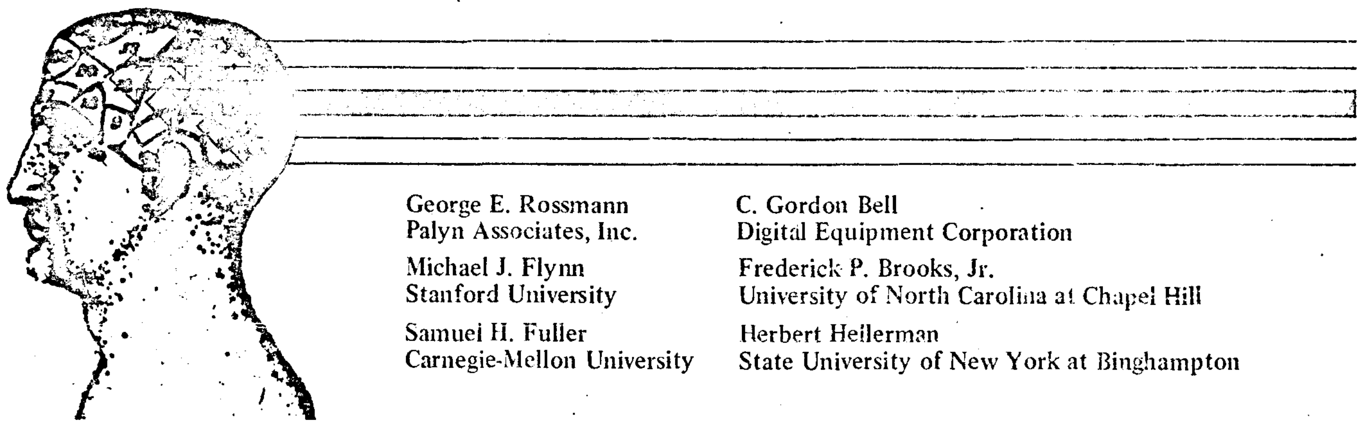
(34)

A Course of Study in Computer Hardware Architecture

Report of the IEEE Computer Society Task Force On Computer Architecture



George E. Rossmann
Palyn Associates, Inc.

Michael J. Flynn
Stanford University

Samuel H. Fuller
Carnegie-Mellon University

C. Gordon Bell
Digital Equipment Corporation

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

Herbert Hellerman
State University of New York at Binghampton

Module 0 — Preface

The subject of computer architecture as currently taught in most computer engineering and computer science programs is a mixture of architectural principles, organizational strategies, and implementation techniques. This blurring of the hierarchy of system levels that characterize the structure of a computer has made it very difficult for students (and often instructors as well) to determine what were the forces that led to the design decisions they have seen reflected in machines.

Guidance for the development of courses in computer architecture has come from the ACM Curriculum Committee on Computer Science [ACM68] and the COSINE Committee of the Commission on Engineering Education [COS68]. In 1968 these committees both proposed syllabuses for a course in computer organization. The ACM committee also proposed the content for an advanced course in computer organization. The material in these courses is characterized by the confusing mixture of topics we have already mentioned. The courses also suffer other shortcomings. They contain sketchy topic outlines rather than detailed specifications. They pay insufficient attention to the fact that to a user the essential part of any computer system is its visible facilities: language processors, operating system, and other software. Therefore, they do not support the integration of hardware and software design that is required to create computer systems which satisfy the user. Finally, computing environments and basic technologies have changed significantly since the ACM and COSINE committees made their recommendations, and these changes need to be reflected in a computer architecture course description.

In view of these circumstances, a task force was established by the IEEE Computer Society to prepare a detailed specification for a course of study in computer architecture for students whose major interest is in computer engineering or computer science. The members of the task force were: George E. Rossmann, chairman, Palyn Associates, Inc.; C. Gordon Bell, Digital Equipment Corporation; Frederick P. Brooks, Jr., University of North Carolina, Chapel Hill; Michael J. Flynn, Stanford University; Samuel H. Fuller, Carnegie-Mellon University; Herbert Hellerman, State University of New York at Binghampton.

Plan of the Report

The structure of this report has been motivated by three considerations. First, the report is concerned primarily with computer hardware architecture. Second, the course of study is intended for students for whom computer architecture supports their interest in computer engineering or computer science. These people need to be familiar with all of the topics discussed here. This course of study is not sufficient, however, for the training of the professional computer architect. The introduction to Module 1 explains why. Third, the universities using this report will vary considerably with respect to plans for packaging this material into academic courses, instructor capability, student ability, and student preparation.

The first and second considerations determined what material should be included, and the third consideration determined how it should be organized. The material presented has been restricted to those topics which we feel every computer engineer and computer scientist ought to know and to those computer systems which have been in the mainstream of commercial equipment. As a result, topics such as the early history of computers; parallel, pipeline, associative, and array processing; the effectiveness of various machine structures in performing various computations; implementation details including busing structures and interfacing conventions; and switching structures have been conciously excluded from this course of study. We have organized the material presented into 11 modules, each dealing with a fundamental aspect of computer hardware architecture. The modules are:

Module No.	*Title*
1	Introduction and Meta Representation
2	Data Representation
3	Instructions and Addressing
4	Interpretation and Control
5	Memory Hierarchies
6	Protection Mechanisms and Hardware Aids to Supervision
7	Specialized Processors
8	Multiple Computers
9	Performance Evaluation
10	Reliability
11	System Design Evaluation

The packaging of these modules into academic courses is up to individual instructors. However, they must generally be taught in the order indicated. The material in modules 1 and 11 may be interspersed throughout the course of study rather than being treated as separate items. We have tried to help by suggesting the amount of time which should be devoted to each module and its sub-topics, and by precise specification of the references which support each topic. One grouping of the modules we have found effective is:

Module 1: Meta Representation
Modules 2-5: Classical Processor/Memory/Switch Aspects of Computer Architecture
Modules 6-8: Aspects of Systems Architecture
Modules 9-11: Evaluation Methods and Analysis.

In addition, depending upon the composition of the class, whether computer engineers or computer scientists, the instructor may choose to emphasize, respectively, the hardware implementation or software engineering consequences of various architectural decisions.

