Course Handout - Short-Term Memory Subtypes in Computing and Artificial Intelligence

Part 5 - A Brief History of Computing Technology, 1959 to Date

Copyright Notice: **This material was written and published in Wales by Derek J. Smith (Chartered Engineer). It forms part of a multifile e-learning resource, and subject only to acknowledging Derek J. Smith's rights under international copyright law to be identified as author may be freely downloaded and printed off in single complete copies solely for the purposes of private study and/or review. Commercial exploitation rights are reserved. The remote hyperlinks have been selected for the academic appropriacy of their contents; they were free of offensive and litigious content when selected, and will be periodically checked to have remained so. Copyright © 2003-2024, Derek J. Smith.**

First published online 09:59 GMT 19th January 2003. This version [refresh hyperlinks after 21 years and remount as a .pdf] dated 10:00 GMT 5th March 2024. (The refresh does NOT, however, change the narrative present from 2003 to 2024 since no additional research has been incorporated.)

This is the fifth 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 further introduced the vocabulary necessary for Parts 6 and 7. To go back to Part 4, click here**.**

Part 5: An optional introductory and reference resource on the history of computing technology from 1959 to date. This material follows below and further introduces the vocabulary necessary for Parts 6 and 7. The main sections are:

- **1 - Mainframe Computing 1959-1970 - Second and Third Generation Technology**
- **2 - Microcomputing, 1968 to Date**

- **3 - Mainframe Computing, 1971 to Date - Fourth and Fifth Generation Technology**

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. To go directly to Part 7, click here.

1 - Mainframe Computing 1959-1970 - Second and Third Generation Technology

So to restate Part 4 succinctly, the average mainframe in 1958 was different in several technical respects to the machine which had existed in 1950, and was being used in exciting new ways both commercially and academically. Ferrite core memory had become the de facto standard for Main Memory construction, and machines now came complete with an operating system and an assembly language. However, two innovations in particular were already making such an impact on the industry that they are nowadays seen as the defining characteristics of an entirely new breed of system, namely the **"second generation" computers (1959-1964)**. The first of these class characteristics, a technological advance, was that electronic circuitry could now be transistorised, and therefore made far smaller than in its first generation predecessors; and the second, a design advance, was that second generation computers used microcode. These machines were then followed by the even more advanced **"third generation" computers (1965-1970)**, where the class characteristics were (a) the replacement of the chassis-mounted transistor with the integrated circuit, in which all functional components except the ferrite core Main Memory were miniaturised onto a silicon wafer, known as a "silicon chip", (b) the fact that programming was now increasingly done in compiled high-level languages such as COBOL, and (c) the provision of a virtual memory operating system. In this section, we review the impact of microcode, together with a dozen or so other noteworthy developments from the period 1959 to 1970.

1.1 The Impact of Microcode

Microprogramming is "a systematic technique for implementing the control unit of a computer. It is a form of stored-program logic that substitutes for sequential-logic control circuitry" (Smotherman, 1999). To understand what this means, we have to remember the sequence of events in the idealised Eckert von-Neumann machine's basic processing cycle, namely that individually simple instructions were interpreted by the Control Unit, which then passed an appropriate signal pattern to the logic gates of the Arithmetic/Logic Unit (either bit-by-bit down a single wire if a serial design, or simultaneously down a number of wires if parallel). Unfortunately, the more you wanted the ALU to do at a time in the interests of processing speed, the more complicated the logic circuitry had to be, and the more complicated the circuitry, the more it would cost, the longer it would take to develop, and the more often things would go wrong. The resulting pressure to avoid complex instructions left programmers struggling to reduce complicated mathematical procedures to sequences of forcedly simple steps, and led, in turn, to very long programs.

The win-win solution, on the other hand, was to find a way of providing a high functionality instruction set without paying the price in CPU complexity, and this is what EDSAC's Maurice Wilkes achieved in 1951. **What Wilkes proposed was that the Control Unit should abandon the one-to-one relationship between the program instruction coming in and the ALU instruction going out.** Instead, a repertoire of more complex program instructions was provided, each of which, when detected by the Control Unit, was simply replaced on a many-for-one basis by a logically equivalent series of simpler instructions. The user-visible instructions were called **"macro-instructions"**, and the machine-visible ones were called **"micro-instructions"** or **"microcode"**. It thus became the microcode which controlled the action of the ALU, and not some momentary input pattern derived on a one-for-one basis from the program instruction. Wilkes continued to perfect this approach during the 1951 to 1958 development of his EDSAC II, and then IBM were among the first to buy in to the concept (and a fortune they were to make out of it), fielding it in their 1964 System/360 [see Section 1.6].

> **<< AUTHOR'S NOTE: As repeatedly pointed out previously** [e.g., Part 4 (Section 1.3)]**, neuroscience does not know enough about biological information processing to judge whether the unit of decision making - that is to say, the biological equivalent of a single logic gate - is a single neuron, a small neuron cluster, a large neuron cluster, an entire gyrus or ganglion, or an entire cortical layer. This makes it difficult to judge whether biology evolved separate Control, Logic, and Memory Units, and therefore to identify either the unit (or units) of instruction or the unit (or units) of storage. >>**

1.2 The Manchester University Atlas

While its London factory focussed its attention on the Pegasus [Part 4 (Section 2.3)], Ferranti's Manchester works continued its collaboration with Manchester University and started work in 1956 on a transistorised computer. This was to be fitted with core memory, and was given the name MUSE because it aimed to hit the magic one million instructions per second (mips) [MUSE was short for "microsecond engine", the Greek letter "mu" (Cyrillic μ) being the standard technical abbreviation for "millionth"]. In fact, it took a full six years for the product - soon rebadged "Atlas" - to reach the showroom, largely thanks to the sheer intricacy of the engineering required. One major problem was that no one program would be able to utilise such a fast processor, because at 1 mips, all programs were going to be seriously peripheral-bound. **It therefore made sense to allow several programs simultaneous access to the machine, with each program vacating the CPU during its slower input-output operations.** This arrangement is known as **"multiprogramming"** ....

> **Key Concept - Multiprogramming:** In Atlas's case, up to 16 programs - all batch - could be managed concurrently, and it was left to the job execution logic [more on which below] to sort out their relative priority. **<< AUTHOR'S NOTE: The idea of several programs competing with each other for limited resources is quite popular in modern theories of biological attention and high level motor control, especially those of the Norman-Shallice type. For specific examples, click here or here. >>**

Unfortunately, no operating software yet existed capable of supporting the sort of job execution scheduling which multiprogramming called for; and so this, too, had to be written. In the resulting three-pronged development, David B. G. Edwards managed the hardware teams, R. A. Brooker took on compiler development, and David J. Howarth saw to the operating system. Their work is historically noteworthy on two major counts. The first is for including the concept of **"system supervisor"** in the package of functionality provided by a good operating system .....

> **Key Concept - The System Supervisor, "Jobs", and Job Execution Scheduling:** A **"System Supervisor"** is the top level control logic in a hierarchically organised computer operating system. It is the "process control module" at the "top" of a pyramid of lesser modular processes. In Atlas's case, the idea was that each program's momentary demands would be matched as far as possible against the machine's momentary resources; and,

where this meant queueing, that would be handled automatically by a **"contention scheduling"** routine. Another important aspect of the System Supervisor concept was that it required the programmer to consider not just the logic of his code, but the broader package of things needing to be done before that code could be executed. This gave rise to the concept of **"job"**, a request to run a program (or, indeed, a series of programs) against a precisely identified set of resources such as data, peripheral devices, and CPU time. These resources were identified as part of a **"job pack"**, originally a set of punched cards to be read by the Supervisor software, but soon yet another computer file. The idea of job packs soon caught on across the industry. IBM job packs, for example, were written in a set of conventions known as **JCL** (= **"Job Control Language"**). The ICL VME operating system went a stage further in the 1980s, allowing job packs to be compiled as object code, whereupon they could be treated just like the instructions directly provided by the OS. This control language was known as **SCL** (= **"System Control Language"**). In 1988, the author wrote the operational SCL routines to control the execution of seven linked COBOL programs making up a major stores accounting suite, complete with restart points, archive taking, archive pruning, on-the-fly integrity checking, and full audit trail; and the whole thing performed as sweet as a nut. Ed Thelen notes that Howarth's Supervisor is "considered by many to be the first recognisable modern operating system" (Thelen, 2003).

Atlas's second claim to fame - *and it easily ranks in importance with the invention of ferrite core memory* - is that the System Supervisor could be used to apportion the available Main Memory amongst its 16 simultaneous programs in such a way that they all got the whole cake! The secret of this digital conjuring trick lay in notionally dividing Main Memory up into **"pages"**, each the size of a convenient block of disk storage, and by keeping count of how often each page was being used. Infrequently used pages could then be copied off to disc without the program being aware of the loss, so, when running several programs, each would think it had the full range of memory available to it *and could be programmed accordingly*. The cost to the system lay in all the counting and the copying, but the saving lay in the ability to pass the space thereby released over to other users. The name given to memory which looks like Main Memory but is really on disk was **"virtual memory"**, and the name given to the all-important process of copying blocks of it backwards and forwards was **"paging"** ...

Key Concept - "Virtual Memory" (VM), "Paging", and "Thrashing": Given that the typical Eckert-von Neumann machine had only a small amount of Main Memory, but a much larger amount of backing store, and given also that programs typically contain large segments of conditionally used code [see the indent on *Conditional Control* in Part 3 (Section 4)], it occurred to the Atlas design team to push low-usage segments of a loaded program out of the one form of memory and into the other. Because these segments were known as **"pages"**, the process of **"rolling code in and out"** of Main Memory as demand dictated became known as **"paging"** (sometimes **"demand paging"**). Additional registers were added to the CPU to keep note of where each program had got to, and **"page tables"** were stored in the operating system area of Main Memory to accumulate the necessary usage statistics. **"Thrashing"** is what happens in a VM system when paging gets overloaded or goes out of tune. The point is that rolling programs in and out of Main Memory takes time, so, if you do too much of it, the ability to do anything constructive in the in-between-times reduces accordingly.

The Atlas was officially inaugurated on 7th December 1962, but only three full, and two scaled-down, machines were sold, due partly to Ferranti selling off its computer division to ICT in 1963 [see Section 1.8]. The technology did not go to waste, however, in that it helped shape ICT's successful 1900-series. A final Manchester University project - the MU5 - ran from 1966 to 1974 as a testbed for ICL's VME 2900 mainframe range [which we shall be dealing with in Section 3.3].

1.3 The IBM Second Generation Machines

As we saw in Part 4 (Section 1.2), IBM's strategy during the early 1950s had been to apportion its efforts between developing simple tabulator-multipliers for its commercial customer base and producing large defence computers for the US military. By mid-decade, however, it had started to bridge the gap with mid-range products. In 1955 the IBM 608 transistorised calculator was introduced, in 1957 the 7070,

and in 1959 the 1401, "the first computer system to reach 10,000 units in sales". The IBM 1401 has recently been acclaimed "the Model T of the computer business" (Nicholls, 2003), because it was the first mass-produced, digital, all-transistorised, business computer. It was announced in October 1959, came complete with an Assembler language called *Autocoder*, and could be installed on lease from as little as $2500 per month. It is also historically significant within the UK, because it grabbed so much of the market that it forced a series of mergers and rationalisations on the British computer industry [see Section 1.8].

The IBM Stretch was another cutting edge design concept from the late 1950s, and many of its concepts were (and some still are) ahead of their time. The project was fronted by Stephen W. Dunwell, ex-codebreaker, design began in 1955, and the product as first delivered in 1960 contained 169,000 transistors, and offered 96,000 64-bit words of core memory. The development team included Gerrit A. Blaauw, Frederick P. Brooks, John Cocke, Erich Bloch, Werner Buchholz, Edgar Codd [more on whom in Section 3.1], Robert Bemer ("father of ASCII"), and Harwood Kolsky, and of particular importance was an advanced form of pipelining .....

> **Key Concept - "Pipelining" (Continuation):** The basic concept of "pipelining" was introduced in Part 4 (Section 2.3), as a technique "whereby multiple instructions are overlapped in execution" (Patterson and Hennessy, 1996, p125). The IBM Stretch designers set new horizons for this concept by arranging for the FETCH part of the basic FETCH-and-EXECUTE cycle to be *up to four instructions ahead* of the EXECUTEs (Smotherman, 2002). They did this by setting the instruction length at either 32 or 64 bits depending on the number of operands required, and then building into the Control Unit two 64-bit "instruction buffers". Instructions were fetched 64 bits at a time from Main Memory, and when one buffer had been processed the next was processed while the first was refreshed. In this way, the decoding of instruction <n+1> could be taking place during the execution of instruction <n>, with instructions <n+2> and <n+3> already on their way across. There was also allowance for a process known as **"Memory Operand Prefetch"**, a technique which Smotherman describes as "a novel form of buffering called a 'lookahead' unit".

Stretch also introduced the term and concept of the "byte" into the computing lexicon. The idea was Bob Bemer's (although the word itself seems to have been Buchholz's), and it helped you distinguish between bits, and the various clever things you could do with bits if you set your mind to it. Stretch was put to market as the IBM 7030 in 1961, and held the "World's fastest computer" title until 1964. It suffered its fair share of teething troubles due to the sheer number of innovations built into it, but it provided IBM with a body of design experience second to none, which they were about to put to spectacular commercial effect in the System/360 [see Section 1.6]. Metz (2001) therefore bestows upon the 7030 the accolade of the "Most Successful Failure in History".

1.4 The "Stack" Processing Machines

The 1950s had also seen the invention and increasing popularity of a clever alternative to the one- and three-operand instruction sets which had typified the generation zero machines. This involved feeding only one element at a time to the Control Unit, be that either the op code itself or one of its associated operators or operands. The Control Unit would then accumulate these elements in a "stack" (in the sense of "things to be done"), acting upon them whenever enough elements had become available to complete one of the subtasks making up the main task [example below]. It would continue to do this - subtask by subtask - until it had the final answer, whereupon it would discard the elements it had finished with, leaving the stack free to start accumulating elements for the next task. For technical reasons, the most convenient stack structure is **"last in, first out"** or **LIFO**, and, by analogy with the old-fashioned desktop "in-tray", this way of doing things became known as **"stack processing".**

The inventor of stack processing was Charles L. Hamblin (1922-1985), an Australian philosopher and computer scientist. McBurney (2003) explains that Hamblin first learned about computing when working as a radar technician during World War Two. He then did his doctorate on the topic of information theory before taking a post in philosophy at what is now the University of New South Wales. His radar experience then got him involved when the university bought one of English Electric's DEUCES in 1956, and he was set to work producing an in-house programming language called GEORGE, which used a stack-structured instruction queue and something called **"Reverse Polish Notation"** (Hamblin, 1957, 1962) .....

> **Key Concept - "Stack Processing", "Polish Notation", and "Reverse Polish Notation":** An instruction **"stack"** is a constantly changing queue of instructions maintained by the Control Unit to cope with instructions which for one reason or another are not yet ready to be acted upon. Stack processing requires programmers to learn a curious new way of thinking. Specifically, it adopts the reverse of a mathematical notation system known as **"Polish notation" (PN)** (so named, after the Polish mathematician, Jan Lukasiewicz, who invented it in the 1920s). In PN, the brackets which are so vitally important to the conventional "BODMAS" system of algebraic expansion are not allowed, and great care therefore has to be taken to expand complex algebraic terms in the right sequence. Thus in normal arithmetic the expression < **4(a + b)** > binds (a + b) together, meaning that 4 has to be multiplied by all the terms within the brackets, giving < **4a + 4b** >. What Lukasiewicz did was to reorganise the expression as < **x4+ab** >, which we may read by scanning left-to-right as "**multiply** by **4** the **plussing** of items **a** and **b**". This gives the same answer WITHOUT USING BRACKETS. What Charles Hamblin then did was to show that **"Reverse Polish Notation" (RPN)** was an excellent way of presenting arithmetical instructions to a stack processing logic unit. In RPN, < **x4+ab** > becomes < **ab+4x >**, which we may read as "take **a**, take **b**, and **plus** it to what you have accumulated so far, take **4**, and **multiply** it by what you have accumulated so far". The real advantage comes whenever conventional notation calls for nested brackets, because these have to be resolved in matching pairs starting with the innermost. This means holding the entire expression (and it might be long and complex) in memory and making a number of passes through it, resolving one level of bracketing at a time. RPN thus saves a large Control Unit register and lots of time. On the down side, of course, there is the intellectual challenge of converting the expression into RPN in the first place, but, once that is out of the way, the fetch and execute logic is simplicity itself. Because operators and operands arrive one at a time, stack systems have been described as **"zero addresses per instruction"** systems. Stacks can be implemented either in hardware as an array of registers, or in software as a Main Memory array under the control of the machine's operating system. **<< AUTHOR'S NOTE: We shall be considering whether biological cognition uses stack processing, Polish or otherwise, in Part 7. >>:**

News of Hamblin's achievement soon made its way back to English Electric, who were so impressed with the idea that they used it in their 1960 KDF9, giving it a "hardware stack" capable of holding 16 48-bit words. Stack processing was also used in the 1961 Burroughs D825 and D830 and the 1963 B5000, as well as in the Hewlett-Packard HP9100A calculator.

