TK 155.148

KFKI-1980-21

F. VAJDA

COMPUTER SCIENCE NOW AND IN THE 1980s

Hungarian Academy of Sciences

CENTRAL RESEARCH INSTITUTE FOR PHYSICS

BUDAPEST

2017

KFKI-1980-21

COMPUTER SCIENCE NOW AND IN THE 1980s

F. Vajda
Central Research Institute for Physics
H-1525 Budapest 114, P.O.B. 49, Hungary

To be presented on the Course and Conference Universities in World Network of Information and Communication III.
May 13-23, 1980, Dubrovnik, Yugoslavia

HU ISSN 0368 5330
ISBN 963 371 650 0

ABSTRACT

After a brief survey of computer systems and technology progress, the paper presents an overview of curricula developments in computer science and engineering education. Next it gives a mosaic-like picture as to what can be expected in computer science and computer engineering education in the '80s.

АННОТАЦИЯ

После короткого обзора прогресса вычислительных систем и технологии в статье дается короткое резюме о развитии планов инженерно-технического обучения науке вычислительных машин и вычислительной техники. Затем рисуется мозаичная картина предполагаемого прогресса в инженерно-техническом обучении науке вычислительной техники в 80-ых годах.

KIVONAT

A számitógép rendszerek és a technológiai fejlődés rövid áttekintése után a cikk összefoglalja a számitógép tudomány és számitástechnikai mérnökképzés tanterveinek fejlődését. Ezek után mozaik-szerü képet ad a számitógép tudomány mérnökképzés várható fejlődéséről a nyolcvanas években.

"The future is a race between
education and catastrophe"

H.G.WELLS

COMPUTER SCIENCE NOW AND IN THE 1980s

F.VAJDA

Central Research Institute for Physics of the Hungarian Academy of Sciences and the Electrical Engineering Faculty of the Technical University of Budapest

Budapest,Hungary

After a brief survey of computer systems and technology progress, the paper presents an overview of curricula developments in computer science and engineering education. Next it gives a mosaic-like picture as to what can be expected in computer science and computer engineering education in the '80s.

1. A SURVEY OF COMPUTER SYSTEMS.

A computer is determined by many factors, including architecture, structural properties, the technological environment, and the human aspects of the environment in which it was designed and built.

A computer scientist or mathematician sees a computer as *levels-of-interpreters*. An engineer sees the computer on a *structural basis*, with particular emphasis on the logic design of the structure. The view most often taken by a buyer is a *marketplace view*. While these people each favor a particular view of computers, each typically understands certain aspects of the other views.

Adopted from [1] we will consider different views of computer systems.

1.1 Structural levels.

In [2], a set of conceptual levels for describing, understanding, analyzing, designing, and using computer systems was postulated. The model has survived major changes in technology, such as the fabrication of a complete computer on a single silicon chip, and changes in architecture, such as the addition of vector and array data-types.

As shown in Figure 1, there are at least *five levels of system description* that can be used to describe a computer. Each level is characterized by a *distinct language* for representing the components associated with that level, their modes of combination, and their laws of behavior. Within each level there exists a whole hierarchy of systems and subsystems, but as long as these are all described in the same language, they do not constitute separate levels. With this general view, one can work up through the levels of computer systems, starting at the bottom.

The lowest level in Figure 1 is the *device level.* Here the components are p-type and n-type semiconductor materials, dielectric materials, and metal formed in various ways. The behavior of the components is described in the languages of *semiconductor physics and materials science.*

The next level is the *circuit level.* Here the components are resistors, inductors, capacitors, voltage sources, and nonlinear devices. The behavior of the system is measured in terms of *voltage, current, and magnetic flux.* These are continuously varying quantities associated with various components; hence, there is continuous behavior through time, and equations (including differential equations) can be written to describe the behavior of the variables. The components have a discrete number of terminals whereby they can be connected to other commponents.

Above the circuit level is the *switching circuit or logic level.* While the circuit level in digital technology is very similar to the

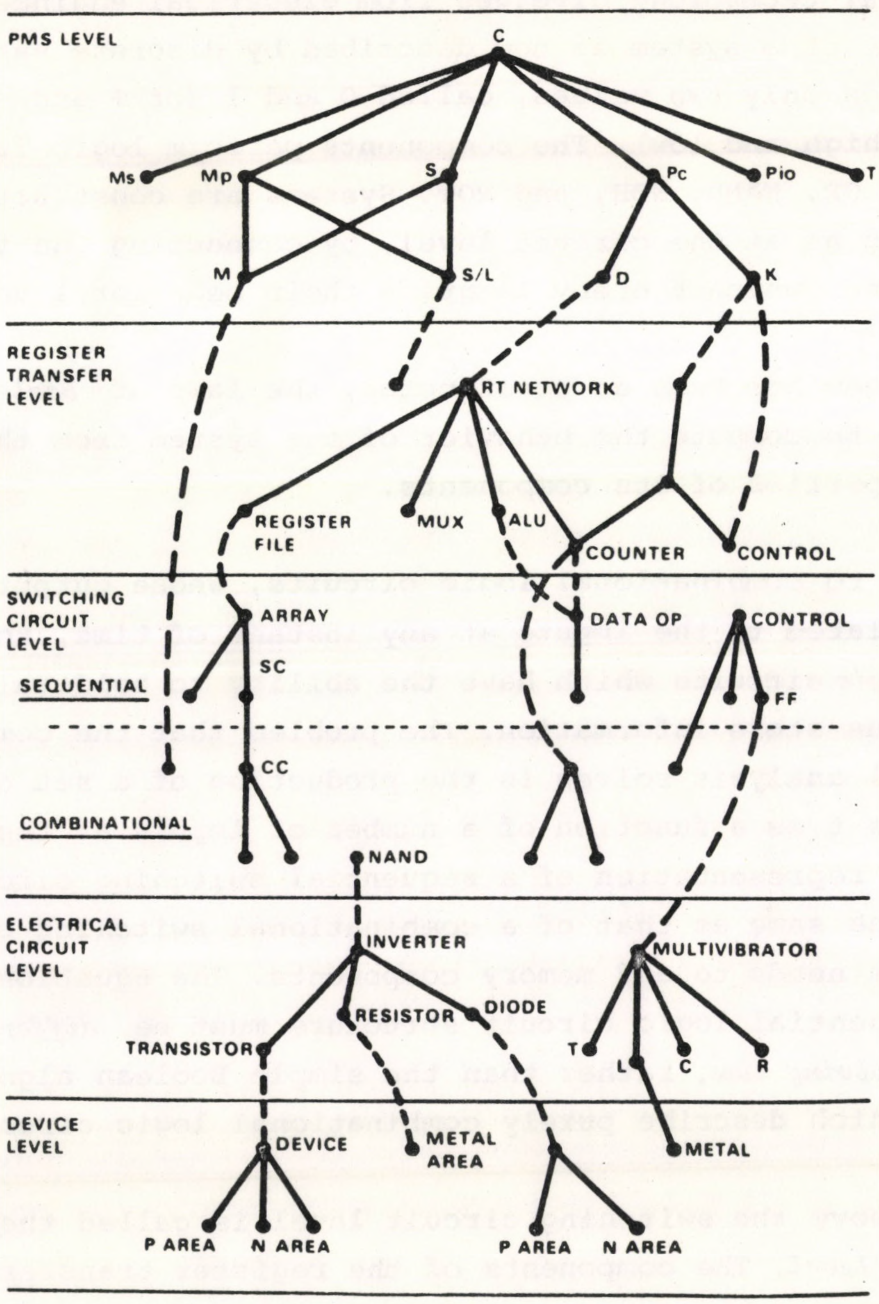

Figure 1.

Hierarchy of computer levels [2].

rest of electrical engineering, the logic level is the point at which digital technology diverges from electrical engineering. The behavior of a system is now described by discrete variables which take on only two values, called 0 and 1 (or + and -, true and false, high and low). The components perform logic functions called AND, OR, NAND, NOR, and NOT. Systems are constructed in the same way as at the circuit level, by connecting the terminals of components, which thereby identify their behavioral values.

After a system has been so constructed, the laws of *Boolean algebra* can be used to compute the behavior of the system from the behavior and properties of its components.

In addition to combinational logic circuits, whose outputs are directly related to the inputs at any instant of time, there are *sequential logic* circuits which have the ability to hold values over time and thus store information. The problem that the combinational level analysis solves is the production of a set of outputs at time t as a function of a number of inputs at the same time t. The representation of a sequential switching circuit is basically the same as that of a combinational switching circuit, although one needs to add memory components. The equations that specify sequential logic circuit structure must be *difference equations involving time,* rather than the simple Boolean algebra equations which describe purely combinational logic circuits.

The level above the switching circuit level is called the *register transfer (RT) level.* The components of the register transfer level are registers and the functional transfers between those registers. The functional transfers occur as the system undergoes discrete operations, whereby the values of various registers are combined according to some rule and are then stored (transferred) into another register. The rule, or law, of combination may be almost anything, from the simple unmodified transfer (A ← B) to logical combination (A ← B ∧ (AND)C) or arithmetic combination (A ← B + (PLUS) C). Thus, a specification of the behavior, equivalent to the Boolean equations of sequential circuits or to the differential equations of the circuit level, is a *set of expressions* (often called productions) that give the conditions under which such transfers will be made.

The fifth and last level in Figure 1 is called the *processor-memory-switch (PMS) level*. This level, which gives only the most aggregate behavior of a computer system, consists of central processors, core memories, tapes, disks, input/output processors, communications lines, printers, tape controllers, buses, teleprinters, scopes, etc. The computer system is viewed as processing a medium, information, which can be measured in bits (or digits, characters, words, etc). Thus, the components have capacities and flow rates as their operating characteristics.

1.2. Layers of interpreters.

In contrast to the structural view, this view is functional. According to this view, a computer system consists of *layers of interpreters*, much like the layers of an onion.

An interpreter is a processing system that is driven by instructions and operates upon state information. The basic *interpretive loop*, shown in Figure 2 is most familiar at the machine language level but also exists at several other levels.

To formalize the notion of levels-of-interpretation, one can represent a processing system by the diagram in Figure 3.

The state information operated on by an interpreter is either internal or external. This can best be understood by considering the "onion skin" levels of the five processing systems that form a typical airline reservation system. These levels are listed in Table 1.

The Level 0 system is the *logic that sequences* the Level 1 *micromachine*. The Level 1 system is a microprogrammed processor implemented in real hardware. It is the machine seen by the logic designer. The Level 2 system is the *central processing unit (CPU)*. It is the machine

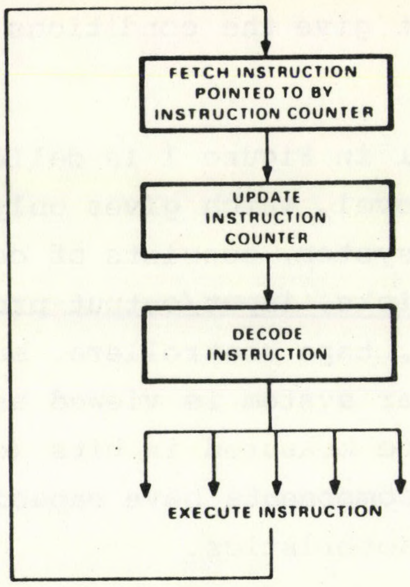
FETCH INSTRUCTION POINTED TO BY INSTRUCTION COUNTER
UPDATE INSTRUCTION COUNTER
DECODE INSTRUCTION
EXECUTE INSTRUCTION

Figure 2.

The basic interpretive loop.

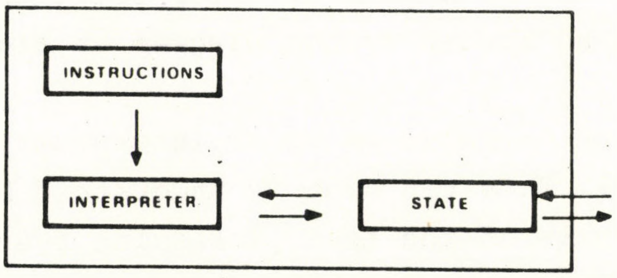

Figure 3.

A processing system.

Table 1. Five Levels-of-Interpreters for an Airline Reservation System [1]

Level4	Instruction:	Seat allocation request message
	Interpreter:	Airline reservation system.
	Internal state:	Number of requests pending at this moment Location of passenger list on a disk file Number of lines connected to system
	External state:	Number of reserved seats on a given flight Airline name for a given flight
Level3	Instructions:	FORTRAN statement codes
	Interpreter:	FORTRAN execution system
	Internal state:	Memory management parameters User name Main storage size Location of disk files Interrupt enable bits Expression evaluation stack Dimensions of arrays
	External state:	Subroutine names Values of data in arrays Statement number Program size Value of an expression DO-loop variable value Printed characters on line printer
Level2	Instructions:	Machine language instructions
	Interpreter:	Processor
	Internal state:	Program registers Condition codes Program counter
	External state:	Data in main memory Disk controller registers
Level1	Instructions:	Microcode
	Interpreter:	Micromachine
	Internal state:	Instruction register Flip-flops holding error status Stack of microprogram subroutine links.
	External state:	Program registers Condition codes Program counter

Table 1 [cont.]

Level0	Instructions:	Hardwired combinational network
	Interpreter:	Sequential machine controlling the micromachine
	Internal state:	Clock, counters, etc.,controlling micromachine timing
	External state:	Micromachine, console

seen by the machine language programmer. The Level 3 system shown here is a FORTRAN *language processing system.* The Level 4 system is *an airline reservation system.* Four of these five systems form the hierarchy shown in Figure 4, where each system is an interpreter that sequences through multiple steps in order to perform a single operation for the next level interpreter. The highest level system, the airline reservation system, is an interpreter operating on messages received from outside of the system. It tests and modifies states and generates messages to send back outside the system, thus performing a single operation for the outermost interpreter.

