reifler, of the University of Washington, Seattle, was one of the first to try to design his way around this particular problem. In Reifler (1955), he proposed carefully allocating separate dictionaries to separate "operational form classes", that is to say, to each of the standard grammatical categories. Form classes with very large memberships include nouns (cup, table, man, etc.), adjectives (big, happy, etc.), and verbs (eat, walk, fly, etc.), and form classes with very small memberships include determiners (a, the, some, etc.), pronouns (I, she, it, etc.), auxiliary verbs (to be/have/get, etc.), prepositions (in, to, by, etc.), and conjunctions (and, but, because, etc.). Clearly recognising the modular nature of biological language processing, he then proposed separate drum memories - up to 15 of them! Thus .....

"If magnetic drums are used for memory storage, we may distinguish between large-drum memories and small-drum memories. A number of each will be needed. [.....] Operational form classes whose membership is large will require large drum memories. [//] *Large-Drum System:* [.....] For the large operational form classes, a total of four large-drum units will be needed, containing the following memories: (a) *The capital memory* for substantives and substantive constituents, (b) *The attributive adjective memory*, including the cardinal numerals, except the few included in the memory of determinative 'pro-adjectives', (c) *The principal verb memory*, excluding the few verbs included in the memory of verbs always or sometimes requiring a predicate complement, [and] (d) *The predicate adjective memory*, including all adverbs of adjectival and numerical origin. *Small-Drum System:* In the small-drum system, an individual memory will be assigned to each of the operational form classes with a comparatively small membership. The ten small operational form classes discussed above may be entered into ten small drums, but it may be desirable with German to subdivide the class of propositions into six sub form-classes, according to whether they require their complements in the following cases: genitive, dative, accusative, genitive and dative, genitive and accusative, or dative and accusative. This would increase the number of small drums to 15. With the smaller number of drums, a greater complexity of equipment and circuitry would be necessary to handle the prepositions. It is, thus, a matter of engineering economics whether 10 or 15 small-drum memories are used." (Reifler, 1955, pp159-160; italics original)

> **TECHNICAL ASIDE:** In retrospect, Reifler's suggestion is nowhere near as silly as it might sound; it was just a couple of years ahead of its time. With the development of database technology over the ensuing decades, it gradually became possible to store fundamentally different datasets such as Reifler was proposing on a single physical resource. In other words, any Database Management System (DBMS) worth its salt could easily make a single magnetic disk look like 15 disks. Figure 2(b) shows the sort of data analysis which would have needed to be done in order to allow this to happen. We shall be saying a lot more about how DBMS do what they do in Part 5 (Section 3.1) and Part 6, and about why they are so important in Part 7.

Coping with noun plurals and (in certain languages) case variations, as well as with verb endings, was, however, just the beginning, for it soon became apparent how complicated were the real-life languages programmers were trying to translate. The problems posed by irregular verbs, or - worse - by the use of ironical or figurative language, or by the intricacies of pronoun resolution, or by the effects of context, or by **"polysemy"** (that is to say, the multiple meanings of words), have still not been overcome half a century later, and the 1950s efforts were often laughable. Here, from Masterman (1967, p202), is an example of just how hopeless simple word-for-word translation can be [compare (2) and (3) for intelligibility, and note that Latin is a heavily inflected language] .....

(1) ORIGINAL LATIN: Possibile est, at non expertum, omnes species eiusdem generis ab eadem specie ortum traxisse.

(2) WORD-FOR-WORD MACHINE TRANSLATION: Possible is however not prove/lacking all species/appearance same genus/son-in-law from same species/appearance arise draw.

> **NB:** Note the underlined alternatives here. These are examples of the multiple meanings mentioned above, where a single L1 word has two or more valid L2 translations, and where the final choice relies on contextual cues.

(3) FINAL ENGLISH: It is possible, though not proved, that all species of the same genus have been derived from the same species.

Now Margaret Masterman (1910-1986) knew Richens thanks to their joint involvement with Cambridge University's Language Research Unit .....

"Research on MT at Cambridge began [.....] with the informal meetings of the Cambridge Language Research Group in 1954 [.....] In 1956 a grant was received from the National Science Foundation to pursue MT research, and the Cambridge Language Research Unit (CLRU) was formed, a research organisation independent of the University of Cambridge, with Margaret Masterman [wife of the philosopher R.B. Braithwaite] as director. In later years research grants were also received from the US Air Force Office of Scientific Research, the Office of Scientific and Technical Information (London), the Canadian National Research Council, and the Office of Naval Research (Washington, DC) [.....] By 1967, active research on MT at CLRU had declined; many of its members had moved elsewhere, mainly within the Artificial

> Intelligence field and often with a continued interest in MT [.....] There were four main themes in its MT research: the thesaurus approach, the concept of an interlingua, 'pidgin' translation, and lattice theory. The focus was primarily on semantic problems of MT, and syntactic questions were treated secondarily. Although procedures were intended to be suitable for computers, most of the proposals were tested only by manual or punched card simulations because access to a computer proved to be difficult for many years." Hutchins (1986)

All four CLRU avenues of attack are directly relevant to our argument, including their investigations of pidgin. Indeed, one of the group's most fruitful observations was that there were some rather telling similarities between the halting and generally awkward word-for-word machine translations [sentence (2) above] and real life "pidgins" .....

> **BASIC LINGUISTICS - PIDGINS AND CREOLES:** A **"pidgin"** is "a reduced language resulting from contact between groups with no common language", and a **"creole"** is "a pidgin or jargon that has become the native language of an entire speech community, often as a result of slavery or other population displacements". Thus Pidgin English is any rudimentary form of English, especially if spoken by aboriginal peoples for trading purposes, and the pidgin spoken in Papua New Guinea contains many quaint phonetic corruptions, such as *lusim* (= abandon), *bigpela man* (= adult), *klostu* (= almost), and *narapela* (= same again).

The academic point was that since pidgins somehow contained all that was necessary to render a complex message meaningful across a language barrier, it might make sense to design some form of **"mechanical pidgin"** into your machine translation system. And if you did this carefully, you would then be able to record most of L1's inflections in coded form, tucked in as **"pidgin markers"** alongside the word stems, giving you what is known as a **"continuous form"** translation. With a continuous form version safely under your belt, you would have a much better chance of producing a contextually, as well as grammatically, sound L2 draft. According to Masterman (1967), the term "mechanical pidgin" came from Richens. Here, still from Masterman (1967, p202), is how sentence (2) above might look in continuous form, complete with inset markers .....

> **(4) CONTINUOUS FORM MACHINE TRANSLATION WITH INFLECTION MARKERS:** Possible z is however not prove/lacking z all m species/appearance same g genus/son-in-law z from same species/appearance o arise z draw p.