1.5 "Supercomputers"

In Part 4 (Section 1.3), we saw how the Remington Corporation's UNIVACs did much to popularise business computing in the mid-1950s. Unfortunately, the corporation was still beset by interdepartmental and political rivalries, and in 1957 one of ERA's original founders, William C. Norris, resigned to found the Control Data Corporation. He took with him Seymour Cray as his Chief Engineer and a dozen or so other engineers and programmers, and so carefully had he chosen his people that the new corporation's first product, the fully transistorised CDC 1604, was ready only a year later. This was then followed by the world's first fully transistorised **"supercomputer"**, the CDC 6600, in 1964.

> **Key Concept - Supercomputers:** A supercomputer "is a computer that leads the world in terms of processing capacity, particularly speed of calculation, at the time of its introduction" (Wikipedia, 2003).

The 6600 was a highly compact design, thanks to deliberately short internal cable runs and an integral Freon liquid cooling system, and this allowed it to achieve 3 mips. It thus "completely outperformed all machines on the market, typically by over ten times" (Wikipedia, 2003 online). Lundstrom (1987) describes some of the machine's technicalities .....

> "The [CPU] consisted of ten separate 'functional units', each optimised to do only one particular operation - multiply, add, divide, shift, etc. The CPU communicated only with its central memory; it had no input/output instructions whatsoever. The CPU operated on a 60-bit word, which is unusually long. Included in the 6600 computer and communicating with the CPU's central memory [128k words, or about one Megabyte by today's measures] were ten independent 12-bit minicomputers, each having its own separate memory as well as access to the CPU's memory. These minicomputers, called Peripheral Processing Units (PPUs), provided the input/output instructions to keep the CPU fed. Thus a model 6600 consisted of eleven (or twenty, if you counted each functional unit separately) discrete programmable computers, all capable of working independently but all capable of communicating with each other through the central memory" (Lundstrom, 1987, p111).

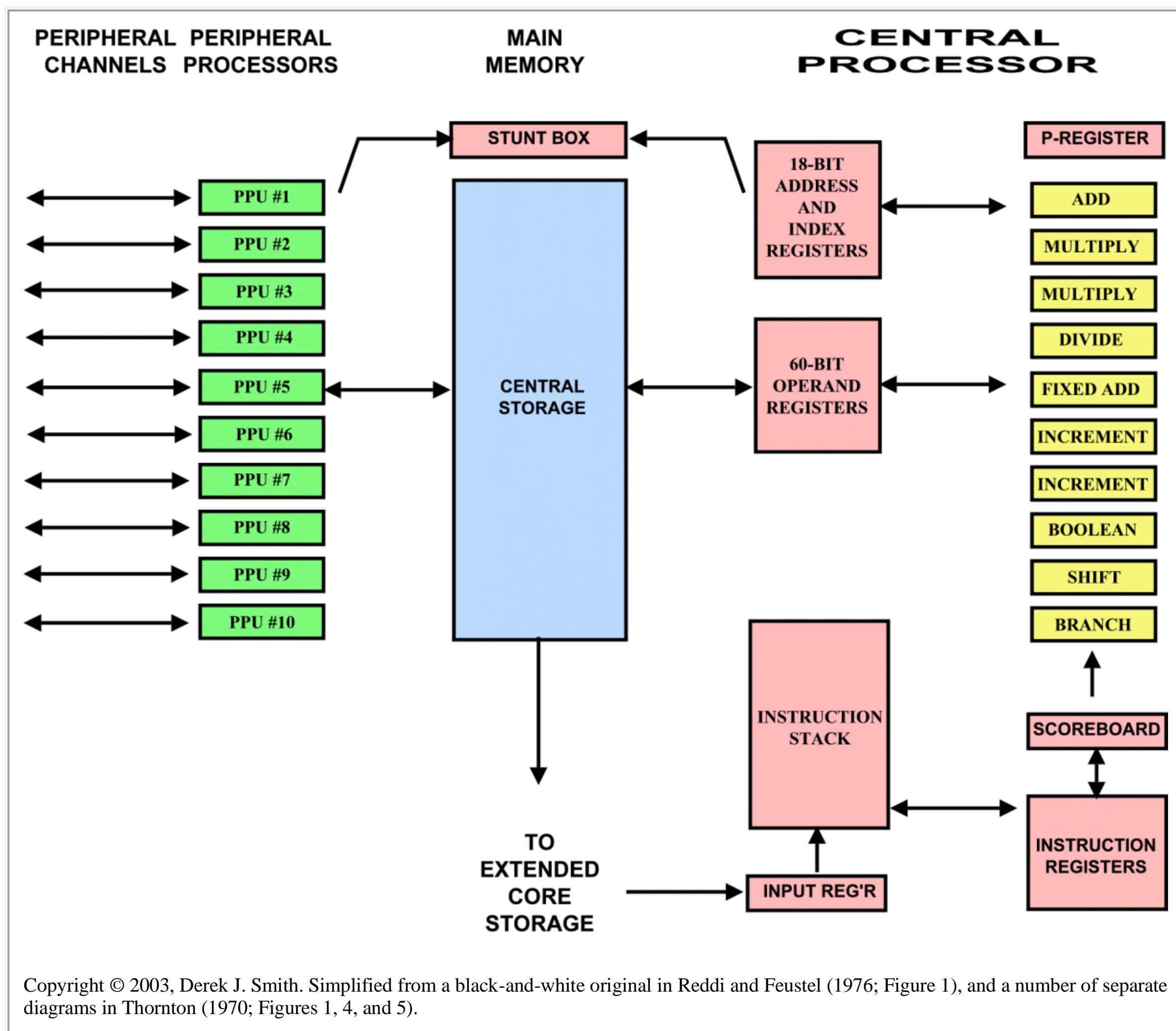
And the machine's principal designer, J. E. Thornton, adds .....

> "The Central Processor of the Control Data 6600 Computer is based on a high degree of functional parallelism. This is provided by the use of many functional units and a number of essential supporting properties [...] The ten functional units are independent of each other and may operate simultaneously. In a typical central processor program at least two or three functional units will be in operation simultaneously." (Thornton, 1970, pp12-13)

Figure 1 shows how these various processing modules were laid out .....

Figure 1 - The "Distributed Computing Architecture" of the Cray 6600 Supercomputer: Here we see the ten **"Functional Units"** (FUs) of the CPU [yellow highlights, right], the ten **"Peripheral Processing Units"** (PPUs) [green highlights, left], and the various elements of the Control Unit [red highlights, right], all neatly set out around the machine's Main Memory [blue highlight, centre]. When the control system decides that a particular FU is needed, it checks with the Scoreboard that that resource is currently free. If it is, the Scoreboard then issues the necessary control instructions, telling the FU in question when to start and what to do with the results when it has finished. If not, then that instruction is queued. The Scoreboard also interlocks the FUs when it is logically impossible to run operations simultaneously. The eight-element Instruction Stack permitted up to seven "old" instructions to be retained within the CPU, so that short program loops could be managed from within the stack without requiring repeated time-consuming FETCHes.

<< AUTHOR'S NOTE: Of all the machines we have encountered so far, the CDC 6600 is the first to start to approach neuroanatomical modular complexity. Its ten carefully synchronised FUs might be compared to the principal functional areas of the mammalian neocortex [readers unfamiliar with cortical mapping in humans may find some value in Kleist (1934)], and its ten PPU channels might be compared to the brain's sensory and motor systems. In fact, if you rotate the entire diagram 90° counter-clockwise, it assumes the same basic shape as the layered control hierarchies so often suggested by cognitive theorists - see, for example, Norman (1990). It is also worth noting that the highest level of control is provided by a complex network of registers and control units, *themselves distributed*. This is a cautionary note for cognitive scientists seeking single seats of biological consciousness and volition. >>



Copyright © 2003, Derek J. Smith. Simplified from a black-and-white original in Reddi and Feustel (1976; Figure 1), and a number of separate diagrams in Thornton (1970; Figures 1, 4, and 5).

The 6600 was actually upgraded "well after first deliveries" (Thornton, 1970, p17) so that two machines could share a common **"extended core storage"**, making the combination, perhaps, even more biologically lifelike [Thornton's Figure 8, for example, broadly resembles the bi-hemispheric vertebrate brain (*op. cit.*, p18)].

Clever though it undoubtedly was, the 6600 lacked the all-round strengths of the latest offering from IBM, the System/360 [see next section], and, as a result, secured only a few highly specialist sales (e.g., nuclear physics, meteorology, and the like). However, one of its lasting benefits was to show how instruction sets - which had grown bloated over the years - could be re-simplified, an approach which would later become known as **"Reduced Instruction Set Computing" (RISC)**. In 1968, Cray designed the CDC 7600 as a replacement for the 6600. This made intensive use of advanced pipelining and a lean, mean, instruction set, and, at about 15MIPS, broke all the speed records of its day. However, it acquired no great reputation for reliability, and again sold only to those who needed processing speed first and foremost, and could afford to pay for it.

1.6 The IBM System/360

By the early 1960s, IBM was receiving more and more complaints about how difficult it was for new models to run software developed for earlier ones. All too often, potential customers were saying that they could afford this or that new piece of hardware, but not if it meant junking all the software they had

struggled to get running on the older hardware. The company therefore set up a brainstrust in November 1961 to look into this problem, and a month later the panel came back with what became known as the **"Spread Report"**. Here is the thrust of that report's recommendations .....

> "The central realisation of the committee, was the need for a fully compatible family of machines. Each machine could specialise on particular areas, could offer different levels of performance, and at different prices, but the essential quality was that of complete compatibility. I.e., every program written for the *family* of machines could be executed on any one of the machines (obviously yielding to peripheral and memory requirements, and at different speeds)."

The result was one of IBM's all-time winners, namely the 1964 **System/360** range of machines. A family of seven machine sizes was offered, to appeal to all sizes of organisation, but operating procedures and programming languages were standard across all of them (Ganek and Sussenguth, 1999). The development team included Eugene M. ("Gene") Amdahl (1922-1998), later to leave IBM to produce the Amdahl range of "IBM-compatible" competitor machines, and the operating system was developed by Frederick P. Brooks. Both Amdahl and Brooks had already contributed to the success of the Stretch system five years previously.

> **ASIDE:** Brooks has since been acclaimed as the father of modern IT Project Management. It was he who first publicly lamented the fact that computer projects could suck in programming hours like the proverbial bottomless pit; also of the phrase "the mythical man month" (Brooks, 1975). He was particularly critical of the tendency for higher management to throw additional staff at problems at the last minute, in the rather naive belief that this would help. It never does. **It takes nine months to deliver a baby, Brooks explained, no matter how many women are assigned to the task!** He was subsequently involved in the development of computer graphics and natural language processing.

Nevertheless, it was Wilkes' concept of microcode which allowed it all to happen. Remembering the need for a *family* of machines, IBM gave the models at the bottom of the range lots of microcoded instructions and a cheap and relatively uncomplicated processor, whilst those at the top of the range were given less microcode (and hence ran faster), but needed more expensive processors to compensate. **At the coding level, however, programs were effectively identical.** In other words, it was finally dawning upon computer designers that in a world of technical and commercial compromises it was a machine's **"architecture"** which determined what it had been optimised for, and therefore who was likely to buy it. Thisted (1997) explains the issue nicely .....

> "A *computer architecture* is a detailed specification of the computational, communication, and data storage elements (hardware) of a computer system, how those components interact (machine organisation), and how they are controlled (instruction set). [...] The System/360 also precipitated a shift from the preoccupation of computer designers with computer arithmetic, which had been the main focus since the early 1950s. In the 1970s and 1980s, computer architects focused increasingly on the instruction set." (Thisted, 1997)

We should also note that the System/360 followed Bemer and Buchholz's lead [see Section 1.3] and standardised on an 8-bit character code known as **"Extended Binary Coded Decimal"** (or EBCDIC, pronounced "ebbsie-dick", for short). This system supports a 256-item character set, using the codes 00000000 (decimal zero) to 11111111 (decimal 255) inclusive, double the capacity of the 7-bit version of the ASCII system. For sake of readability, however, each 8-bit byte was sub-divided into two 4-bit "nibbles", separately transliterated into hexadecimal. Thus 01001011 (decimal 75) was treated as 0100 (hexadecimal/decimal 4) followed by 1011 (hexadecimal B, decimal 11), and shown as 4B.

> **TECHNICAL ASIDE - HEXADECIMAL:** Hexadecimal is the base 16 numbering system, which means that each left order integer represents a power of 16 higher than the one to its right. This rather inconveniently means that you need 16 different characters per digit column, and so hexadecimal takes the decimal digits 0 to

9 inclusive, and extends these by the letters A for decimal 10, B for decimal 11, C for decimal 12, D for decimal 13, E for decimal 14, and F for decimal 15. You cannot go beyond hexadecimal F, because adding one to F will always "carry one" - thus hexadecimal F (decimal 15) plus 1 gives hexadecimal 10 (read this as "one zero", not "ten", because in hexadecimal it signifies < one 16 plus no units >), and hexadecimal FFF (decimal 4095) plus 1 gives hexadecimal 1000 (read this as "one zero zero zero", not "thousand", because in hexadecimal it signifies < one 16 to the power 3, no 16 squareds, no 16s, and no units >).

1.7 On Line Transaction Processing and Time-Sharing

It is important to remember that when the Atlas team invented multiprogramming [see Section 1.2], they were not offering access to multiple "online" users. Rather, there was a single central machine operator - not necessarily a programmer at all - who sat at the computer's control console typing in operating system commands (*not programming language*). The programmers sat at desks in offices and laboratories elsewhere, preparing their programs as punched card packs and submitting them, presumably with some form of authorising documentation, to the operator; and the real "end users" of the software, that is to say, the people who were paying for it all because they had some pressing need for the results it would bring them, may have been elsewhere still. The Cambridge University EDSAC, for example, made itself popular across the campus when the Mathematical Laboratory started to offer a data processing service to other faculties.

How much better would it be, therefore, if a system could support a number of terminals in addition to the operator, making some of these available for software development, and the rest for live departmental applications. Many minor advances were made in this direction during the mid-1950s, but the most famous was "SABRE" (Semi-Automatic Business-Related Environment), IBM's real-time air ticketing system for American Airlines, and "a model for all point-of-sale transaction systems to come" (Waldrop, 2001, p75). This was the birth of the **"on-line transaction processing"** (OLTP) type of modern computing ...

Key Concept - On-Line Processing: Processing is deemed to be "on-line" if and when the decision making elements of the machine are free to respond with only a few seconds delay to a particular enquiry or update. On-line processing is not available with any form of tabulated card system, because it is totally impractical to go directly to a particular card in a particular card set [see the paragraphs on "random access" in Section 3.1]; indeed in this respect, if no other, tabulator systems were a retrograde step compared to 19th century filing cabinet systems. On-line processing is sometime referred to as "on-line **transaction** processing" because there is a specific and brief exchange - the **"transaction"** - between the user and the machine. The OLTP module within an operating system is known as the **"TP Monitor"**. IBM's current TP monitor is its Transaction Processing Facility (TPF), first released in 1979. This evolved out of IBM's Airline Control Program (ACP), a System/360 project dating from the mid-1960s, which was itself heavily influenced, in turn, by the earlier SABRE and SAGE systems. Wilkes (1963) describes the OLTP facilities offered by Ferranti and English Electric during the 1960s, and in due course these evolved into ICL's Transaction Processing Management System (TPMS).

SABRE came about because IBM had spent a lot of time assisting MIT in developing the SAGE air defence system, and it made good sense to leverage these rare skills. The company therefore contracted with American to look into the seat reservation problem in 1957, and by March 1959 a specification had been agreed. Detailed development then lasted another three years and cost an estimated $30 million. The pilot version of the system was installed in 1962, running on an IBM 7090, fitted with "front-end processors" to handle the data communications processing load ...

Key Concept - "Front-End Processors" and "Cluster Controllers": A **"front-end processor"** is a small computer designed to extend mainframe input/output channels outwards onto a data communications network. A **"cluster controller"**, on the other hand, is a small computer designed to extend a data communications

network onwards onto the input/output channels of a number of remotely located access terminals. As to the software needed to manage all this, see firstly the indent on "Network Protocols" below, and then Section 3.2.

