TRENDS IN COMPUTER SYSTEM STRUCTURE AND ARCHITECTURE

W.G. Spruth

IBM Entwicklung und Forschung
7030 Boeblingen, Germany

1. INTRODUCTION

An information processing machine of the most general type is characterized by its ability to generate information for a predefined purpose. This information is the result of a processing operation for which both input data and internally stored data have been utilized. There are many types of information processing machines. Examples are the Touring Machine, the Analog Computer and the Digital Differential Analyzer. The digital computer, or as we nowadays call it, the data processing system, is another special case. It is characterized by the sequential processing of individual instructions of a prestored program and the fact that program and data occupy a common "Main Store". Data Processing Systems can be classified by their major application as Information Systems, Problem Solving Systems, and Object Systems. As to mode of operation, they can be classified as batch systems and interactive systems:

	BATCH	INTERACTIVE
INFORMATION SYSTEM	DATA BASE	RESERVATION SYSTEM, DATA BASE/DATA COMMUNICATION SYSTEM
PROBLEM SOLVING SYSTEM	STANDARD BATCH	TIME SHARING
OBJECT SYSTEM	DATA COLLECTION	AUTOMATION & CONTROL

There are three major elements which make a data processing system into an information processing machine: hardware, system software and application programs. Hardware and system software are supplied by the manufacturer of a data processing system. There is no clear distinction between those two functions, and there are some types of systems which perform functions in hardware which other types of systems perform through system programs.

The application program contains the algorithm which defines how to generate output information using both input and internally stored data. In addition it contains features which adapt it to the executing system. A data processing system is thus a machine which is capable to execute many, independently produced application programs. Its architecture and structure can be described independently of its application programs.

The *architecture* of a data processing system can be defined as the functional appearance of the system to a user, its phenomenology. The structure of a system is characterized by the manner in which the individual building blocks are interconnected to implement the architecture. A modern system architecture has three major components (Fig. 1):

Processing
Storage
Input/Output

A possible implementation is shown in Fig. 2.

2. UNDERLYING DRIVING FORCES

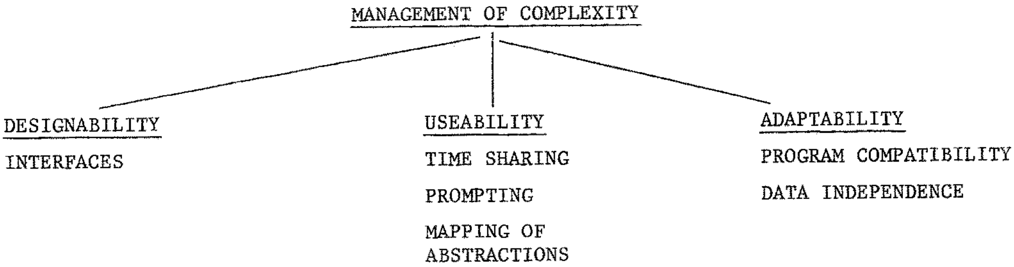
Features and characteristics of a computer system architecture and structure are driven by three underlying forces. The first of these has to do with the *concept* of the digital computer. Characteristics are indicated in Fig. 3.

The second underlying driving force is controlled by consideration of the *technology* used for implementation *and* the resulting *operational efficiency*. During the last 30 years technology developments have to a significant extent, been one of the underlying driving forces in the development of computer structures. Fig. 4 indicates some of these developments. It has been popular to indicate technological progress on an exponential curve, but it is unusual to see exponential development curves continue for periods as long as 30 years. Usually, a technological development curve is S-shaped with a period of slow evolution, rapid growth and final maturity. The extended progress in data processing technology has been sustained by the repeated superposition of several of those curves, as indicated in Fig. 5. The replacement of the Williams tube by the core memory and later by the monolithic memory are an example in point.

Because of technological deficiencies, emphasis has been placed on improving the operational characteristics of a data processing system. Features are implemented to achieve optimum results from given technological capabilities. Examples are:

Multiprogramming
Multiprocessing
Job and Task Management
Allocation and Resource Management

The third underlying driving force has to do with the *"Management of Complexity"*. Data processing systems are among the most complex structures ever invented and built by man. Managing this complexity has been a problem of ever increasing importance. Problems associated with the Management of Complexity have to do with designability, useability and adaptability:



The problem of *designability* in particular is being attacked by creating interfaces within a system structure. Interfaces decrease performance and increase manufacturing cost. They do, however, reduce the development effort, and improve reliability, recovery, and repairability. They are not defined by physical science and strongly impacted by technological progress. Examples are:

/370 channel interface
3830 DCI interface
Machine-micro-nano-instructions
Supervisor/Problem status, SVC interrupt
Operating System control blocks, e.g. TCB, DCB, UCB
Disk physical data organization (count, key, data fields)
Disk access methods, e.g. VSAM
Data Base schema and subschemata, DL/1
Branch and Link
Structured Programming

There are probably many cases of "natural interfaces" which impact performance and cost only to a moderate degree. Finding such natural interfaces appears to be anything but an exact science.

In many cases there are strong pressures for architectural standardization of existing interfaces. It should be emphasized that only well understood functions lend themselves to standardization. Our understanding of data processing functions is much more limited than we frequently believe. For example, a function like the EDIT instruction in the /360 architecture found poor utilization because it is not properly defined. Every system in the /370 product line succeeded in implementing a different address translation buffer scheme. The VS1 and VS2 operating systems have different page replacement algorithms, with only limited understanding as to the advantages and disadvantages of the different algorithms being utilized.

The three underlying driving forces mentioned so far, have had different levels of importance over the years. This is indicated in Fig. 6.

3. MANAGEMENT OF COMPLEXITY

Next to designability, improvements in *useability* are a major issue in the management of complexity. Prompting, checkout and debugging aids fall into this category. Interactive use, especially time sharing in problem solving applications is another case. The most important contribution to improvements in useability have been achieved through the use of abstractions. The user works with *logical abstractions* of a data processing system which are simpler and easier to understand than the actual physical features of the system. To this purpose, simplified logical structures are *mapped* onto the more complex physical structures. A modern system contains several levels or layers of abstractions which in turn apply to programs, data, and system commands. Mapping is performed in 4 major system areas: language translators, system programs, CPU hardware, and channel and control unit hardware. An overview over this multilayered mapping of abstractions is given in Fig. 7.

The most important abstraction level is the layer of machine architecture, characterized by machine instructions, I/O instructions, and data addressing facilities for both main store and external data. Most application programs are written for a higher abstraction level in a "Higher Level Language" like Cobol, RPG, Fortran or PL/1. A given system usually has a single architecture interface but multiple higher level languages, thus the duality of these two layers.