Masterman selected her inflection markers from the following list from Richens and Booth (1955, p35) .....

> a = accusative case; d = dative case; f = future tense; g = genitive case; i = indicative mood; l = locative case; m = multiple; n = nominative case; o = oblique; p = past tense; q = passive form; r = partitive; s = subjunctive mood; u = untranslatable; v = vacuous; z = unspecific

The technical name for an intermediate translation form of this sort is an **"interlingua"** .....

> **Key Innovation - Interlinguas:** An **"interlingua"** is a language form intentionally divorced from both L1 and L2 in a translation, but from which both L1 and L2 (and, ideally, all other languages) can be generated. Strictly speaking, this definition incorporates international languages such as Esperanto, but we restrict ourselves here to any sequence of heavily inflected protosemantic codes produced by parsing. << **AUTHOR'S NOTE: Neuroscience does not know enough about biological processing architectures to judge what is the basic central cognitive code, although there is a broad consensus that it is non-verbal. Nor, indeed, is the code open to introspection because our output sentences (written or spoken) come to us out of the inner darkness more or less ready to deliver [this point from Velmans (1991)]. We shall be returning to this point in Part 7.** >>

Masterman is also important because she suggested a major improvement to the Richens-Booth continuous form interlingua. In Masterman (1957), she simply took her copy of *Roget's Thesaurus* from the shelf, and used its headings as a ready-made semantic interlingua, complete with associated numeric codes. We have ourselves discussed this approach in our paper, **"The magical name Miller, plus or minus the umlaut"**

(Smith, 1997b, in Harris (Ed.)), where we gave some of the credit for the idea to the 19th century professor of linguistics, Max Müller .....

> "[Müller] was one of the first to recognize that word meanings are constantly evolving, and spent a lifetime studying the etymologies of words in a variety of languages with a view to tracking down their derivations (Müller, 1887). Working in this way, he reduced all languages to much smaller pools of word roots and fundamental concepts. To give but one example, the Greek words for lift, compare, tribute, spread, delay, bury, madness, endure, recover, reproach, help, and excel (to name but a few) all derive in various ways from ϕ ε ρ , 'to bear'. Müller's central argument goes as follows: 'Give us about 800 roots, and we can explain the largest dictionary; give us about 121 concepts, and we can account for the 800 roots. Even these 121 concepts might be reduced to a much smaller number, if we cared to do so' (op. cit., p551.). But not all authorities agree on where to draw the line. Whereas Müller chose a pool of 121 'original concepts', Saban (1993) chooses 31 basic 'semantic fields', and in the original 1852 edition of his renowned thesaurus Peter Roget went for 1,000 'related ideas'. So why the disagreement? Well insofar as it was responsible for importing the concept of the bit from engineering into psychology, the answer lies in George Miller's paper [i.e., the Miller (1956) "Magical Number Seven" paper - Ed.]. This is because the logarithmic nature of the bit renders 31, 121, and 1,000 not so different after all: to be precise, they might just represent three points on the *powers of two* dimension. That is to say, we need *five bits* of information to specify a single one of Saban's 31 semantic fields, *seven bits* of information to specify a single one of Müller's 121 original concepts, and *ten bits* of information to specify a single one of Roget's 1000 related ideas [raising two to the powers five, seven, and ten gives 32, 128, and 1024 respectively - Ed.]. Could we, therefore, be looking at a semantic memory which somehow operates according to a binary chop principle?" (pp204-205).

> **NB (14th July 2003):** Although ignorant of her work at the time, we owe it to Masterman to point out that she was there 40 years before we were. This from Masterman (1957, p41): "..... in any kind of writing which builds up into an argument, thesaurus-heads tend to be introduced in powers of two, each topic being introduced concurrently with that to which it primarily contrasts". It is simply a very efficient way to think about things - as Hamlet found out with all his to being and not to being. Nor, it turns out, was Müller really the first, either, for he was simply following suggestions from the 17th century "universal languages" of Cave Beck and Athanasius Kircher. We might also add that Richens' 80 or so "naked ideas" and Parker-Rhodes' (1978) 150 or so "primitive items" are not that far from the line.

So to put things back into perspective, what the early MT researchers had done was to re-open a very old (and more than normally wriggly) can of philosophical worms, and in retrospect it is not really surprising that Booth and Richens did not keep their MT lead (such as it was) for long. The ARC was little more than a hobby kit computer, and the big machines soon overtook it, and - the pessimists apart - there was still a common presumption that things would be better when newer, faster, machines with more powerful programming languages came along. Nevertheless, what the Birkbeck (Booth), MIT (Yngve, Bar-Hillel), Seattle (Reifler), and Cambridge (Richens) groups had been doing was laying the foundations for the modern science of **"computational linguistics"**.

4.2 Semantic Networks

Sadly, MT's problems did not go away, even with interlinguas and inflection markers and more powerful computers. Irony, ellipsis, context, and idiom, still had to be overcome. Reifler himself gives the example of trying to translate "he is an ass" into Chinese, assuring us that you cannot simply substitute the Chinese word for ass because Chinese folklore does not regard the ass as inherently stupid. There are other traditionally difficult and highly illustrative problem phrases, perhaps the most famous of which is "out of sight, out of mind", which in the absence of some extremely nifty programming invites translation as "invisible lunatic" - perfectly logical in its own way, but also perfectly stupid. Another commonly cited howler comes from the English-to-Russian MT software which reputedly rendered "The spirit is willing, but the flesh is weak" as "The steak's off, but the vodka's OK". Warren Weaver himself summarised all the criticisms in a privately (but widely) circulated memorandum on the subject (Weaver, 1949), and it was this which drew broader academic attention to the underlying issues. Thus it was that linguists and philosophers such as Bar-Hillel grew deeply pessimistic about MT's long term prospects [wait and see what he says about

John McCarthy in Section 4.7!]. You could not do MT, in Bar-Hillel's view, until you had acquired a far better understanding of linguistics, and those who already understood linguistics were already wagering that you would not be able to do it then either (and, as we shall be seeing in Part 5, nobody half a century later has come close to relieving them of their money).

