Chapter 1: The Beginning

1. What is the difference between a *dedicated* and *general-purpose* computer?

   SOLUTION

   A *dedicated computer* is constructed to solve a specific problem and cannot be used to solve other unrelated programs. For example, a computer in a microwave oven is able to perform only operation relevant to cooking. A computer may be dedicated because it is hardwired (i.e., its logical structure is fixed) or it may be dedicated because its internal software (firmware) is written to solve one class of problem only. Pocket calculators and GPS navigation systems (sat nav) are examples of dedicated computers.

   A *general-purpose computer* can be programmed to solve any problem because its operations are determined by a user-written program stored in its memory. The PC and Apple iMac are examples of general-purpose computers.

2. Is an aircraft's automatic pilot an example of a dedicated or a general-purpose computer?

   SOLUTION

   The auto-pilot is a dedicated computer because it solves only a single class of problems (controlling the aircraft's flight path). However, because of the complexity of the modern autopilot, it is highly probable that there is a general-purpose computer at the heart of an autopilot. This computer uses a fixed program in read-only memory. However, this program will be in flash memory that can be user-updated to deal with bug-fixes and new revisions of the software.

3. We said that the pattern of 1s and 0s used to represent an instruction in a computer has no intrinsic meaning. Why is this so and what is the implication of this statement?

   SOLUTION

   There is no intrinsic or natural meaning to a binary pattern such as 11001010. Similarly, there is no natural ordering of the bits; whether we write 11001010 or 01010011 (called Big Endian and Little Endian) is a convention. There is no reason why the bit order can't be 7,0,6,1,5,2,4,3 instead of 7,6,5,4,3,2,1,0 and the pattern written as 10110001 (thus scrambling the bits).

   Binary 1s and 0s can be associated with anything. The 1s and 0s can represent numbers, letters, names, instructions, images, and sound – anything we want them to.

   The computer designer decides how the individual bits of a computer word are grouped and what instructions are assigned to them. For example, the first 6 bits of an instruction might define the operations, so that 000000 is ADD, 000001 is SUB, 0000010 is AND, and so on. This assignment is entirely arbitrary.

4. Why is the performance of a computer so dependent on a *range* of technologies such as semiconductor, magnetic, optical, chemical, and so on?

   SOLUTION

   Consider a computer's memory that uses the widest range of technologies. In an ideal world, a computer would have a large quantity of low-cost, very fast, non-volatile memory. Unfortunately, fast memory such as DRAM is expensive and volatile. Non-volatile memory such as magnetic disk is (usually) slow and cheap. Real computers use a combined memory system that makes the computer appear as if it really did have fast, cheap non-volatile memory. That is, by combining memories fabricated with different technologies, the computer manufacturer can hide the negative characteristics of specific technologies.

The fastest memory is cache (usually fast static semiconductor RAM) used to hold frequently-used data. The bulk of the immediate access memory is normally DRAM (typically 4 Gbyte today). This semiconductor dynamic memory holds data and working programs, but is volatile and DRAM must be loaded from disk. A hard disk that stores data by magnetizing the surface of a platter holds programs that are archived and that have to be loaded when the user requires them. The hard disk drive is very slow but non-volatile.

Flash memory is non-volatile semiconductor memory used to hold semi-fixed data (e.g., the BIOS) and CD/DVD is optical memory designed to allow interchangeable media.

Semiconductors themselves are the result of complex technological processes. Even the structure and chemical composition of transistors change. New materials are constantly emerging for use in display systems.

5. Modify the algorithm used in this chapter to locate the longest run of non-consecutive characters in the string.

SOLUTION

At any instant you are either in a run of consecutive elements or you are not. If the current element differs from the previous element, you are in a run of non-consecutive elements so you increment the counter and continue. If the current element is the same as the previous element, you are no longer in a sequence of non-consecutive elements and you clear the non-consecutive element counter.

The following presents the code both as ARM assembly language and pseudocode (to the right of the semicolons).

```
       AREA NonSequential, CODE, READONLY
START
       ADR    r8,Sequ       ;Point to sequence
       MOV    r1,#0xFF      ;Dummy old element ($FF is not a legal element)
       MOV    r3,#1         ;Preset longest non-sequential element length to 1
       MOV    r2,#1         ;Preset current non-sequential element length to 1

Rep    LDR    r0,[r8,#4]!  ;Repeat: read element and point to next
       CMP    r0,#0xFF      ;   If terminator
       BEQ    Exit          ;      THEN exit
       CMP    r1,r0         ;Are new and the last element the same?
       BEQ    Same
       ADD    r2,r2,#1      ;If not same THEN increment non-sequential counter
       MOV    r1,r0         ;Old element becomes new element
       CMP    r3,r2         ;Compare current sequence length with highest
       BGE    Rep           ; IF lower than or same repeat (goto line 'Rep')
       MOV    r3,r2         ;    ELSE save new longest run
       B      Rep           ;         and repeat

Same   MOV    r2,#1         ;Clear current not-in-sequence count
       B      Rep           ; then repeat

Exit   B      Exit          ;Termination point

Sequ   DCD 1,2,3,3,3,2,2,4,5,3,2,1,1,1,1,1,4,0xFF ;List for testing
       END
```