Mapping of one level of abstraction onto the layer below is usually done interpretatively; higher level language compilation being one major exception. CPUs very often execute machine instructions through an interpretation hierarchy. Early computers had relatively limited instruction sets. As time evolved, the instruction sets of intermediate and large size systems became more comprehensive both in the size of the repertoire and the complexity of system control functions implemented in individual instructions.*

The system /360 architecture, introduced in 1964, utilized therefore in most implementations the microprogramming concept originally proposed by Wilkes [1]. The first /360 machines had microinstructions of the horizontal type characterized by the fact that they usually could be executed within one machine cycle. Lateron microinstructions

*It is interesting to note that the increase in instruction set complexity applied most to logical and system control functions. Early computers, e.g. the Harvard Mark II, had special instructions to perform complex mathematical functions (e.g. sin). Modern machines use mostly subroutines for this purpose.

themselves became more complex, in particular with the introduction of the writable control store in intermediate size systems [2]. Because of their complexity, intermediate systems started to use an interpretation approach for the microinstructions themselves, which leads to nano- or picoinstructions, as indicated in Fig. 8. The 2-level instruction interpretation implies some inefficiency in instruction execution. Large systems therefore do not use microinstructions or at least not a dual interpretation hierarchy. Because they perform instruction execution mostly in parallel, the utilization of a complex instruction repertoire is an advantage. With the use of these two parallel implementation approaches we see a trend to utilize the same architecture for both intermediate and large systems. Mini systems (e.g. S/32, PDP 11, Eclipse) feature a functionally more limited architecture to achieve lower cost (Fig. 9).

Data structures are mapped into several layers of abstractions in a similar way as programs. Particular examples are the virtual store concept and the data models and data submodels used in data base systems.

In actuality, the storage part of a system architecture gives to the user the external appearance of 3 independent stores, two of them implemented by a storage hierarchy. This is indicated in Fig. 10. Of particular importance is the fact that data sets are stored in a different type of logical store than program instructions and working buffers. The READ-, WRITE-, GET-, PUT-operations logically map a single record of an external data set logical store into a corresponding work buffer area of the virtual store (Fig. 11). The MULTICS system has tried to merge both types of logical storages into a single storage [3]. However, the industry so far has not been able to find ways and means to implement this concept in today's systems.

Adaptability is the third major element to manage complexity. Emulators, virtual systems, link edit and data independence functions are its major components. In particular data base system structures are to a large extent impacted by the requirement for easy adaption to an ever changing set of external influences.

4. ARCHITECTURAL CLASSIFICATION

System complexity results in a stratification of abstraction levels within a given system architecture. System size leads to a classification of different system architectures. It is interesting to compare subsequent generations of machines of the same architecture. Fig. 12 shows the CPU cycle time vs. the CPU speed for two subsequent generations of machines of the /360 and /370 architecture. Faster machines get their speed, compared to slower machines, partially through faster CPU cycles, and partially by using fewer CPU cycles for each machine instruction execution (and a corresponding increase in CPU hardware). It is also interesting to see that the more powerful, but also more complex /370 architecture requires a noticeably shorter CPU cycle time than the /360 architecture for each average machine instruction execution. A similar trend can be observed if we plot the CPU circuit count against CPU speed, as indicated in Fig. 13. Main store access time seems to have very little relationship to CPU speed (Fig. 14), while the main store transfer rate grows more than linear with CPU speed (Fig. 15). Large systems compensate for slow memory access time through techniques like parallel memory access, cache, etc.

CPU speed is only one of several factors which determine system performance. The industry presently manufactures and uses a wide spectrum of different system sizes. A classification of these system sizes into microcomputers, mini systems, intermediate systems, and large systems is given in Fig. 16. Important classification factors are addressing and I/O architecture, operating system characteristics, main store and disk store size. A significant break can be observed between a microcomputer, which has programmed I/O, and the minicomputer which works with interrupt I/O. The next significant architectural differentiation is between the mini system which advantageously utilizes a 16 bit addressing scheme in a non-virtual memory operation, and the intermediate system which eliminates both these limitations. As indicated before, intermediate and large systems often have fewer architectural differences. Microprocessors and minicomputers have a less complex and less powerful architecture than larger systems; they feature many characteristics which were typical for larger systems 15 to 25 years ago.

The last few years have seen a very significant proliferation of mini computer systems. This has been partially due to the first time availability of fairly inexpensive hardware (especially LSI). Another reason is the "sponsor problem". It becomes increasingly difficult in large organizations to reach agreement between various departments as to optimum computer operation and utilization. This particular "Management of Complexity" problem often gets resolved by a single department taking the responsibility for buying, installing, operating and maintaining its own system. Additional motivation is given by "organizational reliability". This addresses the fact that a particular department can maintain access to its *own* computer more easily, than to a central computer, when another higher priority area in an organization gets into difficulties.

The "Departmental" approach offers another attractive possibility. A single, well understood application of moderate complexity can be shaped such that it fits on a system not only of moderate size but also of moderate functional capability. The machine gets programmed at what is essentially the microprogramming level of an intermediate system. Programming is more complicated than if done in a Higher Level Language on a full function system. On the other hand, the lower functional capability of the mini system tends to decrease programming complexity.

Assuming the application is small, well understood, and needs to be done only "once and for all", this approach is often very attractive. Large organizations therefore now often maintain multiple small "Departmental Systems" in addition to their large computer center system(s). As it turns out, however, applications of these departmental systems are often not as isolated as originally assumed. We therefore observe a recent trend to interconnect these independent and usually architecturally incompatible machines into loosely or tightly coupled computer networks, possibly with interaction by a large central computer. This is sometimes done to share workload, functional capabilities, or I/O devices , more often, however, to share data. Network structures feature layers of abstractions just like individual computer systems: Link Control, Path Control, Session Control. Especially the last two require additional system architecture and system structure innovations which are still in the process of evolution.

A related trend has to do with the observation that many "once and for all" applications grow in time, both in terms of size and complexity. As a consequence, the departmental system grows: more main store, more disk storage, more complex system software. Very often the increased capabilities can only be obtained in a system with a different architecture, featuring more powerful hardware functions like extended addressing, protect mechanisms, supervisor-state functions, I/O channels, more powerful I/O devices (especially disk storage), and more powerful operating system functions like data management, resource control, overlay supervisor and virtual storage. The switch to a different architecture usually implies an expensive conversion process.

5. CONCLUSION

Management of Complexity is the overriding concern in the development of modern Data Processing Systems. We observe a split into systems which are used in a computing center, and systems which are used by an individual department. The first class of systems is characterized by general purpose attributes of its structure and features for adaption to a wide spectrum of individual users. The second class is characterized by a tayloring of hardware and software features to individual applications.