We must not forget that an air ticketing system is an end-user system - it provides a user-friendly (for its day) application system to travel agency personnel, who, needless to say, write no code. Back at MIT, meanwhile, they had been making an altogether different point, looking at fully-fledged **"time-sharing"**, and here we again meet the psychologist-in-computing J.C.R. ("Lick") Licklider (1915-1990). Now we have already heard [see Part 4 (Section 1.1)] how Licklider had joined Bolt, Beranek, and Newman, a firm of Cambridge, MA, acoustics consultants, in 1957. While he was there, he helped them expand into software engineering, earning the (unofficial) title of "Cambridge's third university" (after Harvard and MIT) into the bargain. While thus engaged, he soon became convinced that it was only a matter of time and good design before properly trained operators would be able to hold a conversation of sorts with their CPUs. The concept was given the technical thumbs up by IBM's Bob Bemer in 1957, and one of the first to start work was the same John McCarthy whom we introduced in Part 4 (Section 4.7). The first large time-sharing system was MIT's **"Compatible Time-Sharing System" (CTSS)** in 1961.

Key Concept - Time-Sharing and Related Concepts: "**Time-sharing** is an approach to interactive computing in which a single computer is used to provide apparently simultaneous interactive general-purpose computing to multiple users by sharing processor time" (Wikipedia, 2003). Time-sharing is made possible by the fact that humans type slowly at the best of times, and even more slowly when they have to keep stopping to think. So when User A is busy, the odds are that Users B and C are momentarily inactive. It was in environments such as that provided by CTSS that programmers were first called upon to edit their own programs at their terminals, rather than plotting them out character-by-character on coding sheets for punching elsewhere. It is also when the distinction between **"foreground"** and **"background"** processing emerged. To run a program **"in foreground"** is to have it in constant communication with the initiating user terminal. This has the advantage that output can be inspected as it appears, but the disadvantage that it effectively locks out the terminal for the duration of the program. Foreground processing is therefore reserved for interactive work, or for short programs. All other work is processed **"in background"**, with messages being routed to a job log file, rather than to the user terminal. All the user gets to see is a single prompting message at the end of run, whereupon s/he can check the job log to see if there were any problems. The distinction is readily demonstrated by the fact that readers of this paragraph will be able to print it off in background while continuing to read it in foreground.

There then followed an extraordinary convergence of interests, with civilian systems like SABRE and CTSS suddenly needing the sort of technology developed for military command and control systems such as SAGE, and the military suddenly needing to know more about human factors. In 1962, this earned Licklider a move to the Pentagon's Advanced Research Projects Agency (ARPA) to help develop their Information Processing Techniques Office (IPTO). Here he continued his work on distributed systems, putting forward a number of detailed proposals to raise what would now be called the **"connectivity"** of the hardware then in use [more on connectivity in Section 3.2]. When Licklider left ARPA in 1963, he left the conceptual design to be fleshed out and made to work by his successor, Robert Taylor, under the name ARPANET. Taylor chose a young engineer named Larry Roberts to head the project, and the first problem to be resolved had to be the compatibility problem, because computers from the same manufacturer often could not intercommunicate, and computers from different manufacturers never. The solution came in two stages, namely (a) to build a **"subnetwork"** of telephone lines and switching points, and (b) to create communication **"protocols"** ...

Key Concept - "Network Protocols": Network protocols are standard methods of (a) authenticating which stations should actually be in a network, (b) identifying which station a message is to be routed to, (c) governing when, and at what speeds, stations are allowed to transmit and receive, (d) arranging for data compression and decompression, and (e) detecting errors in transmission and arranging for their correction. Nowadays, there are many highly technical network protocols in force, published by such agencies as the International Standards Organisation, but the **Open Systems Interconnection (OSI) Reference Model** [**ESSENTIAL REFERENCE DOCUMENT**] coordinates them all. It is a reasonably non-technical overriding guideline, and it recommends a

seven-layered analysis for any given communication, with the physical channel (the wire, etc.) consigned to the lowest level, and each layer communicating logically (but not physically) with the matching decoding layer(s) at other node(s). This type of arrangement is known as **"peer-to-peer communication"** (Gandoff, 1990). IBM's OSI-compliant networking system, SNA, is described in Section 3.2. << **AUTHOR'S NOTE: We discuss the metaphoric value of the OSI Model in understanding layered cognitive systems in our e-paper on Shannonian Communication Theory. >>**

Unfortunately, networking calls for a lot of highly technical bits of code, and it took the remainder of the decade to get everything up and running. Bolt, Beranek, and Newman helped with many of the problems, including that of message prioritising, one of the keys to effective time-sharing. And of course all these people across these campuses now wanted to talk to each other as well, and so e-mail had to be invented as well. Here we have another of Bolt, Beranek, and Newman's engineers to thank. This is how *PreText Magazine*'s Todd Campbell tells the story of the first e-mail .....

> "Sometime in late 1971, a computer engineer named Ray Tomlinson sent the first e-mail message. 'I sent a number of test messages to myself from one machine to the other', he recalls now. 'The test messages were entirely forgettable ..... Most likely the first message was QWERTYIOP or something similar.' It seems doubtful that 'QWERTYIOP' will make it into the history books [yet Tomlinson] is the inventor of e-mail, the application that launched the digital information revolution" (Campbell, 1998).

These early campus systems eventually evolved into the world's first "online community" (Waldrop, 2001), and thence, with a little help from Tim Berners-Lee in 1989, into the modern Internet. Licklider rounded off his personal contribution by authoring a number of influential papers profiling the sort of knowledge interrogation software which Internet search engines have since made commonplace.

1.8 The BTM-Ferranti and English Electric Stables

In Part 4, we introduced the British Tabulating Machine Company (BTM) as "British Hollerith", holders of the British punched card licence (signed by Hollerith himself) since 1909 (Humphreys, 1997). Now just as IBM were arch-rivals with the Remington Corporation for the primary card-processing patents, so, too, were BTM and the Powers-Samas Company arch-rivals as the respective British licencees. However, rather than slug it out as IBM and Remington had done, the two British licencees merged in 1959 to become International Computers and Tabulators (ICT). Their first joint product was the 1962 1MHz, all-transistor, ICT 1301, complete with magnetic drum backing store and programmed (at least in the early versions) in machine code. In 1963, a 1301 was the first mainframe to be used by the Metropolitan Police, and a decade or so later it was not unknown for scrapped systems to be bought up by television production companies for use as scenery in science fiction programs.

The 1301, however, was soon obsolete, and the Managing Director of the new company, Cecil Mead, had the strategic vision of bringing all British computer interests together under one roof. Negotiations duly began, and by 1963 ICT had acquired the computing divisions of GEC, EMI, and Ferranti, and, consequently, the job of trying to sell "the largest ragbag of incompatible computers it was possible to imagine" (Humphreys, 1997). The acquisition of the Ferranti interests was a particularly good deal, because Ferranti brought with them their brand new FP6000 [courtesy of their Canadian subsidiary, Ferranti Packard], a machine with everything good about Pegasus and Atlas in its bloodstock. Devonald and Eldridge (1996) recall how they were part of the working party which examined the Canadian system on behalf of Ferranti UK, and drew up a report identifying where still further improvements could be made. In the event, they were not a moment too soon .....

> "..... in April 1964, the whole economic basis of the industry was transformed by the announcement of the IBM System/360 range of third-generation computers. It was estimated that the R & D spend for System/360 was

> of the order of $5 billion. At this point, all the mainframe manufacturers world-wide were forced into announcing computer ranges of comparable performance." (Campbell-Kelly, 1990)

The FP6000 rapidly got the improvements Devonald and Eldridge had suggested, was renamed the 1900-series, and was announced in September 1964. The 1900 series did most things right, and enjoyed considerable success at the heavy end of the British commercial mainframe market, before being upgraded in the late 1970s to the 2900-series, running the VME virtual machine operating system [more on which in Section 3.3].

As for Elliott Automation, their 400-Series machines had, following a strategic alliance with the NCR Corporation, actually outsold IBM for a while, and the company was encouraged to experiment with a transistorised upgrade. This project was led by the designer of the 405, Andrew St. Johnston, together with John Bunt and Roger Cook, and the result was the 800-Series. The first of these, the range-defining Elliott 801 prototype and the experimental 802 did not go into mass production, but around 250 of the 1962 all-transistor Elliott 803 were produced, three of which are currently in various stages of restoration, one at Bletchley Park, spiritual home of British computing.

Lyons, too, put together a third generation upgrade to their earlier range. This was the LEO III, and it came complete with both microprogramming [see Section 1.1] and multiprogramming [see Section 1.2]. Here is how it was received .....

> "There was no question that LEO III was a groundbreaking machine, and its later successes confirmed John Pinkerton's status as a computer designer of genius. Nonetheless, its introduction to the world was fatally hampered by mismanagement. As with LEO II, the design team working on LEO III was ludicrously understaffed - Peter Bird quotes a figure of just six people, compared with the armies developing new machines at IBM. [As a result] despite the excellence of the product, says David Caminer, selling it was still a problem" (Ferry, 2003, p166).

In November 2001, one of the original LEO systems men, David Caminer, spoke at the IEE's Guildhall Conference for Business Leaders, London, on the prospects for business computing as it celebrated its 50th birthday and looked forward to its second half century. His main criticism of his former employers was that they had not exposed their IT division to the harsh financial realities of computer development (he explains that the developers tended to look upon the parent company as their "sugar daddy"). This oversight finally caught them out when the LEO 3 project ran short of cash in 1961, and in 1963 Lyons sold LEO to English Electric. Then the System/360s started to arrive in numbers, another round of rationalisations followed, and by 1967 Elliott Automation and Marconi had followed LEO into the English Electric fold. Finally, in July 1968, English Electric and ICT merged to form International Computers Ltd (ICL). Suddenly, the future of British computing, to say nothing of ICL's 34,000 employees, was subject to the strategic steer of a single board of directors. As to why it had all gone so wrong, Usselman (1993) may have one of the answers .....

> "Another problem BTM faced was that it stood outside the university research establishment. This was true initially of IBM in the United States as well, but after the war that changed under the influence of the liberally funded, procurement-driven American military. In the U.S., moreover, universities and government had never established very firm connections, so IBM and other corporations faced little resistance from entrenched interests. In Britain, however, universities and government had established close links by the time of the war, with business drawn into the alliance only through the mediating agency of the consulting engineer. After the war, Britain's much smaller defence budget did not have the strength to dissolve those traditional bonds. University research remained closely tied to government and the military. This skewed efforts away from development and manufacturing. In the specific case of computing, this had the effect of keeping Britain's remarkable code-breaking machines and the pioneering computers developed at Manchester University

separate from BTM and other manufacturers. In 1949, moreover, BTM severed its ties to IBM, thus depriving itself of ready access to an alternative source of computer designs."

1.9 Machine Translation and Semantic Networks, 1959 to 1970

The period 1959 to 1970 saw few real advances in the world of machine translation, with those who believed the technology had a future facing continued (and at times strident) criticism from those who did not. The arch-realist Yehoshua Bar-Hillel pronounced judgement on the context issue, for example, thus .....

> "**Fully automatic, high quality translation is not a reasonable goal, not even for scientific texts.** A human translator, in order to arrive at his high quality output, is often obliged to make intelligent use of extra-linguistic knowledge which sometimes has to be of considerable breadth and depth. Without this knowledge he would often be in no position to resolve semantical ambiguities. At present no way of constructing machines with such a knowledge is known, nor of writing programs which will ensure intelligent use of this knowledge" (Bar-Hillel, 1960, p135).

There was also an overdue touch of realism on the funding side of things. In 1964, the US government set up the Automatic Language Processing Advisory Committee (ALPAC), to report on progress with automatic intelligence gathering from Russian documents, and its 1966 report on the long-term prospects for MT was very negative. It concluded that MT was "slower, less accurate, and twice as expensive as human translation" (Hutchins, 2003), and observed that it would be a lot cheaper for the analysts involved to go away and learn Russian the old-fashioned way!

So basically it was down to the network-philes to respond to this challenge by demonstrating some genuine "new insights", and as the 1960s got under way they set out to do just that. One of the first to report was Silvio Ceccato (1914-1997). In two linguistically highly technical papers (Ceccato, 1961, 1967), Ceccato developed the idea of the **"correlational net"**. Ceccato was concerned with the relationships established between mental constructions, be they perceived things (i.e., nouns), or done things (i.e., verbs), or whatever. Such constructions, when combined in some way, require that "the mental category of relationship comes about" (Ceccato, 1961, p91), and thought itself is "precisely an opening and closing of correlations" (p94). **This is an important point, and we shall be hearing a lot more about relationships and nodes when we get to Section 3.1.**

Another ex-CLRU member, **Michael A.K. Halliday**, went on to develop **"systemic functional grammar"**, a linguistic theory based on lots of programmer-friendly decision trees called "systems", and generally to promote the structure-vs-function distinction within linguistics (eg. Halliday, 1970). Halliday also met with Berkeley's **Sydney M. Lamb**, and helped develop the latter's **"relational network"** (Copeland, 2000) [more on which in Section 3.9], and influenced the shape of Winograd's SHRDLU software [see Section 1.12].

The late 1960s also saw the first publications in a series of now-classical studies in cognitive psychology. The lead worker here was W. Ross Quillian, and the major papers are Quillian (1968) and Collins and Quillian (1969). What Quillian did was carry out a response time analysis of true-or-false statements of the form <A PARROT IS A BIRD>, <A CANARY IS A FISH>, <A CANARY IS BLUE>, etc. The point which emerged was that some of the things we know about canaries (etc.) seem to derive NOT from the fact that it is a canary, but from the fact that it is a bird, that is to say, from properties at a **"superordinate" conceptual node** in a multi-layered **"conceptual hierarchy"**. The team then drew an explanatory diagram, which identifies both nodal and supranodal properties (i.e., **"inheritance"**), and

in which there is a broad correlation between knowledge access time and distance to travel within the hierarchy, both vertically and horizontally.

1.10 The Turing Test, 1959 to 1970

Although the Turing Test dates from 1950, it was not until the early third generation systems that academics had access to sufficient computing power to try their luck at it. The best known of the first wave efforts was by MIT's **Joseph Weizenbaum** [Wikipedia biography]. .....

> **BIOGRAPHICAL ASIDE - JOSEPH WEIZENBAUM:** Joseph Weizenbaum was born in Berlin but became a US citizen in 1952. He trained as a systems engineer at Wayne University in the early 1950s, and after experience with analog computation was headhunted by General Electric. He joined MIT in 1963.

In 1964, Weizenbaum started to develop a computer program "with which one could 'converse' in English" over a teletype link (Weizenbaum, 1976/1984, p2). Remembering that it had been Eliza Doolittle's fate in "My Fair Lady" to perform a range of linguistic party pieces, Weizenbaum wittily named this program ELIZA. To get ELIZA to work, the program needed to be primed with a "script", "a set of rules rather like those that might be given to an actor who is to use them to improvise around a certain theme" (*op. cit.*, p3). For Weizenbaum's first experiment, he turned ELIZA into DOCTOR, by giving it a script to allow it to "parody" a Rogerian psychotherapist. To see a sample interaction, click here and see the image, top right.

Now Eliza's secret, of course, was that "she" was simply rephrasing input statements as questions! Here is the how the trick worked .....

> "The gross procedure of the program is quite simple; the text is read and inspected for the presence of a keyword. If such a word is found, the sentence is transformed according to a rule associated with the keyword, if not a content-free remark or, under certain circumstances, am earlier transformation is retrieved. The text so computed or retrieved is then printed out. [..... For example,] any sentence of the form 'I am BLAH' can be transformed to 'How long have you been BLAH', independently of the meaning of BLAH" (Weizenbaum, 1966, p37).

After several years of experience with his software, Weizenbaum concluded that it would never be possible to computerise language without first solving the problem of language context. ELIZA had little to say about the potential for computer understanding of natural language, and Weizenbaum was surprised that anybody would think that it did (although he was justifiably proud that his software was able to maintain the illusion of understanding with so little machinery). With much the same sense of sadness and surprise, he also noted "how quickly and how very deeply people conversing with DOCTOR become emotionally involved with the computer and how unequivocally they anthropomorphised it" (*op. cit.*, p6). (And, anecdotally, he was certain that some of his colleagues - despite knowing full well that they were conversing with a machine - nonetheless started telling DOCTOR their deeper secrets!)

Another early contender for the Turing Test prize was PARRY, a program produced by a Stanford University team led by the psychiatrist Kenneth M. Colby. The clever thing about PARRY was that its creators (psychiatrists, remember) had made it ever so slightly paranoid. Its "conversation" was therefore full of lifelike but rather cynical observations on the unfairness of life. Here are some of the software's typical carefully prepared answers .....

<I SHOULDN'T BE HERE>

<COPS DON'T DO THEIR JOB>

<COPS ARREST THE WRONG PEOPLE>

By deliberately playing on the interrogator's emotions in this way, PARRY could be rather convincing as to its inner humanity, and some of the shorter exchanges are decidedly humanlike. On the other hand, so they should be, because they were originally devised by the programmer and stored as entire phrases - no word-by-word sentence building was involved, merely whole-sentence output selection.