6. I was once criticized for saying that Charles Babbage was the inventor of the computer. My critic argued that Babbage's proposed computer was entirely mechanical (wheels, gears, and mechanical linkages) and that a real computer has to be electrical. Was my critic correct?

SOLUTION

I believe not.

There is no fundamental rule that states that a machine such as a computer must have a preferred embodiment. A computer can be mechanical, electronic, quantum mechanical, biological, or chemical. What is required to implement a computer (as we understand the term) is a means of:

- storing information (memory),
- reading information from memory,
- decoding the information,
- executing the instructions corresponding to this information,
- writing back the information.

Moreover, for a computer to perform general-purpose tasks, it must be capable of conditional behavior; that is, the result of one operation must select between two or more alternative courses of action. Otherwise, the computer would always execute the same sequence of operations irrespective of the data.

Of course, a practical computer cannot be mechanical because of the slowness of moving parts compared to electrons in solids. However, one day mechanical computer may be created by using movement at the atomic level (e.g., changing the position of atoms in a crystal or moving nanotubes).

7. What is the effect of the following sequence of RTL instructions? Describe each one individually and state the overall effect of these operations. Note that the notation [*x*] means the contents of memory location *x*.

   a. [5] ← 2
   b. [6] ← 12
   c. [7] ← [5] + [6]
   d. [6] ← [7] + 4
   e. [5] ← [[5] + 4]

   SOLUTION

[5] ← 2	The value 2 is loaded into (memory) location 5
[6] ← 12	The value 12 is loaded into location 6
[7] ← [5] + [6]	The sum of the contents of locations 5 and 6 are loaded into location 7. In this case, the value 2 + 12 = 14 is loaded into location 7
[6] ← [7] - 9	The contents of location 7 (i.e., 14) minus 9 are loaded into location 6; that is, location 6 is loaded with 5.
[5] ← [[5] + 4]	The contents of location 5 are read and then 4 is added to the result. This new value (i.e., 2 + 4 = 6) is loaded into memory location 5. The contents of 6, i.e., 5, are loaded into location 5. At the end of this code fragment [5] = 5, [6] = 5, [7] = 14

8. What are the differences between RTL, machine language, assembly language, high-level language, and pseudocode?

   SOLUTION

RTL (register transfer language) is an algebraic notation used to define machine-level operations such as the transfer of data between registers. Consider the notation:

[r6] ← [r3] + 4

This means that the contents of register r3 are read and 4 added to that value. The total is then copied into register r6.

Machine language is the actual binary code executed by a computer. For example, the binary sequence 1100101000000001 might mean *increment the contents of register r1*. However, this meaning would apply only to a specific computer.

Assembly language is the human-readable form of machine language; that is, it is a representation of machine language in terms of mnemonics; for example, in a specific assembly language, `MOVE.B D3,(A4)` means *add the byte pointed at by register A4 to the contents of register D3*. However, an assembly language normally has special or additional features that make it easier for a programmer to generate code (e.g., the ability to integrate libraries of functions in a program).

High-level language is a computer language that has been designed to facilitate programming. There is no link between a high-level language and the underlying machine code. (However, it would be possible to design a specific architecture that executed a high-level language directly). All programs written in high-level languages have to be compiled into machine code prior to execution (or interpreted line-by-line by an interpreter during execution). Typical high-level languages are C, Java, LISP, and Python.

Pseudocode is an informal high-level language used by programmers to express algorithms. Pseudocode is often a sequence of operations expressed in almost plain English. For example:

```
Repeat
      Add a new number to the total
Until all numbers have been added
```

9. What is a *stored-program machine*?

SOLUTION

A stored program or *von Neumann* machine is a general-purpose digital computer that stores programs and data in the same memory. Instructions are processed in a two-phase cycle called fetch and execute; that is, the instruction pointed at by the program counter (also called the instruction pointer) is read from memory, fetched into the computer, decoded, and then executed. Since an instruction might be of the form `LOAD X` or `STORE Y`, or `ADD Z,5`, a second memory access may take place to read or write to the operand in memory.