We believe that it is possible within the scope of this course of study to undertake term projects that involve design. However, it was unfeasible for us to offer guidance in this area. Instructors will have to define projects based on their resources and the abilities of their students. There is, of course, a temptation to do a machine sketch, but the committee does not encourage such a project because we feel that the essence of the design process lies in probing deeper than the sketch level to understand the interrelations between various design decisions.

The only textbooks currently available that cover a reasonable amount of the material proposed in this course of study are: *Computer Structures: Readings and Examples,* C. G. Bell and A. Newell [BEL71]; *Digital Computer System Principles,* H. Hellerman [HEL73]; and *Introduction to Computer Architecture,* H. Stone (ed.), [STO75].

Background

To be prepared for this course of study in computer architecture the student should have a good understanding of (1) programming in both assembly language and higher-level languages, and (2) how at least one simple computer works.

The necessary experience in programming can be obtained from any course that introduces the notion of computing through the use of an algorithmic language such as FORTRAN, ALGOL, or PL1. Familiarity with assembly language programming and the organization of a simple machine can be obtained from a course based on a text such as [STO72] or [GEA74].

It is desirable for the student either to have had a course in logic design or to have been exposed to logic design in either of the prerequisite courses just described. However, since many students interested in computer architecture may be engaged in software engineering programs, we have arranged this course so that logic design is not an essential prerequisite.

Some of the modules require additional prerequisites such as elementary statistics and probability, and depend on an understanding of preceding modules. Such dependencies are indicated in the introductions to the individual modules.

Module 1 — Introduction and Meta Representation (2 Hours)

Introduction (1 Hour)

What is computer architecture? A number of definitions have been advanced. Architecture describes the attributes of a system as seen by a programmer—i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation. Architecture is the study of those aspects in the analysis and design of computers which specifically relate their structure and their function. Computer architecture is the discipline devoted to the design of highly specific and individual computers from a collection of common building blocks.

We decided not to adopt any of these definitions. Instead we defined computer architecture by deciding what professional architects are supposed to do. The committee determined that the computer architect's task is to define computer systems that use hardware and software technologies so as to best satisfy all the users' needs, including function, economy, reliability, simplicity, and performance. In carrying out this task, the architect must develop an understanding of the potential applications of each system and then bring to bear extensive knowledge of the material in this course of study, operating systems principles [COS71], implementation details, component technologies, and many other things to accomplish its design.

Computer hardware systems are complex. There are at least five levels in any implementation of a computer system:

a. processor/memory/switch (PMS) level
b. programming level which includes operating systems
c. register transfer level
d. switching circuits level
e. circuit or realization level

Each level arises from abstractions of the levels below it and is characterized by a distinct language for representing its components, its modes of combination, and its laws of behavior.

The first three levels of this hierarchy are the proper study of the computer hardware architect, because the architect must decide how to distribute the complexity inherent in a computer system among them. Not much will be said about software and operating systems directly, but their influence on hardware architecture will permeate all the modules. Moreover, we have tried to show how the boundaries between these levels keep changing and how this impacts design.

Meta Representation (1-4 Hours)

The general problem addressed in this submodule should be discussed briefly to introduce the student to the need for formal description. Specific notations can then be introduced naturally when the need for them arises. Therefore, the time allotted to this submodule should be distributed throughout the course of study.

In order to communicate about machines, one must have suitable languages. Communication dialogues include user-vendor, vendor-system designer, system designer-programmer, student-teacher, etc. The earliest communications concerning machines took the form of standard prose and conventional mathematics (including standard and Boolean algebra) and used ad hoc diagrams and flowcharts to describe the structure and operation of the object being built.

As machines evolved into complex structures, however, it became important to use more precise languages in order to communicate the designs exactly but simply. The formality of contemporary descriptions also implies that they can be understood by machines. It is useful to have machine-readable descriptions of machines so that they may be simulated and hence verified prior to their actual construction. For example, in the case of microprograms, programs translate the descriptions from the syntax of an assembler or compiler language into a form for construction, simulation, and verification. There has also been considerable interest in translating the formal description of each system level in a computer structure into a realization of that level automatically.

For the high-level block diagrams that represent the components of computer systems (e.g., central processors, primary memories, and secondary memories), there is a need for precise descriptions simply to permit novices to understand them readily. Such notations also permit comparisons. Descriptions that show the connectivity of a structure lay the foundation for the automatic generation of models for performance and reliability calculations. A formal representation permits automatic verification and construction of configurations. That is, a specific computer system can become the basis for a variety of configurations each of which depends on the number and type of components that are used to form it. It is generally quite difficult to verify that each such configuration will be viable when realization considerations such as power, floor space, interconnection cable lengths, input-output bandwidths, reliability, etc., are accounted for.

Some aspects and uses of notations include description of:

(a) The functional specifications of a system with conventional prose. At the topmost level of system description, users must express their needs and work with computer architects to refine these expressions into meaningful specifications through the medium of conventional prose.

(b) The physical structure of a particular computer system including its various processors, memories, peripherals, and their interconnection.

(c) The instruction set with its interpretation and the definition of data-types and operations on these data-types. Such a description may be used in lieu of standard programming reference manuals. Currently, most machines are described in conventional prose, but also contain an informal register transfer notation description of the behavior of each instruction. Nearly all machines have a simulator written in a high-level language that runs on another machine.

(d) The physical organization of a particular processor, including its registers, data operators, data flow, and control. Usually this is ad hoc on a functional diagram basis. The behavior of such a system is specified using register transfer languages and conventional programming languages. The outcome is a description suitable for simulation. For microprogrammed machines the behavior is expressed as an assembly language listing and/or flowchart.

(e) The syntax and semantics of data structures. In currently evolving hardware structures, especially communication links, more complex data structures are being directly interpreted. The interconnection of computer components (computer-computer, computer-terminal, etc.) depends on complex communications protocols. These message protocols are expressed in Backus-normal form, state diagrams, flowcharts, and timing diagrams.

(f) Combinational and sequential logic with logic diagrams and Boolean algebra. Such descriptions enable gate and circuit level simulation of the objects.