1.11 Neural Networks, 1959 to 1970

In Part 4 (Section 4.5), we saw how Rosenblatt's Perceptron had set the early standard for artificial neuron studies. Not surprisingly, there were many imitators, and another early pattern recognition machine, Oliver Selfridge's (1959) **pandemonium**, went back to the first principles of perceptual theory for its inspiration. Selfridge anticipated Marr's (1982) theory of perceptual stages by identifying four successive "layers" of image analysis, each deploying a network of small neural elements to act in parallel to analyse some particular feature (a line or edge contour, say) of an incoming retinal image. The output from each level of analysis then became the input to the level of analysis immediately above. The metaphors of demons and clamour come from Selfridge's description of activated elements as "shrieking" for attention from above. Finally, a particular pattern of activation will prompt a high level "decision demon" to hazard a guess as to the identity of the external stimulus [Boeree (2003) talks us through some useful worked examples of this process, if interested].

TRE's Albert Uttley was also still active in this area. Having played with mathematical predictions of dendrite density and the like [see Part 4 (Section 4.5)], he went on to model the process of neuronal maturation in the human infant's brain (Uttley, 1959). He identified two separate, but complementary, basic processes, firstly the strengthening of connections which were conditionally relevant to the system's current needs, and secondly the active deletion of those which were not. Subsequent microanatomical studies have proved him right, for by and large nature takes no chances when putting nervous systems together: far more neuroblasts are produced than are needed, and cells which fail to migrate or pathfind are simply allowed to die off. Indeed, according to Levitan and Kaczmarek (1991), "as many as three quarters of the neurons destined for a specific neuronal pathway may die during early embryonic development" (p305).

All in all, however, the early perceptrons were intrinsically very simple machines, and, when put to the test, did not really work that well. The purse strings gradually tightened, and in 1969 Marvin Minsky and Seymour Papert of the Artificial Intelligence Laboratory at MIT published a book entitled **"Perceptrons"** [Amazon] in which they threw serious doubt on how much such devices would ever be able to achieve. Funding duly collapsed, research teams were redeployed, and things went quiet for a decade or so [to go straight to the story of the rebirth of the perceptron in the guise of the modern neural network, jump to Section 3.6].

1.12 Machine Chess and Problem Solving, 1959 to 1970

The 1960s were a period of consolidation for the developers of chess playing software, and in 1966 Richard Greenblatt's MacHack was blooded in the Massachussetts Amateur Chess Championship, where it lost four matches and drew one. It returned in improved form the following year, and became the first program to defeat a human champion. As for machine problem solving in its more general form, the thrust was beginning to shift away from attempts to simulate mathematical reasoning, and towards the mysteries of natural language processing instead. Thus when the "Advice Taker" program was proposed

by McCarthy (1959), it attempted to solve problems by processing natural language statements, that is to say, its behaviour was designed to "be improvable merely by making statements to it, telling it about its symbolic environment and what is wanted from it". Which yet again brings us back to the science of linguistics, for what this new approach brought with it was the need for software capable of "parsing" a sentence.

Unfortunately, linguists have not always agreed amongst themselves how best to go about parsing, and it was over ten years before anything really impressive came along. The honour here has to go to Terry Winograd, then of MIT's Artificial Intelligence Laboratory, who wrote a parser called "SHRDLU" as part of his 1968 to 1970 PhD studies (Winograd, 1970, cited in Weizenbaum, 1976/1984). This program engaged its human operator in a textual dialogue, concerning a screen display of apparently three-dimensional solid blocks. The commands and questions were analysed for linguistic structure - the parsing - and then applied to the machine's mental model of this three-dimensional world. Thus <PICK UP A BIG RED BLOCK> was successfully parsed and understood as VERB (IMPERATIVE) <PICK UP> rather than, say, <GRASP>, ADJECTIVE <BIG> rather than <SMALL>, ADJECTIVE <RED> rather than <BLUE> or <GREEN>, and so on. However, the process of relating each word to its **"referent"** (object or action) in the real world moves beyond the syntactic nature of the word to the semantic, and gradually drew Winograd into the world of the semantic network, forcing him to build a "cognitive-deductive" ability into his software. SHRDLU could also cope with basic **"pronoun resolution" tasks** [glossary], and respond with a request for clarification when it was unsure, thus ...

> **FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE HOLDING AND PUT IT INTO THE BOX.**
>
> **BY 'IT', I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING.**

Requests for clarification such as this are commonplace in human verbal interaction, but require highly sophisticated psycholinguistic processing, often involving common sense cues or reference to earlier sentences.

> **<< AUTHOR'S NOTE: Our own view is that this sort of request for clarification is an "OSI Level 6" holding response, a touch more complicated than a simple "say again", but less complicated than a fully ruminative "well that all depends on what you mean by taller". Read again the indent on *Network Protocols* in Section 1.7, and then check out the "missionaries" example (i.e., Figure 8) in our e-paper on Shannonian Communication Theory. >>**

The story of Edward Feigenbaum's 1965 DENDRAL system belongs here by project start date, but is deferred to Section 3.5, and there treated as an early instance of a new type of reasoning system known as an **"expert system"**.

1.13 Machine Consciousness, 1959 to 1970

Which brings us to the problem of **"embodiment"**. This issue was first raised by McCulloch (1965), who, with more than twenty years of practical experience in the field, simply pointed out that it might eventually prove impossible to build an artificially conscious computer brain for the simple reason that it would have no biological body to go with it. It was not **"embodied"**, and as a result there would simply be too little for it to be conscious of. We shall be developing this point further in Section 3.12.

2 - Microcomputing, 1968 to Date

The story of the modern microcomputer can be told in many ways, and we are going to begin with the stories of four inventors and a "killer app", before looking at some of the technicalities .....

2.1 An Wang and his Calculators

The first of our inventors is Harvard Computation Laboratory's An Wang, previously introduced as the part-inventor of ferrite core memory. Not realising what the long-term impact of his invention was going to be, Wang was concerned to find that opportunities at Harvard were drying up following the Computation Laboratory's spat with IBM, and in April 1951 he resigned to set up on his own "as a sole partnership with his six hundred dollars of savings as capital, no orders, no contracts, and no office furniture" (Weiss, 1993, p63). Fortunately, Wang's reputation within the industry stood him in good stead, and his company grew slowly but surely through the 1950s. However, as core memory was developed at MIT, he got sucked into an unsuccessful patents dispute over licensing rights. Wang then did what only the very best geniuses manage to do - he geniused again, but this time he was careful to deliver the complete and legally unassailable package. His breakthrough came in the early 1960s, when Wang devised circuitry to compute logarithms using fewer than 300 transistors. He patented this in 1964, and used it in a desktop calculator named LOCI. The machine retailed at $6700, and the marketing edge came from the fact that purchasers got digital computing power at a fraction of the hardware cost of a mainframe, and without having to employ any professional programming staff. This was followed in 1965 by the Wang 300-series at from around $1700, and at a stroke the electromechanical calculator market - with a pedigree going all the way back to Pascal and Leibnitz - was doomed. By 1986, Wang Laboratories employed 30,000 staff.

2.2 The Invention of Integrated Circuitry

Our second inventor is Robert Noyce, and the invention is the "planar process" of semiconductor production. In September 1957 Noyce was one of the eight founders of Fairchild Semiconductor, and the new company's first director of R & D. All eight men had previously worked for Shockley Transistors, the company founded by William Shockley, co-inventor of the transistor in the late 1940s, and were keen to put their knowledge of semiconductors to the commercial test. The critical invention came after only a few months ...

> "During his tenure as head of R & D, Noyce patented an idea for an integrated circuit [..... His] great insight was that a complete electronic circuit could be built on a single chip, using specialised techniques (called planar processing) recently developed by his colleague [Jean] Hoerni. Noyce's conception, and the subsequent two-year development effort [would] eventually prove to be of monumental importance to Semiconductor, and, indeed, to the entire electronics industry" (Berlin, 2001).

The 1961 **"Fairchild chip"** had four transistors on it, and high profile projects such as NASA's Apollo Guidance Computer were buying them up at around $1000 apiece. By 1968, better fabrication techniques had both slashed the price and pushed the transistor count up to 200 per chip, allowing entire systems to be mounted on a single chip, giving the world the **"microprocessor"**.

Our third inventor is Jack St. Clair Kilby, one of the "few living men whose insights and professional accomplishments have changed the world" (Texas Instruments). Kilby joined Texas Instruments in 1958, and immediately did for them what Noyce was doing for Fairchild. In other words, he "conceived and built the first electronic circuit in which all of the components, both active and passive, were fabricated in a single piece of semiconductor material half the size of a paper clip. The successful laboratory demonstration of that first simple microchip on September 12, 1958, made history" (*ibid.*). In the event,

however, the Noyce-Hoerni technique was to prove markedly more cost-effective in production, and Texas Instruments eventually bought that method in.

And our fourth inventor is Gilbert Hyatt, who in 1968 trademarked the name **"microcomputer"**, and who then, with an "effective date" of 28th December 1970, filed for a patent for an entire computer on a chip. However, due to prolonged litigation, this patent was not granted until 17th July 1990, and then reversed on appeal in 1996. The chip had a large central array of logic gates, surrounded by six registers, two timing units, a channel control unit, and a **"scratchpad memory (SPM)"** unit, a rudimentary sort of "cache memory" ...

> **Key Concept - "Cache" Memory:** A memory "cache" is a memory resource physically situated on the CPU chip rather than at a distance away in Main Memory. It is intended to hold high demand memory pages without the need for chip-to-chip data transfers, thus saving time.

What Hyatt did well was to set up a company called Micro Computer, and amongst the venture capitalists who bought in were Robert Noyce and a colleague of his named Gordon Moore. What Hyatt did not do well was remain friends with Noyce and Moore. He refused in 1971 to sign his patent rights over to them, funding was withdrawn, his company ceased trading, and Noyce and Moore went on to prosper as the Intel Corporation. Twenty five years later, US law pronounced to the effect that Hyatt had tried to patent too general an idea, and that that was not protectable under patent law.

2.3 The "Killer App"

As for the "killer app" [this quaint phrase describes a computer application which more or less perfectly fills a previously unrecognised commercial need, and sells accordingly - Ed.], this was without doubt word processing. The antecedent technology was the 1961 IBM "Selectric" (or "golfball") typewriter. This delivered each keystroke via an electrically activated spherical typehead, in which the letter to be impressed was chosen by determining the angles of azimuth and pitch of the typehead at the time of striking. All these angles were set, in turn, by electronic signals generated by the keyboard, and in 1964 the system was upgraded to store those electrical signals on a magnetic tape, thus allowing the same document to be typed time after time, from memory. Moreover, by pausing the tape, it also allowed errors to be detected, and minor corrections/insertions made. For the first time, "typed material could be edited without having to retype the whole text" (Kunde, 1986). In 1969, IBM replaced the magnetic tape with a magnetic card, and in 1971, replaced the card with Al Shugart's recently invented "floppy disk". That same year, Wang Labs brought out the Wang 1200 word processing system.

> **ASIDE:** In 1976, the present author did commercial word processing on an Olivetti TES501, a desk-sized electronic typewriter, complete with eight-inch floppy disk, a single line side-scrolling display, and a daisy-wheel character printer, retailing for around £6000. Ten years later he could do more, and quicker, on an Amstrad PCW retailing for around £300.

2.4 The Integrated Circuit Industry

We have already recorded the birth of the chip in Section 2.2. As for the story of its subsequent exploitation, we present this in timeline form, collated from Siewiorek, Bell, and Newell (1982) and the sources specifically referenced .....

- 1968 - In July 1968, ex-Fairchild experts Robert Noyce and Gordon Moore set up Intel Corporation to develop and exploit DRAM (= "dynamic random access memory") memory chips. The commercial motivation here was that

ferrite core store was actually technically difficult to miniaturise, so that anyone managing to produce a workable solid state flip-flop resource would command a massive marketplace.

- 1969 - In April 1969, Masatoshi Shima, of Busicom Corporation, discussed with Intel his ideas for the chip architecture for a decimal desktop calculator. Intel agreed to produce these, but in September 1969 one of their engineers, Marcian E. ("Ted") Hoff counter-proposed with the plans for what would become the binary Intel 4004 chip. This was something of a masterstroke, because it would not just satisfy the specific Busicom calculator contract, but would also service small general purpose computing needs as well. That November, Datapoint Corporation's Victor D. Poor proposed an 8-bit programmable microprocessor. Poor offered the design to both Intel and Texas Instruments, and the latter took it on.
- 1970 - During 1970, Hoff, Shima, Stanley Mazor, and Federico Faggin finalised the 4004 design using around 2000 transistors and with parallel wiring.
- 1971 - In March 1971, Intel began deliveries of their 4004 to Busicom. In May, Texas Instruments' prototype 8-bit microprocessor was demonstrated to Datapoint. In July, Texas Instruments' Gary W. Boone produced the prototype of the TMS 1000 microcontroller chip, the first "computer-on-a-chip" (that is to say, a CPU microprocessor - as above - PLUS the supporting RAM, I/O, and ROM). The TMS 1000 family went into production in 1974, and was used across TI's calculator range. In November, Intel announced the 4004 as heralding "a new era of integrated electronics".
- 1972 - In September, Texas Instruments announced that it was entering the calculator market to leverage its chip know-how. In November, Intel announced the 8008 chip.

ASIDE: The Intel lineage went on to define by number the generations of the modern PC. The 4004 and 8008 were followed by the 8080 in 1974, the 8085 in 1976, and the 8086 in 1978. In that seven-year period, processing power increased about a hundred fold, and price decreased by about the same factor (from $300 a chip down to $3). The 8086 was followed by the 80186 and 80286 in 1982, by the 80386 in 1985, by the 80486 in 1989, and by the first of the Pentiums in 1993. There is thus a direct historical line from the biggest and best of the modern "Intel Insides" back to the Bell Labs team who discovered semiconduction in the late 1940s.

- 1975 - IBM introduced the IBM 5100. It bombed commercially, and for the next five years the company suffered as the home computer market exploded around it.

These, then, were the specific physical devices which defeated Hyatt in his long-running patents dispute. TI's microcomputer patent #4074351 describes the final item as "a one-chip device about one-fifth of an inch square, typically with more than 20,000 transistors or related electronic elements, and is the first integrated circuit to contain all of the essential functions of a computer: program and data memory, arithmetic unit, control, and input-output circuits" (TI website, 2003).

2.5 The PC Software Industry

A chip, of course, does nothing without some sort of operating software, and the story here begins in 1972 when William H. ("Bill") Gates and Paul G. Allen, set up the Traf-O-Data company to develop an automatic traffic recording system based around the Intel 8008 chip. This was one of many small company attempts to put the new technology to work, and in 1974 a lecturer at the US Navy's Monterey Postgraduate School, Gary Kildall, produced the first microcomputer operating system, CP/M. This was designed for the April 1974 Intel 8080 chip, and - foreseeing the as-yet-non-existent home computer market - intended to provide it with a range of simple user commands. Leaving the navy in 1976, Kildall founded Digital Research to market the product, and was soon selling copies at $70 a time. In August 1980, Tim Paterson, of Seattle Computer Products, started shipments of QDOS as a (just legal) CP/M clone.

Attention now turned from operating systems to programming languages, and this is where Traf-O-Data - now renamed Microsoft - comes in. While Kildall and Paterson had been developing their rival operating systems, Gates and Allen had been developing a language called BASIC for the Intel 8086. This was a user-friendly interpreted language (that is to say, it executed on an instruction-by-instruction basis from the source code), and it first saw the light of day in early 1975, just a few months before the first home computer shops started opening. These players and these products then proceeded to make a little history. In 1980, IBM approached tiny Microsoft, wanting to commission an operating system for a proposed new range of home computers. Microsoft, more interested in BASIC, simply recommended CP/M. IBM went away, but their negotiations with Kildall failed. They returned to Gates with their chequebook open, and Gates and Allen - reconsidering their options - bought in Paterson's QDOS for $50,000, and licensed it straight back out again to IBM for $15,000. To normal mortals, of course, this looks like a $35,000 kick in the pants, but not to the visionaries at Microsoft, who saw that for every PC they helped IBM to sell there was a decent chance for them to follow up with a sale of BASIC! And the rest, as they say, is history.

3 - Mainframe Computing, 1971 to Date - Fourth and Fifth Generation Technology

In this section we complete our review of milestone innovations in the history of computer memory, by looking at mainframe systems from 1971 to date. Broadly speaking, this divides into the period of the **"fourth generation" computer (1971 to 1979)**, in which "large scale integration" (LSI) was replaced, in turn, by "very large scale integration" (VLSI), followed by the period of the **"fifth generation" computer (1980 to date** [= 2003, remember]**)**, in which component size would be less and less critical, but processing architecture more and more so. Specifically, the fifth generation machines would be departing from the Eckert-von Neumann serial processing model in favour of ever more parallel architectures.