1.3. Packaging levels of integration.

This is a structural view that *packages the various components* (hardware and software) into levels. The levels for computers are as follows:

9	Applications
8	Applications components
7	Special languages
6	Standard languages
5	Operating systems
4	Cabinets (to hold complete hardware systems)
3	Boxes
2	Modules (printed circuit boards)
1	Integrated circuits

There are *three major changes* taking place:

1. Changes in the *hardware levels,* where the shrinking in physical size of functions has three effects:
   a. Lower levels subsume higher levels.
   b. The semiconductor component supplier is forced to assume higher and higher level design responsibilities.
   c. Levels disappear.

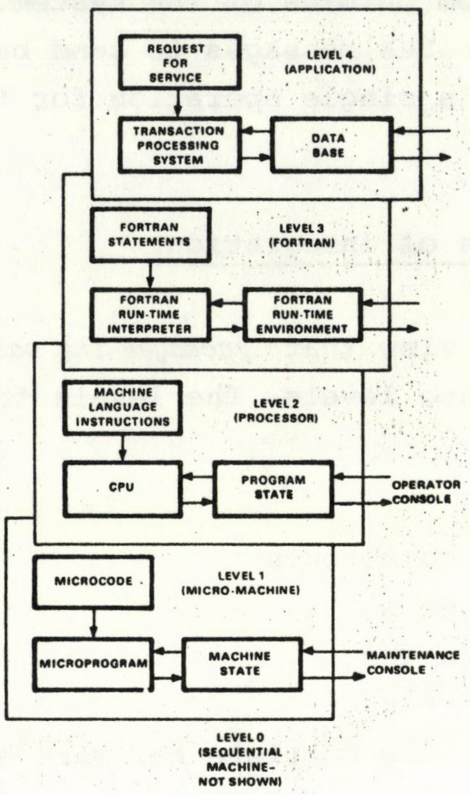

Figure 4.

A hierarchy of interpreters [1].

2. Changes in the *software levels,* again with three effects:
   a. Each level grows in size as more functionality is added over time.
   b. More levels are added as minicomputers are applied to a broader range of applications.
   c. Functions migrate downward from level to level.
3. Changes in the *hardware/software interface,* where software functions migrate into hardware for higher performance.

Figure 5 shows the *costs of various levels-of-integration versus time* for small computers. The cost depends partly on implementation and architecture word length. As the word length is made shorter, there are some savings, particularly for very small computers, because some levels-of-integration cease to exist. For example, most hand-held calculators are implemented using 4-bit, stored program computers with fixed programs that occupy a single integrated circuit. There are associated modules, backplanes, boxes, and cabinets - but all are contained in a single package that fits in the hand.

Semiconductors, the lowest level of technology, have had the greatest price decline (Figure 5). *Modules* have a lesser price decline because they are a mix of integrated circuits, printed circuit boards, component insertion labor, and testing labor. The price decline for the integrated circuit portion of the module cost is moderated by the labor-intensive nature of module fabrication, thus producing a price decline for modules that is markedly less than that for integrated circuits. At the *box level-of-integration,* power supplies and metal or plastic boxes are also labor-intensive and further moderate the price decline provided by the integrated circuits. Finally, as boxes are integrated (by people) and applied at *a system level* (by people), the price decline almost disappears.

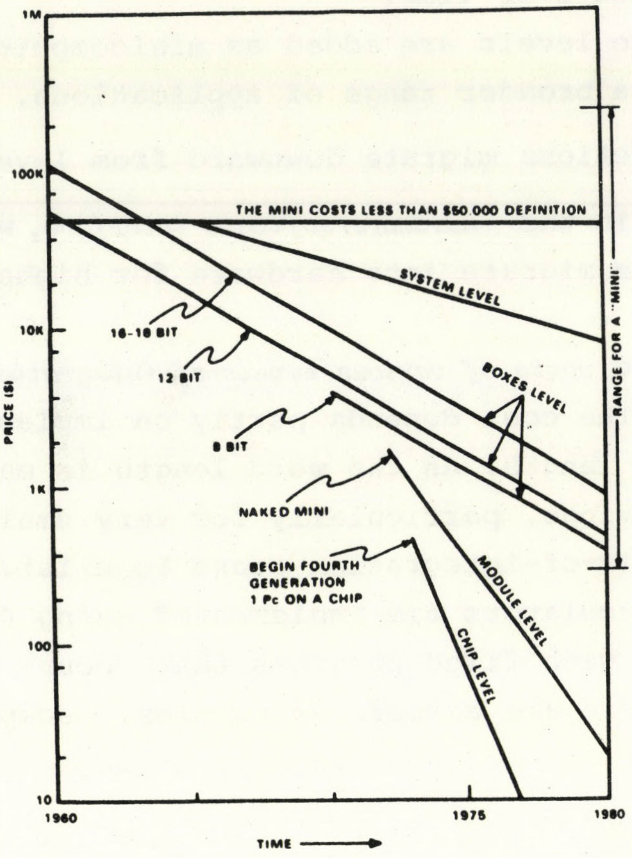

Figure 5.

Machine price vs. time for various levels of integration [1].

1.4. Computer classes.

Because it is the complete marketplace process that produces the computer, this view is the most complex. In terms of *marketability*, a computer can be characterized as a function of price, performance, and time of introduction in what might appear to be a commodity-like environment.

Because various computers operate at different performance rates and at various costs, computation can be purchased in multiple ways, and *price/performance ratios* will thus affect marketability. For example, computation can be supplied by a shared large, central batch computer; each organizational entity can own and operate a shared minicomputer; an individual can operate a single desk-top system; or each individual can operate a programmable calculator.

The price/performance ratio is not the sole factor determining marketability, however. *Program compatibility* with previous machines is important. Compatibility considerations are based on the economic necessity of using a common software base.

In a similar way, *compatibility over a range of machines* at a given time allows a user to select a machine that matches his problem set while having the comfort that the problems can change and there will be a sufficiently large or small machine available to solve the new problems.

For these reasons, nearly all modern computer designs are part of a compatible computer family which extends over price and time. Technology provides basic improvements with each new generation at approximately six-year intervals, and most new designs usually provide increased performance at constant price.

The influence of technology on the computers that are built and taken to the marketplace is so strong that the *four generations* of computers have been named after the technology of their components:

vacuum-tubes, transistors, integrated circuits (multiple transistors packaged together), and large-scale integrated (LSI) circuits.

When an improved basic technology becomes available to a computer designer, there are four paths the designs can take to incorporate the technology:

1. Use the newer technology to build a *cheaper* system with the same performance.
2. Hold the price constant and use the technological improvement to get an *increase in performance.*
3. Push the design to the limits of the new technology, thereby increasing *both performance and price.*
4. Find a *drastically new structure* using the computer as a basic archetype such that the design can be considered off the evolutionary path.

In the first design style, the performance is held constant, and the improved technology is used to build lower price machines which attract new applications. This design style has as its most important consequence the concept of the *minimal computer.* The minimal computer has traditionally been the vehicle for entering new applications, since it is the smallest computer that can be constructed with a given technology. Each year, as the price of the minimal computer declines, new applications become economically feasible.

The second, constant cost alternative uses the improved technology to get better performance at a constant price and will usually yield the best increase in *total system cost and effectiveness,* for reasons which will be discussed shortly.

The third alternative is to use the new technology to build the most powerful machine possible. New designs using this alternative often solve previosuly unsolved problems and, in doing so, *advance the state-of-the-art*. This design alternative must be used cautiously, however, because going too far in price or performance (i.e. building beyond the technology) is dangerous and can lead to a zero performance, high-cost product.

Applying the three design styles over several generations produces the plot given in Figure 5. These figures lead to one of the most interesting results of the marketplace new, which is that *computer classes* can be distinguished by price and named as follows: *submicro, micro, mini, midi, maxi, and super*. The classes midi and maxi are sometimes referred to by the single, nondescriptive name, *mainframe*.

When one distinguishes computer classes by price, a new range of price can be made possible by new technology and create a new class. The new class appears at the low end of the price scale where the minimal computer is introduced at a significantly lower price level than existing computers.

The measure used to define a new class is price, whereas the measure defining an established class is performance. This is because once a new class has become established in the marketplace, the users become familiar with what computers of that class can do for their applications and tend to characterize that class on a performance basic. The characterization of existing classes on a performance basic is important to this discussion because at each new technology time, performance increases by one category, and midi performance becomes available on a mini, for example.

1.5. An applications-functional view.

Because of the general purpose nature of computers, all of the functional specialization occurs at the time of programming rather than at the time of design. As a result, there is remarkably little shaping of computer structure to fit the function to be performed.

The shaping that does take place uses four *primary techniques*.

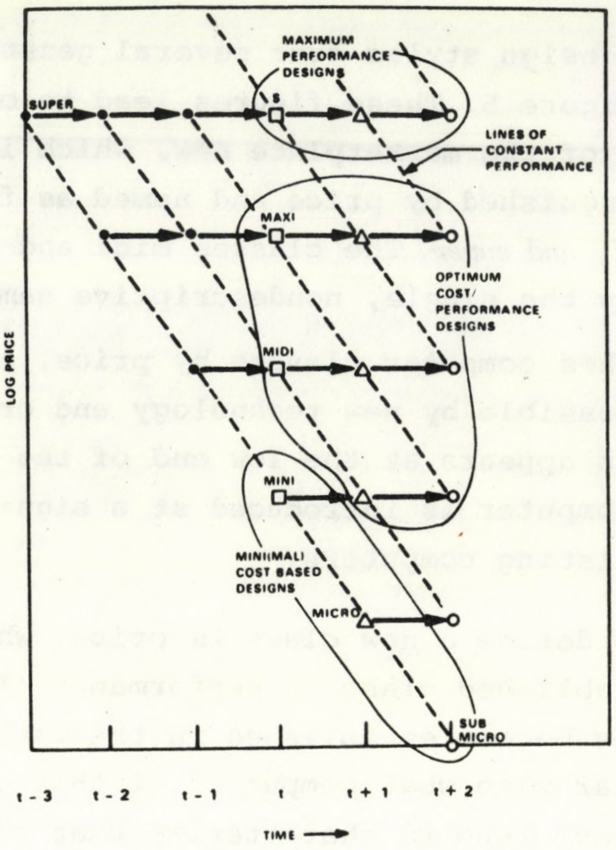
MAXIMUM PERFORMANCE DESIGNS
SUPER
S
LINES OF CONSTANT PERFORMANCE
MAXI
OPTIMUM COST/ PERFORMANCE DESIGNS
LOG PRICE
MIDI
MINI
MINIMAL COST BASED DESIGNS
MICRO
SUB MICRO
t - 3
t - 2
t - 1
t
t + 1
t + 2
TIME

Figure 6.

Price vs. time for each machine class[1]

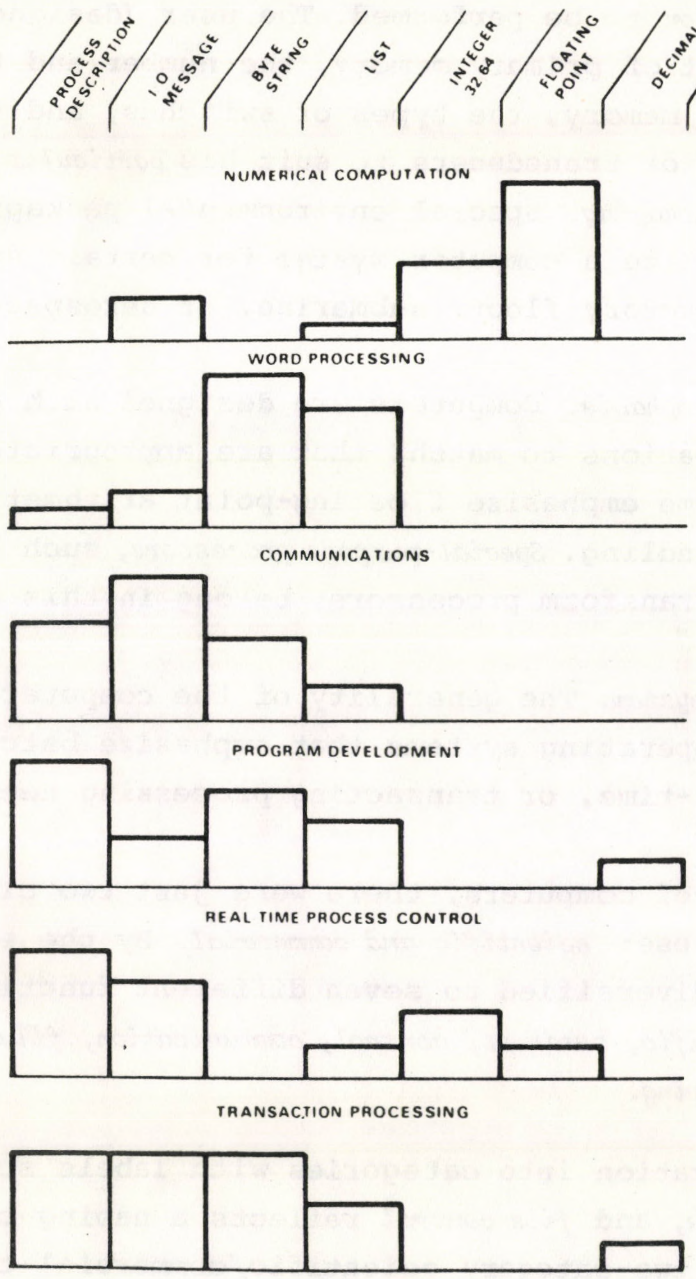

Figure 7.

Data type usage by application [1].