10. I would maintain that *conditional behavior* is the key element that makes a computer a computer. Conditional behavior is implemented at the machine level by operations such as `BEQ XYZ` (branch to instruction *XYZ*) and at high-level language level by operations such as `IF x == y then do THIS else do THAT`. Why are such conditional operations so necessary to computing?

SOLUTION

Without conditional behaviour, each program would consist of a sequence of operations that were always executed in the same way every time the program was run. For example; you could build a computer to evaluate π to 50 decimal places by using a sequence of arithmetic operations without conditional operations.

Conditional operations allow a computer to change the sequence of operations it will execute, according to the outcome of a test. For example, when simulating a game of chess, a computer may first move each chess piece onto a different square and then evaluate the goodness or *figure of merit* of that position. That calculation can be done without conditional behavior. Suppose at a given stage of play there are seven possible moves, and the figures of merit of the moves are 2, -3, -10, 20, 1, 0, 9 with the highest number signifying the best. In this case the computer would select the move with the figure of merit 20 as best and then continue from that point. It is this conditional behavior that gives the computer its great power.

11. What are the relative advantages of one-address, two-address, and three-address computer architectures?

SOLUTION

From a programmer's point of view the difference is between elegance and verbosity. For the purpose of this question we will assume that an address refers to a location in memory (I make this point because you could also regard a register as an address, for example, r3, r5, r12).

A three-address machine allows you to specify three operands in a single instruction, which lets you implement an operation like P = Q + R in the form `ADD P,Q,R`. Here, P, Q, and R are the three addresses of the three operands in memory (or they could be registers). Such an instruction would access memory three times during its execution phase.

A one-address machine specifies only one address, which means that all operations must take place between the contents of a memory location and an implicit internal register (sometimes called the accumulator). To execute P = Q + R we would force you to write something like

```
LDA P          ; Load accumulator with P
ADD Q          ; Add Q to P in the accumulator
STA R          ; Store accumulator in R
```

This is inelegant code and the accumulator is a bottleneck because you have to keep loading and storing as all data goes through the accumulator.

A two-address machine allows you to specify two operands, which means that one operand acts as a source that is overwritten by the result. For example, `ADD P,Q` adds P to the contents of Q. The old contents of Q are lost.

Although there are no three-memory-address machines, RISC processors like MIPS or ARM use a three-address instruction format and specify three registers, for example, `ADD r1,r2,r3`.

Most real computers fall into one of two categories; those that have three register addresses (like ARM and MIPS), and those that have two addresses (like Intel's IA32 architecture). Three-address machines must also provide two memory access instructions, load and store, that transfer data between registers and memory.

Some two-address computers are called *one and a half address machines* because they use one memory address and one register address. You could say that such a machine is essentially a one-address machine with multiple accumulators (e.g., Intel's IA32 series).

One-address machines are simple devices and are implemented as 8-bit microcontrollers in low-cost applications. This course does not deal with these devices.

The relative advantages of two- and three-address machines are debatable and both processor architectures continue to thrive. Some would argue that a three-address (RISC-style) processor should be faster than a two address machine because all data processing operations are applied to registers that have a far faster access time than memory.

12. What is the difference between a computer's *architecture* and its *organization*?

SOLUTION

A computer's architecture is an *abstraction*; that is, it is the assembly language programmer's view of the computer in terms of its instruction set. A knowledge of a computer's architecture is necessary to write (machine-level) programs that run on the computer.

A computer's organization is the actual organization or structure of the hardware that implements the architecture. Any given architecture can be implemented by many different computer organizations. A computer's organization determines its price-to-performance ratio. In short, architecture tells us *what* a computer does and organization tells us *how* it does it. Today, the term *microarchitecture* is sometimes used in place of organization.

13. Can you think of other systems besides computers that may be said to have both an *architecture* and an *organization*?

SOLUTION

The clock or watch falls into this category. Its architecture is determined by its dial and functions (date, stopwatch etc.). Its organization determines how it calculates the time; for example, mechanical clockwork, analog electronics driving a stepper motor to move the hands, or digital electronics.

Automobiles are another example: some automobiles have identical functionalities. However their organization (gasoline or diesel, turbocharged or normally aspirated engine) determines their size, speed, and price.

14. What is the difference between an exo- and an endo-architecture?

SOLUTION

Dasgupta popularized the terms exo-architecture and endo-architecture. The exo-architecture of a computer is the external view (or black box view) of its architecture. The endo-architecture is a description of a computer's architecture at the level of its implementation. For example, the exo-architecture refers to the add instruction and the endo-architecture refers to what an adder does. These two terms correspond to 'architecture' and 'organization' as used in this text.