3.1 The Arrival of Database

We begin by reminding readers of IBM's 1957 invention of the RAMAC, the first commercial hard disk drive, a two-ton monstrosity capable of storing 5 megabytes of information (equivalent to less than four of today's standard density floppy disks, weighing a few grams and costing pennies). We offer this reminder, because, despite its girth, RAMAC was the key to giving computer users **"random access"** to their data, provided only that it could be **"keyed"** in some way. Given a suitable unique key (e.g., a National Insurance Number), and using either an **"index"** or a **"hashing algorithm"**, the physical machine address of the desired record - even if one record out of a file of 10 million similar records - can be obtained; and, using that machine address, the read-write heads on a magnetic disk can position themselves above the appropriate track in milliseconds. As a programmer, therefore, this enables you to state what you want, and have it arrive in your input record area a second or so later. Random access technology thus broke down barriers which had been troubling systems designers since the late nineteenth century, namely how to locate one person's data out of a census-full, say, or one account holder's details at a bank, etc., etc., etc.

But "direct-hitting" of this sort was not the only benefit. Conventional (i.e., pre-Hollerith) filing systems and Hollerith style punched card systems had encouraged single long records, each containing all the data relevant to the target entity, for in those days it did not matter if you pulled a person's card or a customer's folio to check on only one or two details - the cost lay in locating and retrieving the card or

folio, not in examining its contents. With electronic data, however, long records meant dragging around a lot of bytes when only a subset thereof was actually needed, and so the search began for ways of spreading your data out in a predetermined fashion, and of then using data management software to get you only the fragments you needed. The result was the first **"Database Management System"**, or DBMS .....

> **Key Concept - The Database Management Systems (DBMS):** A DBMS is "a software system that facilitates (a) the creation and maintenance of a database or databases, and (b) the execution of computer programs using the database or databases" (Alliance for Telecommunications Industry Solutions). It is an extension [that is to say, you have to buy it separately] to a computer's standard operating system, and it allows data to be fragmented, stored, and subsequently retrieved to order, all without cross-record confusion.

The pioneer of database was **Charles W. Bachman** [Wikipedia biography]. Here is the early story as told by Hayes (2002) .....

> "In 1961, Charles Bachman at General Electric Co. developed the first successful database management system. Bachman's integrated data store (IDS) featured data schemas and logging. But it ran only on GE mainframes, could use only a single file for the database, and all generation of data tables had to be hand-coded. [//] One customer, B.F. Goodrich Chemical Co., eventually had to rewrite the entire system to make it usable, calling the result integrated data management system (IDMS). [//] In 1968, IBM rolled out IMS, a hierarchical database for its mainframes. In 1973, Cullinane Corp. (later called Cullinet Software Inc.) began selling a much-enhanced version of Goodrich's IDMS and was on its way to becoming the largest software company in the world at that time."

During the 1960s, different users developed the early concepts and applied the early products in different ways, seriously impairing the exchange of data between systems, and it was therefore not long before some sort of industry-standard approach was being called for. This came in the shape of a comprehensive statement of database principles made by the Codasyl Database Task Group in 1969 (Codasyl, 1969, 1971; ANSI/SPARC, 1976). These advisory reports cover every aspect of the interaction between non-technical system users and their highly technical computing hardware, and were inspired throughout by the single central axiom that the internal complexities of a database should at all times remain totally transparent to the end-user. **A DBMS, in other words, should allow users to concentrate upon their data rather than upon the tool they are using to view it.**

Another new approach was popularised by an English-American mathematician called Edgar F. ("Ted") Codd (1923-2003). Codd had joined IBM in 1949, and had served time on the SSEC and Stretch teams, before switching to research into methods of data management. Here is Hayes (2002) again .....

> "Meanwhile, IBM researcher Edgar F. ('Ted') Codd was looking for a better way to organise databases. In 1969, Codd came up with the idea of a relational database, organised entirely in flat tables. IBM put more people to work on the project, codenamed System/R, in its San Jose labs. However, IBM's commitment to IMS kept System/R from becoming a product until 1980. [//] But at the University of California, Berkeley, in 1973, Michael Stonebraker and Eugene Wong used published information on System/R to begin work on their own relational database. Their Ingres project would eventually be commercialised by Oracle Corp., Ingres Corp. and other Silicon Valley vendors. And in 1976, Honeywell Inc. shipped Multics Relational Data Store, the first commercial relational database."

> **Key Concept - The "Flat File" or "Table":** A "flat file" is a file composed of large, identically formatted, data records, which, properly indexed, is ideal for random access storage and retrieval of uniquely keyed individual records. At heart, it is the technology of the filing cabinet, made digital; brilliant for "read only" or non-volatile files, or the *ad hoc* analysis of "large subset" enquiries, but guaranteed to struggle with volatile data and lacking the flexibility of traversal needed for flexible enquiry routes on a single record.

And the ACM .....

"While working for IBM in the 1970s, [Codd] created the relational model, the first abstract model for database management, which facilitates retrieval, manipulation, and organisation of information from computers. It led to commercial products in the 1980s, originally for mainframe systems and later for microcomputers. Relational databases are now fundamental to systems ranging from hospital patient records to airline flights and schedules" (ACM website).

And IBM .....

"In a landmark paper, 'A relational model of data for large shared data banks', Codd proposed replacing the hierarchical or navigational structure with simple tables containing rows and columns. 'Ted's basic idea was that relationships between data items should be based on the item's values, and not on separately specified linking or nesting. This notion greatly simplified the specification of queries and allowed unprecedented flexibility to exploit existing data sets in new ways,' said Don Chamberlin, co-inventor of SQL, the industry-standard language for querying relational databases [...] Janet Pema, general manager of Data Management Solutions for IBM's Software Group, expressed her admiration for the inventor of the product for which she is now responsible. She remarked that 'Ted Codd will forever be remembered as the father of relational database. His remarkable vision and intellectual genius ushered in a whole new realm of innovation that has shaped the world of technology today' " (IBM website).

So how did everything fit together? Well basically, your database team had to do a number of very precise things in a very precise order, as follows ...

(1) Carry Out Data Analysis: The first step is to carry out a detailed investigatory study of the business (or department) in question, in order to document the data stored and used within it. The most important technique here, whether working towards an IDMS or a flat file system, was **"Entity-Relationship Modelling"** (ERM). As its name suggests, an ERM exercise involved recording "the entity and relationship types in a graphical form called, Entity-Relationship (ER) diagram" (Chen, 2002), and this invariably invites the application of **"Set Theory"** whenever the analysis reveals one-to-many relationships within your data types .....

Key Concept - The One-to-Many Relationship of Entities: A company may have many customers, each placing many orders. A warehouse may have many zones, each with many aisles, each with many bays, each with many bins. And so on.

When you have completed your ERM, it will provide you with a "corporate non-technical view" of your data, a real world view, in which the organisation's processes and data, having been identified by the skills of the business analyst, are set out in a set of diagrams and memoranda known cumulatively as a **"business model"**. This provides what is known as a "logical" view of how the organisation operates, and one of the most important specific models is a diagram known as a **"data model"**, which records what real world objects the organisation is concerned with, how they behave, and what their properties are.

(2) Set Up a "Database Schema": The next step is to convert the data model into an equivalent set of declarations and descriptions known collectively as a **"database schema"**. Unlike the data model, however, the database schema is now in a form which can be stored within, and manipulated by, the DBMS. This is a more technical view of the data than hitherto, and constitutes the first major step in bridging the gap between the data as the user knows it and the hardware on which it is eventually to be stored. This is the point at which you finally have to decide between a network system such as IDMS, a hierarchical system such as IMS, or one of the proprietary flat file systems.

(3) Set Up Database "Subschemas": The third step is to create a "departmental" view of the data. This is another technical view, and reflects the fact that no single application program will ever need access to all the available data. Each individual end-user - and that includes even the most senior executives - only needs access to a fraction of that data, and for him/her to be shown too much is at best inefficient, and at worst a breach of system security punishable by civil or criminal law (or both). This "need to know" facility is provided by subsets of the schema known as **"subschemas"**, each one allowing an individual application program to access only the data it is legitimately concerned with.

(4) Set Up Database "Storage Schemas": The final step is to create a "machine level" view, and allows the data as defined in the schema to be "mapped onto" the particular installation's physical storage devices. This is achieved by declaring what is known as a **"storage schema"** to the DBMS, which the DBMS then uses to

translate every user-initiated store and retrieve instruction into a set of equivalent physical store and retrieve instructions. **This is therefore the second major step in bridging the gap between the data and the hardware on which it is to be stored.**

We turn now to the internals of the 1973 Cullinane DBMS product, because it was in managing physical storage that that IDMS really showed how clever it could be. Four major design principles were involved, as now reviewed .....

The first IDMS design principle was that data should be organised into "records" and computer filestore into "pages". The latter become the unit of transfer between the DBMS (and thus the user) and the physical hardware. Their size was decided upon by the design team, and the number of records which will fit onto each page is therefore a function of page size relative to record size. No matter how much data is involved, each record can then be located with total precision and in only a few milliseconds by knowing (a) which page it is stored on, and (b) how far down that page. The resulting two-component page-line address is known as the **"database key"** for that record. Room for future expansion is provided in advance simply by varying the total amount of filestore available. A database might be set up, for example, with its pages only 40% full, to guarantee plenty of room (and time) for growth.

The second IDMS design principle was that each different real world entity type should be represented by a different database record type. Each record type is then allowed to occur as many times as necessary to do the job. Thus for the record type <person> there could be many million **"occurrences"** of that record in a given database, each with its own unique database key, but each laid out internally in exactly the same way (name, sex, age, height, etc.).

The third IDMS design principle was that some data records should be allowed to "own" others. This capitalises on the "set" relationship(s) already identified, such that each given set owner can own one or more set members. The owner-member arrangement is the network database way of coping with the sort of one-to-many relationships which exist between many real world entities. For example, the fact that one real life person may own many real life books could be implemented on the database as record type <person> owning a set of record type <book>. It is up to the design team, incidentally, whether the <book> records are clustered on the same page as their owner or else more widely distributed throughout the remainder of the database, and it creates havoc, incidentally, when they get this sort of decision wrong.

Key Concept - The Computer Data Network: What Bachman and B.F. Goodrich had created, and what the Codasyl Task Force had standardised for general release, was the "network database" (a.k.a. "Codasyl database"). Using the entity-relationship analysis approach to interface the results of Business Analysis with the DBMS schema they created a network structure capable of random access updating and enquiry. << **AUTHOR'S NOTE: Between 1982 and 1989, the author was an IDMS database designer and applications programmer with British Telecom, and found the product both versatile and robust once you got used to it. It was admirably suited to systems needing to update "volatile" (i.e., rapidly changing) data, such as booking systems, banking, and logistics. Interrogation-only systems (eg. marketing data) are better approached with a Codd-style tabular approach. That said, we are dealing at length with the internals of the Codasyl-type database, because it sets up data networks just like those we introduced in Part 4 (Section 4.2) as "semantic networks"; indeed, the central thrust of this seven-part paper is that a computer network database and a biological semantic network are significantly comparable. >>**

The fourth IDMS design principle is simply that the data should be retrievable with the minimum fuss and the maximum accuracy once it has been loaded into the network. Fortunately, the first three design principles allow many different ways of locating a given record, each of which has its own particular merits. The main retrieval options are as follows:

Retrieval Option #1 - Direct Access: This is where a real world data item is designated as a record key, and its contents used to calculate the page element of the database key of the record in question. This is done using a DBMS facility devised by Dumey (1956, cited in Ashany and Adamowicz, 1976), and known as a "hashing algorithm". For example, given the record key "1248116" (a <customer number>, say), the hashing algorithm might return a database key of page 4371, and this would tell the system where physically to store the record in question. Moreover, having made this decision in the first place, *the same routine can be used to calculate where to look for that record whenever it is wanted again*. (Note carefully that whilst the record key is therefore meaningful to the end-users of the system, the database key is not: in other words, the record key is a *logical* key whilst the database key is an internal *physical* key.)

Retrieval Option #2 - Chain Pointer: This is where each record in a set contains the database key of the next record in that set (and also optionally of the previous record and/or the set owner). Each record thus "knows" the address of the next record to be found, so that in our earlier person-book example, we can locate every <book> simply by following the chain pointers around the <person-book> set one by one until we get back to where we started from.

Retrieval Option #3 - Indexes: This is where a large set is sorted according to its record key, and then accessed via a separate index of database keys. To access a given record, its record key is looked up in the index, and the corresponding database key extracted. The index processing adds a small overhead to the total access time, but this is usually well worth paying when sets are otherwise too large to read efficiently using the chain pointer approach.

Retrieval Option #4 - Set Currency: This is where a set member record, having once been accessed by a given program, can be "put on hold" temporarily while other important processing is carried out. What the DBMS does in these circumstances is to store the database key of the record in question so that it - *and the chain pointers it contains* - can be relocated when the next record in that set is eventually required. It does this by maintaining what is known as a **"run time currency indicator"** for every set it knows about, and every time it accesses a record in those sets it copies that record's database key into the appropriate **"current of set"** and **"current of record type"** currency indicator(s). The use and value of this facility is further explored in Part 6.

Retrieval Option #5 - Run Currency: This is where any record, immediately after retrieval by any of the foregoing methods, is totally and centrally available for interrogation and update. It is the record currently being processed. Unlike set currencies, therefore, only one record can be **"current of run"** at a time. The use and value of this facility is also further explored in Part 6.

Key Concept - The Database Currency: A database currency is a software device invented by database systems programmers to help applications programmers "keep their place" at one point in a program while a subprocess is sent off to do something important elsewhere. It works by holding in memory the key parameters of the superordinate search pattern for the duration of an excursion into one or more subordinate search patterns, so that the former can be restored and continued as soon as the excursion is over (just like keeping your finger in one page of a book, while you look quickly to one or more other pages). The device comes as standard with what are known as "Codasyl" databases such as Computer Associates' Integrated Database Management System (IDMS), and is actually nothing more than a small address table held in memory and constantly updated. For a friendly introduction to database currency technology, click here. << **AUTHOR'S NOTE: The importance of currency mechanisms to cognitive science lies in the fact that what they do is uncannily similar to what calcium-modulated synaptic sensitisation does in biological memory systems. >>**

Given a database integrated according to these options, the **"programmer-as-navigator"** (Bachman, 1973) can rapidly get a fragment of data (a person's address, say) from here, and another fragment (that same person's credit rating, say) from there, and a third fragment (that same person's previous purchases, say) from somewhere else, and can gradually build up the required display. As to the fragments to get and those not to get, it all depends on the precise nature of the enquiry or update, given the precise layout of the data network, and in practice a great deal of painstaking analysis has to go into designing the network and a great deal of programming skill has to go into writing the programs which are going to

access it. **It is vital, therefore, that each "traversal" of the database is meticulously thought out, and each access made in precisely the right order.**

> **Key Concept - The "Traversal":** A database traversal is the argument structure by which a network database is interrogated during the storing or retrieval of data. It is the method by which a number of distributed data elements are brought together to form a coherent display. This reflects the fact that most applications need to access far more than one record before they can achieve whatever is expected of them, that is to say, their final output displays - be they to screen or printer - are composites of fragments of data gleaned from hundreds of points within the network.

What happened next is a prime example of how words can often unintentionally misinform. The word in question is **"relational"**, and the context in which the misunderstanding arose was the **"relational database"**. What happened was that the advantages of the flat file implementation of a data model became so easy to market that everybody bought one, **including many for whom the technology was entirely inappropriate**. In other words, having learned how to distribute logically related data fragments around the nodes of an all-singing, all-dancing, data network, the industry decided to go back to long records; and decided into the bargain to describe that approach as relational, not falsely, as such, but with the implication that the network database was not itself equally tightly grounded in relational data analysis and set theory.