1. *PMS level configuration.* A configuration is chosen to match the *function* to be performed. The user (designer) chooses the amount of primary memory, the number and types of secondary memory, the types of switches, and the number and types of transducers to suit his *particular application.*
2. *Physical packaging.* Special environmental packaging is used to specialize a computer system for certain environments, such as factory floor, submarine, or aerospace applications.
3. *Data-type emphasis.* Computers are designed with data-types (and operations to match) that are appropriate to their tasks. Some emphasize floating-point arithmetic, others string handling. *Special-purpose processors,* such as Fast Fourier Transform processors, belong in this category also.
4. *Operating system.* The generality of the computer is used to program operating systems that emphasize batch, time sharing, real-time, or transacting processing needs.

In the early days of computers, there were just two classifications of computer use: *scientific and commercial.* By the early 1970s, computer use had diversified to seven different functional segmentations: *scientific, business, control, communication, file control, terminal, and timesharing.*

Functional segmentation into categories with labels such as *business, control, communication,* and *file control* reflects a naming convention rooted in the old two-category scientific/commercial tradition.

Machines, then, evolve to carry out more and more functions. Since a prime discriminant is data-type, Figure 7 is presented to show an estimate of *data-type usage* for various applications, using mostly high level data-types, e.g., process descriptions. The estimates shown are very rough, because attempts to measure such distributions to data have not shown marked differences across applications (except for numerical versus non-numerical) because the data-types have not been of a sufficiently high level.

1.6. The practice of design.

In [3] Asimow gives a general perspective of engineering design and how the formal alternative generators and evaluating procedures are used. He also indicates where these formalisms break down and where they do not apply. He defines engineering design as an activity directed toward fulfilling human needs, based on the technology of our culture. He distinguishes two types of design: *design by evolution and design by innovation.*

Figure 8 called *Philosophy of Design,* shows the basic design process. In [4] Phister posits a model of this process (Figure 9.).

Blaauw [5] distinguishes between *architecture, implementation,* and *realization* as three separable levels in the construction of anything, including computer structures (Table 2).

The *architecture* of a computer system defines its functionality (behavior) as it appears to the *machine level programmer* and can be characterized by the instruction set processor (ISP). The *implementation* of a computer system is the *actual hardware structure* - the register transfer (RT) level behavior and *data-flow organization.* This also includes varios *algorithms* for controlling a machine as it interprets an architecture. *Realization* encompasses the actual *technologies* used and includes the kind of logic and how it is packaged and interconnected. Realization includes all the details associated with the physical aspects of the machine.

Modern architectures (ISPs) usually have *multiple (RT) implementations.* For example, the LSI-11, PDP-11/40, and PDP-11/60 are different implementations of the same basic PDP-11 instruction set.

2. TECHNOLOGY PROGRESS IN LOGIC AND MEMORIES.

If during the next 10 to 12 years the *evolution of technology* will be as fast as it has been over the past ten years, and there is every reason to believe it will, we expect that by 1989 we will be able to offer functional complexity which is at about 1000 to

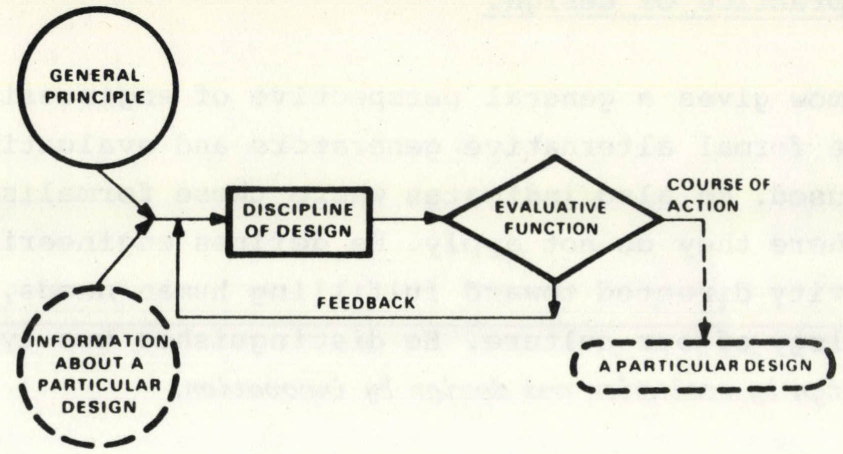

Figure 8.

Philosophy of design [1].

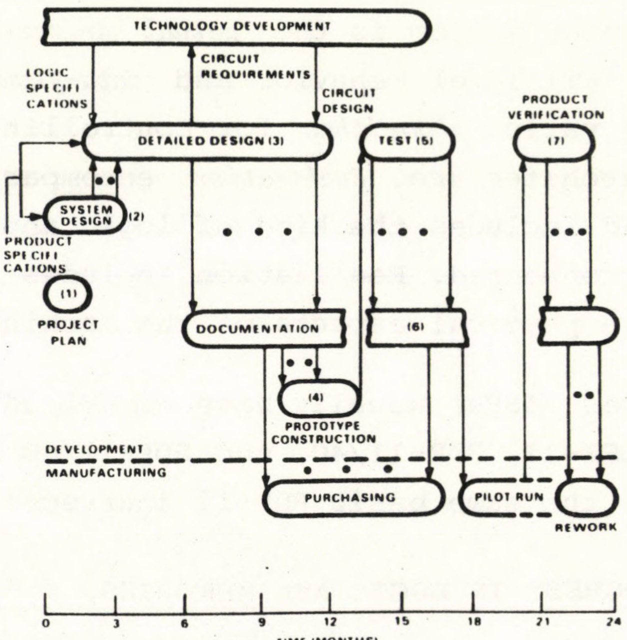

Figure 9.

Hardware product development [4].

Table 2. Characteristics of Design Areas [5]

	Architecture	Implementation	Realization
Purpose	Function	Cost and performance	Buildable and maintainable
Product	Principles of operation	Logic design	Release to manufacturing
Language	Written algorithms	Block diagram, expressions	Lists and diagrams
Quality measure	Consistency	Broad scope	Reliability
Meanings (used herein)	ISP Machine ISP	RT level machine: microprogrammed sequential machine (at logic level)	Physical realization: physical implementation

2000 times what it is today at costs 100 to 1000 times less [6]. This means that 10 years from now almost every one will expect inexpensive computer power in some way accessible at any time. This will mean major changes for all of us in our offices, in our homes, and in our way of doing business.

A recent Rome Air Development Center/State University of New York Symposium on Command Control and Communication predicted that, by 1985, over half of the US workforce will *use computing technology daily* [7].

It is customary when reviewing the history of an industry to ascribe events to either *market (user) pull or technology push.* The history of the auto industry contains many good examples of market pull, such as the trends toward large cars, small cars, tail fins, and hood ornaments. The history of the *computer industry,* on the other hand, is almost solely one of technology push. If the *automobile industry* had had the same evolution as the computer field: a car which in 1945 would have cost about 10.000 dollars and would have driven about four kilometers on one litre, in 1979 an "equivalent" would be costing 500 dollars, be driving 75 kilometers on one litre, at 1200 kilometers an hour and in 1989 such a car would cost 50 dollars, be driving 750 kilometers on one litre at the speed of 10.000 kilometers an hour [6].

Technology push in the computer industry has been strongest in the areas of *logic and memory.*

For *users of digital integrated circuits* there are several relevant parameters [1]:

1. The *function* of an individual circuit in the integrated circuit, the aggregate function of the integrated circuit, and the functions of a complete integrated circuit family such as the 7400-series.
2. The *number of switching circuit* functions per integrated circuit. This quantity and density is a measure of the capability of the integrated circuit and the ingenuity of

the designers.

3. *Cost.*

4. The *speed* of each circuit and the speed of the integrated circuit and set of integrated circuits within a family. The semiconductor device family (transistor-transistor logic = TTL. Schottky TTL = TTL/S, emitter-coupled logic = ECL, metal oxide semiconductor = MOS, complementary MOS = CMOS, silicon on saphire = SOS, integrated injection logic = I^2L) usually determines this performance.

5. The *number of interconnections* (pins) to communicate outside the integrated circuit.

6. The *reliability*. This is a function of the circuit technology, the density, the number of pins, the operating temperature, the use (or misuse), and the maturity (experience) of the manufacturing process.

7. Power consumption and *speed-power product*. A frequently used metric is the speed-power product, where the delay through a typical gate is multiplied by the power consumption of the gate. For a particular technology, the speed-power product tends to be constant because short gate delays usually are accompanied by high power consumption. A technical advance that lowers the speed-power product is considered noteworthy.

Figure 10 shows a *family tree* (taxonomy) of the most common digital integrated circuits. The least complex functions are in the upper portion of the figure, and the most complex are at the bottom. In addition, the circuits are ordered by generation, starting with the second generation on the left side of the figure and progressing to the fifth generation on the right side. The circuits are clustered roughly by the regularity of the function and whether memory is associated with the function. Circuit regularity is important in large-scale integrated circuits because it is desirable to implement regular structures to minimize area-consuming interconnections and, thus, to simplify layout and

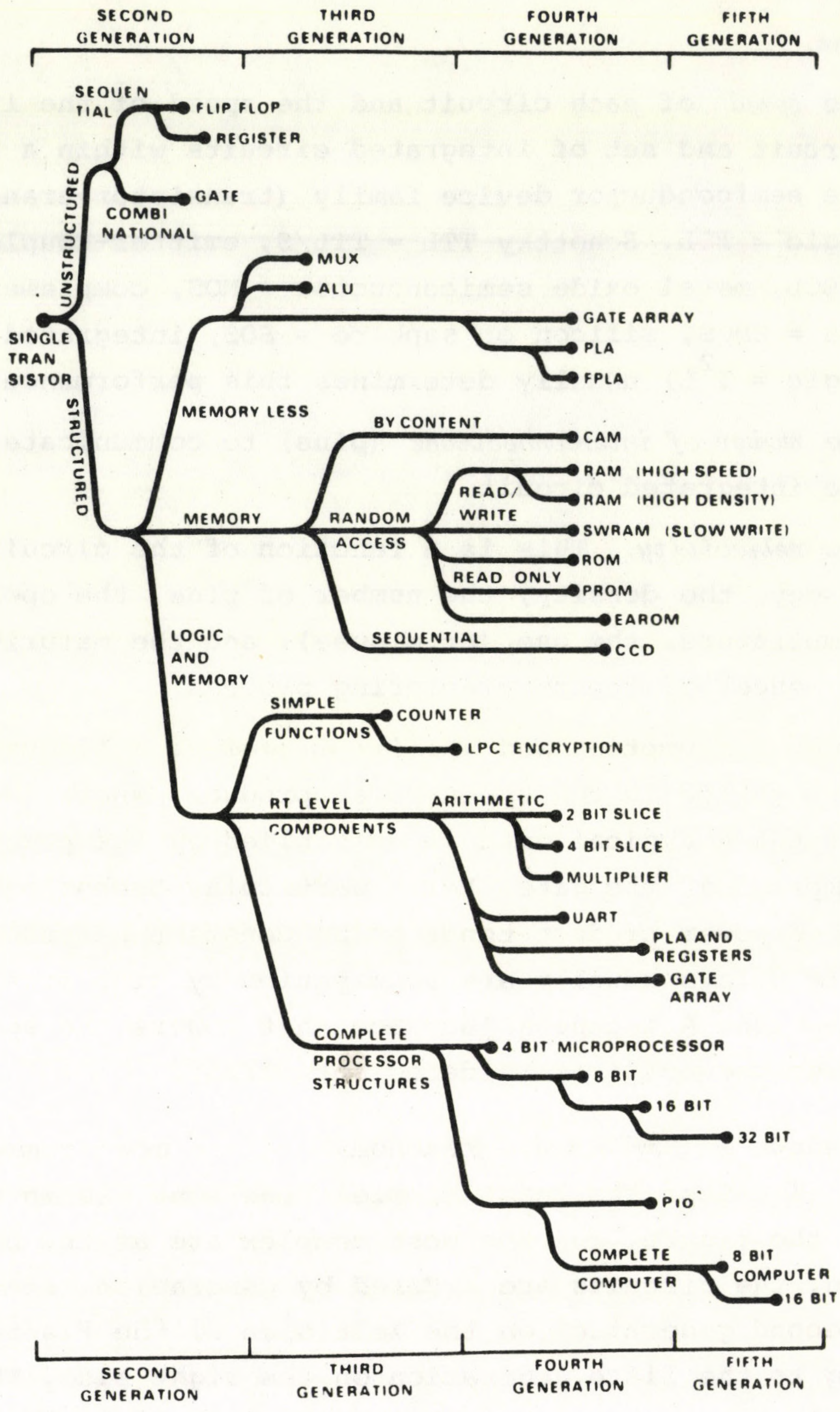

Figure lo.

Family tree of digital integrated circuit function [1].

understanding and to aid testing.

The number of gate circuits per chip not only determines chip *functionality,* it also is the measure of *density* as seen by a user (Figure 11). This metric is the product of the circuit area and the number of circuits per unit area. Progress in lithopraphy has led to a reduction of conductor linewidths and a corresponding reduction of circuit size to yield higher speeds and higher densities. Linewidths have decreased from 10 microns in early large-scale integrated circuit chips to 6 microns in the LSI-11 chips, and more recently to 3 or 4 microns in Intel's 8086. Linewidths of less than a micron have been achieved at the research level, but they require electron beam techniques instead of present photographic methods of production. The processing techniques to create semiconductor materials have also been improved for better manufacturing yields (and lower costs). Circuit and device innovation (such as reducing the number of transistors per memory cell) have also contributes to density and yield increases.

The result given in Figure 11 is exponential and indicates that the number of bits per chip for a metal oxide semiconductor (MOS) memory doubles every two years according to the relationship:

$$\text{Number of bits per ship} = 2^{t-1962}$$

This is the so-called Moore's Law [8].

The cost history of integrated circuits is reflected very dramatically in the cost history of a special class of integrated circuits, semiconductor memory. The *semiconductor memory cost* curves, given in Figure 12 are also interesting because of the important role of memory in past and future computer structures.