15. Over the years, has more computer progress been made in computer architecture or computer organization?

SOLUTION

It is difficult to precisely quantify progress in these two aspects of the computer. The question should be interpreted to mean greater relative progress. It appears to me that greater progress has been made in the area of organization rather than in architecture. For example, the programmer's model of Intel's IA32 architecture has not changed greatly since the 80386. However, the performance of this family has changed massively in the same period. Much of this performance is due to technological advances in manufacturing, and in organization (e.g., the use of on-chip cache memory, pipelining, branch predication, parallel processing, and so on).

16. What is the *semantic gap* and what is its importance in computer architecture? You will need to use the Internet or library to answer this question.

SOLUTION

Humans write code in high-level languages like C and Java. Computers execute code in low-level languages like the Intel IA32 instruction set. The compiler translates a high-level language into low-level language (in machine code form) that can run on an actual processor. The difference between high- and low-level languages is called the *semantic gap*. If a computer were constructed that directly executed C code, then there would be no semantic gap.

17. What is the difference between human memory and computer memory?

SOLUTION

A computer's random access memory is normally accessed by applying an address to interrogate the contents of a specific location within the memory. Human memory appears to be associative. That is, it is searched by providing data as a key rather than address. Memory cells containing that key then respond. For example, the key might be "red" and responses might be "sunset", "color", "rainbow", and so on.

Associative memory is accessed in parallel with a key that is fed to all locations simultaneously. In other words, you access associative memory rather like a web search, by means of a query. Currently, we cannot construct large semiconductor associative memories, because it would mean accessing millions of locations simultaneously and matching the contents of each location with the key. Associative memory is also called CAM (content accessible memory). Small amounts of associative memory are used in specialized applications such as high-speed address translation in memory management units.

18. What is the von Neumann bottleneck?

SOLUTION

The von Neumann machine operates in a fetch/execute cycle with a common memory holding both instructions and data. This means that memory must be accessed (typically) twice for each instruction – once to read the instruction and once to access the operand used by that instruction. Consequently, the path between the computer and memory becomes a bottleneck. The use of separate data and instruction caches can help overcome some effects of the bottleneck.

19. Suppose Intel did not develop the first microprocessor. Was the microprocessor inevitable?

In my view yes. At the time microprocessors appeared, everything that was needed was in place: microtechnology and semiconductor manufacturing were growing, the computer and minicomputer existed, calculators existed. Small and medium-scale logic elements were being produced. There was a need for the microprocessor. All these factors made the microprocessor inevitable.

SOLUTION

20. Identify as many enabling technologies as you can that were required before the computer could be constructed.

SOLUTION

In order to design a mechanical computer of the type envisaged by Babbage, you need chemistry and metallurgy to create the working parts (moving cog wheels and levers and linkages), and engineering technology to create the machines required to build the computer.

For an analog computer you need the development of the theory of electronics, the construction of active (amplifying) devices such as vacuum tubes and transistors, and the availability of components such are resistors, capacitors, inductors, diodes, etc., that are the basic elements of all analog circuits.

For the construction of electromechanical computers (i.e., those using relays) you need some of the same components as general analog circuits plus relays that require the construction techniques of the watch-maker (a relay has moving marts actuated by electromagnets).

For the construction of the electronic digital computer, you need the development of active devices (vacuum tubes or transistors) that can be used to make gates and bistable elements (flip-flops). Of course, to construct practical computers, you need to develop a wide range of technologies: electronics to process signals and transmit data from point-to-point over both short and long distances. You need the development of magnetic and optical technologies as these form the basis of many storage systems. You need the development of display technologies (the CRT, LCD, printer, and so on). Note that many of the technologies are ‘universal’ in the sense that using a technology in one field means that it can be applied in other areas; for example, the technology used to build a CPU is almost identical to that used to create an LCD display.

I asked this question because some introductory texts on computer architecture and organization seem to imply that the computer suddenly emerged. The typical discussion of computers goes something like ...abacus... Babbage analytical engine … ENIAC… PC.

In fact, rather a lot happened between Babbage and ENIAC. Much of the driving force behind the computer was the result of the development of the telegraph and telephone networks. These required the development of metallurgy (metals and wires, magnetics), chemical processing (insulators for wires), fabrication and manufacturing, battery technology, and the development of circuit theory and transmission lines (the behavior of signals in networks).

The design of mechanical and electromechanical switching networks for telephone exchanges gave rise to the body of theory that would later be used to create binary and logic circuits. The vacuum tubes that were used as amplifiers in switching circuits and memories required the development of complex chemistry (cathodes), high-vacuum technology, and a theory of the behavior of electrons in electrostatic fields. Even cosmology played a role in the history of the computer because circuits developed to detect cosmic rays in high-altitude balloons were later adapted as pulse counters in the first generation of vacuum-tube-based computers. The development of the computer required a massive amount of progress across numerous fronts.