Module 1 – Topic Outline

Prose and Informal Diagrams

Initial functional specifications of a system.
Computer reference manuals [e.g., CDC66; DEC10; IBM70].

Block Diagrams (Two-Dimensional) to Represent System Structure

Computer components. Processors, memories, controllers, transducers (I/O equipment), links, switches (buses). [BEL71, Ch. 2 (PMS); system block diagrams from any computer reference manual]
Register transfer level block diagrams.
Combinational and sequential logic.

Block Diagrams (Two-Dimensional) to Represent System Behavior

Flowcharts and/or state diagrams for sequential circuits, microprograms, and communication link protocols.

Languages (One-Dimensional, Formal, Machine Interpretable)

Instruction set definition. [FAL64; BEL71, Ch. 2 (ISP); CHU72; HIL73]
Number representation: mathematical notation and conventions.
Register transfer and logic behavior including microprograms.
APL and APL-derivatives. [HEL73; HIL73]
ALGOL-like. [BEL71, Ch. 2 (ISP)]
FORTRAN-like. [CHU72]
Logical design: Boolean algebra for combinational logic and time difference equations for sequential circuits.
Communication-message formats (syntax and semantics) for communication links. [Backus-normal form]

In summary, to understand, analyze, synthesize, and generally communicate with one another and with machines about machines requires various informal and formal representations. As one considers each aspect of computer architecture, representation is the basis for understanding.

Module 2 – Data Representation (5 Hours)

Computers exist to manipulate information. Therefore, designing the ways in which the information is to be represented is a first and central step in computer architecture. The set of operations to be performed follows from the set of data types to be manipulated. Since one wants the operations for manipulating program information to be as much as possible like other data operations, the selection of data representations creates strong biases as to the formats and representations of instructions.

A good way to teach this module is to lead students through a decision tree representing the data representation design decisions for a general-purpose or special-purpose computer.

The objectives of the module are:

(a) To teach a rather general point of view about representation, since creativity with respect to new representations and a sharp awareness of existing ones lead to innovative program and machine design.

(b) To show the numerous and sometimes subtle considerations affecting each representation decision.

(c) To show how representation decisions interact, thereby introducing the student to the notion of system design, which requires optimizing a set of tightly coupled decisions.

General references: [STO75, Ch. 2; BEL71, Ch. 3; BRO69, Sec. 6.2; KNU68, Ch. 2].

The Notion of Representation (½ Hour)

Representation of concepts by language elements—numbers, letters.
Representation of language elements by bits.
Allocation of a set of bits.
Encoding of values by a bit configuration.
Centrality of representation to architectural process.

Structure of the Representation Space (½ Hour)

Resolution

What will be the smallest named and manipulable element? For example:
1 bit – IBM Stretch, Burroughs B1700, CDC Star
6 bits – Univac 1108, Honeywell 6000
8 bit byte – IBM S/360, DEC PDP-11
18 bit halfword – DEC PDP-10
36 bit word – IBM 7094.

Data Element Sizes

Systems require consistency among the several sizes. Consider the representation families of some systems:
IBM S/360: 4, 8, 16, 32, 64 bits. [AMD64]
Univac 1108: 6, 9, 12, 18, 36, 72 bits.
Burroughs B6700: 4, 6, 8, 48, 96 bits.
CDC 6600: 60, 120 bits. [THO70, Ch. 5]

Data Formats (4 Hours)

Data types are derived from architectural and applications requirements.

Fixed Point Numbers

Choice of base: binary, decimal (IBM 650, IBM 1401).
Sign representation: IBM 7094, CDC 6600, IBM S/360.

Mixed Fixed Numbers

Scaling. [MCR57, Ch. 4; STE72, Ch. 7]

Floating Point Numbers

General reference: [BUC62, Ch. 8; STR74].
Location of radix point. CDC 6600, IBM S/360, Burroughs B5500.
Mantissa base and sign representation. [BRW69; COD73]

Exponent base and sign representation.
Precision, significance, significance alarms and measures. [BRE73; KUK73].
Normalization, guard digits, rounding. [KAN73]

Character Strings

Encoding of characters: ASCII, EBCDIC, BCD. [BUC62, Ch. 6]
Distinctions between internal and external representation of characters.
Inherent length variability arises from applications. Various methods exist for accommodating variable length fields and for specifying lengths. IBM 1401, IBM S/360, Burroughs B5500. [BUC62, Ch. 4]

Bit Strings

If bits are independent and re-orderable by the designer, they really form a set represented as a vector. Special operations for bit vectors [IVE62, Ch. 1]; for bit strings [BUC62, Ch. 17]. Burroughs B1700, IBM Stretch, CDC Star.

Higher Structures

Complex numbers.
Vectors. CDC Star.
Chained representations of vectors, FIFO, LIFO (stacks). Burroughs B5500. [BAR61]
Matrices, trees. Both of these map neatly onto vectors, hence processors with vectorial memories. ILLIAC IV, STARAN. [IVE62, Sec. 1.23; BER71]
COBOL, PL/I structures: trees of inhomogeneous elements.
Lattices, graphs, and nonplanar nets: representation by chained structures. [MCA60; ROS61]
Data flags: embedding control information in data. Burroughs B5500. [ILI68, Ch. 2-3; BUC62, Sec. 7.9; FEU72; FEU73]

Module 3 — Instructions and Addressing (5 Hours)

In all electronic computer technologies up to now, memory speeds have been at least as slow as, and usually much slower than, operation speeds. Therefore, the performance of a computer can be first-order approximated by the *memory bandwidth*, the number of bits per second delivered (or accepted) by the memory, running continuously. For most applications, regardless of the machine, about half of the bandwidth is used for data, about half for instructions.

The architect, therefore, tries to improve the cost-performance ratio of his design by utilizing the memory bandwidth efficiently, in Shannon's sense. The design of data representation has been treated in Module 2. This module treats the design of instruction representations. Whereas many decisions about data representation are almost dictated by the characteristics of the data from intended applications, instruction representation gives wide scope for design differences. Indeed, the most striking differences in computer architectures have historically been concentrated here.