Decentralized departmental systems can achieve significant efficiency improvements through application tayloring, if and when their applications can be treated as isolated from each other. Where this is not possible we observe a development trend towards computing networks with distributed intelligence, very often with a powerful central host system. In this case, issues of architecture integrity, compatibility, program and data portability are a major concern, and will impact future developments to a significant extent.

LITERATURE

[1] M.V. Wilkes, "The best way to design an automatic calculating machine", presented at the Manchester University Comp. Inaugural Conference, Manchester, England, 1951, p. 16

[2] C. Schuenemann, "Micro- and Picoprogram Stores", Proceedings of the IBM Informatik Symposium on Rechnerstrukturen, R. Oldenbourg, 1974, p. 36-74

[3] E.I. Organick, "The Multics System", MIT Press, 1972

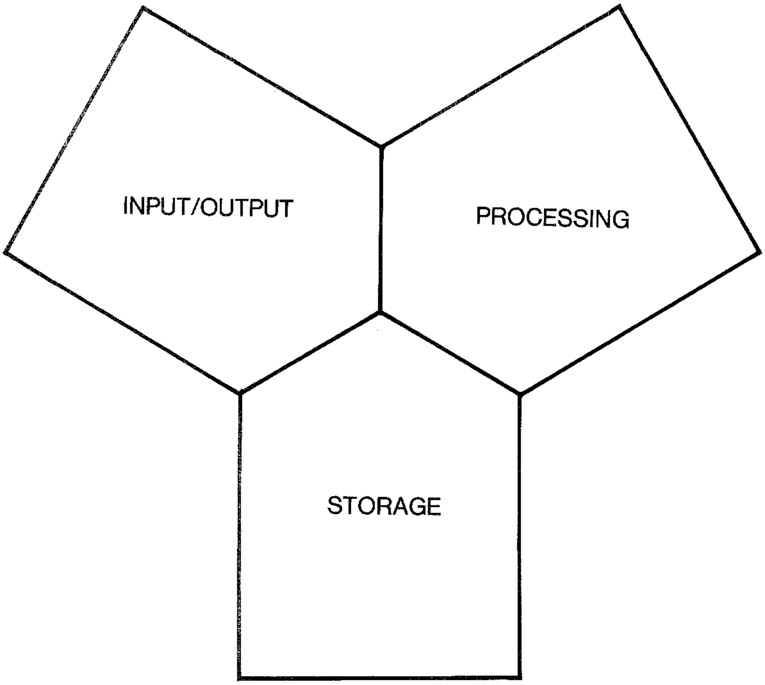


FIG. 1 SYSTEM ARCHITECTURE

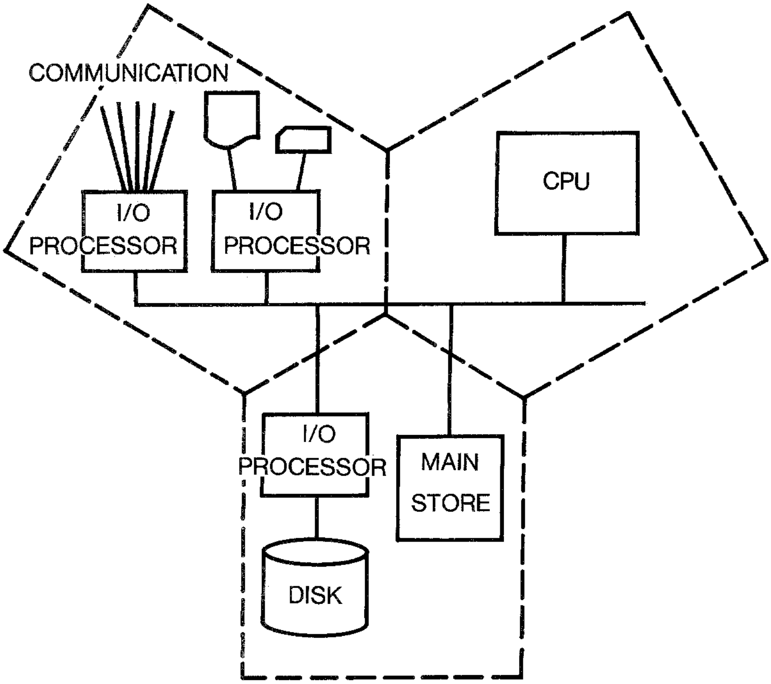


FIG. 2 SYSTEM STRUCTURE

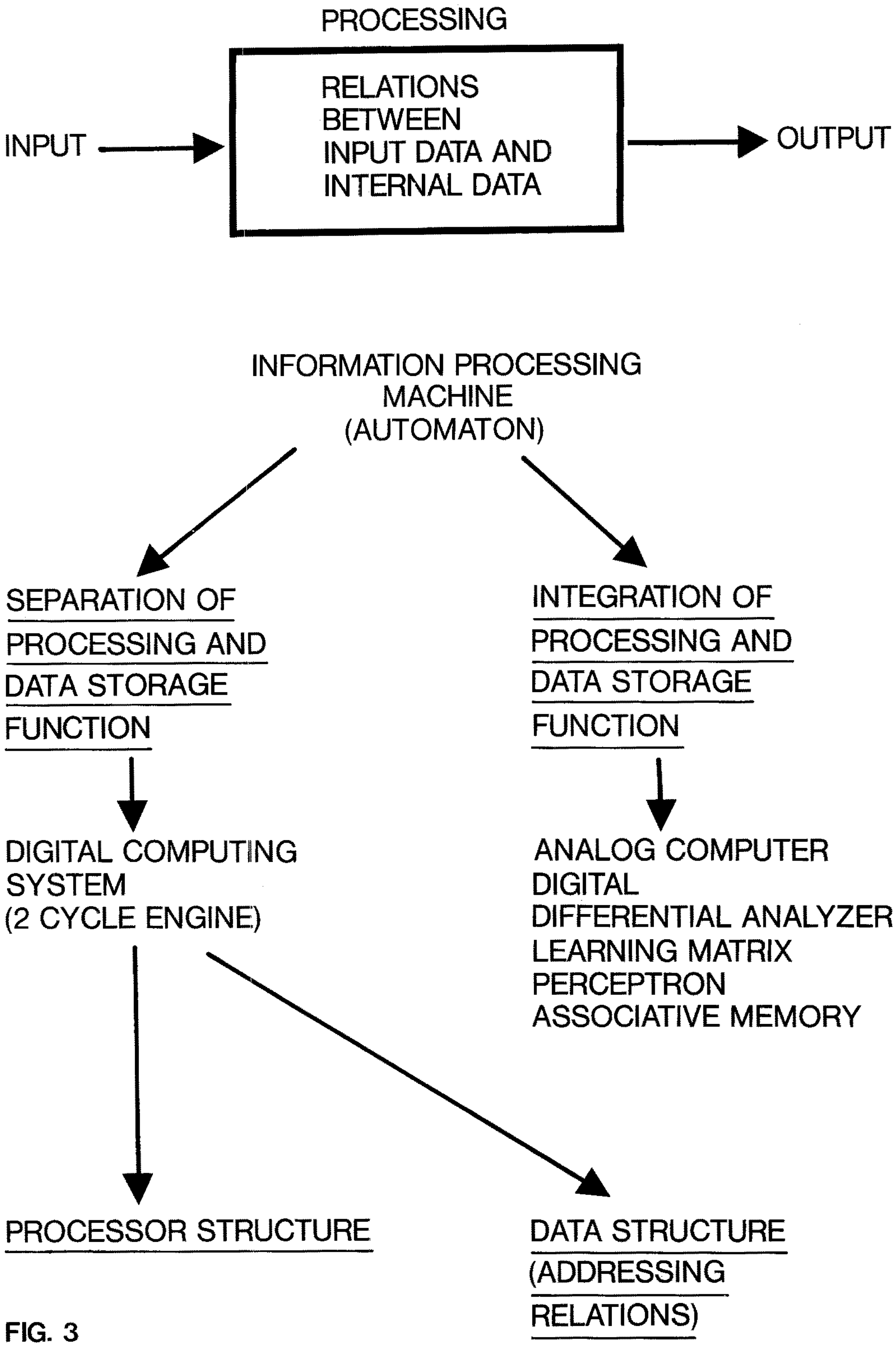

PROCESSING
RELATIONS BETWEEN INPUT DATA AND INTERNAL DATA
INPUT
OUTPUT
INFORMATION PROCESSING MACHINE (AUTOMATON)
SEPARATION OF PROCESSING AND DATA STORAGE FUNCTION
INTEGRATION OF PROCESSING AND DATA STORAGE FUNCTION
DIGITAL COMPUTING SYSTEM (2 CYCLE ENGINE)
ANALOG COMPUTER
DIGITAL DIFFERENTIAL ANALYZER
LEARNING MATRIX
PERCEPTRON
ASSOCIATIVE MEMORY
PROCESSOR STRUCTURE
DATA STRUCTURE (ADDRESSING RELATIONS)

FIG. 3

TECHNOLOGY DEVELOPMENTS

LOGIC	RELAYS → TUBES → TRANSISTORS → → SLT → MSI → LSI
PACKAGING	SOLDER → WRAPPING → CARDS & BOARDS SOCKETS → SLT → C4 → MLC WIRES → FLAT CABLES → LIT
INTERNAL STORAGE	COUNTERS → STORAGE TUBE & DELAY LINES & REVOLVERS → CORES → MONOLITHIC MEMORIES
EXTERNAL STORAGE	PAPER TAPE → CARDS → MAG TAPE DISK → TAPE LIBRARY