21. Suppose Babbage had succeeded in creating a general-purpose mechanical computer that could operate at, say, one operation per second. What effect, if any, do you think it might have had on Babbage's Victorian society?

SOLUTION

Technology is often interlinked. A development in one area is made because there is a need for that development. Technology often develops across a broad front (chemistry, materials science, engineering, electronics and so on). Had Victorian society developed mechanical computers (of the form envisaged by Babbage), engineering and scientific calculations may have been improved, but I doubt whether there would have been a major impact on society. It was the development of the chemical industries and the beginning of electronics that led to our modern society.

22. Use the method of finite differences to calculate the value of 15^2

N	N^2	Δ1	Δ2
1	1		
2	4	3	
3	9	5	2
4	16	7	2
5	25	9	2

SOLUTION

In the above table we’ve provided the integers 1 to 5 and their squares. The Δ1 column contains the differences between successive squares and the Δ2 column contains the difference between successive differences (this is called the second difference). You can see that the second difference is always 2. Using this and addition we can build up the following table. Below is the table from 4 onward with the seconds and first differences. We can add the second differences to the N^2 column to complete the table.

N	N^2	Δ1	Δ2
4	16	7	2
5	25	9	2
6	36	11	2
7	49	13	2
8	64	15	2
9	81	17	2
10	100	19	2
11	121	21	2
12	144	23	2
13	169	25	2
14	196	27	2
15	225	29	2

23. Extend the method of finite differences to calculate the value of 8^3 and 9^3.

SOLUTION

N	N^3	Δ1	Δ2	Δ3
1	1			
2	8	7		
3	27	19	12	
4	64	37	18	6
5	125	61	24	6
6	216	91	30	6
7	343	127	36	6
8	512	169	42	6
9	729	217	48	6

In this case we've written the values of the cubes up to 5. Observe that the third difference is a constant 6. Now the table can be continued by constructing the third, second, and first difference columns to generate the column of cubes without multiplication.

24. Suppose you decided to try and make computers more 'human' and introduce the 'random element.' How would you do that?

SOLUTION

It is possible to make computers random in the sense that we can create random numbers and then use a random number as a means of choosing the outcome of a decision; that is, creating a random element. Random numbers can be generated by taking random noise and converting it into 1s and 0s (random noise is the background hiss present on some radio signals). Random numbers can also be created mathematically by starting with a seed (an initial number) and then applying successive mathematical operations to create a sequence of numbers that appear to be random. These are called pseudo-random numbers because the same seed always generates the same sequence.

It would be possible to model some aspects of human thought processes by using random numbers to 'make a guess'; for example, to model a decision that is 90% certain you could create a random number in the range 0.0 to 1.0 and then take the decision if the random number is less than 0.9. This speculation belongs to the world of artificial intelligence.

25. Computers always follow blind logic. Executing the same program always gives the same results. That's what the computer books say. But is it true? My computer can appear to behave differently on different occasions. Why do you think that this might be so?

SOLUTION

In principle, you would think that computers are entirely predictable and you always get the same response to the same actions. In practice, clicking on an icon may sometimes work and sometimes cause a crash. Why?

Of course, it is true that basic computer operations always yield the same results using the same data; for example, if you perform the operation $a = f(b,c)$ you will always get the same value of a for data b and c and operation f.

However, computers are complex systems and are not entirely synchronous; for example, the state of a signal from an external device may be sampled (read) and it may be read as a 1 or 0, depending at which instant it is sampled. In a sophisticated system with memory management, data may sometimes be in memory and sometimes on disk. If data is read from cache it may be available in 1 clock cycle, getting it from main store may take 50 cycles before it is available. If the data is on disk, it may take more than 20 million clock cycles to retrieve it. Whether data is immediately available or requires a very significant wait is dependent on the current job load. The interaction between individual jobs, asynchronous events such as interrupts and data transmission, means that it is difficult to predict the operation of a computer in many circumstances.

Note that, under certain circumstances, the sampling of digital signals can lead to transitory, random errors called glitches.

The study of systems that require guaranteed behavior (industrial process controls, fly-by-wire aircraft, and nuclear reactors) is a branch of computing called real-time systems. In real-time systems the hardware and software are constructed to take account of the effect of the problems we have highlighted and to minimize them. In the 1980s a processor called the Viper was designed in the UK (sponsored by the UK Ministry of Defence) for applications that required predictability.

26. The value of X is 7. Some computer languages (or notations) interpret X+1 as 8 and others interpret it as Y. Why?

SOLUTION

In everyday life (i.e., natural language), we do not distinguish between the name of a variable, its address, and its value. When we say, $X = X + 1$, we mean that the value we have given the variable X is incremented by 1 to become 8. X is the name of the variable whose value is 7.