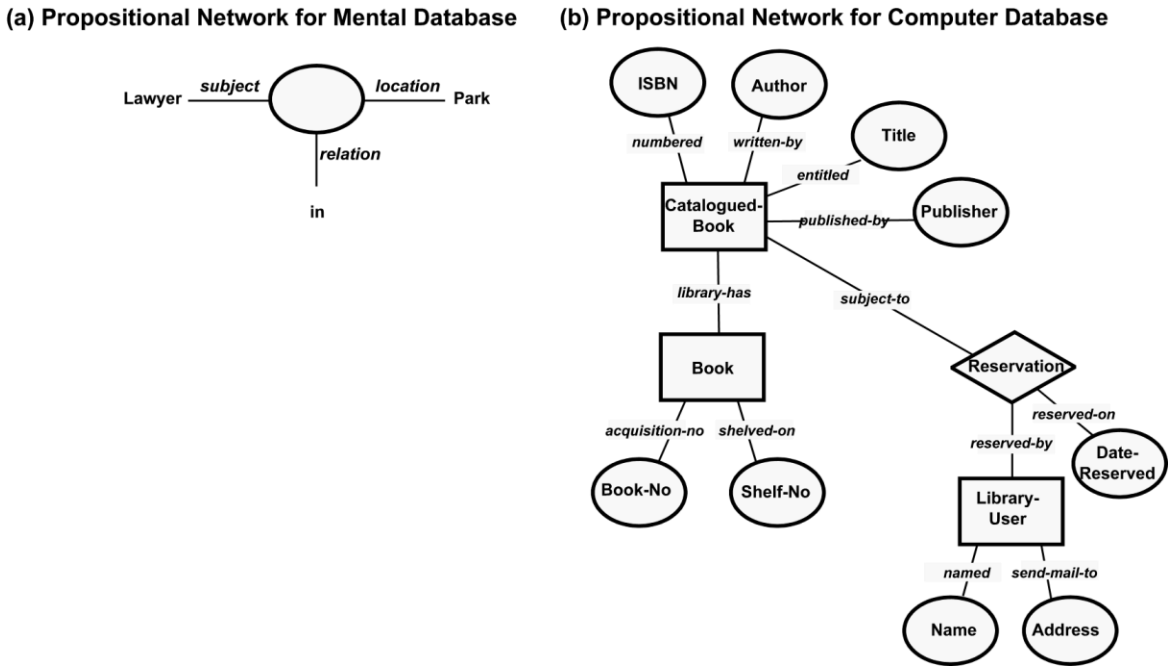
Yet the CLRU still had the fourth of its four main themes - lattice theory - in reserve. The main worker here was a colleague of Masterman, Arthur Frederick Parker-Rhodes, and his basic proposal was that the mathematics of networks (lattice theory arose a branch of Boolean algebra) was also the mathematics of language. Drawing on Parker-Rhodes' ideas, Masterman re-analysed the linear array of meanings laid out in *Roget's Thesaurus* as the nodes of a large knowledge network.

Of course, the idea that the mind might organise its available knowledge as a vast interlocking network of concepts is not new. One of today's leading experts on the technical structuring of knowledge, ex-IBM man and knowledge consultant John F. Sowa, for example, dates the idea to the Greek philosopher Aristotle. Aristotle's early thoughts on the hierarchical nature of definition survived in the network diagram drawn up by Porphyry in the third century CE, a device now known as the **"Tree of Porphyry"**. The idea was then reborn in the **"Associationist"** tradition of Western philosophy (led by David Hartley in the mid-18th century), and was also used by Jung as the basis of his "association of ideas" projective test [the famous "tell me the first word that comes into your head" test]. **We shall be returning to this issue in Part 5, but mention it here because Richens - geneticist, self-taught database designer, and now linguistic philosopher - has recently been identified as the man who coined the term "semantic net", and semantic networks are a very important topic .....**

> **Key Innovation - Semantic Networks:** A semantic network is "a graphic notation for representing knowledge in patterns of interconnected nodes and arcs [..... and] what is common to all semantic networks is a declarative graphic representation that can be used either to represent knowledge or to support automated systems for reasoning about knowledge" Sowa (2003 online). As we shall be seeing in Part 6, semantic networks exist within the computer world in the form of the network-structured (aka "codasyl") database. << **AUTHOR'S NOTE: It is worth noting at this juncture (a) that the concept of the semantic network has made some impact on the psycholinguistic side of cognitive science, but (b) that research has been fragmented between linguistics, psycholinguistics, and mainstream cognitive theory, and (c) that one major perspective has been strangely overlooked, namely the portfolio of diagramming and analysis tools used to build data networks for use in computer databases. The most ambitious attempt so far had been by Cycorp's Douglas B. Lenat, whose Cyc project has been under development since 1984, but more on that in Part 5. Our own preference is for the "Bachman Diagram" approach, but there is nevertheless considerable explanatory value in both Anderson's and Lenat's approach. For his part, Sowa explicitly equates his Definitional Network with Tulving's (1972) "semantic" type of LTM, and his Associational Network with the "episodic" type. However, to the best of our knowledge, there has not been a single attempt to use codasyl-type database in AI simulations. The full evaluation will not be available until 2004. >>**

The main psychological input to the semantic network debate has been from Anderson (1983) and his **"propositional network"**, as reproduced at Figure 2(a). However, very similar diagrams are heavily used within the IT industry to formalise the layout of large computer databases, and a specimen of one of these is reproduced at Figure 2(b). Curiously enough, attempts to import database concepts into psychological explanation are few and far between.

Figure 2 - Propositional Networks, (a) Biological and (b) Computer: In diagram (a) we see how a simple network diagram can be used to analyse something as philosophically complex as a unit of truth. As is common with this sort of diagram, the individual concepts are shown as **concept nodes** in the network, whilst the *real* knowledge exists in the form of **propositional nodes** (the oval ones). The propositional nodes are vital because they "encode" facts (ie. propositional truths) about the concepts they link. The oval symbol on this occasion serves to link the otherwise separate conceptual nodes <LAWYER> and <PARK>, and in so doing it identifies a number of essential linguistic properties as shown in italics on the line linkages. Thus <LAWYER> is the *subject* of the proposition, <PARK> is the *location*, and <IN> specifies the *relation* between the two. In diagram (b), we see similar principles at work in a computer data model. This particular modelling convention uses four types of symbol. The rectangles and diamonds denote different types of concept, or *entity*, nodes, the ovals denote the attributes, or properties, of those entities, and the lines denote the relationships between entities and other entities or between entities and attributes. Again, the essence of the relationships between entities is marked on the line linkages. Properly implemented by a Database Management System, every occurrence of every entity type, attribute, and relationship can be safely stored in rapid access computer filestore, and retrieved in short order. **This is precisely the sort of technology which would have allowed Reifler [see Section 4.1] to squeeze his 15 drum dictionaries onto a single storage resource.**



 Previously in Smith (1996). Redrawn from black-and-white originals (a) in Anderson (1983, p26), and (b) in Oxborrow (1989, p36).

For a rich selection of other diagrams and approaches, see Sowa (2003 online). In fact, Sowa identifies no less than six subtypes of network, as follows .....