> **<< AUTHOR'S NOTE: In our 1982 to 1989 IDMS system, we supported a central heavy duty network database with a number of ancillary flat file systems. The network system handled the primary second-by-second updates, the big prints, and the routine small enquiries, whilst the flat file systems handled the data management and large enquiries using *QueryMaster*, ICL's own structured query language. Each suite of programs therefore played to its inherent strengths, to the ultimate benefit of the company. Flat file or otherwise, databases are important to cognitive scientists because they are the computer industry's only implementation of semantic networks. As such, they generate several noteworthy metaphors in a number of areas of memory theory, again not least that of calcium-sensitised medium-term memory. >>**

3.2 "Connectivity", 1971 to Date

We have already seen [see Section 1.7] how the 1960s witnessed major breakthroughs in online transaction processing, time-sharing, and campus-wide networking software. By 1971, it was possible to link a mainframe via a front-end-processor onto a telecommunications network, and thence either (a) via a cluster controller to a nest of remote user workstations, or (b) via a remote front-end-processor to another mainframe. In 1974, IBM capitalised on this body of experience by introducing their **Systems Network Architecture (SNA)**. This was a package of the hardware, peripherals, system software, networking software, cabling standards, and communication protocols necessary to set up a mainframe computing network. The development was guided by, and fully compatible with, the aforementioned **Open Systems Interconnection (OSI) Reference Model** [reminder] (albeit it substituted IBM's own house terminology whenever it could), and automatically took care of such everyday networking problems as line or modem failures and node overloads. Upgraded functionality was later provided to cope with the subtly different requirements of microcomputer systems, and the resulting product was called **Advanced Peer-to-Peer Networking (APPN)**.

The OSI Model also helped steer the design process as the demand arose for what is today known as **"client-server" networking**. This is the name given to Internet-type networks in which the processor at one of the nodes has been set up as a **"server"**, specifically to service the data requirements at the remaining nodes. Client-server is not a product as such, but rather a conceptual "file-sharing architecture" model which manufacturers can map their specific products onto, and the main controlling

protocols are the US Department of Defence's TCP and IP (usually seen together as "TCP/IP"). Here are the technicalities ...

> **"Transmission Control Protocol" (TCP):** TCP is a "transport-level" protocol, and "is responsible for verifying the correct delivery of data from client to server". This means taking all necessary steps to detect accidental data loss in the network, and, once detected, taking further steps to have it retransmitted.
>
> **"Internet Protocol" (IP)**: IP is a "network level" protocol, and "is responsible for moving packets of data from node to node". This means taking all necessary steps to determine the optimal transmission path through the available intermediate routing stations.

3.3 The ICL-Fujitsu Experience, 1971 to Date

The ICL 1900s [see Section 1.8] were well-built third generation machines, and thus had a decent enough commercial life. Nevertheless, by the late 1960s they were beginning to show their age and the company started to consider how they should be replaced. Procter (1998) explains how plans for the upgrade started to take shape in 1969, when a "jury" was set up to evaluate various optional ways forward. They considered four options for what they were already referring to as their "new range", namely, (a) to develop the 1900-series, (b) to go for the "Basic Language Machine", a concept computer being worked on in-house by a team led by John Iliffe, (c) to buy in Manchester University's "Mark V", the MU5, or (d) to go for the "synthetic option", a combination of the others. They decided on the latter, and, after five years intensive work, the 2900-series was launched in April 1974, complete with a highly versatile operating system known as the **Virtual Machine Environment (VME)** [Wikipedia briefing].

Now the 2900s were typical fourth generation systems. They were robust and efficient, and could cope with the heavy processing loads presented by transaction processing and database applications, and they made good use of the newly arrived **"Fourth Generation Languages"**, or "4GLs" ...

> **Key Concept - Fourth Generation Languages:** A 4GL is a proprietary software package designed to write 3GL code for a programmer more cost-effectively than s/he could do it her/himself. 4GLs do this sort of **"code generating"** by constantly referring to previously entered **"metadata"** - data about data - contained in a **"data repository"** or **"data dictionary"**. Programmers using 4GLs are thus relieved of the need to become experts in the underlying 3GL. Unfortunately, it is not unknown for the benefits of 4GLs to be overstated at point of sale. Yes they work, but the code generated is rarely anywhere near as optimised for processing efficiency as expert hand-written code. Accordingly, 4GLs should only be used when response time is not a critical issue.

The 2900s actually reached new heights of sophistication in their VME TP Monitor, TPMS. Here is how ICL described the transaction processing problem, and their solution to it .....

> "Transaction processing operates in its own environment known as a *transaction processing service*. The environment provided is one in which many people can hold simultaneous conversations with computer programs about some business they have in common. The computer's store is used to maintain the business information on which the users of the service all depend, and it can be interrogated directly about that information from a computer terminal and the answers to critical questions can be obtained with only seconds delay. Moreover, only a few copies of the relevant applications programs are required to support a high volume of simultaneous enquiries from a large number of connected terminals. [//] A transaction processing service and its users interact by means of *message-reply pairs*. The user inputs a *message* or query at his terminal as the first half of the pair. This is then processed by a user-written *application program*. Finally, the service provides an appropriate response to that message as the second half of the message-pair and sends a reply to the terminal of origin. Typically, the message concerns one of the many transactions that make up the daily life of the business, for example the receipt or despatch of an order, an invoice amendment or stock update, and frequently it will cause the service to update the associated data files. The information in these files will therefore always reflect the overall state of the business, and can be used to answer queries or generate reports

as and when required, using completely up-to-date information." (International Computers Limited, 1987, p1; italics original)

The way TPMS organised its application software was to bundle together sub-selections of applications programs, and to load these as compound object code files known as **Application Virtual Machines (AVMs)**, thus ...

"The program modules which process the various messages may also be grouped into related sets, known as *AVM types*, which provide a means of controlling the allocation of system resources and of optimising performance. Any one AVM type may contain one or many different program modules, and there may be many duplicate copies of a particular AVM type concurrently active. This means that a number of identical or similar messages may be input from different terminals, and processed simultaneously. (International Computers Limited, 1987, p5; italics original)

The question then arises, of course, as to what controls the moment-to-moment deployment of the AVMs, and the answer is "a set of controlling modules" known as a **Control Virtual Machine (CVM)**. Here is ICL's description of this TPMS facility ...

"The CVM, as its name implies, is in control of the TPMS service. Its main functions can be summarised as follows: 1. Control of the terminal network, including (a) connection and disconnection, and (b) input of messages and output of replies. 2. Formatting of replies incorporating the required template. 3. Routing of messages to the correct application code for processing, in the required priority order. 4. Control of privacy on messages according to terminal user. 5. Control of AVM concurrency according to the actual service requirements at any time. 6. Statistics collection. 7. Control of run-time changes to the service necessitated either (a) by control commands issued by the service controller, or (b) by failures within parts of the system; for example, failures in files or application code. 8. Logging of messages. 9. Control of service start-up and close-down, including any recovery on a restart" (International Computers Limited, 1987, p4).

So what happens when a user wants to make an enquiry which either (a) cannot be completed within the three or four seconds typically allocated to a given message-pair, or else (b) produces results which will not fit within the dimensions of the available screen display? Well the answer is to do a screen-by-screen display, with each screen showing a subset of the total output. Of course, it takes time for the user to do whatever it is s/he wants to do with each screen-load of output, which means, in turn, that subtotals etc. and restart pointers such as database keys have to be safely maintained from one screen display to the next. This problem was solved by the use of a temporary storage facility known as **"partial results"**. This is how ICL describe this facility .....

"In the simplest case, a transaction consists of just one message and one reply, in which case it is known as a *single-phase transaction*. However, many transactions comprise more than one interaction between the terminal operator and the TP service. Each of these interactions may produce intermediate data that is to be used in conjunction with the input to the next phase of the transaction. Such transactions are known as *multi-phase transactions*. Data produced by one phase and saved for use in one or more subsequent phases is known as partial results, and is stored [in a system file] between phases, along with other recovery data generated to enable the TP service to be restarted tidily after a service break." (International Computers Limited, 1987, p5; italics original)

Key Concept - Partial Results: Partial Results is the name given to a relatively small area of Main Memory reserved by a TP Monitor to allow a program to pause and recommence across processing phases, thus providing users with an apparently continuous display across what is objectively a discontinuous exchange. With a multi-phase enquiry program, Partial Results would typically be used to "keep one's place" in the underlying data so that display *n+1* could begin where display *n* left off, and with a multi-phase update program (where convention requires that file updates are only ever done in the concluding phase), it would be used to carry forward potential updates in short-term storage until committal to long-term storage was required. **<< AUTHOR'S NOTE: The brain is constantly having to keep track of things across the processing phases of complex operations, and it is tempting to speculate that some form of mental partial results might be involved. >>**

Two further useful concepts from the world of transaction processing are those of the **"success unit"** and the **"rollback unfinished"** .....

> **Key Concepts - "Success Units" and "Rollback":** A processing success unit is a succession of referentially related file or database update operations **which - because in some way they belong together - cannot sensibly be interrupted by a system failure**. For example, if you want to book a flight and the system fails after having taken your name and address, but before having taken your destination, then those partial updates should be forcibly discarded as soon as the system has been recovered, and the transaction begun again in its entirety. What the system does, therefore, is to set an **"unfinished indicator"** immediately before the first partial update, and unset it only after full and satisfactory completion. In the event of processing failure during the updates, any updates coded as **"unfinished"** in this way can then be "rolled back", that is to say, reversed out. This form of **"rollback unfinished"** recovery action is a way of maintaining data integrity in an imperfect world. It recognises the fact that all systems are bound eventually to fail, and that when they do they may be part-way through a number of logically related file updates. **<< AUTHOR'S NOTE: Reaching the end of a success unit is like taking a belay when climbing a cliff or executing a <SAVE> when word processing: you are safe against the unexpected, but only for the present. Now it may well be that biological cognition has its own version of the success unit, thus defined. The classic cite here comes from the Russian psychologist Bluma Zeigarnik, who noted that if subjects are exposed to an incomplete pass through something they are familiar with, it creates a state of tension within them, and plays upon their mind until some way is found of completing it (Zeigarnik, 1927). This state of tension is akin to the Gestaltist "Law of Closure", and is popularly known as "the Zeigarnik Effect". >>**

In due course, the 2900s were further upgraded in the mid-1980s by the 3900 series. We mention these, because they were reasonably typical fifth generation machines. They were technically state-of-the-art, had the class-characteristic "nodal architecture", and required minimal operator supervision.

To complete the sorry history of the British computer industry, ICL was taken over by Standard Telephones and Cables in 1984, who then sold it on to Fujitsu in 1990, who then forced it to concentrate on software rather than hardware provision. In 1997, the 3900s were replaced by the Fujitsu Trimetra range. The ICL marque was finally withdrawn 2nd April 2001.

3.4 Supercomputers, 1971 to Date

We left Control Data at the end of the third generation, trying to make a living from the low-sales-volume world of supercomputer design [see Section 1.5]. Time and again, Seymour Cray and his colleagues would push down clock speeds as far as the physics of the day would support, or do ever more things simultaneously in increasingly parallel architectures. However, Cray's initial design for a four-processor machine, the 8600, was too far ahead of its time to be practicable, and in 1972 he fell out with Norris, and resigned in order to go it alone as Cray Research. He put his ideas to work (albeit in somewhat scaled-down form) in the 1976 Cray-1, this time winning the title of the world's most powerful computer in his own name. The Cray-1 retained the built-in Freon cooling system from the CDC 6600, but brought in pipelining with four sets of instruction buffers, and introduced **"vector processing"** ...

> **Key Concept - "Vector Processing":** So far as the computer industry is concerned, a "vector" is an ordered list in memory. It has a starting address, a number of elements, and a constant "stride" (i.e., the address increment for each successive element). Once created, vectors can be used to speed up the processing of large arrays whenever an instruction needs to be carried out on every element. This calls for a specially designed CPU, complete with "vector registers" and an upgraded pipeline. It also requires careful software design to play to the strengths of the hardware, specifically because vectoring can only be done from the innermost loop of a DO loop (Chiang, 2003). The pay-off comes in reductions of up to 90% in cycles per element.

3.5 Expert Systems, "Inference Engines", and Fifth Generation Fever

The 1970s also saw the arrival of what were rather grandly called **"expert systems"** .....

> **Key Concept - Expert Systems:** "An **expert system** is a class of computer programs developed by researchers in artificial intelligence during the 1970's and applied commercially throughout the 1980's. In essence, they are programs made up of a set of rules that analyse information (usually supplied by the user of the system) about a specific class of problem, and provide an analysis or recommendation of the problem, and, depending on their design, a recommended course of action." (Wikipedia, 2003). Alternatively, "an expert system is a computing system which embodies organised knowledge concerning some specific area of human expertise [...] sufficient to be able to do duty as a skilful and cost effective consultant" (Michie, 1979). The body of organised knowledge is known as the **"knowledge base"**, and the software which interprets and applies that knowledge is known as the **"inference engine"**.

The first recognised expert system was called DENDRAL, and was developed in the period 1965 to 1975 by a student of Herbert Simons, Edward A. Feigenbaum. DENDRAL was designed to take the human expert out of the day-to-day medical decision making loop, and it did this by constant recourse to a knowledge base of pre-loaded rules and statements. The idea was that the expert knowledge of professors and senior consultants would be "captured" in the first instance by interview, and, once codified, would be made available through the machine for use by more junior grades.

DENDRAL inspired many imitators, including MYCIN, developed between 1972 and 1980 by a research team at Stanford University led by Edward Shortliffe. Here is MYCIN's conceptual specification .....

> "MYCIN is an interactive program that diagnoses certain infectious diseases, prescribes antimicrobial therapy, and can explain its reasoning in detail. In a controlled test, its performance equalled that of specialists. In addition, the MYCIN program incorporated several important AI developments. MYCIN extended the notion that the knowledge base should be separate from the inference engine, and its rule-based inference engine was built on a backward-chaining, or goal-directed, control strategy. Since it was designed as a consultant for physicians, MYCIN was given the ability to explain both its line of reasoning and its knowledge. Because of the rapid pace of developments in medicine, the knowledge base was designed for easy augmentation. And because medical diagnosis often involves a degree of uncertainty, MYCIN's rules incorporated certainty factors to indicate the importance (i.e., likelihood and risk) of a conclusion. Although MYCIN was never used routinely by physicians, it has substantially influenced other AI research" (source).

The area was often dismissed as marketing hype, and trials of expert systems were dogged by resistance from the field. What they did do, however, was inspire the Japanese, who in 1980 declared that the "fifth generation" of computing had arrived. The announcement was made by the Japanese Ministry of International Trade and Industry (MITI) as an attempt to deliver artificial intelligence based upon "large scale parallel processing". A major conference followed in October 1981 in Tokyo, where, in addition to expert systems, up-beat proposals were heard for distributed storage, high power DBMSs, and inference software to allow human-like knowledge-processing. It all went down rather too well, and in Britain, fifth generation fever inspired the creation of the Department of Trade and Industry's "Alvey Flagship initiative". Procter (1998) describes ICL's involvement in Alvey during the period 1983 to 1987, and names some of the high performance parallel machines which then followed.

Yet the Japanese had bitten off far more than they could chew, and project after project failed to deliver against the earlier promises. The fifth generation initiative was therefore stood down in the late 1980s. Scott Callon (1996) describes it as having been doomed to failure from the beginning, and as having been little more than "a public relations ploy" by MITI. Others have pointed to the over-reliance on the PROLOG programming language, which was not up to the task. There are then two schools of thought about where that left things. On the one hand there are those who believe it put the world back in the fourth generation, and on the other there are those who believe it ushered in the sixth generation. Either

way, a fortune awaits any cognitive scientist who can unravel the joint complexities of brain and mind and set the fifth generation going again. For further details, see Kahaner (1993). Callon (1997) provides an interesting exposé of Japanese bureaucratic participation in the hype and the technical failure, if interested. The latest products in the spirit of the fifth generation are heavily Internet-oriented, and include **"avatars"** and **"intelligent agents"** .....

> **Key Concept - Avatars:** In Hindu mythology, an avatar is an incarnation of a God (Wikipedia, 2003 online), that is to say, a way in which God makes h/self accessible to humans. Thus the God Vishnu may manifest himself as a fish, a tortoise, a boar, a man-lion, etc. The word was then adopted by the computer industry to refer to a two-dimensional virtual person, a latter-day "talking head", complete with a synthetic personality to complement its synthetic voice. To see the avatar *Vandrea* at work, click here; for her rival *Ananova*, click here.
>
> **Key Concept - Intelligent Agents:** An agent is a robot implemented entirely in software, and thus differs from an avatar in that it is able to do things off-screen, that is to say, within cyberspace itself. The idea arose in the early 1990s as a way of searching inadequately indexed databases such as the World Wide Web. You launch an agent applet (= small application) from your home terminal and it goes off, if necessary for days, to inspect web content site by site, looking for relevant content, but far more thoroughly than would be possible using the dozen or so keywords available on browsers like Google.

Jansen (1996) provides a convenient and detailed introduction to agent technology, and for more on expert systems, click here.

3.6 Neural Networks, 1971 to Date

The 1969 Minsky-Papert critique [see Section 1.11] brought about a period of soul searching, and prompted the development of more powerful methods, which culminated in the modern **"neural network"** (NN). Unlike perceptrons, modern NNs typically have *three* layers of artificial neurons. Now there is an input layer, an output layer, and - sandwiched between these - an "invisible", or "hidden", layer. All communication between the input and output layers has to go via the invisible layer, giving you two sets of connection weightings to play with, instead of one. These three-layered NNs soon proved far more competent than the older two-layered ones.