An instruction consists of the specification of (usually) a single operation and the operands for it. The design decisions, therefore, are:

- What set of operations shall be allowed?
- How shall a particular operation be specified?
- How shall the operands be specified (*addressed*)?
- How shall all the specifications be fitted together into an instruction format?

The objectives of the module are:
(a) To teach the student to think of the bit as his raw material and its efficiency of use as a first-order estimator of goodness of design.
(b) To survey the wide variety of techniques that have been used in instruction specification, particularly the efficient specification of addresses.
(c) To show the tight interaction among representation decisions enforced by the necessity to make instruction formats commensurate with data formats, which enforces a bit-budget.
General references: [BEL71, Ch. 3; GEA74, Ch. 2; HEL73, Ch. 8, 9].

Operation Sets (2 Hours)

What to include?
Frequency of use the chief test. [FOS71; CON70; WIN73]
Fewness of operands an important test.
(WIR68 and ANA73 discuss operation-set shortcomings of the IBM S/360).
Operations determined by data types selected. [BEL71, Ch. 3]
Housekeeping. [FLY74]
Processor state changes: Loads and stores.
Operations on addressing and indexing mechanisms.
Movement of data, partitioning, shifting.
Arithmetic.
+, −, ×, ÷, shifting. [CHU62; STE72; STO75, Ch. 1; HEL73, Ch. 7]. In either this section or in **Data Formats** (pp. 47), algorithms which implement the common arithmetic operations +, −, ×, ÷, and shifting should be discussed. These algorithms, of course, depend on the method of representing data. The advantages and disadvantages of various forms of data representation should be discussed.
Square root, radix conversion and other options. [CHU62]
Floating point. [SWE65; STE72; STR74]
Comparisons and tests.
Sequencing.
Branching.
Subroutine linkage.
Operations on sequencing mechanisms such as enabling/disabling interrupts.
Synchronization of concurrent activities.
Input-output operations.

Addressing (2 Hours)

Name-Space Structure (Main and Secondary)

Numeric names. (IBM 604 Electronic Calculating Punch, circa 1948, had arbitrary names.)

Associative (stored name) – MU5 vs. geometric (built-in name).
Vectorial (most machines) vs. matrix (STARAN, ILLIAC IV) vs. tree ([MCA 60]–LISP) vs. network (Burroughs B6500–segmentation) name structure.
Number of independent name spaces. [DEN65; ARD66; DEG70; RAN69]
Overlapping name spaces. General purpose registers are addressable memory locations. DEC PDP-10, Univac 1108

Operand Specification Within the Name Space

Full address in instruction. DEC PDP-10, IBM 7094
Abbreviated addresses.
  Necessity due to large memories and program locality (nonuniform-access distributions). [AMD64]
  Bank Registers–The contents of a bank register are concatenated to the left end of the displacement field to enlarge the effective address. DEC PDP-8, HP 2100
  Base Registers–The contents of a base register can be added to a displacement field (IBM S/360) or used in relative addressing (Univac 1108).
  Descriptors or control words, referenced by short address. Burroughs B5500 and B6500, MU5. [ILI68; KIL68]
Effective address calculation. [BLA59]
  Index registers. IBM S/360, CDC 6600, DEC PDP-11, DEC PDP-10, Univac 1108
  Indirect addressing.
    Single level. IBM 7094, DEC PDP-11
    Multilevel. HP2100, DEC PDP-10
    Programmed–Indirection achieved through Load Address instruction. IBM S/360
  Desciiptors–A Special form of indirect address. Burroughs B5500
  Immediate Address–Direct implementation of operand in instruction. IBM Stretch

Formats (1 Hour) [LAW68; KIL68]

Importance of Frequency Arguments in Format Design

CDC 6600 NOP's waste bandwidth.

Format Components

Format Specifier.
Operation Code: fixed vs. variable length vs. extensions.
Addresses (including abbreviated ones).
Address Modification Specifications.
Sequencing.

Format Sizes

Size must be commensurate with data representation and subfields must be commensurate with readily-accessible data units.

16, 32, 48 bits: IBM S/360
15, 30 bits: CDC 6600
8, 16, . . ., 96 bits: Burroughs B6500
36, 72, 108, 144 bits: Honeywell 6000

Operation Code Specification

Fixed-size. IBM S/360, CDC 6600
Variable-size. DEC PDP-11
Huffmann-coded. IBM Stretch, Burroughs B1700
Direct bit-significance. DEC PDP-8

Number of Addresses

Examples:

0	1	1 + 1
Burroughs B5500	IBM 7094	IBM 650
2	3	3 + 1
IBM S/360	CDC 6600	EDVAC

Stacks as a mechanism for eliminating addresses. [BAR61; BRO63; BAR69; AMD64; STO75, Ch. 7]
Accumulator, multiplier registers as implicit second, third addresses. IBM 7094, IBM S/360, Univac 1108
Multiple accumulators for abbreviated second, third addresses. IBM S/360, CDC 6600, DEC PDP-11. IBM S/360 fixed and floating point registers distinguished by operation code.
Next instruction address. IBM 650, EDVAC. Explicit sequence specification is rare because redundancy is so high.

Module 4 – Interpretation and Control (6 Hours)

The writing of interpretive routines is usually taught independently and separately from the notions of instruction execution and interrupt handling. This module attempts to stress the unity of these ideas.

Interpretation is the assignment of meaning to an expression applied to specific data, whether the expression is in a high-level, assembly, or low-level (so-called microprogram) language. It is invoked by a control mechanism which selects a series of operations to be performed at the (virtual) machine level that implements the interpretation (execution). The control mechanism then makes the next expression ready for interpretation (sequencing). The emphasis of this module will be on control within an interpretive routine and control between interpretive routines.

For example, if the expressions are assembly language instructions, the control mechanism selects a sequence of register transfer level operations to decode and execute the operation specified in an instruction. The control mechanism then either fetches the next instruction for interpretation or, if an external event (interrupt) has been signaled, initiates an interpretive routine to handle the interrupt.

One way to introduce this module would be to discuss the interpretation of assembly language macroinstructions. Have the students program a small number of short interpretive routines in assembly language and then show them how to link and sequence through these routines. The macro sequence should include setting and testing conditions, branching, and executing the interpretive routines. One interpretive routine should be devoted to the handling of interrupts. Some of the concepts of Module 3 could also be brought out during these exercises: passing of parameters, generation of addresses, representation of an instruction, operation sets, etc.