(1) Definitional Networks: These are networks which define a concept in terms of its similarity to other concepts. Sowa calls this a "subtype", or an "is-a" relation, and places its value in the fact that the properties of the supertype are automatically "inherited" by the subtype. [*Our Example:* <ROBIN> is a <BIRD, SMALL, GARDEN, NON-MIGRATORY>.]

(2) Assertional Networks: These are networks which assert propositional truths. [*Our Example:* <DAVID> <IS EATING> <CAKE> or <WHEN HE WAS 7> <DAVID> <VISITED> <SEATTLE>.]

(3) Implicational Networks: These are networks which "use implication as the primary relation for connecting nodes". [*Our Example:* <THE ROCK IS AT THE TOP OF A SLOPE> which implies ,..... AND SO IS LIKELY TO ROLL IF PUSHED>.]

(4) Executable Networks: These are networks which have mechanisms for actively locating content according to a prescribed search mask. [*Our Example:* A network which could respond appropriately to an instruction in the form LIST ALL CONTENT ASSOCIATED WITH THE TARGET <SLOPE>.]

(5) Learning Networks: These are networks which can extend themselves, either by adding new nodes or new arcs between existing nodes, or by modifying the association strengths, or **"weights"**, of arcs. [*Our Example:* A face recognition network which can learn to recognise new faces (new nodes), or attach new attributes to existing ones (new arcs).]

(6) Hybrid Networks: These are networks which combine two or more of the preceding types.

4.3 Mechanised Game Playing

In 1769, the inventor-showman Wolfgang von Kempelen built a chess-playing automaton which he demonstrated at theatres and fairs where it proved capable of beating most contestants. Unfortunately, von Kempelen was pulling a fast one, and the functional CPU turned out to be the world's first minicomputer - a human dwarf secreted within the device. The idea remained appealing, however, and one of the pastimes of the first computer programmers was to investigate whether computers could be programmed to play games such as draughts [US = checkers], noughts and crosses [US = tic-tac-toe], or chess. The pioneer thinker in this area seems to have been Germany's Konrad Zuse [reminder], inventor of *Plankalkül*, the world's first higher-level programming language. In the period 1942-1945, Zuse sketched out designs for a chess playing program for his Z4 computer, but it never got written and the design was only published in 1972. The formal glory goes instead to another of Warren Weaver's protegés, Claude Shannon of the Bell Telephone Company. In a manuscript dated October 1948, Shannon analysed the problems which were likely to be encountered when getting a computer to play chess (subsequently published as Shannon, 1950/1993). He saw this as one of many ways in which computers could move in the direction of processing "general principles, something of the nature of judgement, and considerable trial and error, rather than a strict, unalterable computing process" (Shannon, 1950/1993, p637). He saw "the chess machine" as an ideal problem with which to start the quest for what he called "mechanised thinking", and discussed methods of codifying and then evaluating proposed moves. Not surprisingly, Shannon's papers inspired some ground-breaking computer programming in the early 1950s. The following bullets come from a number of sources, including Copeland (2000) .....

- The Bletchley Park veteran, Donald Davies, published an article on the mechanisation of board games (Davies, 1950), and coded a simple noughts-and-crosses program for the NPL's Pilot ACE.

- In May 1951, the aforementioned Christopher Strachey [see Section 2.3], wartime radar boffin turned schoolteacher, scrounged time as a technical hobbyist on the NPL's Pilot ACE, and wrote what has been acclaimed as the world's first AI program - a draughts program. Unfortunately, the code had a few bugs in it, and Strachey did not get it working tidily until the following summer on Manchester's Ferranti Mark 1 (Copeland 2000).

- In November 1951, Manchester University's Dietrich Prinz used Turing's newly produced "Programmer's Handbook" to produce the first experimental chess program, also on the Ferranti Mark I (Copeland).

- IBM's Arthur Samuels heard of Strachey's program at a 1952 conference, and adapted it for the IBM 701. He then had to spend the next ten years improving it, eventually adding the rudimentary ability to learn from experience, whereupon the code turned rather churlishly on its creator and started to beat him (Samuels, 1959).

- IBM's Alex Bernstein wrote the first reasonably competent chess program on an IBM 704 in 1957.

- With even more improvements, Samuels' 1959 program beat a human draughts champion in 1962. The program is historically significant because it used **"heuristics"** rather than **"algorithms"** to evaluate and score possible moves.

> **Key Innovation - Algorithms vs Heuristics:** The word "algorithm" derives from the Arabic "al Khowarasmi", the popular name of the ninth century Arab mathematician Abu Jafar Mohammed Ben Musa, and refers to a sequence of formal actions by which a given problem can be systematically resolved. It is "a systematic list of instructions for accomplishing some task" (Wikipedia, 2003 [this phrase no longer appears in March 2024]). Thus computer programs are algorithms. The word "heuristic" derives from the Greek root ευρισκειν - "to find" (thus being related to "Eureka" = "found it!"), and, strictly speaking, refers to anything which promotes an act of discovery. However, where algorithms consist of specific actions, heuristics consist of rule-action combinations, so with an heuristic you do not know in advance what you are going to do - you have to wait and see what circumstances trigger what rules. The beauty of the if-then approach is that it can save you a lot of time. When playing chess, for example, it is rarely a good idea to exchange a queen for a pawn, and so an entire branch of the exhaustive search tree can be discounted at a stroke. Insofar as the word is used in game theory, the point about heuristics is that they attempt rapid penetration of a problem rather than exhaustive analysis, by virtue of which they are often referred to as **"rules of thumb"**. Ultimately, of course, both algorithms and heuristics are only as good as their designers. If either is not particularly powerful, then the quality of the penetration will be limited.

4.4 Man-Machine Conversation

The fourth line of enquiry was a more philosophical one, and looked at what needed to be done to turn a cleverly programmed calculator into something qualitatively more than a cleverly programmed calculator. What, for example, might a cash register have to do to be accepted as containing a mind; something more than just a thing which can be switched on and off on demand? What might your laptop have to do to be accepted as possessed of some kind of vital spark? This question was first put forward in a paper from Mr GPC himself, Alan Turing, who, when not codebreaking, was "really quite obsessed with knowing how the human brain worked and the possible correspondence with what he was doing on computers" (Newman, 1994, p12). Taking time out from his work on the Manchester Mark 1, Turing wrote a paper entitled "Computing machinery and intelligence" in which he suggested that the question "can machines think?" was philosophically unsafe, due to problems agreeing the meaning of the word think (Turing, 1950).