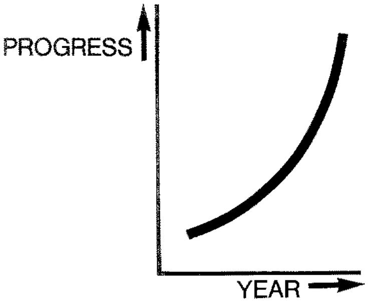


FIG. 4

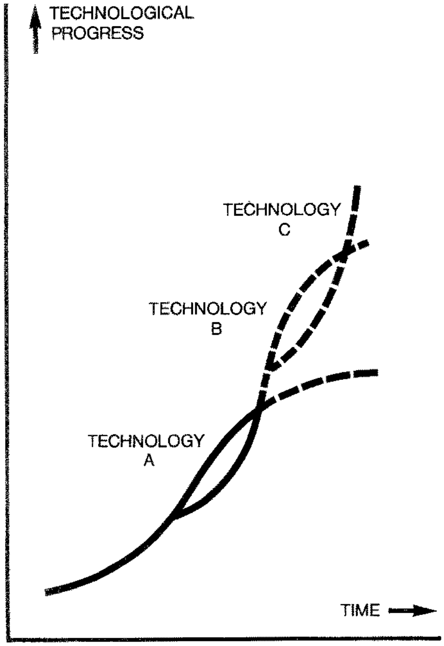
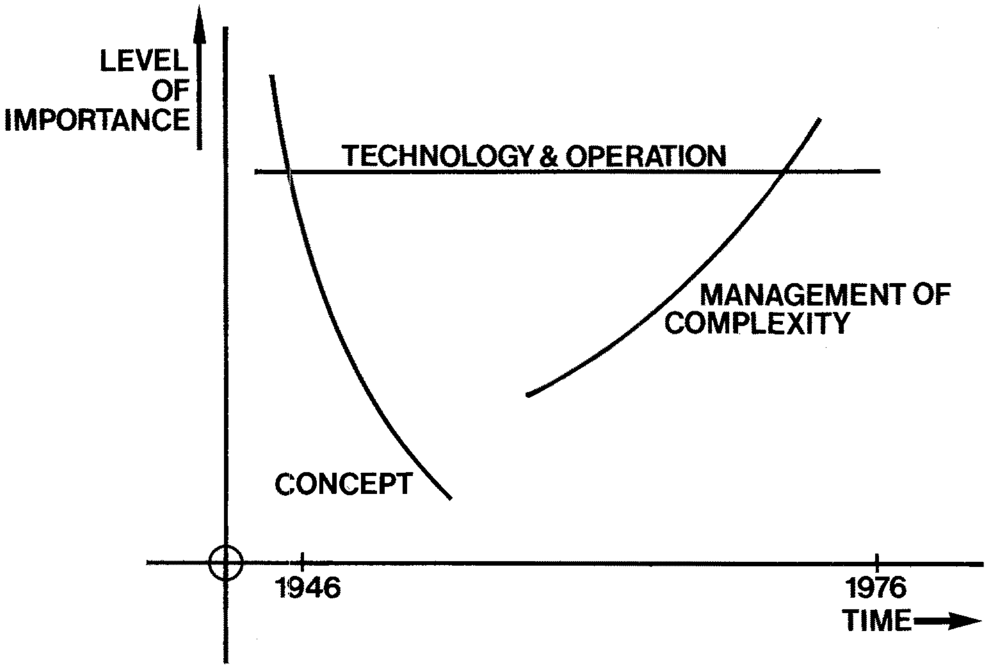
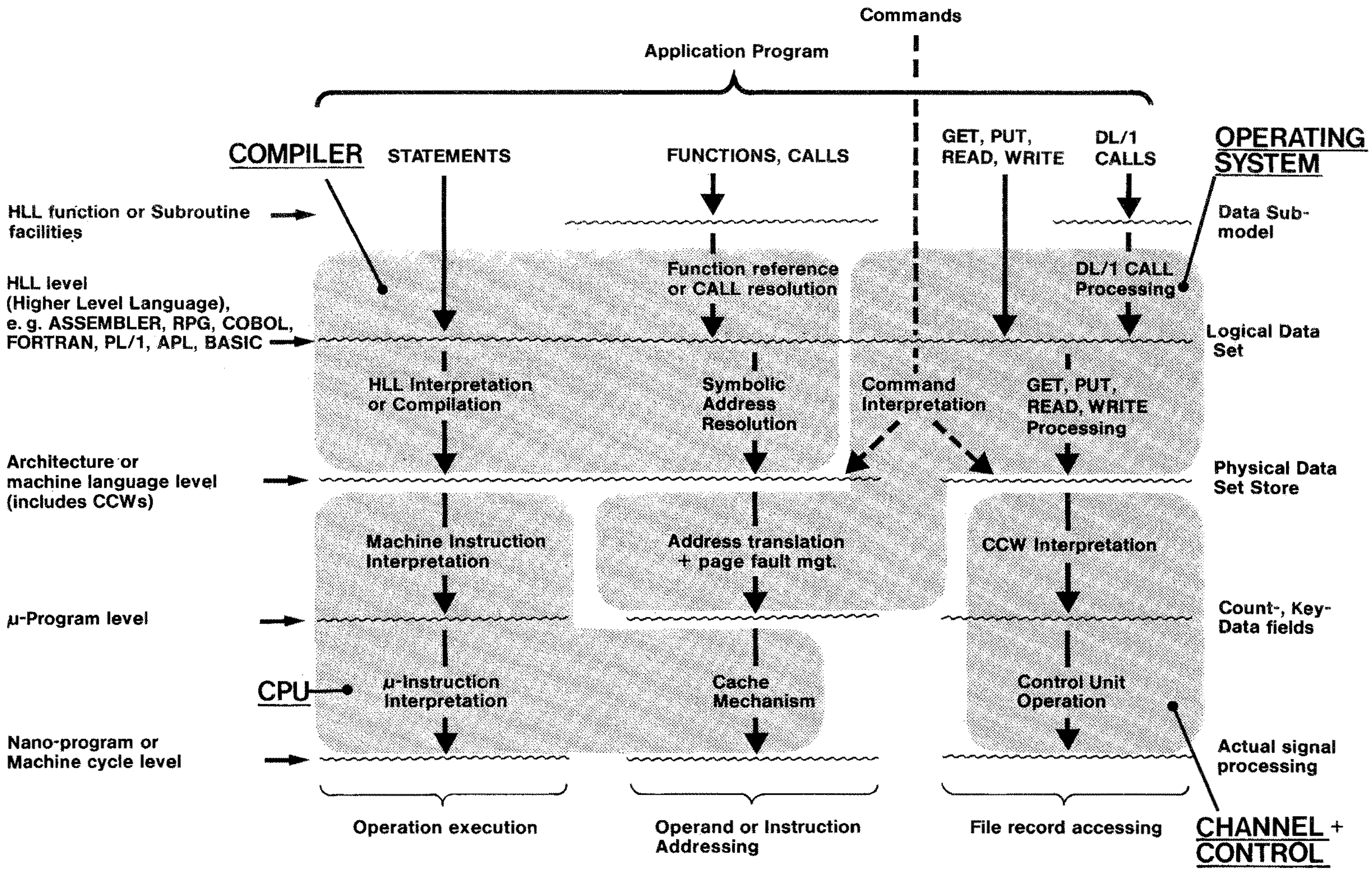