After this introduction, a detailed treatment of assembly language instruction interpretation should follow. Sequencing between instructions should be discussed. The definition of the conditional branching mechanism should be especially stressed because it plays such an important role in hardware/software interaction. The original "IF" and "DO" specifications of FORTRAN represent the effects of attempting to build a language in which programs would be as easy as possible to translate into IBM 704 machine language. They were direct descendants of the IBM 704 compare and TIX instructions. There are numerous converse examples of machine designers mapping higher level language artifacts into hardware. The treatment of conditional operations should also include a discussion of the modifications that can be made once a test is complete. That is, once one control path is selected, exactly how is a new instruction address determined? The role of meta-instructions such as REPEAT and EXECUTE, which are themselves interpretive routines, should be discussed.

Sequencing within instruction interpretive routines is a direct extension of the foregoing discussion on sequencing between them. The additional material to be discussed is related to physical timing of a system. It is important to introduce the student to the notion of a register state transition (internal cycle) as being a fundamental quantum of timing in the system. Concepts such as I and E cycles, the general decoding process, control points and data paths, as well as simple and iterated execution cycles (multiply, translate, etc.), should be discussed. Hardware decoders involving rings, counters, and logic blocks should be introduced and a discussion of microprogramming should follow.

General references: [BRO69, Ch. 8; KNU68, Sec. 1.4.3; CHU72; STO75, Ch. 10].

Introduction to Interpretation (½ Hour) [KNU68, Sec. 1.4.3; BEJ73]

Sequencing and execution of interpretive routines.

Sequencing Between Instructions (2 Hours)

Sequence Determined by Instruction Action

In-line.
Explicit control. IBM 650

Sequence Determined by Instruction and Result Data

Conditions and testing. IBM S/360, DEC PDP-11, IBM 7094
Branching action.
Hardware-software implementations of control structures: IBM 7094–FORTRAN, Burroughs B5500, MU5–ALGOL. [LIN68]
Modification of instruction sequence.
Methods of address generation for branch target: relative, skip, absolute, etc.

Meta Instructions

Univac 1108–REPEAT, IBM S/360–EXECUTE. [BRO60]

Sequence Determined by External Signals–Interrupts

IBM S/360, DEC PDP-11, DEC PDP-8, CDC 6600. [CHU72]
Types.
Priority and levels.
Methods of handling.

Sequencing Within Instructions, Execution, and Microprogramming (3½ Hours) [ROI69; CHU72; STO75, Ch. 10; IBB72]

The student must fully understand the basics of instruction fetching, decoding, and execution and be able to time out the interpretation of an instruction in a simple machine.

Concepts [FLY67]

Cycle–as a state transition in a finite state machine.
Data paths and control points.
  Control timing and data path timing.
  Control finite state machines.
  Execution finite state machines.
Decode of an operation.
  Gating descriptions.
  Sequencing.
Execution.
  I and E cycles. [LOR72; AND67]
  Simple execution.
  Iterated execution cycles (e.g., MULTIPLY, TRANSLATE).
  Concurrent execution. [TOM67; THO70]

Implementation [WIK53; ROI69; HUS70]

Hardwired. [GSC67, Ch. 8; ROI69]
  Timing rings, counters, and decode logic.
Microprogramming. [ROI69; HUS70; TEC71; TEC74; CHU72; DAV72; FUR74; STO75, Ch. 10; TUC67]
  The subject of microprogramming, although strictly speaking an implementation topic, has had much greater significance in computer architecture over the past ten years than a narrow view of implementation would have admitted. Rosin [ROI74] credits this to the notion that "microprogramming is the implementation of hopefully reasonable systems through interpretation or unreasonable machines." This point should be discussed in terms of how well computer systems have met their users' needs.
    Read only storage. [HUS70]
    Emulation. [ROI69; TUC65; WEB67]
    Read-write microstorage. [WIL72a]
    Contemporary microprogrammed processors: Burroughs B1700, Nanodata QM-1, Data Saab FCPU.

Module 5 – Memory Hierarchies (8 Hours)

This module deals with the architecture and structure of the memory system. The fundamental reason for using memory hierarchies in computer systems is to reduce the system cost. Computer architects must balance the system cost savings accruing from a memory hierarchy against the system performance degradation sometimes caused by the hierarchy.

The architecture of a memory hierarchy defines the logical memory structure available to a program (process). There is a strong interaction between this structure and the design of an operating system for a machine.

Properties of Memory Reference Patterns (1 Hour)

An effective analysis or design of a memory hierarchy requires an understanding of the dynamic behavior of a program in execution. For the purposes of memory hierarchy analysis, it is useful to abstract the execution of a process to derive the "reference string" from an executing program. The reference string is simply the ordered sequence of memory references made by the process. The properties of the reference string have a significant impact on the performance of the memory hierarchy. The most important property of real reference strings is their locality—the tendency for almost all references to be directed to a limited region of the address space.

General references: [BEY66; COF68; BRA68; SIS69; MAT70; JOS70; SAL74a].

Memory Components (4 Hours)

Overview of the Memory Hierarchy

The memory hierarchy includes cache memory, main memory, on-line secondary storage units such as drum and disk, and bulk and off-line storage such as a magnetic tape library. The primary parameters of a memory hierarchy—capacity, access time, and bandwidth at each level of the hierarchy—are often the most important determinants of a computer system's performance. Therefore, it is not surprising that the readings dealing with memory hierarchies are largely concerned with performance.

General references: [KUC70; BEL71, Ch. 3; ANC69; STO75, Ch. 5].

Cache Memories and Other High-Speed Memory Buffers

These memory buffers are an important implementation technique that has a major impact on the performance of central processors. They form a level in the memory hierarchy that is between the main memory and the processor. This level differs from the other levels (i.e., main, secondary, and bulk storage) in the important respect that it is not visible nor available for direct manipulation by the programmer.