Two factors influence the cost of integrated circuits: *density in bits per integrated circuit* and *cost per bit.* The two factors have not had equal influence in reducing costs because, while chip density has improved by a factor of 2 each year, the cost per bit has not declined by a factor of 2 every two years. The equation for the

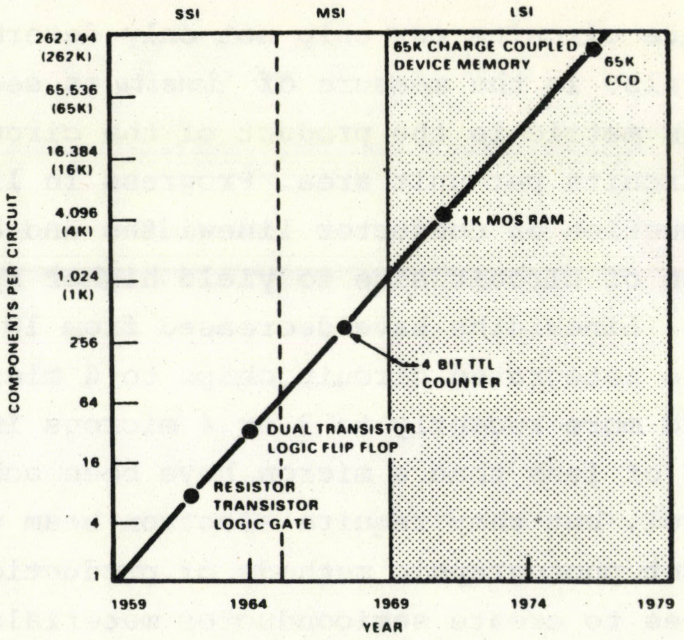

Figure 11.

Components per single integrated circuit die vs. time [1].

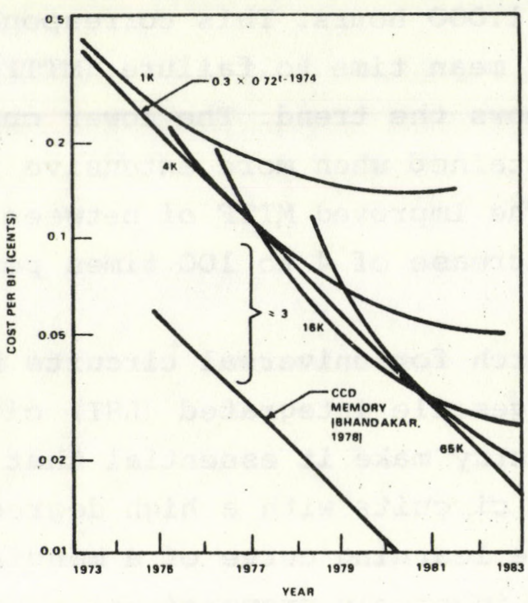

Figure 12.

Cost per bit of integrated circuit RAM memory vs. time [9].

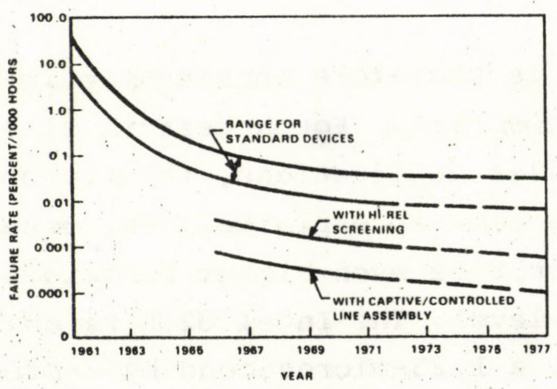

Figure 13.

Failure rate of silicon integrated circuits [10].

line drawn in Figure 12 is [9]:

$$\text{Cost/bit (c)} = 0{,}3 \times 0.72^{t-1974}$$

Over the past 15 years, the *failure rate* for standard integrated circuits has been reduced by *two orders of magnitude* to the neighborhood of 0.01 percent per 1.000 hours. This corresponds to 10^7 hours (about a millenium) mean time to failure (MTTF) per component. Figure 13 [10], shows the trend. The lower curves show the higher reliability obtained when more extensive testing and screening are employed. The improved MTTF of between 10^8 and 10^9 are obtained at a cost increase of 4 to 100 times per component.

A dilemma involving a search for universal circuits has developed in the manufacture of largescale integrated (LSI) circuits. The economics of the LSI industry make it essential that integrated circuit suppliers produce circuits with a high degree of *universality.* This is because the learning curve of a manufacturing process causes cost to be inversely proportional to volume, and for a design to be sold in high volume, it must be usable in a large number of applications. However, the trend in circuit *complexity,* which allows semiconductor manufacturers to put more transistors on a constant die area each year, tends to increase specialization of function, lowering the volume and raising the price.

The LSI product designer is therefore continually in search of *universal primitives or building blocks.* For a certain class of applications, such as controller applications, the microprocessor is a fine primitive and has been so exploited. For other applications, circuit complexity can embrace even higher functionality at the processor-memory-switch level. The Intel 827X is an interesting example: two processors, a 1.25-microsecond byte-processor and a 2.50-nanosecond bit-processor, are combined in one large-scale integrated circuit.

The characteristics of microprocessor and read-only memory design methods of creating *customized* results from universal large-scale integrated circuits are summarized, along with the characteristics of a number of other methods, in Table 3.

With the advent of the processor-on-a-chip, *digital system design* has been, or soon will be, converted completely to *computer system design*. The hardware part of the design, the interface to the particular equipment, is straightforward. The major part of the design is the programming. Since the late 1940s, three generations have learned about computer design, especially programming. The first generation *discovered and wrote* about it. Then it was *rediscovered and applied* to minicomputer systems. This time, it is being learned by everyone who must use and program the microcomputer. Each time, for each individual or organization, the story is about the same: people start off by programming (using binary, octal, or hexadecimal codes) small tasks, using no structure or method of synchronizing the various multiple processes; the interrupt mechanism is learned, and the symbolic assembler is employed; and finally some more structured system, possibly an operating system, is employed. Occasionally, users move to high level languages or macroassemblers.

In view of this *cyclical history*, it seems likely that current digital systems design practice, which consists of building simple hardware interfaces to relatively poorly defined buses together with programming the applications, will be relatively short lived. The design method of the future (fifth generation) will be at the PMS level component, although at the moment there are several factors that prevent this from being done reliably and cheaply by large numbers of enigneers.

One factor which impedes this progress to the fifth generation is the (fundamental) *interconnect problem*. Currently, many small-scale integration components are required to handle the mismatch between microprocessor chips and memory and I/O subsystems. Furthermore, buses are hard to specify.

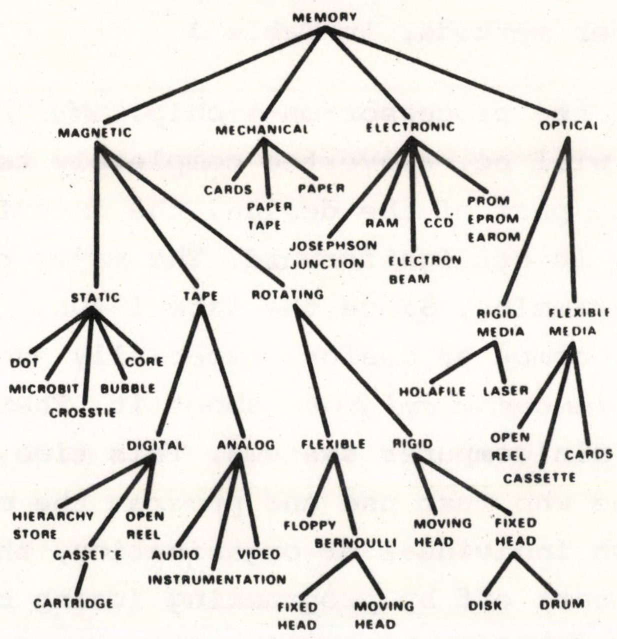

Figure 14.
Family tree of memory technology.

Table 3. Design Techniques for Various LSI Building Blocks [1]

Building Block	Technique for Varying Function	Degres of Generality	Permanence of Change
Computer module	Program	Very high	None
Micro processor	Program	High	Low to medium
Bit-slice	Microprogram	Medium	Medium
ROM	Factory mask change	Very high	Irreversible
PROM	Field change	Very high	Irreversible
EAROM EPROM	Field change	Very high	Low
PLA	change		
FPLA	Field change	Medium	Irreversible
Gate array	Factory mask change	Medium	Irreversible
RAM	Write	Very high	None

Another impediment is that *system level behavior* (the interaction of processors, memories, and transducers via switches and links) is less understood than is interaction at the register transfer level.

Of substantial assistance in easing the transition to the fifth generation would be *base level operating systems* that were embedded in hardware. These should be placed in read-only memory to give a feeling of permanence so that users would be less likely to embark on the expensive, unreliable rediscovery path.

Figure 14 shows the various *technologies* employed in *computer memory applications*, while Figure 15 shows current devices of solid-state memory technology and a likely progression of develepment to come [1]. Table 4 separates *storage systems* by their *function* and *size*.

3. TRENDS AND PERSPECTIVE IN COMPUTER SCIENCE AND ENGINEERING EDUCATION.

It is not easy to trace the evolution of the computer science and engineering (CSE) education. At the beginning two types of courses were most prominent; the *logic design courses* primarily emphasizing the computer design techniques based on switching theory and circuit design and *programming courses* emphasizing machine and assembly language programming. The former was generally taught by electrical engineering professors, while the programming courses usually by specialists of the university's computing center under the auspices of the mathematics department. Later electrical engineering (EE) departments naturally included *computer design courses* in their curriculum whereas the mathematics departments generally handled programming and numerical analysis courses.

Subsequently computer science (CS) departments began to appear as offshots of mathematics departments in the colleges of art and sciences. At present, the CSE education in major universities and

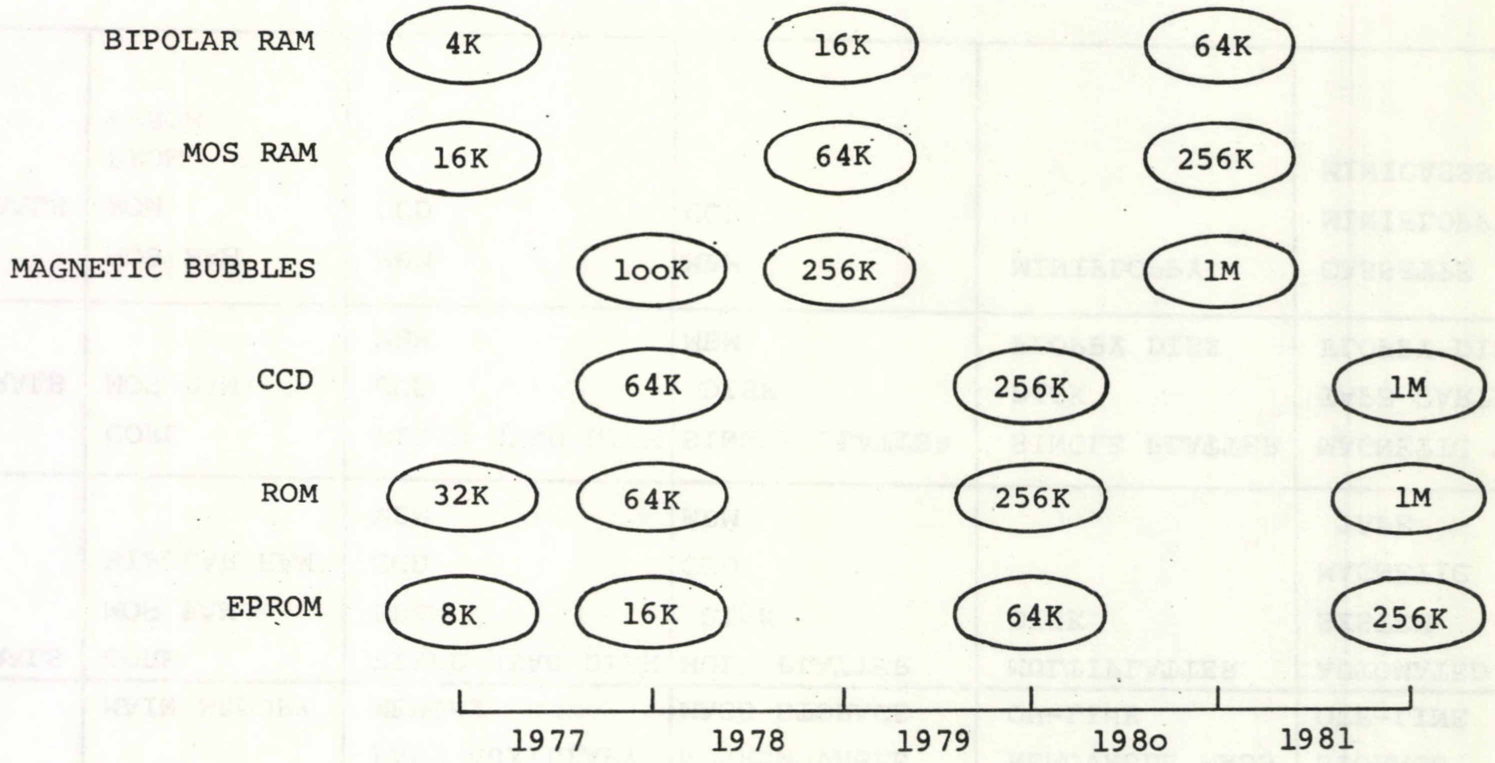

Figure 15.

Solid-state memory technology trends[11].