Now the beauty of NNs is that each electronic synapse is purpose-built to take care of its own weighting. The network, in other words, is free to acquire its knowledge the same way biological brains acquire theirs - **by learning from experience**. In computing terms, NNs are therefore very important **because they need no programming**. NNs are now being put to a growing variety of commercial applications, such as telling coin from coin in vending machines (they "recognise" the different bounces), checking your credit card spending pattern, and steering robots. They have also had a major impact on cognitive theory. Neuroscientists now see NNs as validating the traditional cell assembly concept: each weighted connection in an NN is simply the electronic equivalent of biology's variable synaptic strength. Indeed, McNaughton and Nadel (1990) rate NNs as such an important development of this concept that they term all such networks **"Hebb-Marr networks"** [glossary]. Other authors prefer the term **"matrix memory"** to convey the same underlying concept. And the new science even has a new name - **"Connectionism"** [see stand-alone handout].

And yet, as with expert systems, even NNs were running out of steam by the end of the 1980s. As they were built bigger and bigger and given harder and harder problems, failures started to be reported. The genre had come up against a major theoretical and practical barrier. A concise and highly illustrative example of the problem is provided by the Medical Research Council's Dennis Norris, who describes how a system of three interlinked small NNs was able to solve a problem which had been beyond the

competence of a single large one. In other words, there exist tasks for which you need **many separate NNs**; and which, moreover, you need to connect up in a very precise fashion. **Giving you, of course, a *network of neural networks*, in which the connections within AND between each module are both vitally important.** The secret, in other words, was to break the problem down into its logical components. "Instead of having to solve one large problem," Norris writes, "[the system then] simply had to solve three far smaller problems" (**Norris, 1991**, p295). The specific problem Norris gave his network was how, given any date in a century, to reply with what day of the week it was ...

Exercise - What Day of the Week?

Norris divided the date-day problem into three sub-problems. Can you do it in less, or would you prefer to do it in more? Familiarise yourself with this task. What day of the week was/will be .....

seven days ago

ten days' time

49 days ago

48 days ago

7000 days' time

6999 days' time

11th September 2001

18th March 2042

29th February 2080

"Going modular" in this way certainly helped, and two years later Hinton, Plaut, and Shallice (1993) built a modular connectionist network capable of rudimentary reading out loud. And - tantalisingly - their network of neural networks ended up being about as granular as the transcoding model popularised by Ellis and Young (1988).

3.7 Machine Translation, 1971 to Date

When we left MT in 1970, teams were busily drawing up semantic networks as mechanisms of providing the topic context which human translators rely on so much, and eagerly awaiting cheaper and faster machines to cope with the large dictionary files which were clearly going to be needed. Well the PC revolution of the 1980s gave them the hardware capacity they were looking for, and prompted a new wave of development, and we shall be looking at the network/context issue in Section 3.9 below. However, it is still very easy to catch these products out, for they hate any form of linguistic or cognitive complexity. Indeed, the standard advice to get the best out of the available MT tools is to use simple, grammatical, structures, write 20 words or so per sentence, avoid jargon, and if you insist on using more than one clause per sentence take care not to omit the "whos" and "thats" which mark the clause junctions (O'Connell, 2001).

3.8 Machine Chess and Problem Solving, 1971 to Date

Once Greenblatt's MacHack system had shown that chess playing software could defeat a human champion [see Section 1.12], system performance gradually improved throughout the 1970s and 1980s. The next major milestone was the 1988 **"Deep Thought"** system, built at Carnegie Mellon University by a team of graduate students including Feng-Hsiung Hsu and Murray Campbell, and capable of analysing

750,000 possible moves per second (Powell, 1997). An upgraded system was beaten by Gary Kasparov in 1989, but came back in 1993 to beat Judit Polgar. Not wishing to be outdone, IBM countered in February 1996 with **"Deep Blue"**, developed by a team led by C.J. Tan and advised by US chess champion Joel Benjamin. It ran on an IBM 32-node RS/6000, and could evaluate a phenomenal 200 million moves per second (Apte et al, *op. cit.*). Yet even this did not stop it losing four-two to Kasparov (although it won the 1997 rematch 3.5-2.5). Apte, Morgenstern, and Hong (2000) tell the Deep Blue story like this .....

> "Perhaps the most famous of IBM's game-playing programs is Deep Blue [...] which made headlines when it beat Gary Kasparov, thus becoming the first chess-playing program to defeat a reigning world champion. Developing a champion-class chess-playing program had long been a challenge problem for AI. Game-playing programs have traditionally used a combination of evaluation functions to determine how good a particular position is, and effective search techniques to search through a space of possible states. The difficulty in developing a champion-level chess-playing program lies in the facts that first, the state space is so large (since the *branching* factor or ratio, the number of states that can be reached from a given state of the game, is 30 - 40) and second, the sophisticated evaluation of chess positions can be difficult to develop and computationally expensive to perform" (p7).

As for the more general world of machine problem solving, the problem of context was again rearing its ugly head, although this time it went by the name of **"common sense" knowledge**. Programs were now getting quite adept at following rules and implications, provided they stayed within the immediate problem domain. But more and more lapses of general knowledge were being reported. The pivotal paper here was by the father of AI himself, John McCarthy, who pointed out in McCarthy (1959) that there was invariably far more to a problem than would initially meet the eye. There could be thousands of fragments of knowledge - some obvious, but others not so obvious - which might somehow prove to be relevant, and without that background general knowledge - the common sense - the machine simply did not know enough. In the next section, we look at attempts to get around the problem.

While we are here, however, there is time to introduce the concept of **"production systems"**, perhaps the best known theoretical framework for explaining real time human cognition. Here are the basic concepts, as taught on the Trinity College (Hartford[CT]) website ....

> "A production system is a model of computation that provides pattern-directed search control using a set of **production rules**, a **working memory**, and a **recognise-act cycle**. [//] The productions are rules of the form C -> A, where the LHS is known as the **condition** and the RHS is known as the **action**. These rules are interpreted as follows: *given condition, C, take action A*. The action part can be any step in the problem solving process. The condition is the pattern that determines whether the rule applies or not [//] **Working memory** contains a description of the **current state of the world** in the problem-solving process. The description is matched against the conditions of the production rules. When a condition matches, its action is performed. Actions are designed to alter the contents of working memory. [//] The **recognise-act cycle** is the control structure. The patterns contained in working memory are matched against the conditions of the production rules, which produces a subset of rules known as the **conflict set**, whose conditions match the contents of working memory. One (or more) of the rules in the conflict set is selected (**conflict resolution**) and **fired**, which means its action is performed."

The production system is thus an eclectic combination of the cybernetic loop, with the "TOTE" control cycle proposed by **Miller, Galanter, and Pribram**'s (1960) **"Plans and the Structure of Behaviour"** [Amazon], but extended by the concept of working memory imported from computer science. It has been suggested that the concept of the production rule derives originally from the theories of the Polish logician, Emil Post (e.g., Post, 1943), but the modern theory is the work of Carnegie Mellon University's John R. Anderson (e.g., Anderson, 1976, 1983, 1990, 1993). Anderson's production system is called ACT* (pronounced "act-star"), and relies on the distinction between declarative, procedural, and

working memory, with declarative memory being presumed to be "a semantic net linking propositions, images, and sequences by associations".

> **Key Concept - "Production Systems" and "Production Rules":** "A production system consists of a collection of if-then rules that together form an information-processing computer simulation model of some cognitive task, or range of tasks" (Young, 2003).

3.9 Semantic Networks Revisited

We are now tangentially mentioning semantic networks in just about every section, so this is probably a good point to bring their story, too, up to date. We have, of course, consistently taken the line that semantic networks are by far the most important of all the AI threads opened up in Part 4 (Section 4), for the simple reason that they promise a powerful explanatory package to anyone attempting to make sense of biological memory. And many agree. This, for example, is how Simon (1983) puts it .....

> "..... in recent years, there has been considerable interest in systems for drawing inferences from information stored in large data bases, often in the form of semantic nets." (Simon, 1983, p9)

However, the worker who has done most [in 2003, remember] to develop physical storage networks is **Doug Lenat** [Wikipedia biography], who, in an attempt to provide computers with common sense knowledge, has spent 20 years loading a computer database with hundreds of thousands of rules and definitions. The resulting product is Cyc (pronounced "sike") - part electronic encyclopaedia, and part sleeping giant .....

> "The Cyc product family is powered by an immense multi-contextual knowledge base and an efficient inference engine. This knowledge base is built upon a core of over 1,000,000 hand-entered assertions (or 'rules') designed to capture a large portion of what we normally consider consensus knowledge about the world. For example, Cyc knows that trees are usually indoors, that once people die they stop buying things, and that glasses of liquid should be carried rightside-up" (2003 corporate promotion, now lapsed).

There are also a number of network offerings in the Richens - Masterman - Halliday - Ceccato (psycho)linguistic tradition [see Section 1.9]. For example, the link from Halliday's "systems" to Sydney M. Lamb's **"relational networks"** is explicit .....

> "A symposium at Rice University in 1984 featured among others, papers by Lamb and Halliday and their co-workers, and focused on a comparison of systemic analyses with various cognitive analyses of a common oral text captured on video [...] Halliday's functional work has influenced Lamb's theoretical work in a number of important ways, and vice-versa. Lamb's relational work notation took important cues from Halliday's systemic diagrams ....." (Copeland, 2000)

Then there is the work of Surrey University's Ian J. Thompson on **"layered cognitive networks"**. Thompson begins by summarising the recent history of "semantic nets", thus .....

> "Semantic nets [various citations] are an attempt to represent the meanings of sentences by means of a formal network structure. Various forms of these structures are possible [citations], but typically objects are represented by nodes in a network, their relations by arcs between the nodes, and inferences involving them by production rules which manipulate the network" (Thompson, 1990).

It is, however, difficult to build nets capable of loosely applying prior knowledge. After all, Thompson explains, we immediately recognise a four-leafed clover, despite the fact that there can be no pre-existing node for it. He goes on to discuss how a layered net architecture might assist .....

"The ability to recognise *new* instances of a specific concept means that the concept-node cannot be activated by purely historic links, and means that some essential kind of rule-based operation will have to be built into the system. Once this is done, I argue that both the symbol and the processing features of traditional symbolic systems can be incorporated within connectionist networks. [//] The architecture proposed to handle both connectionist links and pattern-directed rules involves *layers* of distinct networks, so that the relations within a layer are given explicitly by the links of the graph, whereas the relations between layers have a functional or rule-based interpretation. Specific types of semantic contents will be proposed for the distinct layers, guided to some extent by Piagetian theories of development" (Thompson, 1990).

Finally under this heading we must re-introduce **John F. Sowa** [previously mentioned in Part 4 (Section 4.2)], who has capitalised on his experience turning data models into databases by serving on the advisory panel during the drafting of the American National Standards Institute (ANSI) standard for **"Conceptual Graph Interchange Form" (CGIF)**. This is an attempt to specify "the syntax and semantics of conceptual graphs (CGs) and the representation as machine-readable character strings in [CGIF]" (Source). To conform to this standard, an IT system must be able to read in a sequential description of a conceptual network, use those data to establish the physical network within its own filestore, and then be able to convert that network, duly updated if necessary, back out into a sequential description again, for onward transmission. Here is how he sees the network content <John is going to Boston by bus> translating into CGIF syntax .....

[Go *x] (Agnt ?x [Person: John]) (Dest ?x [City: Boston]) (Inst ?x [Bus])

3.10 Robotics, 1971 to Date

When we introduced the science of robotics in Part 4 (Section 4.6), we saw how the concept of the robot emerged from antiquity, firstly into science fiction, then into remote controlled weapon technology, and then into guided missile technology and modern military robotics. Nowadays, however, the science is mature enough to support "pure" research as well as applied, and probably the single most influential figure in modern robotics is **Rodney A. Brooks** [Wikipedia biography], Director of the MIT Artificial Intelligence Laboratory and Fujitsu Professor of Computer Science. Having joined MIT in 1984, his first headline-grabbing development was of a six-legged insect walking robot called Genghis, where the design problem was how to synchronise six simple limbs into one overarching system of directed locomotion. The solution was to strap together 57 separate processors, 48 working at reflex level (eight per leg!), four more working to deliver level-platform coordination, and five more (and only five) dedicated to "central control" (Brooks, 1993, p359).

Brooks used this experience to draw a broader point ...

"This exercise in synthetic neuroethology has successfully demonstrated [that] higher-level behaviours (such as following people) can be integrated into a system that controls lower level behaviours such as leg lifting and force balancing, in a completely seamless way. There is no need to postulate qualitatively different sorts of structures for different levels of behaviours and no need to postulate unique forms of network interconnect to integrate high-level behaviours. [//] **Coherent macro behaviours can arise from many independent micro behaviours [and] there is no need to postulate a central repository for sensor fusion to feed into**" (Brooks, 1993, pp362-363; bold emphasis added).

Other MIT projects have since focussed on different areas of biological cybernetics and/or cognition .....

Coco: Coco is a quadruped robot with "gorilla-like" proportions, and articulated arms, legs, neck, and eyes. It is being used for research into postural and gaze adjustment during locomotion and visual inspection of the world.

Cog: Cog is an attempt to coordinate robotic eyes, head, and hands, and to use the resulting exploratory subsystem to discover for itself things about its world.

Kismet: Kismet is the robot with the highly mobile eyes and eyebrows, just like "Johnny Five" in the movie "Short Circuit". **<<AUTHOR'S NOTE: Our species is highly sensitive to facial body language, which is capable of changing the emotional underpinning of ongoing cognition in moments. Mehrabian (1971) has estimated that facial expression is responsible for approximately 55% of the information transfer during normal one-to-one communication, with tone of voice accounting for a further 38%, and the words themselves for a mere 7%! >>**

Another guru of modern robotics is **Hans Moravec** [Wikipedia biography], Principal Research Scientist at the Robotics Institute at Carnegie Mellon University. Moravec is particularly concerned with going beyond the engineering (important though this clearly is) and contemplating the nature of the robotic mind, past, present, and future. He believes we still have a long way to go .....

> "It is not the mechanical 'body' that is unattainable; articulated arms and other moving mechanisms adequate for manual work already exist, as the industrial robots attest. Rather it is the computer-based artificial brain that is still well below the level of sophistication needed to build a humanlike robot." (Moravec, 1999, p86)

In Moravec's view, we are approaching robotics' third generation quantum leap. He profiles **"third generation robots"** (3GRs) as being able "to learn very quickly from mental rehearsals in simulations" (p93), and as requiring about 5 million MIPS of brainpower to do so. 4GRs would have "a humanlike 100 million MIPS" and "be able to abstract and generalise". Robot intelligence will then start to exceed human by about 2050!

3.11 The T-T-T-Turing Test

The Turing Test has also gone modern, in the form of a competition, the **Loebner Prize** ($100,000). This competition was established in 1990 by Hugh Loebner, a New York philanthropist, and the Cambridge Centre for Behavioural Studies. It is held each year, and (so far at least) the judges have always won. Specifically, they have always been able to phrase the all-important man-machine conversation in such a way as to catch the computers (by which we really mean the *programmers* of those computers) out. Bronze models and $2000 consolation prizes are awarded annually to the "most human" runner-up. For details of the competition, including the names of past winners and details of their entries, click here. The 2000 and 2001 bronze awards went to Richard S. Wallace's Alicebot system. This was first developed in 1995 written in the SETL programming language, and was then migrated to the Java language in 1998.

Others, meanwhile, have been moving the goalposts. For example, Harnad (1991) points to a major weakness in the original Turing Test (TT), namely that it is executed over a keyboard-to-keyboard link. In his view, Turing was wrong to discount bodily appearance, and he proposed a more stringent test, the **"Total Turing Test" (TTT)**, as follows ...

> "Most people have the intuition that these machines may be doing something clever, but not clever enough. There's also doubt that the machines are doing it the 'right way'. We, after all, are not digital computers [...] The only trouble is that those who know what kind of 'hardware' we are - the anatomists and physiologists, especially those who study the brain - have absolutely no idea how our mind works either. So our intuitive conviction that computers work the wrong way is certainly not based on any knowledge of what the 'right' way is. [...] The candidate must be able to do, in the real world of objects and people, *everything* that real people can do, in a way that is indistinguishable (to a person) from the way real people do" (Harnad, 1991, pp43-44).