Instruction Buffers. [THO70; AND67]

Cache Memories. [KIL62; GIB67; LIP68; COT69; MEA70; KAP72; BEJ74]

Main Storage [THO70, Ch. 4; BOL67]

- Name space vs. real memory space size.
  - Marketing considerations in specifying maximum memory capacities for various IBM S/360 models. [BEL71, Ch. 44]
- Bandwidth.
  - Low order interleaving. [BOL67; BUR70]
- Memory reconfiguration.
  - High order interleaving. Univac 1108

Secondary Storage

- Extended main storage. [THO70, Ch. 4; BOL67]
- Fixed head disk or drum. [COF73, Sec. 5.3; FUR75]
  - IBM 2305 Drum Storage—[IBM23]
- Moving head disk. [COF73, Sec. 5.4; TEO72]
  - IBM 2314 Direct Access Storage Facility. [IBM24]
  - IBM 3830 Storage Control and 3330 Disk Storage [IBM33]

Mass Storage [DAM68; HOA72; JOH75]

IBM 3850 Mass Storage System. [IBM38]

Address Translation Mechanism (3 Hours)

One of the most significant developments in computer architecture has been the distinction between the virtual address space (or name space) of the processor and the physical address space of memory. A range of mechanisms have been developed for mapping one to the other. The following references discuss the more important ones. General references: [KIL62; ARD66; RAN68; DEG70].

Base Register Relocation

Single. CDC 6600 [THO70, Ch. 4]
Double. Univac 1108

Paging

IBM S/370. [KIL62; JOS70; SAL74]

Segmentation

Burroughs B5500/B6500. [DEN65; HAU68]

Implementation of Dynamic Address Translations. [DAL68; LAV71; IBM70; IBM68]

Virtual Machines. [GOL73]

Module 6 — Protection Mechanisms and Hardware Aids to Supervision (4 Hours)

Prerequisite: Module 5

Modern computer systems are often designed for shared use of their resources by several concurrently executing processes (program jobs). The most vital protection mechanism is one designed to ensure that no job's execution can possibly interfere with the information used by any other job. This storage protection is, however, only one important aspect of a general class of *supervision* functions concerned with signaling, response, and assignment activities required for efficient computer resource allocation and acceptable response times. A judicious combination of software and hardware is used to implement supervision functions. Hardware is needed for sequences that occur with high frequency. This module and Module 5 are the areas of greatest interaction between computer architecture and the principles of operating systems.

General Discussion of Protection and Supervision (½ Hour)

General references: [DEG71; HEL73; NEE72]

Motivations for Shared Use of Storage and Processor.

Timesharing of the CPU. [WIK72]

Space-Sharing of Main Storage

Register and stored key protection.
Read/write protection.

Interprocess Communication (2 Hours)

General references: [LAM69; HEL73; HAN73; ORG73]

Interrupt Principles

Saving and restoring system states. RCA Spectra 70 had four operating states: program, executive, interrupt, and machine; whereas the IBM System/360 has only two, program and supervisor.
CDC 6600 exchange jump.

Interrupt Signaling Categories

Clock (timer), I/O condition, address violation, operation invalidity, external signals, explicit program call, etc.

Semaphores. [DIJ68]

Privileged Mode (½ Hour)

General references: [CLA64; HEL73; IBM70].

Storage Protection

Interrupt Handling

Mode-Change Mechanisms

Hardware-Operating System-Problem Program Interactions

Virtual Memory Mechanisms for Protection (1 Hour)

General references: [DEN66; DAL68; ORG72; NEE72; SCH72; IBM70].

Typical Major Examples to Illustrate Module Topics

Although almost every system has its distinct features, the following can illustrate most principles and many variations found in actual practice. The asterisks denote virtual storage systems.

CDC 6600
DEC PDP-11
DEC PDP-10
IBM System/360
Burroughs B5500/B6500* [ORG73]
IBM System/370*
Multics* [DAL68; ORG72; SAL74b]

Module 7 – Specialized Processors (5 Hours)

A number of functionally specialized processors have been developed. These processors are defined primarily by the data types which they can process. They do not interpret general programming languages. This module deals with some of the more common specialized processors.

Input/Output Processors (2 Hours)

An input/output processor specializes in the management of peripherals. It controls the details of transmission of information between secondary memories or terminals and main memory. An input/output processor does not usually alter information; it is merely an interpreter for moving information. Computational capability is only required if recoding and reformatting of information is necessary, or if operations are to be carried out between second memories without central processor intervention.

General reference: [STO75, Ch. 6].

Copy Logic, Buffers and Direct Memory Access Capability

DEC PDP-8, HP2100, DEC PDP-11. [GSC67, Sec. 8.4]

Channels

IBM 1800 Special Data Channels. [BEL71, Ch. 33]
IBM 7607 and 7909 Data Channels. [BEL71, Ch. 41]
Selector and Byte Multiplexer Channels. [PAD64]
Block Multiplexer Channel. [BRZ72]

Peripheral Processors

CDC 6600 PPU's. [THO70, Ch. 7 and 8]
CDC 7600 PPU's. [CDC76, Ch. 5]
Univac 1108 I/O Processor.

Microprocessors and Microprocessor Applications (2 Hours)

Over the 30 years in which computers have existed, several thousand species of machines have been built by a variety of organizations for a variety of functions. The implementations of these machines were based on available semiconductor and magnetic technologies. Currently, semiconductor technology is such that a small stored program processor can be fabricated on a single silicon die. These processor-on-a-chip devices, when executing a fixed program, form the basis for hand-held calculators, appliance and automotive controllers, and simple terminals. They also form the basis for many programmable products: point-of-sales terminals, instruments, factory data collection terminals, display terminals, etc.

The important aspect of nearly zero cost processor-on-a-chip computers is that they will be applied on a wide scale in nearly all man-made objects. Though these computers are revolutionary in functional ability, their architectural development has been strictly evolutionary. All of the topics discussed in this report are relevant to their design.

General references: [COM74a; COM74b; LAP72; HOT74].

Terminal Computers

Terminals of this type include CRT-keyboards whose uses range from programming to order entry, point-of-sale, factory data collection, communications control, etc.