FIG. 5 TECHNOLOGICAL PROGRESS OVER TIME



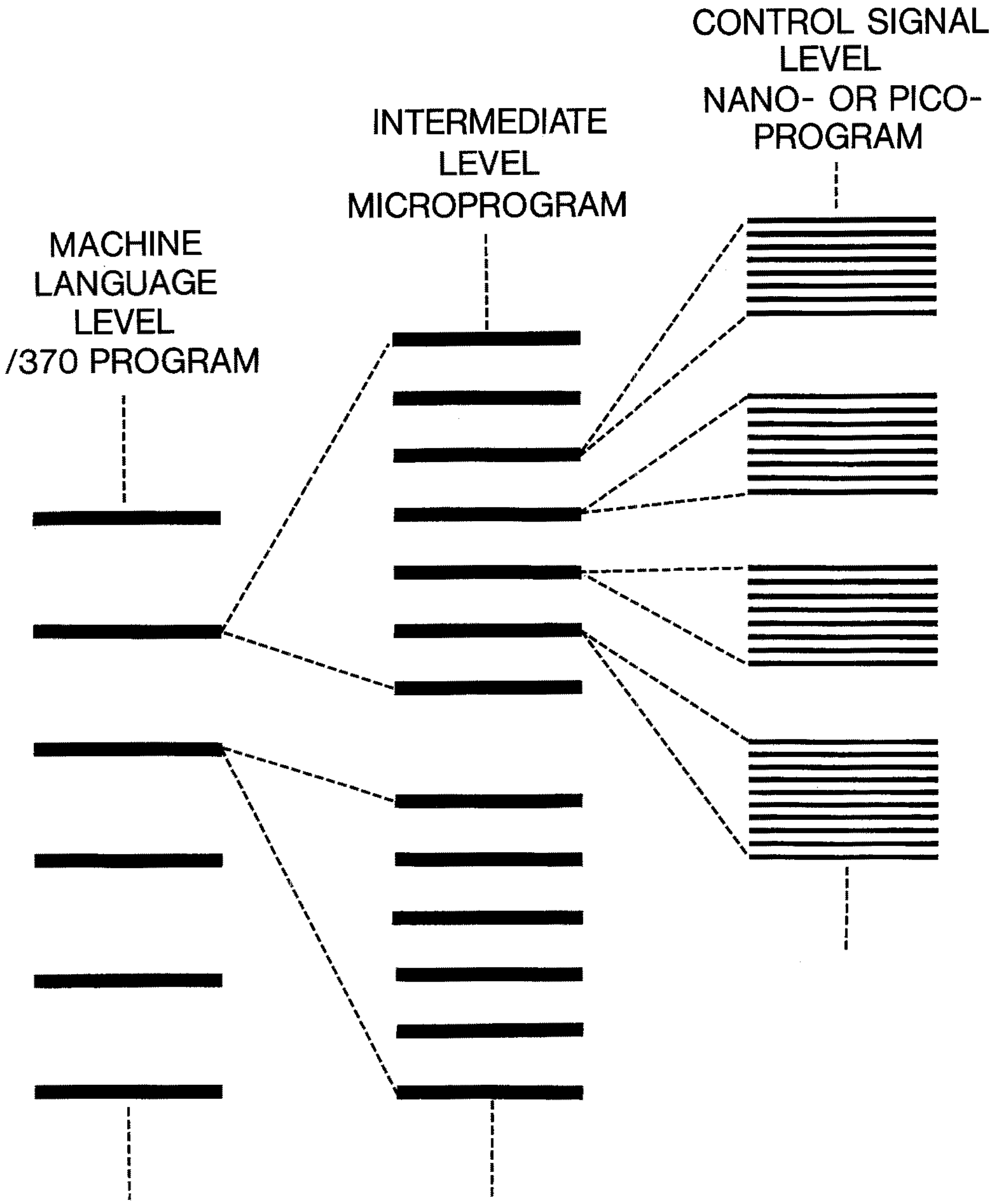
TRENDS IN THE LEVEL OF IMPORTANCE

FIG. 6


Commands
Application Program
COMPILER
STATEMENTS
FUNCTIONS, CALLS
GET, PUT, READ, WRITE
DL/1 CALLS
OPERATING SYSTEM
HLL function or Subroutine facilities
Data Sub-model
HLL level (Higher Level Language), e. g. ASSEMBLER, RPG, COBOL, FORTRAN, PL/1, APL, BASIC
Function reference or CALL resolution
DL/1 CALL Processing
Logical Data Set
HLL Interpretation or Compilation
Symbolic Address Resolution
Command Interpretation
GET, PUT, READ, WRITE Processing
Architecture or machine language level (includes CCWs)
Physical Data Set Store
Machine Instruction Interpretation
Address translation + page fault mgt.
CCW Interpretation
μ-Program level
Count-, Key-Data fields
CPU
μ-Instruction Interpretation
Cache Mechanism
Control Unit Operation
Nano-program or Machine cycle level
Actual signal processing
Operation execution
Operand or Instruction Addressing
File record accessing
CHANNEL + CONTROL UNIT