If we regard X as the representation of the character X, then X is 0x58 (the ASCII code for X). Adding 1 to X gives us 0x59 which is the ASCII code for Y.

Some computer languages allow you to operate on the name of a variable. This question demonstrates that it is important to appreciate the difference between name, address, and value.

27. Carry out the necessary research and write an essay on the history of the development of computer memory systems (e.g., CRT memory, delay-line stores, ferrite core stores, etc.)

SOLUTION

This is an open-ended question. Some professors might require a short note highlighting the details and others may require an extended essay covering memory technologies and the biographies of those involved. Historically, the first form of computer memory was the punched card. This was used in the Jacquard loom in 1801 to control the weaving of patterns in textiles – a hole or no hole at a point in a card could be used to cause the horizontal thread to go in front of or behind the vertical threads. Babbage proposed the punched card as a storage mechanism in his analytical engine.

Early EDVAC and ACE computers used mercury delay lines to store data. Information was converted into pulses in columns of mercury and the pulses were continually recirculated. Mercury delay lines are impractical because you have to wait for the data you want to reach the end of the tube where it is read and recirculated.

One of the very first random access fully electronic storage devices was the *Williams tube* that stored data as an electrostatic charge on the surface of a cathode ray tube. These had small storage capacities but they made possible some of the first computers at the end of the 1940s and the beginning of the 1950s.

Drum memories appeared in the late 1940s. These used a drum or cylinder as a magnetic recording device; that is, it was rather like a 3-dimensional hard disk (instead of coating a flat platter with a magnetic material, the surface of a rotating drum was coated with a magnetic material). Data was written along tracks and the disk rotated. This was also a non-random access device, but it could store more data than previous technologies.

Jay Foster investigated the properties of ferromagnetic materials at MIT in 1949 and this work led to the development of the ferrite core memory used by the Whirlwind I computer in 1953. The ferrite core stored binary data as clockwise or anticlockwise magnetization in a tiny ferrite bead. This allowed random access and relatively high speeds (e.g., 1 µs). Ferrite core memory dominated computing for three decades.

Today's mainstream semiconductor memory is constructed with DRAM, dynamic memory that stores data as an electrostatic charge in a transistor. DRAM was first commercially produced by Intel in 1970. It is still the dominant form of main store in computers today.

Secondary storage has been implemented as paper tape (obsolete) magnetic tape (still going strong in its modern forms), and disk drives. Magnetic tape recording was invented in Germany in 1928. The first use of tape to record data was in 1951 on the UNIVAC I.

The disk drive uses a flat rotating platter with data recorded in concentric tracks on a magnetized surface. It is the same as tape in principle. The first disk drive was introduced by IBM in 1956 in their RAMAC computer. The first relatively low-cost hard disk drive appeared in 1973 as the IBM 3340 Winchester drive. Today, the hard disk drive has evolved into small, low-cost systems with capacities of the order of 3 TB.

The *hard*-disk drive is contrasted with the *floppy*-disk drive. The floppy disk drive (now virtually obsolete) used low-cost plastic disks to store data in the early days of the PC revolution (e.g., capacities of 360 KB, 720 KB, 1.4 MB and 2.88 MB). These capacities are miniscule today and the flash drive has totally replaced the floppy disk drive. Although the PC revolution owes everything to the floppy magnetic disk, this recording medium is now almost entirely forgotten.

28. Of all the early computers, which do you think should be called the first computer if you are judging the world by today's standards?

SOLUTION

This question is difficult, if not impossible, to answer. The person closely associated with an invention in the public's mind is not necessarily the person who really made the invention. For example, Edison didn't invent the incandescent light. Similarly, many now believe that Bell didn't invent the telephone. In 2002 the US Congress recognized the Italian-American Antonio Meucci as the true inventor of the telephone. It is said that Meucci demonstrated the telephone in New York in 1860, sixteen years before Bell took out a patent. I was surprised to find that the transistor was not invented at Bell labs in 1947. Julius Lilienfeld filed a patent for the transistor in 1925.

Claims to the development of the computer are as convoluted and controversial as claims to any other invention. Claims are often influenced by national chauvinism or economic pressures (e.g., in the UK many believe that John Logie Baird invented television, even though very few people in the USA have heard of Baird). The invention of television is analogous to the invention of the computer. Baird demonstrated a means of displaying moving images at a distance. However, the quality of the image and the resolution were very poor, because his scanning process was entirely mechanical. A year or so later, Vladimir Zworkin demonstrated moving images using the iconoscope, an electronic device that became the basis of modern television. Baird's invention was a dead end. So, does Baird or Zworkin deserve the credit?