Calculators [BEL71, Ch. 20; STO75, Ch. 3]

Electronic calculators (especially the hand-held versions) are excellent examples of specialized processors. They are characterized by decimal rather than binary arithmetic units, a spartan simplicity dictated by strict price constraints, and a very primitive I/O system.

Display Processors (1 Hour)

A display processor is a complex system that manipulates information for display terminals. It must perform a substantial number of local operations on a set of specialized data types which are representations of complex graphical objects. These representations typically include character strings, points and vectors to be displayed, and control structures which define pictures.

General references: [WAT69; NEW74; BEL71, Ch. 25].

Module 8 – Multiple Computers (4 Hours)

Various forms of multiple computers have been designed as a means of (1) increasing the reliability and improving the performance of computer systems, and (2) distributing computing according to physical location needs to reduce communication link costs. Two standard forms are the multiprocessor computer (which consists of two or more processors that share a common memory) and geographically distributed computer networks (which are usually a collection of physically separated computers).

Multiprocessor computers which share the same physical memory and addressing space but have only a few processors have existed for some time. Systems are now being built with a large number of processors, though. Although some of the multiprocessor's problems are still in the research domain, the system designer must be aware of the issues of reliability and performance (e.g., conflicts arise when multiple processors access a common resource) and the mechanisms for intercommunication which permit the processors to share common resources.

Understanding computer networks requires an understanding of their constituent components (the interconnection links, the computers, and their operating systems) and the tasks which are to be distributed among them. Computer network analysis is similar to the analysis of other networks in which multicommodities are transferred among links and nodes on a dynamic basis. Many of the studies that have been carried out for telephone communication (especially switching) apply.

Finally, there are ad hoc, tightly coupled computer structures which are neither interconnected via standard communications links nor share the same memory. Examples of these structures include IBM's Attached Support Processor system, communications front ends, and specialized file, array, and display processors.

The advent of microprocessors clearly forces the growth of various multiple computer structures. The simplest structures will be based on communications links with processes being assigned to specific processors on a functional basis.

Multiprocessor Computers (2 Hours)

General references: [BEL71, Part 5; HEL73; ENS74].
Systems:
Burroughs D825. [BEL71, Ch. 36]
Burroughs Interpreter. [DAS72]
Carnegie-Mellon University C.mmp. [WUL72]
Bolt, Beranek and Newman. [HEA73]
Univac 1108. [STN67]
IBM System/370 Model 168 MP. [MAC74]

Switching and Interconnection Structure. [BLA64b; DAS72]

Performance as a Function of the Number of Processors. [LEH66; WUL72]

Sharing and Resource Contention. [SKI69]

Reliability.

Computer Networks (2 Hours)

General references: [BEL71, Ch. 40; IEEE73; COM73; BEL74].

Structures and Analysis. [ROB70; FRA71; ORN72]

Communication Links and Message. [MAR72]

Facilities and Use. [ROB73; KLE74]

Module 9 – Performance Evaluation (9 Hours)

Prerequisites: Modules 2, 3, 5, and 6, plus a basic course in probability. [FEL68]

The performance evaluation of a computer system consists of deriving indices of its quality, such as speeds, storage capacities, and efficiencies of resource use. This assessment requires an understanding of the purposes the system is to serve as well as the hardware and software components that constitute it. These purposes are given explicit definition in the selection and representation of workloads and performance indicators.

Performance evaluation is frequently concerned with more than just producing the values of a few performance indicators such as bandwidth, throughput, response time, and resource utilization for a given configuration. It must determine the dependence of these indicators on various parameters of the system so as to help designers choose among a rather large collection of alternative configurations and options in their efforts to meet user's specifications.

Different types of performance studies (e.g., different levels of detail) are appropriate to various classes of computer professionals such as architects, hardware and software engineers, installation managers, system programmers, problem programmers, etc. Despite the different needs of these groups, they all use the same basic classes of evaluation tools: (1) mathematical analysis, (2) simulation techniques, and (3) measurement. All of these are introduced in this module in various contexts, and some time should be spent comparing their strengths and weaknesses.

General references: [CAL67; SMI68; LUC71; BEL71, Ch. 3; FRE72; DRU73; HEL75; STO75, Ch. 11].

Overview (1½ Hours)

Nature of Computer Performance Evaluation.

Evaluation Classes (Hardware, Software, Systems).

Definition of Performance Measures such as Bandwidth, Throughput, Response Time, Resource Utilization, etc.

Measurement Techniques.

Hardware. [ROE69; BON69; DRU73; IBM70 (Program Event Recording)]

Software. [CAM68; CHN69; SED70; BAD71; LUC71]

Introduction to Techniques of Analysis (2½ Hours)

Elementary Queueing Theory

When the information necessary to evaluate a computer system's performance exceeds the values of simple parameters of the hardware structure, performance evaluation must address the underlying stochastic nature of operating a computer system. Requests for service arriving at processors within a computer system often can only be modeled as a random process.

References: [KLE75, Ch. 3-4; STO75, Ch. 11; HEL75, Ch. 5].

Simulation Theory

There are instances when a more detailed analysis of system behavior is needed than can be obtained from queueing models, and instances when a system to be analyzed cannot be adequately approximated by known analytical methods. In these cases, simulation techniques are the most appropriate tools available to the analyst.

References: [NIE67; MCD68; GOR69].

Workload Selection and Characterization (1 Hour) [FER72; WIN73]

Total System Load. [FRM68; WAL67]

Benchmarks.

Synthetic Workloads. [BUC69; SRE74]

Components and Subsystems Performance (1 Hour) [DRU73; HEL75]

Central Processor. [MUR70; BEL71, Ch. 3; SOL66]

Kernels. [CAL67]

Mixes. [RAI64; GIS70]

Main Storage

I/O Devices and Subsystems. [HEL70]

Disk and Drum. [TEO72; FUR75]

Printer.

Computing System Performance (3 Hours) [HEL75]

The references cited for system studies are only suggestions. Actual studies may involve components, subsystems, or systems.

Batch Systems.