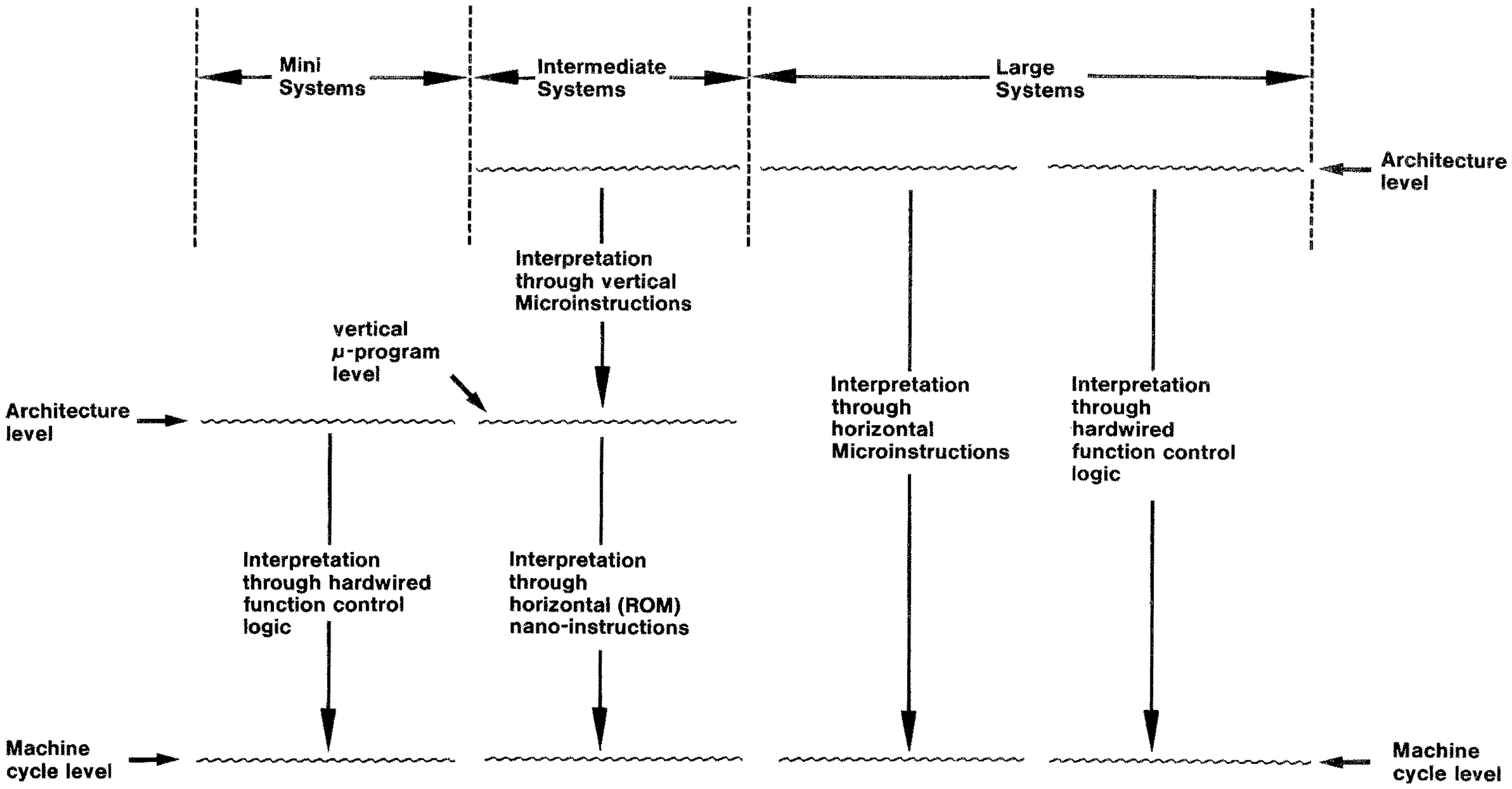
MULTILAYERED MAPPING OF ABSTRACTIONS

FIG. 7



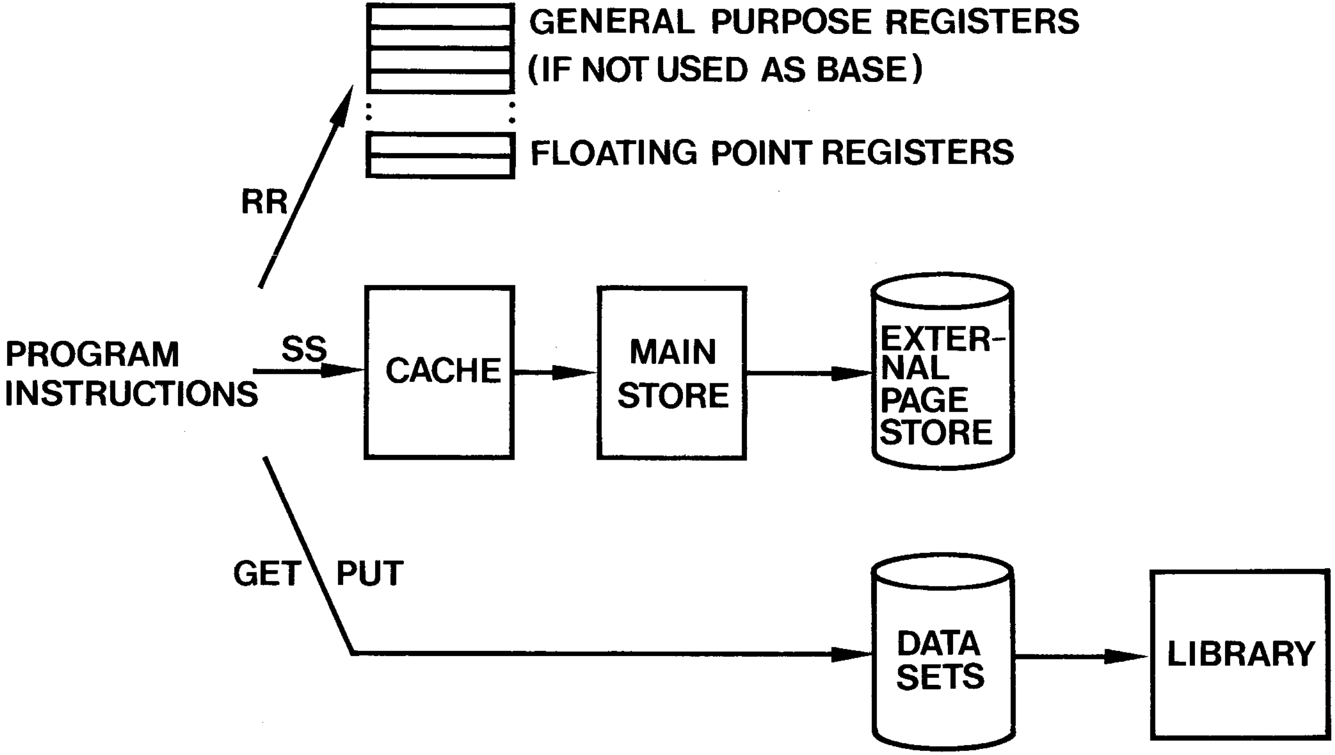
INSTRUCTION INTERPRETATION HIERARCHY

FIG. 8



INSTRUCTION EXECUTION CONTROL FOR VARIOUS MACHINE SIZES