Table 4. Storage peripherals [11]

	MAIN MEMORY	FAST AUXILIARY MEMORY	NONREMOVABLE MASS STORAGE	REMOVABLE MASS ON-LINE	STORAGE OFF-LINE
PERIPHERALS	CORE MOS RAM BIPOLAR RAM	FIXED HEAD DISK EBAM CCD MBM	MULTIPLATTER DISK CCD MBM	MULTIPLATTER DISK	AUTOMATED TAPE SYSTEM MAGNETIC TAPE
MINI-PERIPHERALS	CORE MOS RAM	FIXED HEAD DISK CCD MBM	SINGLE PLATTER DISK MBM	SINGLE PLATTER DISK FLOPPY DISK	MAGNETIC TAPE TAPE CARTRIDGE FLOPPY DISK
MICRO-PERIPHERALS	MOS RAM ROM PROM EPROM	MBM CCD	MBM CCD	MINIFLOPPY	CASSETTE MINIFLOPPY MINICASSETTE

colleges are administered mainly by separate CS and EE departments or by a combined EE and CS department. A few departments identified computing engineering (CE) or call themselves electrical and computer engineering (ECE) departments (Figure 16).

We have used the terms CS and CE without defining them. The *computer scientist* is interested in theory and science of computation and programming, while the *computer engineer* is interested in the specification design and implementation and utilization (operation) of data processing systems including both hardware and software. Taking a closer look at each, the distinction between them is decreasing nowadays.

The need to transform software design and development into an engineering-type discipline called into existence *software engineering* (SE) and the software engineer whose speciality is the design and construction of software systems.

The curriculum development of the CSE education of the USA can be identified by a few basic milestones.

1. The ACM's Curriculum Committee on Computer Science (C^3S) published a set of recommendations called the *Curriculum '68* in 1968 [13]. The prerequisite structure of courses of Curriculum '68 is shown in Figure 17.

2. The *COSINE Committee* of the National Academy of Engineering discussed various aspects of computer education in the mid sixties [14] [15] [16].

3. The *ACM Curriculum Committee* on Computer Education for *Management* issued Curriculum Recommendations for Undergraduate and Graduate Programs in Information System in 1972 and 1973, respectively [17] [18]. The core course sequences are given in Figure 18, while the course relationships in Figure 19.

4. The Education Committee (Model Curriculum Subcommittee) of the *IEEE Computer Society* presented the revised version of its Curriculum in Computer Science and Engineering in January,

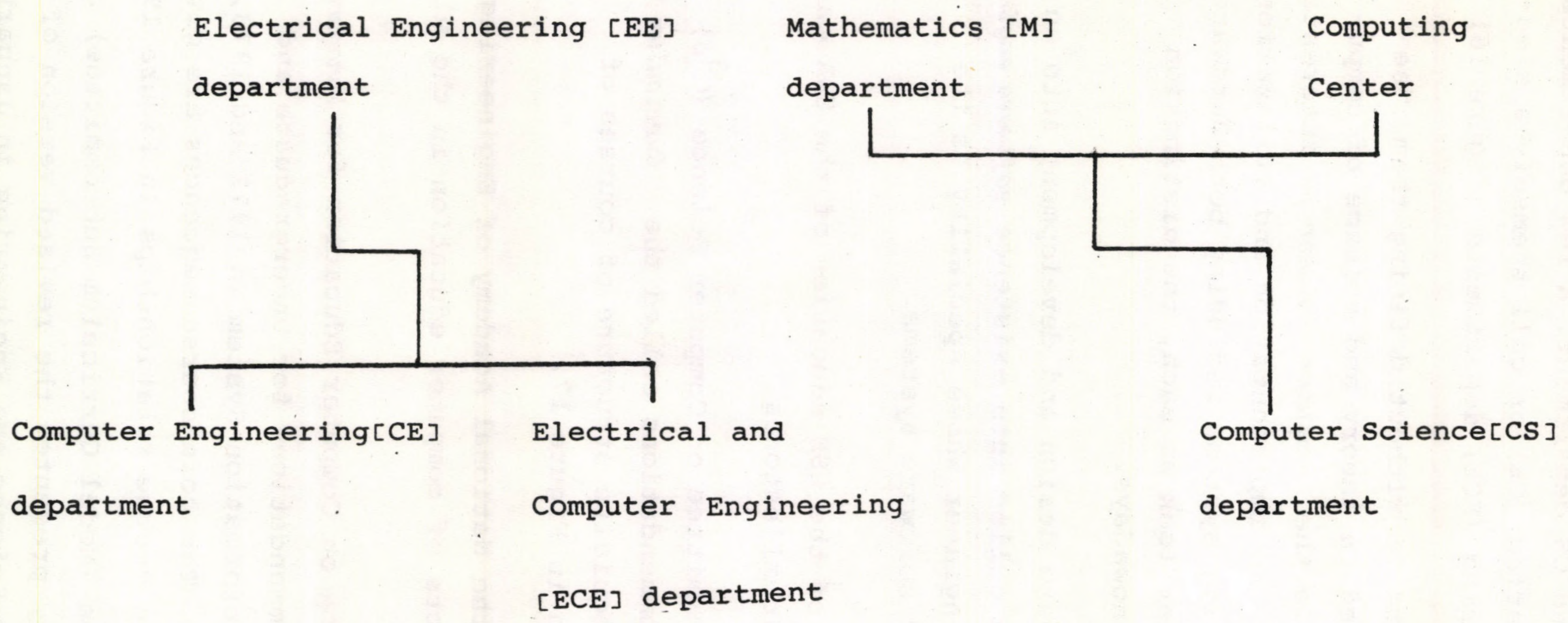
Electrical Engineering [EE] department
Mathematics [M] department
Computing Center
Computer Engineering[CE] department
Electrical and Computer Engineering [ECE] department
Computer Science[CS] department

Figure 16.

Family tree of computer science education [CSE] departments.

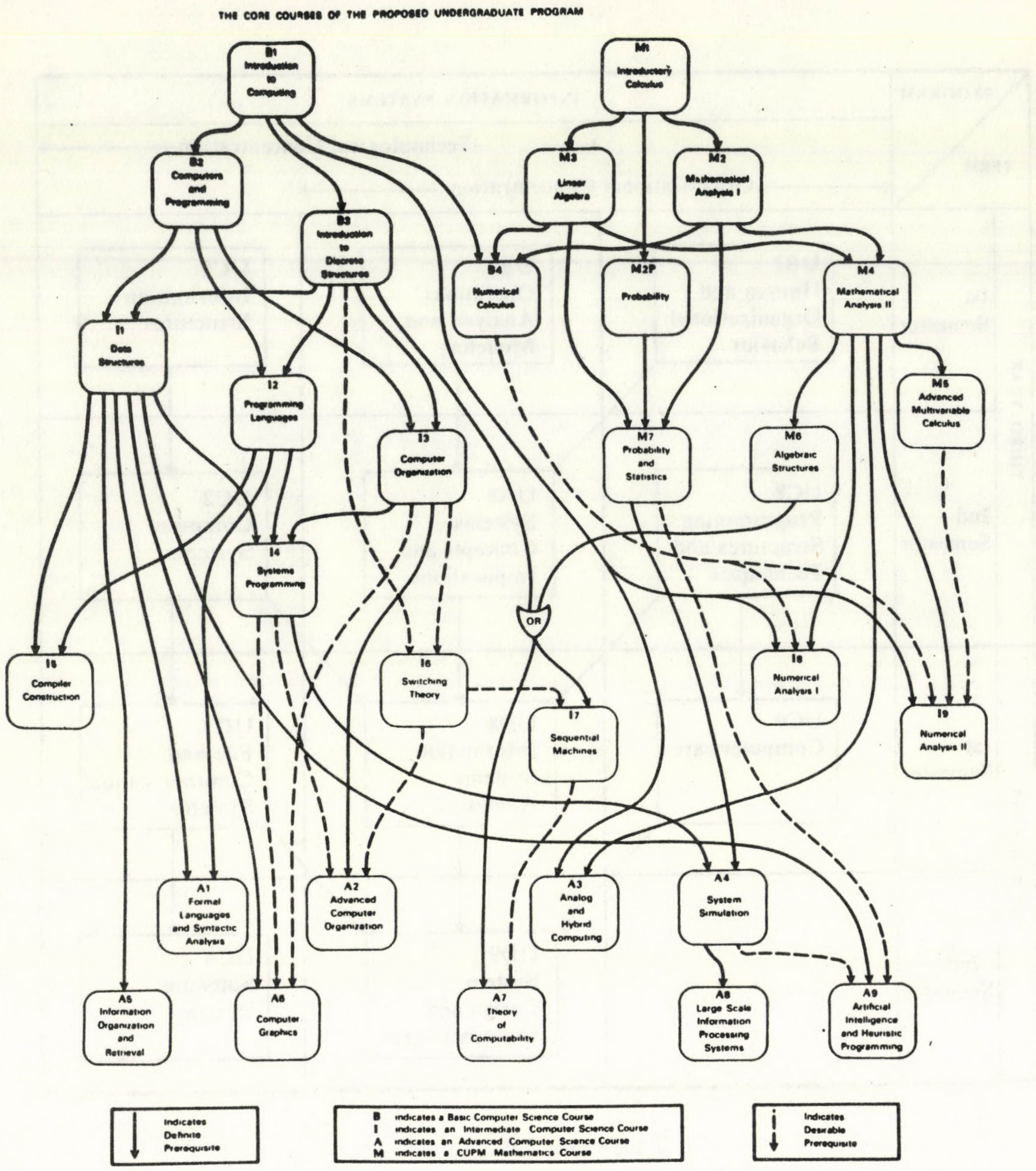
THE CORE COURSES OF THE PROPOSED UNDERGRADUATE PROGRAM
B1 Introduction to Computing
M1 Introductory Calculus
B2 Computers and Programming
M3 Linear Algebra
M2 Mathematical Analysis I
B3 Introduction to Discrete Structures
B4 Numerical Calculus
M2P Probability
M4 Mathematical Analysis II
I1 Data Structures
I2 Programming Languages
M5 Advanced Multivariable Calculus
I3 Computer Organization
M7 Probability and Statistics
M6 Algebraic Structures
I4 Systems Programming
OR
I5 Compiler Construction
I6 Switching Theory
I7 Sequential Machines
I8 Numerical Analysis I
I9 Numerical Analysis II
A1 Formal Languages and Syntactic Analysis
A2 Advanced Computer Organization
A3 Analog and Hybrid Computing
A4 System Simulation
A5 Information Organization and Retrieval
A6 Computer Graphics
A7 Theory of Computability
A8 Large Scale Information Processing Systems
A9 Artificial Intelligence and Heuristic Programming
Indicates Definite Prerequisite
B indicates a Basic Computer Science Course
I indicates an Intermediate Computer Science Course
A indicates an Advanced Computer Science Course
M indicates a CUPM Mathematics Course
Indicates Desirable Prerequisite

Figure 17.

Prerequisite structure of courses in

Curriculum '68 [13].

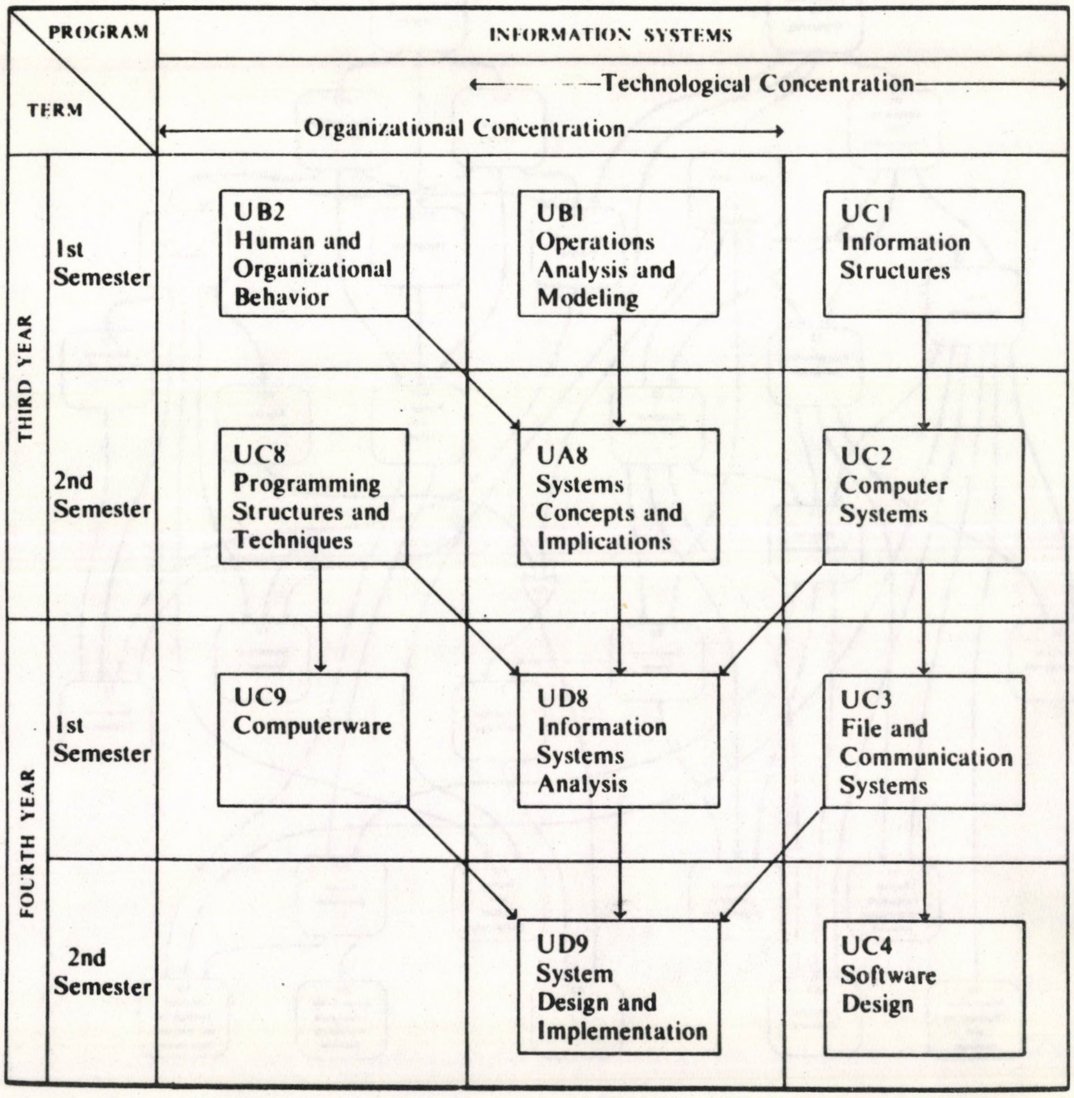
PROGRAM
TERM
INFORMATION SYSTEMS
Technological Concentration
Organizational Concentration
THIRD YEAR
FOURTH YEAR
1st Semester
2nd Semester
1st Semester
2nd Semester
UB2 Human and Organizational Behavior
UB1 Operations Analysis and Modeling
UC1 Information Structures
UC8 Programming Structures and Techniques
UA8 Systems Concepts and Implications
UC2 Computer Systems
UC9 Computerware
UD8 Information Systems Analysis
UC3 File and Communication Systems
UD9 System Design and Implementation
UC4 Software Design

Figure 18.