Of course, Turing had long believed that all but the most obscure of problems could be solved by a sequence of simple actions, and already for a decade programmers had been proving him right. He now recommended that this sequence of actions be thought of as having an existence and meaning in itself: a program, once devised, was a solution *in prospect*, as it were, and simply "awaited" a machine to execute it. Turing then brought these two propositions together by arguing that the mind would not only prove one day to be "programmable", but that the eventual program would be "implementable" on a machine. And once you could do that, he said, you would have a machine which would be indistinguishable from a human; that is to say, a machine which could think. As to how you would test this indistinguishability, he proposed objectively establishing whether the machine, so programmed, could perform as successfully as a human in fooling an interrogator in an "imitation game", in which a man (A) and a woman (B) have to fool (C) as to which is the man and which is the woman. The knowledge pertaining to (A) and (B) is accumulated in the mind of (C) by asking questions, the only restriction being that the answers to those questions should be typewritten so as to prevent vocal clues being given.

Turing's imitation game evolved somewhat over the years, and in its later form became popularly known as the "**Turing Test**". This runs as follows: if a human in room A were to communicate via keyboard and screen with an entity in room B which might be a human but which might also be a computer trying to appear human, then the definition of "humanness" would rest on whether the real human could tell the difference or not after five minutes of questioning. Serious academic attempts to "pass the Turing Test" began in the 1960s, and we shall be looking at some of these in Part 5. Reader (1969) even sketches out the processing modules which would be necessary to build such a machine [check it out], but the layout he

proposes turns out to be yet another instance of the cognitive control hierarchy, indistinguishable in essence from Lichtheim's (1885) "House" diagram, and surpassed in supporting detail by Frank's (1963) "Organogramm".

4.5 "Neurodal" Networks and Machine Learning

The fifth line of enquiry was pitched more at the hardware side of things, and can be traced to a number of seminal papers from the 1940s, led by the American neuropsychologist Warren S. McCulloch (1898-1968). The first of these was McCulloch and Pitts (1943), an attempt at mathematically modelling the workings of biological neurons. The authors called their artificial neurons **"neurodes"**, and each was a simple arrangement of electronic components designed to do two main things, namely (a) to output a signal similar to the bursts of action potentials output by biological neurons, and (b) to ensure that this output was *modulatable* - sensitive to what other neurodes in the vicinity were doing. In this way, quite simple combinations of neurodes were capable of performing the logical operations AND, OR, and NOT to much the same end effect as the electronic circuitry then being developed for use in computers. Pitts and McCulloch (1947) then went an important step further, by discussing what would happen if you took many such neurodes and wired them together. Specifically, they proposed that it would be possible to end up with some sort of artificial pattern recognition system. And McCulloch and Pfeiffer (1949) considered how binary principles might exist within biology. They portrayed the nervous system "as if it were a wiring diagram and the physiology of the neuron as if it were a component relay of a computing machine" (p369). Their aim was then to "set up working hypotheses as to the circuit action of the central nervous system as a guide to further investigation of the function of its parts, and as a scheme for understanding how we know, think and behave" [*ibid*.]. This is part of their observations .....

> "Judging by the speed of response, neurons fall into the middle range of man-made relays. They are about a thousandth as fast as vacuum tubes, about as fast as thyrotrons, but faster than electromechanical and mechanical devices. Because they are much smaller than electronic valves, though the voltage gradients are about the same, they take lower voltages and less energy. A computer with as many vacuum tubes as a man has neurons in his head would need the Pentagon to house it, Niagara's power to run it, and Niagara's water to cool it. The largest existing calculating machine has more relays than an ant but fewer than a flatworm. Neurons, cheap and plentiful, are also multigridded, the largest having upon them thousands of terminations, so that they resemble transistors in which the configuration of electrodes determines the gating of signals." (McCulloch and Pfeiffer, 1949, p370)

> **HISTORICAL ASIDE:** In fact, the idea of nerve net computation is far older than the attempts to construct it. One of the first to suggest that learning was accompanied by the formation of new synaptic connections was the Aberdeen logician, Alexander Bain (1818-1903). Inspired by the work of Lionel S. Beale (1828-1906), pioneer in medical microscopy, and specifically by Beale (1864), Bain tried to find a likely neuroanatomical substrate for the sort of semantic networks popular with psychologists of the "Associationist" persuasion. The process relied on the "contiguity" - that is to say, closeness together in time or space - of the elements to be associated. "Contiguity," he wrote, "joins together things that occur together, or that are, by any circumstance, presented to the mind at the same time ....." (Bain, 1879, p127). He set out his technical proposals in his 1873 book "Mind and Body", from which the following clearly indicates his line of argument: "For every act of memory [.....] there is a specific grouping, or co-ordination, of sensations and movements, by virtue of specific growths in cell junctions" (Bain, 1873, p91; cited in Wilkes and Wade, 1997). Bain then analysed differently interconnected clusters of neurons using simple abstract circuit diagrams (reviewed by Wilkes and Wade (1997). The next major historical figure was Santiago Ramon y Cajal (1852-1934) (e.g., Ramon y Cajal, 1911), whose general idea was that plasticity in neuron-to-neuron connections - what are now known as "synapses" - made it possible for neural pathways to connect themselves up *on demand* - that is to say, as learning took place. Pathways could thus appear where previously just disconnected neurons had existed. Moreover, the biological act of creating those pathways underpinned the psychological act of memorising the experience in question. Some years later, Rafael Lorente de Nº (1902-1990) extended the debate by microscopically analysing the layout of the synaptic "buttons" on neural cell bodies (Lorente de No, 1938). Working at the limits of magnification of the microscopes then available, he found many hundreds of synaptic buttons dotted about the neural membrane, thus reinforcing suspicions as to their role in neural circuitry. Subsequent studies have confirmed this view, and with today's very powerful microscopes veritable forests of synaptic buttons can be seen, and not just on the neural soma but out on the dendritic trees as well. Another of Lorente de No's contributions was to show how *interneurons* could be used to feed excitation back into a given neural circuit, thus keeping that circuit active long

after the original source of excitation had been removed. This arrangement is commonly referred to as a **reverberating circuit**. However, the most influential exposition of neuronal net theory came from Donald O. Hebb (1904-1985). In his 1949 book, "The Organisation of Behaviour", Hebb described the interlinking of neurons as creating what he called a **cell assembly**, which he described thus: "..... a diffuse structure comprising cells in the cortex and diencephalon (and also, perhaps, in the basal ganglia of the cerebrum), capable of acting briefly as a closed system." (Hebb, 1949, p. *xix*). Hebb's ideas of how cell assemblies form and function can be gathered from the following: "The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other" (*Ibid*, p70). The idea of cells repeatedly assisting each other's firing is particularly well brought out in what has since come to be known as **"Hebb's Rule"**.