Then there is the **"Total Total Turing Test" (TTTT)**. This, writes Harnad, calls "for Turing indistinguishability right down to the neurons and molecules [but] my own guess, though, is that [the] TTT already narrows down the degrees of freedom sufficiently" (*Ibid.*, p53). The TTTT was soon followed by Bringsjord's (1994) **TTTT***, which insists on there being a structural equivalence of machine and human at flowchart level, and Schweitzer's (1998) **"Truly Total Turing Test" (TRTTT)**, which points out weaknesses with Harnad's TTT, namely that we are being asked to make judgements about a species - intelligent machines - of which we have no real prior experience. Computers are "toy world" entities, he argues, not real world, and his proposed TRTTT therefore looks for long term evidence of machines accumulating achievements of their own, comparable to human achievements. Machines now need to invent languages and board games, Schweitzer suggests, not just use them. So all in all, it is perhaps fortunate that Turing himself is no longer with us, for, being a stutterer, he might have taken all this t-t-testing personally!

3.12 Machine Consciousness Revisited

> "The question of whether computers can think is like the question of whether submarines can swim" (Edsger Dijkstra).

The various forms of the Turing Test are now merely part of a newer, much larger, science - consciousness studies. The last two decades have seen consciousness become a legitimate topic for scientific investigation, and so much interest has this generated that the science has recently been described as "the race for consciousness" (Taylor, 1999). Molecular biologists and quantum physicists now vie with neurologists, linguists, and cognitive psychologists for the next critical discovery, while philosophers alternately baffle and inspire them all. Unfortunately, all the old problems still remain. The issue of embodiment [see Section 1.13], for example, has grown in importance, not just because it poses an explanatory challenge in itself, but also [see Section 3.10] because body language modulates consciousness much of the time. **Lakoff and Johnson**'s (1999) **"Philosophy in the Flesh"** [Amazon] is perhaps the most widely read source on the subject. Here is how the authors state the problem ...

> "Any reasoning you do using a concept requires that the neural structures of the brain carry out that reasoning. Accordingly, the architecture of your brain's neural networks determines what concepts you have and hence the kind of reasoning you can do. [...] We have inherited from the Western philosophical tradition a theory of faculty psychology, in which we have a 'faculty' of reason that is separate from and independent of what we do with our bodies. In particular, reason is seen as independent of perception and bodily movement. In the Western tradition, this autonomous capacity of reason is regarded as what makes us essentially human, distinguishing us from all other animals. [...] The evidence from cognitive science shows that classical faculty psychology is wrong. There is no such fully autonomous faculty of reason separate from and independent of bodily capacities such as perception and movement. The evidence supports, instead, an evolutionary view, in which reason uses and grows out of such bodily capacities. [...] These findings [...] are profoundly disquieting in two respects. First, they tell us that human reason is a form of animal reason, a reason inextricably tied to our bodies and the peculiarities of our brains. Second, these results tell us that our bodies, brains, and interactions with our environment provide the mostly unconscious basis for our everyday metaphysics, that is, our sense of what is real" (Lakoff and Johnson, 1999, pp16-17).

Nowhere are the effects of embodiment more clearly to be seen, the argument then runs, than in the pervasiveness of embodied metaphor in everyday language ...

> "Our subjective mental life is enormous in scope and richness. We make subjective judgements about such abstract things as importance, similarity, difficulty, and morality, and we have subjective experiences of desire, affection, intimacy, and achievement. Yet, as rich as these experiences are, much of the way we conceptualise them, reason about them, and visualise them comes from other domains of experience. These other domains are mostly sensorimotor domains [citation], as when we conceptualise understanding an idea (subjective

experience) in terms of grasping an object (sensorimotor experience) and failing to understand an idea as having it go right by us or over our heads. The cognitive mechanism for such conceptualisations is conceptual metaphor, which allows us to use the physical logic of grasping to reason about understanding. [//] Metaphor allows conventional mental imagery from sensorimotor domains to be used for domains of subjective experience. For example, we may form an image of something going by us or over our heads (sensorimotor experience) when we fail to understand (subjective experience). A gesture tracing the path of something going past us or over our heads can indicate vividly a failure to understand. [//] Conceptual metaphor is pervasive in both thought and language. It is hard to think if a common subjective experience that is not conventionally conceptualised in terms of metaphor" (*Ibid*., p45).

And to make matters even worse, new problems continue to be raised. For example, the 1990s had opened with a most insightful observation as to what the true sequence of language processing might be by the University of London's Max Velmans. Velmans (1991) began by reviewing data from the selective attention literature [see, for example, our **e-précis of Cherry (1953)**], the general thrust of which was (a) that there had to exist a degree of "preconscious analysis" of incoming messages, and (b) that, although involuntary, this nevertheless presents a considerable extra workload to the cognitive system. The fascinating but inevitable consequence of preconscious analysis is that we must become aware of what we are saying at about the same time as our listeners become aware of it! Thus .....

> "In assessing whether the planning of *what* to say is conscious, it is hence instructive to examine what one experiences during a hesitation phrase (where we have good reason to infer such planning to be taking place). This simple thought experiment reveals that during a hesitation pause one might experience a certain sense of effort (perhaps the effort to put something in an appropriate way), but [...] no more is revealed of conceptual or semantic planning in hesitation pauses that is revealed of syntactic planning in breathing pauses. **The fact that a process demands *effort* does not ensure that it is *conscious*. Indeed, there is a sense in which one is only aware of what one wants to say after one has said it!**" (Velmans, 1991, pp663-664; italics original; bold emphasis added).

This line of argument was then taken up by Gray (1995), who incorporated it into his **"comparator system"** theory of consciousness, thus ...

> "..... the contents of consciousness consist of the outputs of a comparator system [refs] that has the general function of predicting, on a moment-by-moment basis, the next perceived state of the world, comparing this to the actual next perceived state of the world, and determining whether the predicted and actual states match or do not [...]. The hypothesis states, therefore, that the contents of primary awareness consist of subjective qualities whose neural and computational equivalents have been processed by the comparator system, and determined by that system to be familiar or novel."

Anatomically, Gray places this system as follows ...

> "..... the heart of the comparator function is attributed to the subicular area [helpful diagram]. This is postulated (1) to receive elaborated descriptions of the perceptual world from the entorhinal cortex [helpful diagram], itself the recipient of input from all cortical areas; (2) to receive predictions from, and initiate generation of the next prediction in the Papez circuit [helpful diagram] [...]; and (3) to interface with motor programming systems [...]. Second, the prefrontal cortex is allotted the role of providing the comparator system with information concerning the current motor program (via its projections to the entorhinal and cingulate cortices, the latter forming part of the Papez circuit). Third, the monoaminergic [useful definition] pathways which ascend from the mesencephalon to innervate the [septal area, entorhinal cortex, dentate gyrus, hippocampus, and subicular area] are charged with alerting the whole system under conditions of threat and diverting its activities to deal with the threat; in the absence of threat, the information-processing activities of the system can be put to other, nonemotional purposes."

Of course, the fact that consciousness studies is still a comparatively young science, means that articles are still debating how we are actually going to recognise machine consciousness when we eventually

see it. For example, Aleksander and Dunmall (2003) have recently proposed a number of tests for **"minimal consciousness"**. They define "being conscious" as follows ...

> "Being conscious is defined by '*having a private sense*: of an "out there" world, of a self, of contemplative planning and of the determination of whether, when and how to act'" (Aleksander and Dunmall, 2003, p8).

Aleksander and Dunmall then identify three basic types of test, namely behavioural, introspective, and "tests based on a knowledge of mechanism and function" (p9). The first type can be immediately discounted, they argue, because the behaviours of conscious and non-conscious systems are often identical. Nor are introspective data any safer, because they lack third party objectivity. The real question, therefore, **"is what mechanism does the conscious agent need to possess for *it* to have a sensation of its own presence"** (p9). They then propose five axioms, as follows (where *A* is "an agent in a sensorily-accessible world *S*") ...

> **"Axiom 1 (Depiction):** *A* has *perceptual states* that depict parts of *S*.
>
> **Axiom 2 (Imagination):** *A* has internal *imaginational states* that *recall* parts of *S* or *fabricate* *S*-like sensations.
>
> **Axiom 3 (Attention):** *A* is *capable of selecting* which parts of *S* to depict or what to imagine.
>
> **Axiom 4 (Planning):** *A* has *means of control* over imaginational state sequences to *plan actions*.
>
> **Axiom 5 (Emotion):** *A* has additional *affective states* that evaluate planned actions and determine the ensuing action." (*op. cit.*, pp9-10)

The authors then use these five axioms to derive specific hypotheses, and plead for close cooperation between neuroscience and computer science, for the simple reason that "hypotheses in neurology can be tested by computation" (p17).

In another recent paper, Cotterill (2003) tells us about "CyberChild", a detailed simulation of the neural systems involved in some of the most basic biological systems of the human infant, namely voice activation, limb activation, stomach state, and bladder control. Critically, "the simulated child has its own drives" (p33), so that the system can be left to its own devices in deciding when to initiate this or that behaviour. Its creators thus become observers of system behaviour, in the same way that the ethologists made a hard science out of observing animal behaviour. As to the all-critical underlying mechanisms, Cotterill has argued in a number of papers since 1995 that consciousness and a cybernetic phenomenon known as **"efference copy"** are intimately related, thus ...

> "... we have just been emphasising the importance of *sequences* of muscular movements, so we must ask what might be involved in the appropriation of new *context-specific reflexes*, that is to say reflexes which are galvanised into action only when triggered by the correct sequence of elements in the environmental feedback. [Such] acquisition makes surprisingly large demands on a nervous system [and] is possibly seen only in mammals (though perhaps also in the birds). **In fact, we will argue that it actually requires consciousness, which thus becomes the *raison d'etre* of the phenomenon**" (Cotterill, 2001, p9; italics original, but bold added).
>
> "internal feedback made possible by the efference copy routes [and] the premotor and supplementary motor regions [of] the cerebral cortex [are] the key sources of that feedback" (p32).
>
> **Key Concept - "Efference Copy" and "Reafference":** These terms were introduced by Von Holst and Mittelstaedt (1950) to describe a clever way of coping with the potentially excessive feedback found in most biological systems. The problem is that motor activity - **"efference"** - does not just induce movement, but also affects what is picked up by the senses. As soon as you start moving, your proprioceptors will tell you about limb position and balance, your cutaneous receptors will tell you about changes in touch, pain, temperature, and pressure, your homeostatic systems will signal requests for blood pressure and blood glucose maintenance,

and your special senses (eyes and ears) will detect the changing visual and auditory shape of the world. ***The senses, in short, are involved every bit as much in motor activity as are the motor pathways.*** What efference copy systems do, therefore, is subtract what you expect your senses to tell you next from what they actually tell you next. This gives zero if things are going to plan, but a non-zero error signal if they are not. This comparison is achieved by momentarily storing an image of the main motor output - the **"efference copy"** - and by then monitoring what is subsequently received back from the senses - the **"reafference"**. The principal benefit, of course, comes when the two flows totally cancel each other out, because this leaves the higher controller free to get on with more important things. It is a cybernetic trick which deploys a large short term memory resource with a view to simplifying subsequent processing [to see these information flows shown graphically, go to our e-paper on "Cybernetics" and take a look at Figure 2]. Every now and then, however, the system encounters some sort of external obstacle, or **"perturbation"**. This causes the reafference *not* to match the efference copy, and this, in turn, causes the higher controller to be interrupted with requests for corrective action. And because the sensors in an efference copy system thereby become capable of confirming for themselves that the effectors are working to plan, this is a highly efficient way of reducing unnecessary network traffic. Here are the formal definitions: "[Efference copy is] basically a copy of the motor output [and] in some way represents the expected pattern of sensory messages that would be produced in the absence of external interference with the action of the effectors. [Reafference is] the message returned to the nervous system, in consequence of the issue of the command" (Roberts, 1978:141). **<< AUTHOR'S NOTE: Gray's comparator theory and Cotterill's efference copy theory both remind us that the nervous system is a control system, and, as such, is no less subject to the laws of control systems simply because it is biological. Note the use of the word "interrupted" above - processing interrupts are vital to modular cognition, but are rarely discussed in cognitive science. >>**

3.13 Networks, Speech Acts, and Consciousness

What we are increasingly seeing, therefore, is a blurring of the lines of demarcation between philosophy, neuroscience, AI research, and mainstream computer science. For example, the UK Computing Research Committee, a British Computer Society panel chaired by (ex-Pegasus man) Sir Tony Hoare, has recently identified two of the biggest problems as (a) "developing systems that never fail", and (b) as "getting computers to act like humans" .....

> "'Understanding the brain is arguably the most fundamental and important scientific challenge facing mankind in the 21st century', the professors say. [//] 'The proposal is to create a computational architecture of the brain and mind which is inspired both by the neuronal architecture of the brain and high level cognitive functioning in humans, captures the information processing principles present in the brain, and describes how low level neuronal processes are linked and integrated with high level cognitive capabilities such as adaptability, self-awareness, and creativity [...] Such systems could work as domestic robots, helping for example to care for disabled people, and provide new facilities for teaching, entertainment, and intelligent access to information." (Hoare, 2003, p19)

To achieve this, we need to integrate many separate lines of enquiry, not least agreeing a definition of consciousness and solving the problems of embodiment and language. What we need to do, therefore, is to bring networks linguistics, comparative psychology, and cognitive philosophy together. Sowa (2002) is already there. Consider .....

> "The lack of progress in building general-purpose intelligent systems could be explained by several different hypotheses: (1) Simulating human intelligence on a digital computer is impossible. (2) The ideal architecture for true AI has not yet been found. (3) Human intelligence is so flexible that no fixed architecture can do more than simulate a single aspect of what is humanly possible. [//] Many people have presented strong, but not completely convincing arguments for the first hypothesis [citations ..... Others] have implemented a variety of at best partially successful designs. The purpose of this paper is to explore the third hypothesis: propose a flexible modular framework that can be tailored to an open-ended variety of architectures for different kinds of applications. [...] The idea of a flexible modular framework (FMF) is not new. [Its] philosophy is characterised by four design principles: (1) A small kernel that provides the basic services of resource allocation and process management. (2) A large, open-ended collection of highly modular utilities, which can be used by

> themselves or be combined with other modules. (3) Glue languages, also called scripting languages, for linking modules to form larger modules or complete applications. (4) A uniform data representation, based on character strings, which constitute the storage [and transmission] format. [...] The most promising candidate for a glue language is Elephant 2000, which McCarthy (1989) proposed as a design goal for the AI languages of the new millennium. [...] **As an inspiration for Elephant, McCarthy cites the *speech acts* of natural languages**" Sowa (2002; bold emphasis ours).

Other workers in this area are the Australian researchers Graeme Halford, Bill Wilson, and Steven Phillips, who have published a number of papers promoting the importance of relational systems to higher cognitive functions - see, for example, Halford, Wilson, and Phillips (2003).

But still the philosophical problems loom large, and we will close with the philosopher David J. Chalmers, then at the University of California, Santa Cruz, getting to the heart of the first-person problem. Chalmers reviewed progress in neuroscience and identified his now famous **"hard problem"**, as follows ...

> "Researchers use the word 'consciousness' in many different ways. To clarify the issues, we first have to separate the problems [and] for this purpose, I found it useful to distinguish between the 'easy problems' and the 'hard problem' of consciousness. [The] easy problems of consciousness include the following: How can a human subject discriminate sensory stimuli and react to them appropriately? How does the brain integrate information from many different sources and use this information to control behaviour? How is it that subjects can verbalise their internal states? [these questions] all concern the objective mechanisms of the cognitive system. Consequently, we have every reason to expect that continued work in cognitive psychology and neuroscience will answer them. [//] The hard problem, in contrast, is the question of how physical processes in the brain give rise to subjective experience. This puzzle involves the inner aspect of thought and perception: the way things feel for the subject. When we see, for example, we experience visual sensations, such as that of vivid blue. Or think of the ineffable sound of a distant oboe, the agony of intense pain, the sparkle of happiness or the meditative quality of a moment lost in thought. All are part of what I am calling consciousness [and] it is these phenomena that pose the real mystery of the mind. [...] Remarkably, subjective experience seems to emerge from a physical process. But we have no idea how or why this is" (Chalmers, 1995, pp62-64).

4 - Still to Come

That concludes our review of the history of computing technology. We have moved slowly and (we hope) surely from the groundwork inventions of the 18th and 19th centuries [see Part 1] through the heydey of the analog computer to the emergence of the digital alternative [see Part 2], and thence to the number-crunching pioneers who cracked codes, programmed air defence systems, and designed thermonuclear devices [see Part 3]. This then set the scene for a progressive walk through the concepts and inventions which have made modern computing what it is today, pausing regularly to highlight this or that clever use of computer memory, and searching all the time for potentially fruitful explanatory metaphors to assist those struggling to explain "the go" of biological information processing. In Part 6 we shall be looking in greater detail at the various ways computer memory can be used, and we do this for two products in particular, namely (a) the COBOL programming language, and (b) the IDMS DBMS. And finally, in Part 7 we undertake a comparative review of how well the full richness of the subtypes of computer memory has been incorporated into psychological theory.

5 - References