Core course segnences for information

systems programs [18].

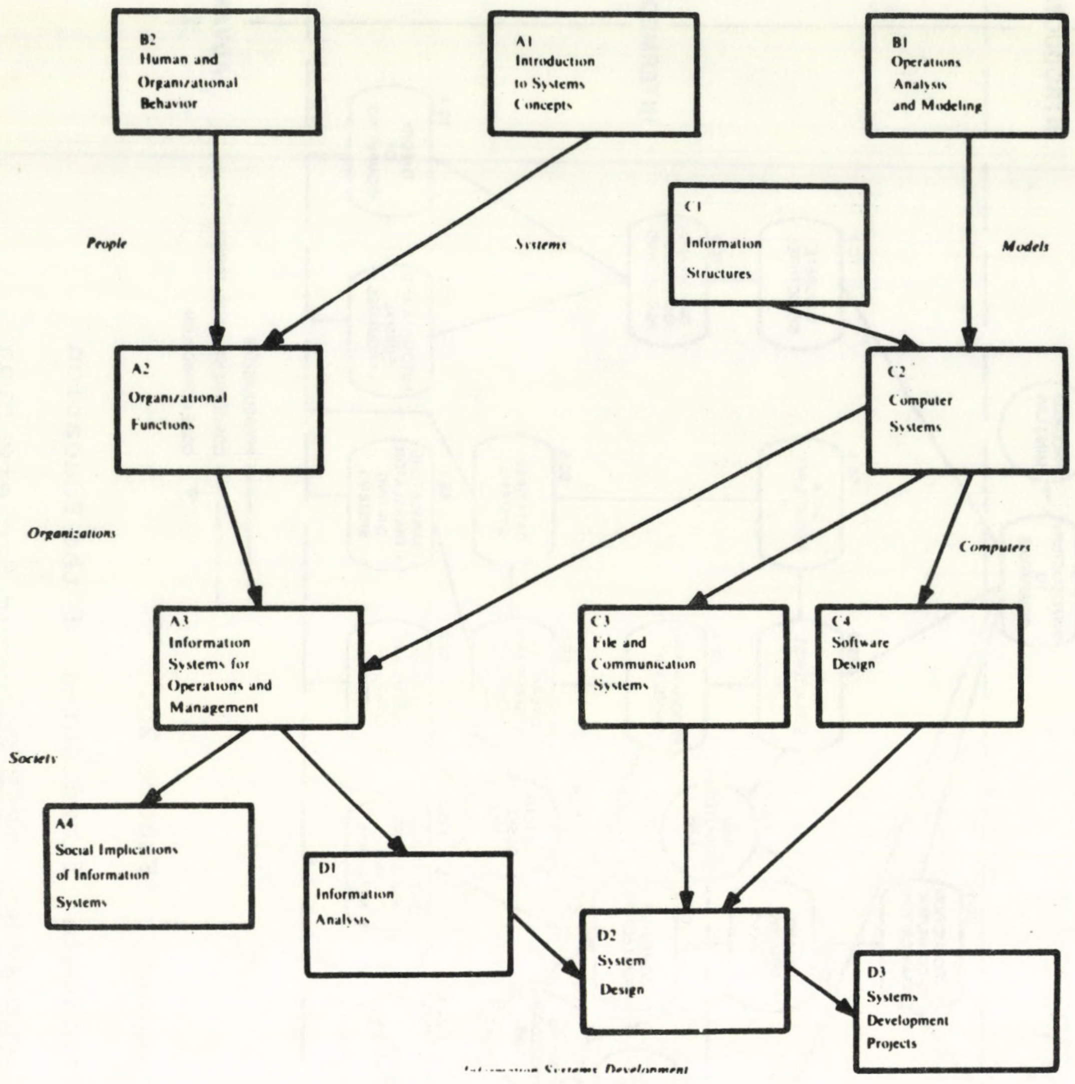
B2 Human and Organizational Behavior
A1 Introduction to Systems Concepts
B1 Operations Analysis and Modeling
C1 Information Structures
People
Systems
Models
A2 Organizational Functions
C2 Computer Systems
Organizations
Computers
A3 Information Systems for Operations and Management
C3 File and Communication Systems
C4 Software Design
Society
A4 Social Implications of Information Systems
D1 Information Analysis
D2 System Design
D3 Systems Development Projects
Information Systems Development

Figure 19.

Course relationship [17].

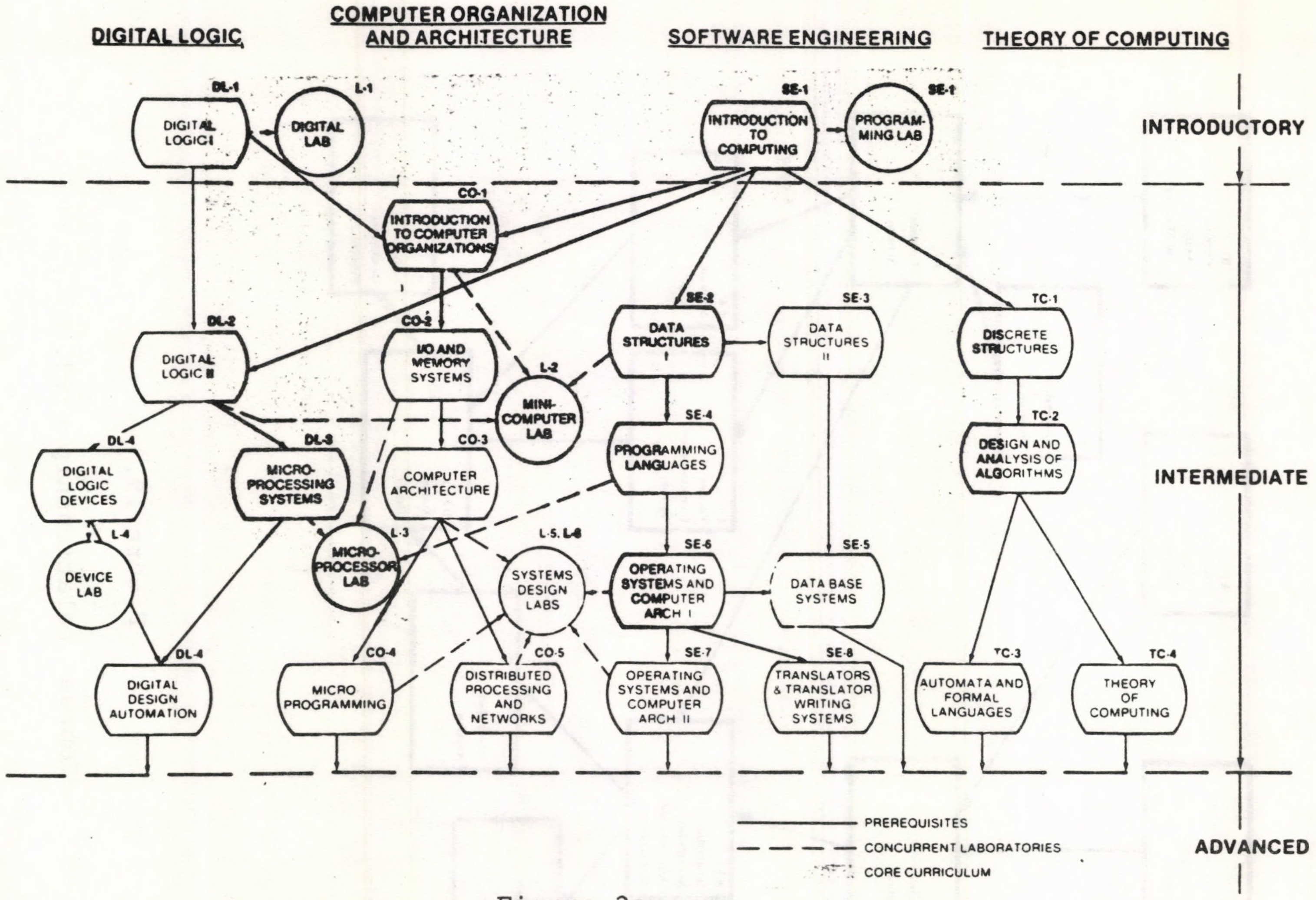
DIGITAL LOGIC
COMPUTER ORGANIZATION AND ARCHITECTURE
SOFTWARE ENGINEERING
THEORY OF COMPUTING
INTRODUCTORY
INTERMEDIATE
ADVANCED
DL-1 DIGITAL LOGIC I
L-1 DIGITAL LAB
SE-1 INTRODUCTION TO COMPUTING
SE-1 PROGRAMMING LAB
CO-1 INTRODUCTION TO COMPUTER ORGANIZATIONS
DL-2 DIGITAL LOGIC II
CO-2 I/O AND MEMORY SYSTEMS
L-2 MINI-COMPUTER LAB
SE-2 DATA STRUCTURES I
SE-3 DATA STRUCTURES II
TC-1 DISCRETE STRUCTURES
DL-4 DIGITAL LOGIC DEVICES
DL-3 MICRO-PROCESSING SYSTEMS
CO-3 COMPUTER ARCHITECTURE
SE-4 PROGRAMMING LANGUAGES
TC-2 DESIGN AND ANALYSIS OF ALGORITHMS
L-4 DEVICE LAB
L-3 MICRO-PROCESSOR LAB
L-5, L-6 SYSTEMS DESIGN LABS
SE-6 OPERATING SYSTEMS AND COMPUTER ARCH I
SE-5 DATA BASE SYSTEMS
DL-4 DIGITAL DESIGN AUTOMATION
CO-4 MICRO PROGRAMMING
CO-5 DISTRIBUTED PROCESSING AND NETWORKS
SE-7 OPERATING SYSTEMS AND COMPUTER ARCH II
SE-8 TRANSLATORS & TRANSLATOR WRITING SYSTEMS
TC-3 AUTOMATA AND FORMAL LANGUAGES
TC-4 THEORY OF COMPUTING
PREREQUISITES
CONCURRENT LABORATORIES
CORE CURRICULUM

Figure 2o.

CSE Curricula flowdiagram of the Education Committee of the IEEE Computer Society [19].

1977 [19] [20]. The subcommittee divided the CSE program into four subject areas i.e. digital logic, computer organization and architecture, software engineering and theory of computing (Figure 20). It was the first real effort to bridge the gap between computer science and computer engineering and to integrate hardware and software as well as theory and practice. Another significant contribution of the Computer Society is the report on computer architecture curriculum [21]. An extensive survey of the literature in computer science education since Curriculum'68 was published in January 1977 [22].

5. In March 1979 the ACM Curriculum Committee on Computer Science published its new recommendations for the undergraduate program in computer science, called *Curriculum'78* [23]. Its course structure is illustrated in Figure 21.

4. ISSUES FACING COMPUTER SCIENCE AND ENGINEERING EDUCATION.

4.1. Keeping up with the computer revolution.

Before addressing the question of how to keep up with the computer revolution,first we must understand the *computer revolution* itself [24]. The computer revolution is the product of the availability of cheap high level functions in small boxes. Currently complete processors are available on *single chips* in the future even large processors and complete computers will be placed on single chips [25] (See figure 22). This drastically affected the *economics of computers* making them available for use in a wide range of products. However, basic computer architectures have *not* been revolutionized by the computer revolution. Instead, it has continued a steady evolution. No one has really invented substantially new architecture recently. Basics, well taught, readily leads the engineer into new technology. The revolution of the computer revolution is the revolution of computer applications, not computer theory.