The next step was to go ahead and wire a few neurodes together, and in 1951 Marvin Minsky did just that (Minsky, 1994). As part of his postgraduate researches in the mathematics department at Princeton, and in conjunction with Dean Edmonds, he built SNARC (= "Stochastic Neural-Analog Reinforcement Computer"), the first machine to display learning behaviour, thanks to an inbuilt reinforcement-sensitive 40-neuron **"neural net"**. This was followed a year later by William Ross Ashby (1903 - 1972)'s "Design for a Brain" (Ashby, 1952). Ashby was the cyberneticist who had popularised the concept of **"homeostasis"** in biological systems. He therefore took a more macroscopic view of neural architecture than had McCulloch or Minsky, that is to say, he was more concerned with what the nervous system was trying to achieve as an organic whole with interrelating modules, rather than with what individual neurons were up to. His work with biological homeostasis had convinced him that many biological control phenomena were simple negative feedback systems, save that they had biological sensors and comparators, and during the 1940s he had experimented with a simple machine called **"the homeostat"**, which was sensitive to environmental events and could initiate corrective behaviours to neutralise any "perturbation" in their stable settings. Angyan (1959) reports on a more advanced machine called *Machina Reproducatrix*, fitted with enough sensors and motors to give it a light-seeking **"tropism"**, however we need to talk about this in Part 7, and so will leave the details until then.

ASIDE: The term "tropism"was coined by Loeb (1888, 1890) to describe the wholly involuntary behaviour of organisms lacking a nervous system. Just as plants can turn towards the sun using non-nervous mechanisms, so, too, is much simple animal behaviour unwilled. To be classified as a tropism, the relationship between the stimulus and the response must be rigid and resulting directly from the action of some external energy source. Tropisms can be subcategorised by energy type and direction of response as follows: *Geotropisms: positive* - turn towards the pull of gravity; *negative* - turn away from the pull of gravity, *Heliotropisms (or phototropisms): positive*: turn towards the light; *negative* - turn away from the light, *Thermotropisms: positive*: turn towards heat; *negative* - turn away from heat, and *Rheotropisms: positive*: turn upstream (in air or water); *negative* - turn downstream. In animals with nervous systems, this type of behaviour is provided by **reflexes**. For a longer introduction to the various types of innate and learned animal behaviour, see Section 1 of our e-paper "Communication and the Naked Ape".

In fact, 1952 was a busy year, because it also saw Claude Shannon following up his chess machine with the blueprints for a maze-solving mechanical mouse (Shannon, 1952/1993), and Anthony G. Oettinger writing a program called "Shopper", the first program (as opposed to learning hardware) capable of learning from experience (Oettinger, 1952). Copeland (2000) describes Shopper as operating in a virtual world containing a mall of eight shops stocking a number of items. Shopper would be sent off to purchase a particular item, and would have to visit shops at random until successful. The learning lay in the fact that Shopper would memorise some of the items stocked in the unsuccessful visits. Subsequent shopping expeditions, either (a) for repeat purchases of the first item, or (b) for first time purchases of one of the memorised items, would be right first time, thus showing the value of short term memory resources in supporting adaptive behaviour.

We must also mention Albert M. Uttley, whom we first met in Section 2.2 as the TRE boffin who designed TREAC, the first British parallel processing computer. Over the years, Uttley gradually extended his interests to include biological as well as workbench electronics. Like McCulloch, he was particularly fascinated by the logic by which neurons were interconnected, and was one of the first to describe the sort

of statistical connection rules which might control the biological learning process. In Uttley (1955), he derived mathematical formulae to describe such things as dendrite distribution and axonal connections.

However, the most famous of the early machines was built by Frank Rosenblatt in the mid-1950s, and called a **"perceptron"**. The essence of Rosenblatt's design was that the neurodes were arranged into two banks, or "layers". One of these was connected to an artificial eye and called an **input layer**, and the other was connected to the chosen output mechanism (usually an array of flashing lights) and called an **output layer**. Each point in the input layer was wired to each point in the output layer, and the effective strength of these connections was specifically designed to be varied. Learning was then a process of changing the strength of the connections - a process known as **weighting**. As we shall be seeing in Part 5, McCulloch's neurodes and Rosenblatt's perceptron formed the basis for one of modern science's most active and exciting research areas, namely **"Connectionism"**.

The point about perceptrons is that the weighting process delivered learning by experience. There was no pre-written program telling the hardware what to do - rather the machine worked it all out for itself. Perceptrons took the academic world by storm, therefore, especially where there were problems which were proving beyond the programmers in the first place - **such as most of higher cognition**. Thus .....

> "It is not easy to say precisely what is meant by learning or what corresponds to it in a machine. [.....] The machine is able to learn up to a point, but once that point has been passed it is helpless. Here it differs from a human being, who is able to learn and to go on learning. [//] A program which would enable a machine not only to learn but also to go on learning, and to adapt itself to each new situation as it arose, I will call a 'generalised' learning program. If such a program could be constructed it would presumably consist of a relatively small initial nucleus of orders, and would adapt and extend itself as circumstances required, replacing its subroutines and switching sequences by others of a more general kind, and creating new subroutines to perform fresh functions. There does not appear to be any reason for believing that the construction of such a program is inherently impossible [.....] Although limitations of speed and storage capacity would undoubtedly soon make themselves felt, **it is not these so much as shortage of ideas which is at present holding up progress**." (Wilkes, 1956, pp291-293; emphasis added)

4.6 Robotics

The defining essence of a robot is that it carries out work on behalf of its master. Strictly speaking, therefore, a robot needs bear no physical resemblance to any biological form (although, for reasons which are still being looked into, we seem to prefer that they do). As such, the broad concept can be traced back several thousand years. Even in 1950, the name "robot" was already at least 30 years old, being usually credited to Karel Capek's 1921 play "Rossum's Universal Robots", and Isaac Asimov's "Laws of Robotics" were 10 years old. As for the technology itself, it was at least 40 years old, and had been forged, as always, on a war anvil .....