FIG. 9



STORAGE ADDRESSING HIERARCHY (AT ASSEMBLER LEVEL)

FIG.10

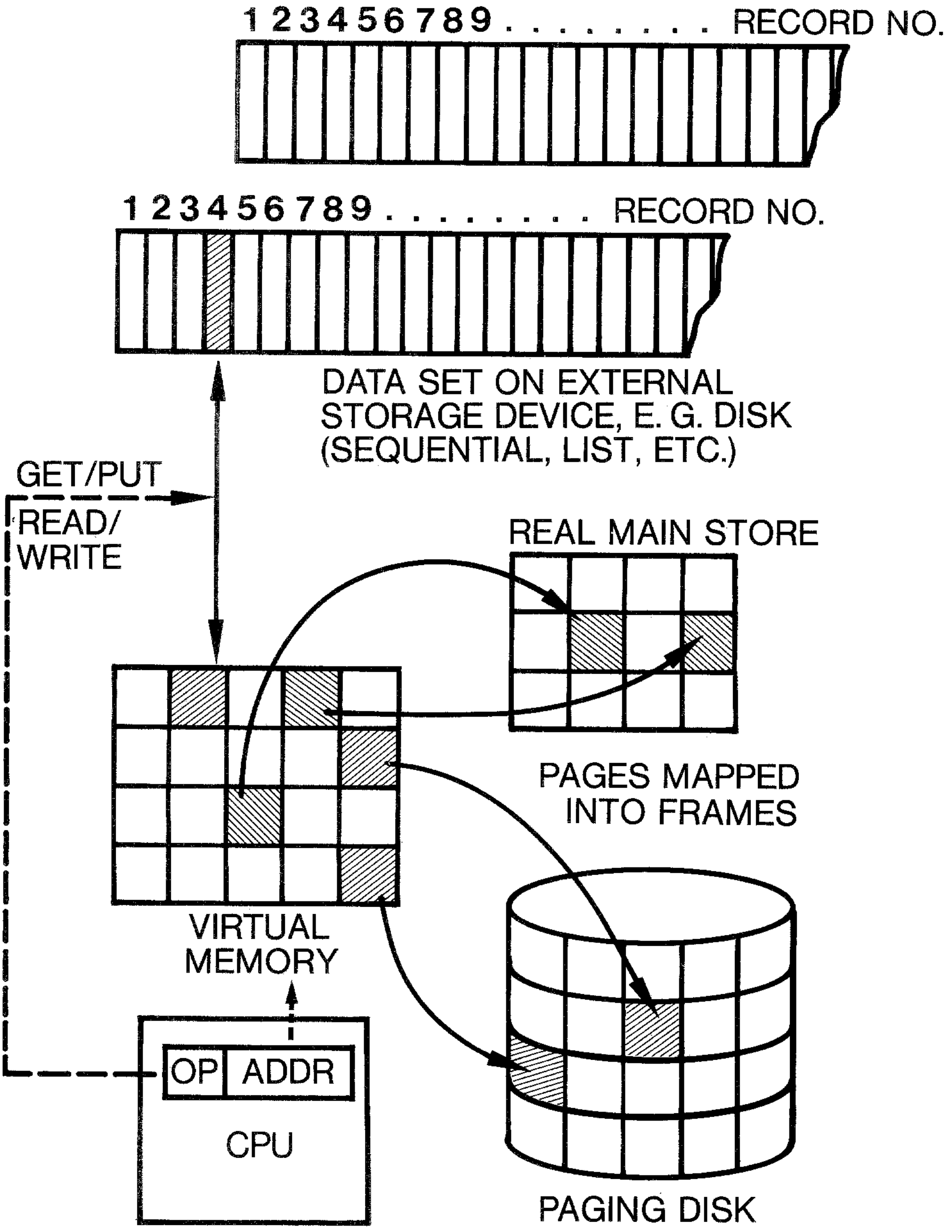
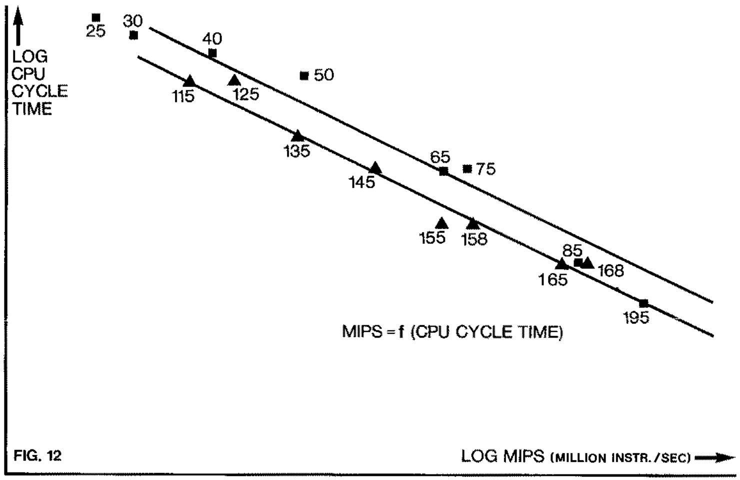
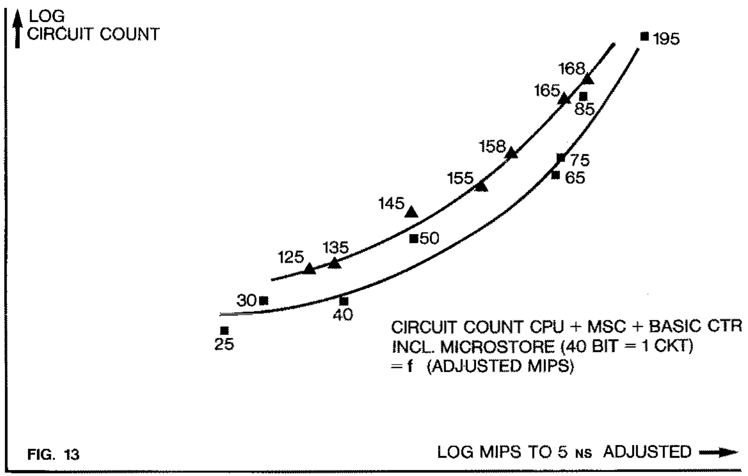