In the world of computing we have a similar problem. When did the computer emerge? If you regard a computer as a concept, Babbage deserves the credit because he suggested the storage of instructions in memory, a mill or processor that performed operations, a means of storing intermediate results, and a mechanism for conditional computation. Babbage never built his mechanical system. However, we should note that today we often view Babbage's analytical engine through modern eyes as if it were conceived of as a computer (Babbage certainly did not see it that way).

Konrad Zuse is now given credit as one of the inventors of the computer. He designed an electro-mechanical computer and even had a binary -floating point unit in 1936. Because he was in Germany in WW2, the outside world learned little of his work until historians re-evaluated his contribution. Zuse's Z1 computer (1938) was the first program-controlled computer and his Z3 (1941) was the world's first fully functional programmable computer.

Atanasoff and Berry claim to have invented the first digital computer in 1937. This was an electronic computer, but not a stored program computer. It was a calculating engine designed to solve linear equations and was not a computer in the current sense of the word (electronic calculator would have been a better term). Many regard the ENIAC as the first digital computer. This was also an electronic machine built at the University of Pennsylvania and completed in 1964. The ENIAC was not programmable and you created a program by rearranging the hardware.

However, in 1973 ENIAC's patent of 1963 was ruled invalid in favor of Atanasoff-Berry. Basically, two giants of computing, Sperry Rand and Honeywell, sued each other because Honeywell alleged that Sperry Rand had an illegal monopoly as Sperry Rand held the ENIAC patent and Honeywell was championing Atanasoff. Atanasoff won the case. However, many historians feel that the judgment was flawed; not least because the ABC (Atanasoff-Berry Computer) computer was not programmable.

It appears that the first true stored-program computer (if we forget Zuse) was the so-called Manchester Baby built at the University of Manchester in the UK in 1948. The EDSAC at Cambridge University, also in the UK, was completed in 1949 and is also frequently credited with being the first stored-program computer.

The computer (as we know it) resulted from gradually accelerating developments during the 1930s and 40s. Its development was inevitable given the need for high-speed computation, the widespread notion of an *electronic brain* and the availability of technology.

29. In what applications have computers been most successful and in what applications have they been least successful or even useless?

SOLUTION

Computers are ubiquitous today in both control systems, and human systems. The microprocessor has brought low-cost controllability to almost any system, from the washing machine to the automobile. Microprocessors are at the heart of digital cameras, mobile phones, MP3 players, iPads/tablets and GPS systems.

Probably the first really successful popular use of the computer was word processing. It allowed countless millions to create and manipulate documents at home. With the growth of communications technology, the microprocessor gave us the web and distributed information – not to mention computer games and multimedia. All these are successes. One of the most surprising and unanticipated products of the microprocessor revolution was the spreadsheet. That revolutionized commerce.

Computers have probably been least successful in the field of artificial intelligence; that is, the mimicking of human behavior. Certainly, the humanoid robot (e.g., Asimov's robots) that has a very long history in SF has not emerged. Nothing remotely like it has appeared. Operations that are apparently simple to humans, such as speech understanding, have not been fully achieved (there are, of course, some respectable speech recognition and understanding systems such as Apple's Siri, but these are not yet perfect). Similarly, robots cannot generally perform operations that we regard as easily; for example, climbing stairs. We have created excellent special-purpose robots (e.g., in the automobile world) but nothing that comes close to a machine with human-like capabilities.

30. Why is the bus so important to a computer?

SOLUTION

The bus distributes data in a computer between functional units and between the computer and peripherals. The bus provides a computer with connectivity. The bus is to a computer what a highway is to a human. Without highways we would not be able to go from one place to another with great ease. Every journey would have to be strictly point-to-point. We share roads with others and have protocols (traffic signals, driving conventions, and laws) to ensure the smooth and orderly flow of traffic. The same is true of computers and buses. Processors, memory units, displays and countless peripherals have to be interconnected. Buses allow this to take place. There are fast buses between a computer (CPU) and its external memory (DRAM). There are slower buses between computers and peripherals (e.g., USB). Protocols exist so that information flows in an orderly fashion in a computer; for example, one device can request the bus, another device currently controlling the bus can give it up, and the would-be bus controller take control in an orderly fashion.

31. It is common to hear the argument that the development of the CPU (microprocessor) in terms of its size, power, and speed has driven the computer revolution. What other aspects of the computer system have driven the computer revolution?

SOLUTION

The previous question pointed out the importance of the bus. Without USB/FireWire and WiFi/Bluetooth, interconnectivity would not be possible and mobile Internet applications would not exist. Similarly, the development of low-cost peripherals such as printers, scanners, displays, mice and keyboards has made the personal computer an almost essential household item.