> **HISTORICAL ASIDE - MILITARY ROBOTICS:** The military have always proved generous sponsors for companies specialising in technological innovation, and we have already described elsewhere [reminder] how the Sperry Gyroscope Company grew into a major international corporation on the back of mechanical and electromechanical naval control systems. Sperry were also quick to see the same opportunity for fully automated aerial weapon systems, and as early as 1913 were using gyroscopic stabilisers as part of a radio-controlled aircraft (Pearson, 2003). The demands of the First World War extended the science of military robotics into the control of aerial torpedoes and the design of better bombsights and perhaps the best known products of this early research were the World War Two German V-1 and V-2 pilotless rocket systems, which both had autopilots and onboard guidance computers capable of steering them several hundred miles. Much less spectacular, but deadly nonetheless, were the Ruhrstahl SD1400 "Fritz X" and the **Henschel Hs293A** "stand-off" bombs [Wikipedia briefing]. These were "Lenk-", that is to say guided, "bomben", and were both small winged glider bombs designed for anti-shipping use. The SD1400 was heavy and free-falling, with an armour-piercing capability, whilst the Hs293 was smaller and rocket-assisted, but lacked the armour-piercing capability. They were released from their mother aircraft while still a number of miles from the target, and steered on their short one-way flight by (initially) radio signals generated by a

small joystick. The *Lenkbomben* were developed in the period 1940-1943, and when ready for field deployment were issued to specially equipped Luftwaffe squadrons codenamed KG40 and KG100. History's first successful guided missile attack on a surface ship then took place on 27th August 1943 in the Bay of Biscay, when the corvette *HMS Egret* was sunk and the Canadian destroyer *Athabascan* damaged by Hs293s launched from KG100 aircraft. The following month, the cruisers *HMS Uganda* and *USS Savannah*, together with the battleship *HMS Warspite*, were seriously damaged by Fritz Xs while supporting the Salerno landings, and the Italian battleship *Roma* was sunk by one or two hits [accounts differ] from Fritz Xs while attempting to defect to the Allies under the terms of the Italian surrender. The battleship *Italia* was also hit on this occasion, and her escort, the battleship *HMS Valiant*, damaged by a near miss, but both managed to make their way to Malta. In November 1943, the British troopship *Rohna* was also sunk en route from Oran to Bombay carrying American troops, and this incident, in terms of lives lost (1138 dead), was actually a worse disaster than the attack on Pearl Harbour (Cassanello, 2003). Still in the Mediterranean, the cruiser *HMS Spartan* was then sunk off the Anzio beachhead in January 1944. The Allies responded very quickly to this new age threat, analysing the German radio control frequencies, and adding radio countermeasure systems (="RCM", or "jamming" equipment) to their front-line naval assets as fast as the equipment could be built and installed. This upgrade extended to a number of landing ships being fitted out as Fighter Direction Tenders (FDTs) in readiness for the 1944 invasion. These vessels were selected because they would be stationed off the landing beaches in a command and coordination role alongside existing headquarters ships such as *HMS Largs*, *Hilary*, and *Bulolo*. As a result, when the invasion went ahead in June 1944 the Allied fleet emerged relatively unscathed. For example, we find the following entry in the log of the battleship *USS Texas*: "the day [7th June] started with a detection of several anti-ship missile radio signals and subsequent jamming". KG100 was active for a few days off the Normandy beaches, and scored at least one sinking, that of *HMS Lawford* on 8th June [divers examined the wreck in 2002, and the nature of the damage confirms that a glider bomb was the most likely culprit]. Off Arromanches, they also hit, but did not sink, *HMS Bulolo*, the Force G (Gold Beach) headquarters ship, striking her - fortunately for the ship as a whole, if not those actually up there - high on the superstructure [the specialist German bomb-aimers were trained to aim at their target's waterline for maximum effect, but even under ideal conditions were lucky to hit within a five metre radius]. The Germans were apparently slow to respond to Allied RCM with "counter-countermeasures", but eventually replaced the radio controlled Hs293A with the wire-guided, and therefore RCM-immune, Hs293B. However, as the war moved progressively inland the opportunity to field this improved system was lost, and the specialist squadrons were disbanded. In all, 319 Fritz X and Hs293 missiles are reported to have been launched in action, of which roughly 100 scored hits. 40 warships and 30 merchant ships were damaged or sunk [source]. In addition to the ships named above, German war historians also claim the destroyers HMS *Jervis*, *Inglefield*, *Boadicea*, and *Intrepid* as sunk, and the cruiser *USS Philadelphia* damaged [confirmation]. In addition, one of the aforementioned FDTs, *FDT 216*, is on official record as having been sunk by an air-launched "torpedo" on 7th July 1944, but it is unclear from the sources available to us whether this was of the guided or free-running type. The next major advance with tactical rocketry was the **"homing"** missile. These were developed typically for use against either aircraft or submarines, and the functional principle is that the missile **"locks on"** to an external energy source (typically the engine exhaust heat of an aircraft or the propeller noise of a submarine) and then twists and turns as much as is necessary to keep that source in its sights (allowing for deflection as necessary). This type of system reached a state of some sophistication in air-to-air missiles such as Sidewinder. Laser guided bombs were developed during the 1980s and saw action during the 1991 Iraq War. They have since been partly replaced by JDAM smart bombs, where the projectile homes on coordinates programmed into a highly flexible tactical command system, and conveyed to the weapon after its release by signals from a satellite. JDAMs are a few metres less accurate than laser-guided munitions, but are cheaper, can be used in larger numbers at a time, are **"fire and forget"**, and are not affected by cloud, darkness, or smoke.

So we now have something of a pattern emerging. With conventional munitions, the projectiles are totally "dumb" - mere lumps of iron, devoid of any form of intelligence; no more robots than spears or musket balls are robots. With the operator-guided weapons, the projectiles are responsive, but still dumb - they make no decisions of their own, but instead receive input (from the remote joystick) and generate output (to their steering mechanism). And with the early pilotless rocket systems, all the control systems were pre-set before launch, and the missile followed the programmed trajectory as closely as it was physically able. This mixture of technologies means that in the typical air-ground combat engagement of modern warfare we often pitch significantly different control systems against each other. We might, for example, have a tactical dual between a ground unit armed with a fire and forget shoulder-launched surface-to-air missile such as a

Stinger, and a helicopter armed with a laser-spotted air-to-ground round such as **Hellfire** [Wikipedia briefing], or between a city's SAM batteries (radar-guided, passive or active), and a B-2 loaded with JDAMs. The common denominator, of course, is that all forms are ultimately placed on target by a human, even if the act of aiming took place at a keyboard many thousands of miles away and many weeks beforehand. The intelligence (and we use the word cautiously) is in the mind of the operator whose eye is on the target (literally or figuratively) and whose hand is on the joystick. So the machines are not really intelligent at all. Even the JDAM has no "flexible control over the sequence of its operations" (Berkeley, 1949, p5), nor is it likely to acquire any for a good few years yet. In fact, Carnegie Mellon University's Hans Moravec [more on whom in Part 5] doubts that such **"hazardous duty" systems** should be called robots at all - he prefers the title "autonomous machines", because "they have the bodies of robots, but not the brains".

> **ASIDE:** We use the word "intelligence" cautiously, because it is one of those words which can mean whatever the speaker at the time wants it to mean. It can mean advanced philosophy to some, contemplative planning to others, or just the ability to hold a target in the crosshairs, and it is precisely this vagueness which has kept the Turing Test [see Section 4.4] so relevant all these years. The focus of our attention must therefore turn away from robotic mechanisms and design architectures, and look instead directly at the intelligence we are trying to inject into our robotic brain .....