1 2 3 4 5 6 7 8 9 . . . . RECORD NO.
1 2 3 4 5 6 7 8 9 . . . . RECORD NO.
DATA SET ON EXTERNAL
STORAGE DEVICE, E. G. DISK
(SEQUENTIAL, LIST, ETC.)
GET/PUT
READ/
WRITE
REAL MAIN STORE
PAGES MAPPED
INTO FRAMES
VIRTUAL
MEMORY
OP
ADDR
CPU
PAGING DISK

FIG. 11



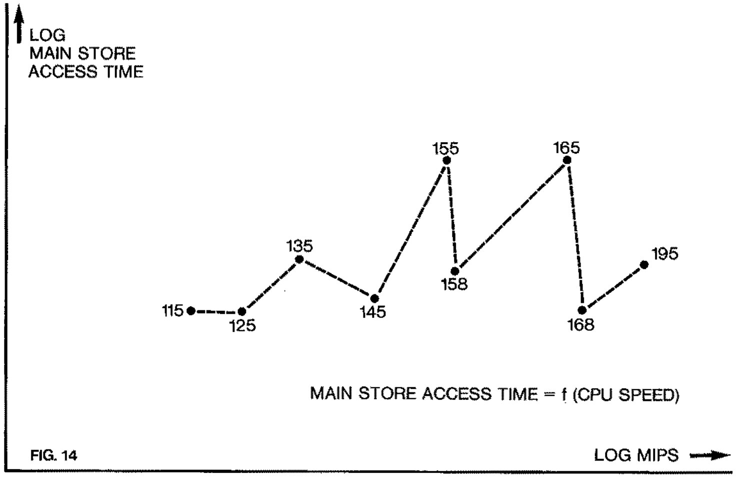
MIPS = f (CPU CYCLE TIME)

FIG. 12



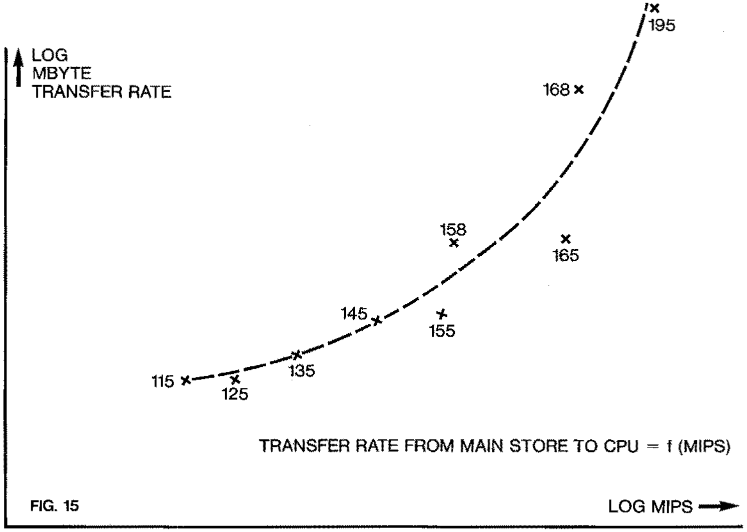
CIRCUIT COUNT CPU + MSC + BASIC CTR
INCL. MICROSTORE (40 BIT = 1 CKT)
= f (ADJUSTED MIPS)

FIG. 13



MAIN STORE ACCESS TIME = f (CPU SPEED)

FIG. 14



TRANSFER RATE FROM MAIN STORE TO CPU = f (MIPS)

FIG. 15

	Main Store Addressing Architecture	I/O Channel Function	Operating System	Main Store Size K Byte	Disk Store Size	
					Spindles	M Bytes
Micro Computer	16 Bit	programmed I/O	macro Assembler	.256-1-4	–	–
Mini System		Interrupt I/O CPU Control	single back ground partitions Limited foreground facilities	4-16-64	1	10
Intermediate System	Register or equival.	Single independent channel	Single Virtual Memory	64-256-1024	4	500
Large System		multiple independent channels	Multiple virtual memory	1-4-16 MB	16	10000

SYSTEM SIZE CLASSIFICATION

FIG. 16