Three other developments that have been vital to the growth of the computer are:

a. The display. Portable computing would be impossible without low-cost, high-resolution, energy-efficient color displays.

b. The development of the disk drive provides large quantities of low-cost data storage. Although performance has increased relatively little (time to read and write data), storage capacity has risen from about 5 MB to about 5 TB which represents a million-fold increase in capacity.

c. The development of flash-memory. The hard disk is relatively bulky and has high power consumption. Flash memory now provides the non-volatile storage required by MP3 players, digital cameras, and some notebook/netbook computers.

32. Where do you think the bottlenecks (limitations or barriers) to the growth of the computer, in terms of its computational power and its abilities/application, lie?

SOLUTION

This is partially answered by other questions on, for example, Moore's Law.

Limitations depend to some extent on the nature of the user. This answer is largely aimed at the home or office computer user, rather than the scientific or government user, because scientific/government users can spend their way out of bottlenecks (e.g., by means of massive parallelism).

In terms of data storage (capacity) we are doing well. The greatest storage requirement comes from high definition moving images. At the moment personal computers and tablets can hold a reasonable amount of data; for example, hundreds of hours of video. As the resolution of displays increases, there will be a continued need for more storage capacity (and some will wish to carry their entire video storage with them).

Processing power is not usually an issue in desktop computing. However, those working on very high resolution still and moving images continue to require more power. The same is true for high-resolution dynamic games. High-performance games computing will probably be a driving force for years to come.

AI applications will continue to soak up computing power as fast as it can be generated for the foreseeable future. For example, face recognition could be used to compile databases of all scenes and actions across many movies, a gigantic task.

A practical limitation to computing is bandwidth, either for a fixed computer (e.g., via cable modems) or for mobile computers (via WiFi). It may well be that the lack of bandwidth will prove to be the single most limiting factor for the average user.

Another limitation is power. This manifests itself by limiting computing speed (the need for more energy) and displays (the need to provide backlighting). Another power limitation is dissipation in terms of the need to remove heat from a computer. This is a particular problem in mobile computing. Finally, when the public electricity supply is not available (e.g., mobile computing), the principal limitation is battery life.

In non-home and office computing (commercial, military, government, medical) it is hard to see any limit to the demand for both storage capacity and computing power. Medical imaging alone has increasingly larger and larger storage requirements. Simulation also has endless requirements from the simulation of nuclear explosions, to the simulation of weather systems, to the simulation of molecular processes in physics. There is simply no foreseeable limit to computational requirements.

33. Is Moore's law a law?

SOLUTION

No, not in the sense of a physical law. It is based on an observation that has been turned into a prophecy or conjecture. Because it has appeared to hold (or at least approximately so) for four decades, manufacturers have *used* Moore's Law to begin designing the next generation of microprocessors before the current generation is in production; that is, the manufacturer can start to design a system with twice the number of transistors as the current generation without the technology needed to fabricate the device being currently available. I call it 'Mr. Micawber's Law', because it relies on something turning up. In recent years, many have stated that Moore's Law has indeed come to an end because we have reached (or are very close to) the physical limits to computer performance.

34. Why do you think that Moore's law exists? What drives it or makes it possible?

SOLUTION

There are several versions of Moore's Law and corollaries of it. The basic law states that the number of transistors on a chip doubles every 18 months. This is similar to saying that the size of individual transistors will be reduced by a factor of 0.7 every 18 months. A popular corollary is that processing power double every 18 months. The fact that Moore's Law has worked for so long indicates that there may be several factors driving it.

Moore's Law has been kept afloat by the astonishing progress in semiconductor manufacturing technologies that are able to fabricate smaller and smaller transistors, year by year. In turn, this requires progress in many different areas – semiconductor manufacture, encapsulation, the development of new materials such as the high-k metal gate, new transistor structures (different geometry) and, above all, progress in the photolithography used to fabricate chips. Moreover, as technologies improve they can themselves be used to create better chip-manufacturing tools.

Moore's Law will eventually come to an end. That is not an observation from recent trends or a guess or a conjecture; it's a certainty. Semiconductors are constructed from real materials that have an atomic structure. There is a limit to how small things can be made in the atomic world. By about 2020, transistors will be reaching the atomic limit. Once interconnections between transistors chips become just a few atoms wide, the conventional rules of electronics will no longer hold and quantum mechanical effects will begin to dominate.

Some believe that Moore's Law will continue beyond 2020 by exploiting three-dimensional circuit design, or other forms of computing that take place at the atomic level using quantum mechanical effects or magnetism.