4.7 Machine Problem Solving

We turn, therefore, to the highest of all forms of guidance, namely reasoned action, and the nearest AI got to this in the 1950s was in its experiments with machine problem solving. The most successful of these was the collaboration between the Rand Corporation's Allen Newell and J. Clifford Shaw, and Carnegie Mellon University's Herbert A. Simon (1916-2001) to investigate how machines might profitably mimic (and potentially help explain) human mathematical problem solving. One of their programs is still regarded as defining the science .....

> **The "Logic Theorist":** This 1956 program was designed to test the truth of specific mathematical theorems. It did this by allowing the programmer to input five basic propositions (called "axioms"), against which were applied a number of **"rules of inference"**. This enabled further logical propositions to be derived, by a process based upon the human process of formal mathematical argument, and as each derived sentence is generated it can be checked to see if it matches the theorem under test. If it does, then the audit trail of successive inferences by which it was derived automatically constitutes the mathematical proof required. If it does not, then the program advances to the next permutation. Unfortunately, the problem with this approach is basically one of time, because ideally you would have to explore every combination of axiom and rule through every possible iteration (a brute force computational approach sometimes known as the "British Museum algorithm"). Remembering the distinction we raised in Section 4.3, this is therefore an excellent example of problem solving by algorithm rather than by heuristic.

Logic Theorist was the central exhibit in 1956 at a conference on AI organised at Dartmouth College by John McCarthy, one of their mathematics lecturers .....

> **BIOGRAPHICAL ASIDE - JOHN McCARTHY:** McCarthy had been following developments in machine cognition very closely, and had already coined the general descriptor "artificial intelligence" in 1955. He would go on in 1958 to develop the LISP programming language, and in 1959 wrote an influential paper entitled "Programs with Common Sense" in which he defined machine common sense as follows: "..... a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows." (McCarthy, 1959). As to how this might be achieved, reference should be made to the five system features proposed in the original paper, if interested. However, the fifth of these features in particular has turned out to be remarkably prescient .....
>
> > "5. The system must be able to create subroutines which can be included in procedures as units. The learning of subroutines is complicated by the fact that the effect of a subroutine is not usually good or bad in itself. Therefore, the mechanism that selects subroutines should have concepts of interesting or powerful subroutine whose application may be good under suitable conditions." (*Ibid*., p3.)

> McCarthy's 1959 paper is also noteworthy as an early example of the ill feeling and interprofessional disdain which can sometimes be seen between linguistics professors and the engineers who gravitate so frequently into AI. "Dr McCarthy's paper," wrote Bar-Hillel in the peer review section of the cited paper, "belongs in the Journal of Half-Baked Ideas [and] before he goes on to bake his ideas fully, it might be well to give him some advice ...." (*Ibid*., p11). Be that as it may, McCarthy had pointed unerringly at the central problem of any hierarchical control system, namely where to place (and how to build) the bit that makes the all-important judgements of interesting, powerful, etc. McCarthy is currently professor of computer science at Stanford University, and is well on the way to making his lifetime's work available on the Internet.

Another program, the **"Geometry Theorem Prover"** was reportedly inspired by a tentative program structure drawn up in 1956 by MIT's Marvin Minsky. This was a sequence of instructions capable of proving a particular geometrical problem. Minsky passed his notes to a young IBM researcher named Herbert Gelernter, who then implemented it on an IBM 704. The resulting software was capable of analysing a geometrical diagram for its constituent propositions, and of then attempting to derive further proofs in much the same way as Newell, Shaw, and Simons' Logic Theorist.

Newell, Shaw, and Simon struck back in 1957 with the **"General Problem Solver"**, a program expressly designed to simulate the processes of human problem solving. Here is Copeland's (2002/2003 online) summary .....

> "..... work continued on the project for about a decade. General Problem Solver could solve an impressive variety of puzzles, for example the 'missionaries and cannibals' problem: How are a party of three missionaries and three cannibals to cross a river in a small boat that will take no more than two at a time, without the missionaries on either bank becoming outnumbered by cannibals? GPS would search for a solution in a trial-and-error fashion, under the guidance of heuristics supplied by the programmers. One criticism of GPS, and other programs that lack learning, is that the program's 'intelligence' is entirely second-hand ....."

The period under consideration ended with a paper by Newell, Shaw, and Simon (1958) on what they called "the problem of complexity". In it, they surveyed the attempts to date (including their own) and the heuristics used, and concluded that the complexity of heuristic programs was beyond the power of the programming languages then available, and appealed for more powerful languages. In Part 5, we shall be looking at what came along.

4.8 Artificial Consciousness

Finally, there has been a line of enquiry which has gone one important step past machine intelligence, looking for the secret of biological consciousness itself. Unfortunately, this remains an area where there is no hard data, little simulation, and a lot of speculation. As with intelligence, we simply do not know what consciousness is in philosophical terms, and so we need to be ready to come across it by accident in any of the various sub-areas of AI. It might be in the interlinguas of the verbal mind (Section 4.1), or in the data networks of the knowing mind (Section 4.2), or in the game playing heuristics of the game-playing mind (Section 4.3), or in the ability to pass the Turing Test (Section 4.4), or in some lucky artifact of the neural network (Section 4.5), or in the onboard intelligence of a third generation robot (Section 4.6), or even in the reflection which characterises good problem solving (Section 4.7). In short, we do not know what we are looking for, and that always makes life difficult for programmers [time, in fact, to call in the systems analysts ;-)]. We therefore close with quotable quotes on this very point by two of generation zero's greatest .....

> "We know the basic active organs of the nervous system (the nerve cells). There is every reason to believe that a very large-capacity memory is associated with this system. We do most emphatically *not* know what type of physical entities are the basic components of the memory in question." (von Neumann, 1958, p68; emphasis original)

> "There is no difficulty in programming a universal machine to perform any set of operations **when once the rules for performing them have been stated explicitly**; this is true however complex the rules may be and however many alternative procedures of a conditional type are specified" (Wilkes, 1956, p291; emphasis added.).

5 - Still to Come

That concludes our review of the history of first generation computing technology during the period 1951 to 1958. In Part 5, we bring the review up to the present day, and identify several more key technical innovations, not least Database Management Systems, in Part 6 we look at computer memory in greater detail, especially as used in the COBOL programming language, the IDMS DBMS, and the TPMS TP monitor, and in Part 7 we undertake a comparative review of how well the full richness of computer memory subtypes had been incorporated into psychological theory.

6 - References

[Previous**][**Next**][**Home**]**