With this understanding, the question becomes not how to keep up, but what *fundamental principles and techniques* of computing and computer hardware should be taught. If educators try to keep up with

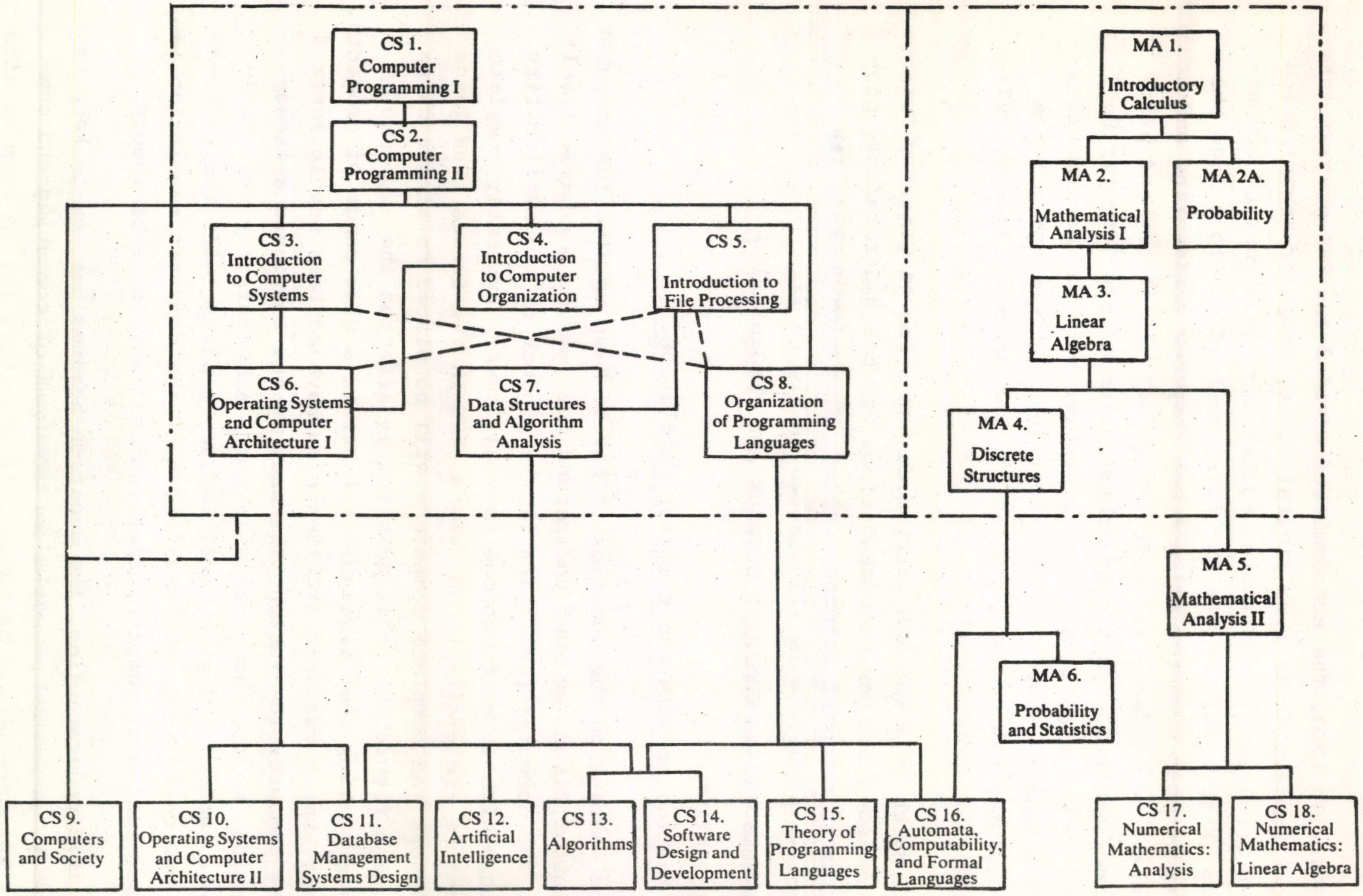
CS 1. Computer Programming I
CS 2. Computer Programming II
CS 3. Introduction to Computer Systems
CS 4. Introduction to Computer Organization
CS 5. Introduction to File Processing
CS 6. Operating Systems and Computer Architecture I
CS 7. Data Structures and Algorithm Analysis
CS 8. Organization of Programming Languages
CS 9. Computers and Society
CS 10. Operating Systems and Computer Architecture II
CS 11. Database Management Systems Design
CS 12. Artificial Intelligence
CS 13. Algorithms
CS 14. Software Design and Development
CS 15. Theory of Programming Languages
CS 16. Automata, Computability, and Formal Languages
CS 17. Numerical Mathematics: Analysis
CS 18. Numerical Mathematics: Linear Algebra
MA 1. Introductory Calculus
MA 2. Mathematical Analysis I
MA 2A. Probability
MA 3. Linear Algebra
MA 4. Discrete Structures
MA 5. Mathematical Analysis II
MA 6. Probability and Statistics

Figure 21.

Recommended courses of Curriculum'78 [23]

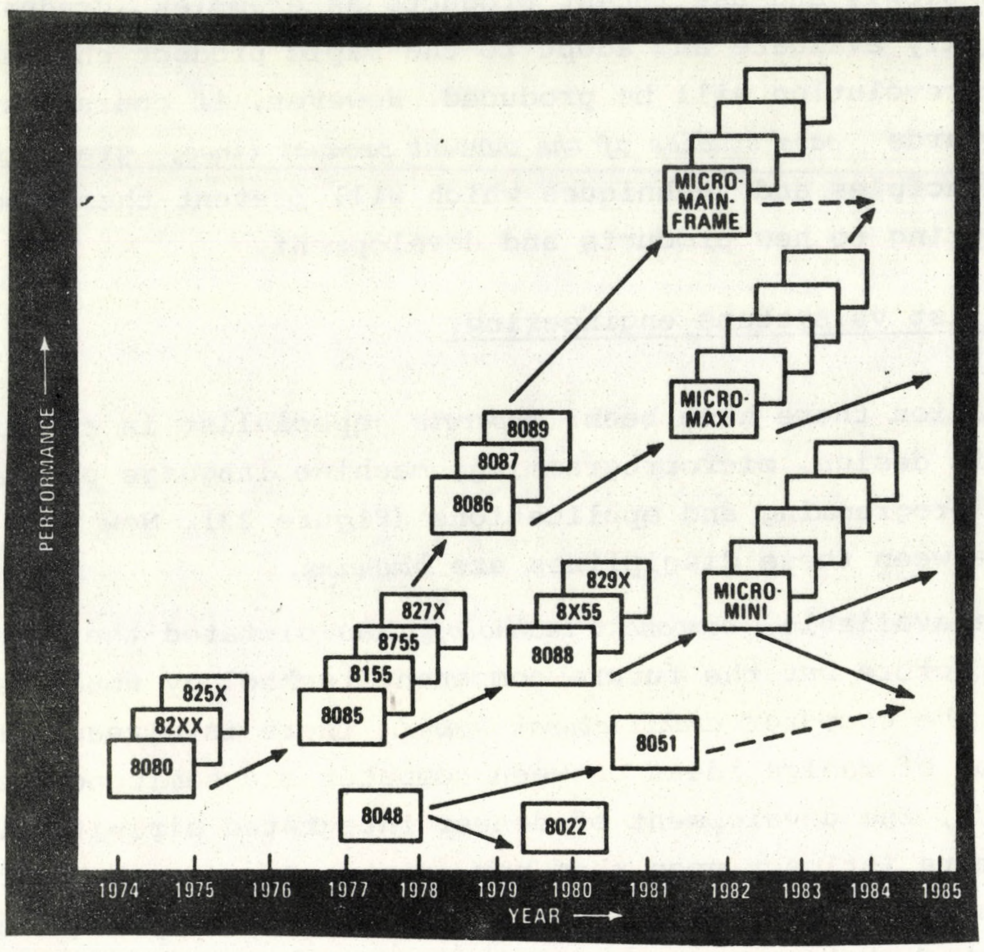

Figure 22.

Intel's plans for the 8o's [25].

every manufacturer's latest hardware and software product, they will be swamped with data that are out-of-date by the time they have been incorporated into a course. Instead, if courses are oriented toward the fundamental principles and techniques, which evolve more slowly and use recent products as examples, graduates who can quickly evaluate and adopt to the rapid product changes of computer revolution will be produced. However, if courses are oriented towards *case studies of the current product lines,* graduates may miss principles and techniques which will prevent them from rapidly adopting to new products and development.

4.2. Specialist vs.systems engineering.

For a generation there have been *separate* specialist in circuit design, logic design, microprogramming, machine language programming, system programming and applications (Figure 23). Now the *boundaries* between these disciplines are *blurring.*

Till now the available *component technology* has dictated the current micro-architecture but the future component technology would be directed by the computer *architectural demand.* There is already cross fertilization of design ideas between computer and semiconductor manufacturers, the development of denser integrated circuits will force even more intimacy upon them now exists. Because the hardware engineering is no longer insulated from software pruduction and the *logic design is merging with semiconductor fabrication,* the engineer's education has to be broden.

Logic design and programming, as seperate specialities, are going the way of the dinosaur. There is a widening gap between supply and demand of systems-oriented computer engineers [26].People (called *system engineers*) in this field are engineers who can pull together from various disciplines all the elements needed to satisfy an application. One of these disciplines is semiconductor design. The systems engineer responsible for specifying custom chips to the semicondustor industry must be knowledgeable of *semiconductor processing* rules. Not only must he know exactly what he needs-which means he must thoroughly understand the application-but he must also figure out what kind of a program it

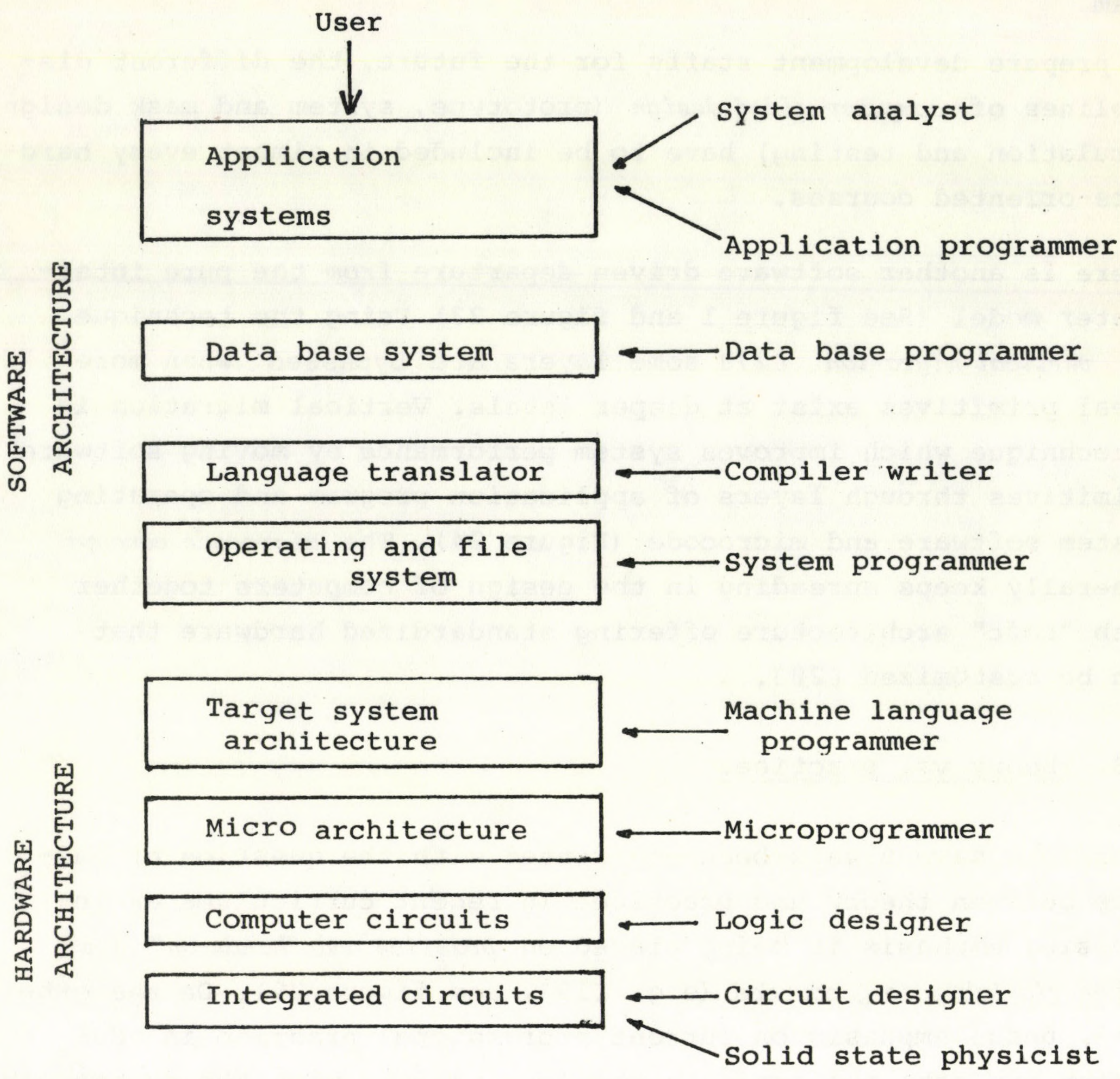

Figure 23.

"Architecture" of computer sysrems and its relationship with different specialist.

will take and what kind of processor can best execute this program.

To prepare development staffs for the future, the different disciplines of *computer-aided design* (prototype, system and mask design, simulation and testing) have to be included in almost every hardware-oriented courses.

There is another software driven departure from the pure interpreter model (See figure 1 and figure 23) Using the technique of *vertical migration* [27] some layers are bypassed when more ideal primitives exist at deeper levels. Vertical migration is a technique which improves system performance by moving software primitives through layers of application program and operating system software and microcode (Figure 24). The *microcode concept* generally keeps spreading in the design of computers together with "soft" architecture offering standardized hardware that can be customized [28].

4.3. Theory vs. practice.

Educators have always been confronted with the question of balance between theory and practice. In recent curriculums an increasing emphasis is being placed on *practice and "hands-on" experience via laboratory courses* (e.g. [19], see figure 25). On the other hand, undue emphasis on current professional practice in education can make the graduate obsolete rapidly with the changes in technology. Here then the theoretical emphasis serves as "tonic"- or as "vitamin" toward building a solid intellectual foundation on which advances in technology can be absorbed and built upon. While theoretical emphasis is desired, one has to bear in mind that computer engineering is a complete set of activities including the use of taxonomies, theories, models and heuristics, associated with the design and construction of computers. It is like other engineering and the definition [1] is especially appropriate: engineers first turn to science for answer and help, then to mathematics for models and intuition, and finally to the seat of their pants. It also means that longer period of

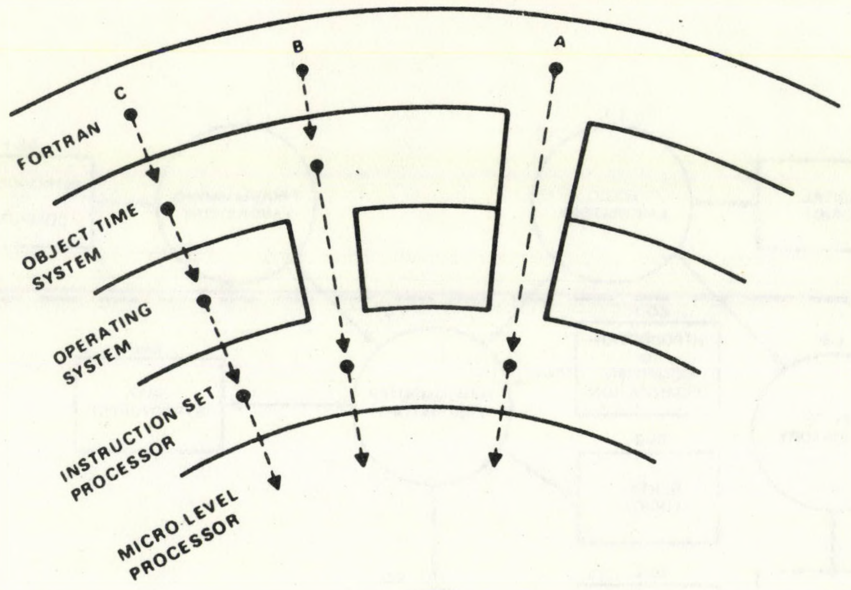

Figure 24.

Levels-of-interpreters with

"pipes" that bypass levels [1].

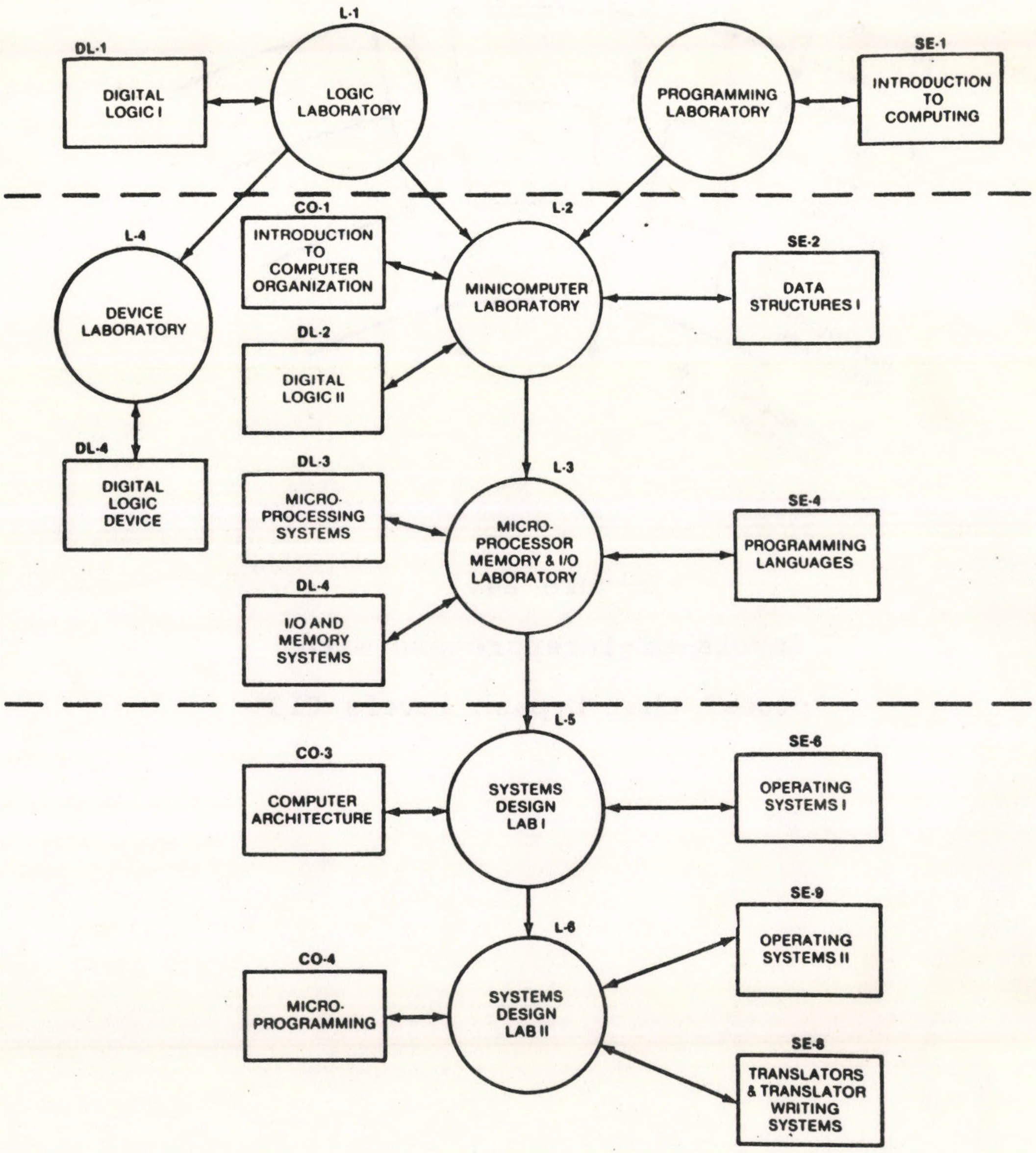

Figure 25.
The segnence of laboratories [19].

training in the industry is necessary before the new graduate is "broken-in" into professional activities.

Regarding the *role of theory* in the computer science and engineering curriculums the difficult question of "why, what, when and where" (and the biggest problem: to decide what material can most easily be left out) we refer to [29].

4.4. Engineering vs. software engineering.

Because of the cost of hardware is dropping rapidly and software productivity improves only slowly, the *cost of software* relative to hardware is *increasing*. The software/hardware ratio was about 2:1 in 1973 [30] 5:4 in 1978 [31], and it is projected as going to 10:1 by 1985 [32].

Because of a great deal of software is already in existence, some of it of low quality, more and more effort, of necessity has to be devoted to *maintenance*. It was estimated that up to 75 [33] or 80 [34] percent of all programming activity might be maintenance, including enhancement and modification of existing programs. Without real changes the software cost would set limits to the utilization of VLSI (considering the yearly incresae the USA would need in excess of 1 million software engineers by 1990.) It is described as a "*programmers catastrophe*" [25].

In software engineering, the very term, engineering implies that the *entire* development of a product from initial conception through testing and maintenance is organized in an *orderly, manageable way*.

Nowadays the *practical software process* differs from the hardware process in many ways. For one thing there is too much freedom in "software land". For example it is entirely possible to design software components using textbook examples, friends' recommendations, or your own immaginations for that matter. Another factor is the *backgrounds of programmers*, they may have degrees in mathematics, in engineering, history, journalism, teaching or whatever . This lack of common background may be part of the cause, - although modern programming parctices have been ready for use for a considerable time, - that conservatism is a legacy forced on the computing community by past and present development practices.

Software engineering is just becoming recognized as a *legitimate academic discipline.* As a result, software engineering education is in a very primitive state but will develop rapidly in the years ahead. Strong computer science education is, and will be a vital part of the education of a software engineer. Recent *curriculum proposals* for computer science education have identified several options within a computer science undergarduate program [33].

4.5. Continuing education.

Generally accepted estimates state that the available knowledge in most areas of computing science and related technologies *doubles* approximately *every 5-8 years.* Although a computer professional is expected to continue to learn "by osmosis" from his peer group for the major part of his productive life, continuing education can clearly assist him in acquiring the necessary new knowledge in rapid and systematic manner. Universities and colleges, industry and professional societies play different roles in this process [34] [35]. The major problems are motivating the employer to support an employee's participation and the employee to participate.

4.6. Impact of microsystems.

There are contraditory opinions regarding this effect, e.g. "The *'microcomputer-revolution'* is best understood by realizing that a microcomputer is basically just an *inexpensive computer.* Most microcomputer concepts are already covered in a good engineering curriculum [36].

"The *proliferation* of microcomputer systems will have an anormous impact on computer science and computer engineering education" [34].

However, almost all of the outhors agree that a micro-lab facility can provide students with exposure to concepts and problems such as actual hardware, computer operation,operating systems, backup procedures, program size problems, inter-computer communications, scheduling, maintenance and computer management.

4.7. Computer and society.

One of the biggest challenges of the 80's may be educate the *average citizen* about computers. In particular, people should be able to separate fact from fiction with regard to computer-related stories on TV and in newspapers and popular magazines.

The growing interest and public support for the computer literacy concept was cited as a reason in a current US recommendation [37] that all high school graduates be computer literate.

To achieve the desired *computer literacy*, graduates should have knowledge of the historical perspective of computing, the computer anatomy (includes parts, work and problem solving) basic uses of computers, social implications and futuristics.

5. REFERENCES.

[1] C.G.Bell,J.C. Mudge and J.E.McNamara: Computer engineering. Digital Press,1978.

[2] C.G.Bell and A.Newel:Computer structureas Readings and examples. McGraw-Hill,New-York,1971.

[3] M.Asimov: Introduction to design. Prentice-Hall, Englewood Cliffs,N.J., 1962.

[4] M.Phister: Data processing technology and economics.Santa Monica Publishing Co., Santa Monica,Calif. 1976.

[5] G.A.Blaauw:Hardware requirements for the fourth generation. In Fourth generation computers: User requirements and transition, F.Gruenberger(ed.),Prentice-Hall, Englewood Cliffs,N.Y. 1970,155-168.

[6] J.C.Peterschmitt: The challange of the eighties. Digital Europa, December 1979,p.3.

[7] E.Bloch and D.Galage: Component progress: Its effect on high-speed computer architecture and machine organization. Computer, April 1978, 64-76.

[8] G.Moore: VLSI:some fundamental challenges. IEEE Spectrum, April 1979,30-37.

[9] R.N.Noyce: Large scale integration: What is yet to come? Science, 195(1977), 1102-1106.

[10] D.A.Hodges:Progress in electronic technologies for computer. National Bureau of Standards Report T73219,March 1977.

[11] D.P.Bhandarkar: The impact of semiconductor technology on computer systems. Computer,September 1979,92-98.

[12] A.Csákány and F.Vajda: The impact of computer availability on engineering education in Hungary. IEEE Transactions on Education, February 1977, 2-6.

[13] Curriculum'68. Recommendations for Academic Programs in Computer Science. (A report of the ACM Curriculum Committee on Computer Science) Communications of the ACM, March 1968, 151-197.

[14] COSINE Committee, Some specifications for computer-oriented first course in electrical engineering. Commission Engineering Education, Washington. D.C., September 1968.

[15] COSINE Committee, An undergraduate electrical engineering course on computer organization. Commission Engineering Education,Washington. D.C., October 1968.

[16] COSINE Committee, Some specifications for undergraduate course in digital systems. Nat.Acad.Eng.Washington D.C., November 1968.

[17] Curriculum Recommendations for Graduate Professional Programs in Information Systems. (A Report of the ACM Curriculum Committee on Computer Education for Management) Communications of the ACM, May 1972, 364-398.

[18] Curriculum Recommendations for Undergraduate Program in Information Systems. (A Report of the ACM Curriculum Committee on Computer Education for Management) Communications of the ACM, December 1973, 727-749.

[19] A Curriculum in Computer Science and Engineering. (Prepared by the Education Committee of the IEEE Computer Society) IEEE Computer Society, EH 0119-8 January 1977.

[20] Special Supplement: Computer Science and Engineering Education. Computer, December 1977, 70-135.

[21] G.E.Rossmann et al.: A course of study in computer hardware architecture. Computer, December 1975, 44-63.

[22] R.H.Austin et al.: A survey of the literature in Computer Science education since Curriculum'68. Communications of the ACM,January 1977,13-21.

[23] Curriculum'78. Recommendations for the undergraduate program in Computer Science (A Report of the ACM Curriculum Committee on Computer Science) Communications of the ACM, March 1979, 147-165.

[24] Keeping up with the computer revolution. IEEE Transaction on Education, May 1979, 39-43.

[25] J.G.Posa: Intel takes aim at the 80's. Electronics, February 28, 1980,89-95.

[26] D.D.McCracken et al.: An ACM Executive Committee position on the crisis in experimental Computer Science. Communication of the ACM, September 1979, 503-504.

[27] J.Stockenberg and A.van Dam:Vertical migration for performance enhencement in layered hardware/firmware/software systems. Computer, May 1978, 35-50

[28] A.Durniak:VLSI shakes the foundations of computer architecture. Electronics, May 24, 1979, 111-133.

[29] B.M.Barnes: Theory in the computer science and engineering curriculum: Why, what, when and where. Computer, December 1977, 106-108.

[30] B.W.Boehm: Software and its impact: A quantitative assessment. Datamation, May 1973, 48-59.

[31] A.I.Wasserman and P.Freeman:Software engineering Education: Status and prospects. Proceeding of the IEEE, August 1978, 886-892.

[32] W.Myers: The need for software engineering. Computer, February 1978, 12-26.

[33] A.I.Wasserman and P.Freeman: Software engineering concepts and Computer Science curricula. Computer, June 1977, 85-91.

[34] U.W.Pooch et al.(Computer Science and Computer Engineering education in the '80s. Computer, September 1978, 69-83.

[35] R.Chattergy and U.W.Pooch: Continuing education for the computer professional: The role of the professional society. Computer, December 1977, 124-128.

[36] J.F.Wakerly and E.J.McCluskey: Microcomputers in the computer engineering curriculum. Computer, January 1977, 32-38.

[37] R.H.Austing and G.L.Engel: Recent developments in computers and society research and education. Proc. AFIPS.vol. 48.1979 National Computer Conference.AFIPS Press, New York, 1979, 407-410.

63.002

Kiadja a Központi Fizikai Kutató Intézet
Felelős kiadó: Sándory Mihály
Szakmai lektor: Csákány Antal
Nyelvi lektor: Jávor András
Példányszám: 270 Törzsszám: 80-233
Budapest, 1980. április hó