Multiprogramming Systems. [ONE67; CAN68; STU71]

Timesharing Systems. [SCE67; MCK69; DOH70; BAD71; NIE71; SHE72]

Virtual Memory Systems. [SAL70; ORG72; SCE73; HEL75]

Module 10 – Reliability (5 Hours)

Reliability affects everyone connected with computer systems, from machine designer to end user. As more complex systems are designed and fabricated, their designers must exercise greater care during design in order to ensure that the systems they develop operate with an acceptable level of reliability. The material in this module helps the student to understand and determine various reliability parameters of a system, given knowledge of its constituent parts, and to design (or configure) systems for increased reliability.

Basic Measures and Calculations (2 Hours)

Attempts to evaluate a computer system quantitatively require an understanding of the fundamental measures of reliability (e.g., mean-time-to-failure, mean-time-to-repair, expected-system lifetime). In addition, there are a few elementary computations that are useful when attempting to estimate system reliability.

General references: [ESA62; BOU71; MAH71; HEL73, pp. 443-452].

Codes for Error Detection and Error Correction (1 Hour)

An essential basis for any computer system that must operate with a high degree of reliability is an error detection, and possibly an error correction, scheme for transmission and storage of data. In memory systems, redundant error detection and/or correction information is held in the memory. Different codes are often required for distinct levels in the memory hierarchy, and each memory technology may impose different demands on its code. Similarly, communication links are not error-free, and thus additional information must be transmitted with a message to ensure its integrity.

Another area in which error detection and correction plays an important part is the use of arithmetic codes for the detection and correction of errors in the results produced by arithmetic operations.

General references: [HIL68, Ch. 8; AVI71b; PET72; RAO74].

Diagnostic Procedures (1 Hour)

Diagnostic procedures are the set of programs that exercise each of the components of a computer system to validate that it is working correctly or that help isolate the component that is failing. The frequency with which diagnostic procedures are unable to detect failures when a system is known to be malfunctioning underscores the need to discuss the principles of testing. The student should comprehend the impossibility of exhaustive testing, the need to understand the structure as well as the function of the component under test, and the concept of "bootstrapping" diagnostics–i.e., validating a small "core" of hardware and then using the core to test the remaining facilities.

General references: [BAT70; CHA70; JON71].

System Reliability and Serviceability (2 Hours)

As computer systems continue to be applied to an increasing variety of real-time environments, stringent demands are made on the computer system's availability and serviceability. The increasing complexity of large computer systems also requires the inclusion of more sophisticated machine-checking capabilities.

General references: [CAR64; FOX75; ORN75].

Machine Checking Techniques.

Error Logging and Recovery.

Fault Tolerant Techniques. [AVI71a].

Module 11 – System Design Evaluation (6 Hours)

Introduction

To appreciate how the architectures of computer systems develop, one must analyze complete systems. Since formal techniques for the analysis of these systems do not yet exist, there is no substitute for studying some existing systems closely. The purpose in doing this is to try to deduce the reasons for various design decisions and to see how design decisions in some aspects affect those in others. Moreover, these studies provide the opportunity to compare the techniques adopted by different systems for solving fundamental problems. The instructor may prefer to introduce such studies gradually throughout the course of study rather than treating them as a comparative analysis at the end.

The subject matter to be covered is best represented by a matrix of systems and concepts (Figure 1). The horizontal dimension compares different techniques for solving the same fundamental problems, and the vertical dimension shows how solving one of the problems in a particular way constrains the solution of other problems. Each of these dimensions is to be evaluated in as quantitative a manner as possible.

Two points are to be stressed here: environment and technology. The first is dealt with because it is imperative that the student develop an understanding of the user environments for which these systems were actually intended. By environment we mean the complete set of users' specifications: the functional capabilities that were

	COMPUTER SYSTEMS			
Computer System Design Problems	Large Systems	Medium Systems	Minicomputers	Microcomputers
Data Representation				
Instruction and Addressing				
Interpretation				
Memory Hierarchy				
Protection Mechanisms and Hardware Aids to Supervision				
Input/Output				
Facilities for Multiprocessing				
Reliability				
Performance				

Figure 1. System design evaluation matrix

expected, the relative cost of the system (relative to others of its day) [SHA69], the language processors and operating systems that were expected, and the reliability of the system in its environment.

The second is addressed because it is impossible to compare systems without considering their technological base. The instructor should normalize these considerations whenever possible, but also call attention to them when they represent technical constraints in the system designer's environment. For example, the availability of different memory elements and the relative cost of circuits frequently affect architectural decisions.

Computer Systems

The systems selected for comparison should include an interesting disparity of environments as well as some similarity. The selection must be influenced, of course, by the availability of reference materials and the students' and instructor's familiarity with specific systems. Here are some suggestions.

Large Systems

a. IBM System/360 and System/370 in general and System/370 Model 158 [IBM58] or Model 168 [IBM68] in particular.
b. DEC PDP-10.
c. CDC 6600.
d. Burroughs B6500.
e. Univac 1108.

Medium Systems

a. Burroughs B1700.
b. IBM System/370 Model 145 [IBM45].
c. DEC PDP-11/45.
d. Honeywell Series/60 level 64..

Minicomputer Systems

a. DEC PDP-8.
b. HP2100 and HP21MX.
c. Microdata 1600.
d. Data General Nova.
e. IBM System/32.

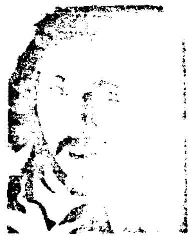
Microcomputer systems

a. Intel 8080.
b. National IMP-16.

Acknowledgements

The committee would like to express its appreciation to Professor E. J. McCluskey of Stanford University, who was instrumental in establishing the task force, and to the distinguished group of computer architects who served as referees for this report. Their constructive comments were extremely influential in the preparation of the final version of this report.

The committee would also like to thank Ms. Patricia Stoaks of Palyn Associates who typed all the drafts of this report. ■



George E. Rossmann is on the staff of Palyn Associates, where he has been involved with the design and evaluation of medium and large scale processors, the design and evaluation of memory hierarchies, and the evaluation of soft machine architectures. He has worked for IBM in the areas of advanced technology, circuit design, and large systems architecture. From 1971-1973 he was an assistant professor at the University of Delaware.

He received the BS degree in electrical engineering from Lafayette College, the MS from Penn State University, and the Ph.D from Syracuse University.

Dr. Rossman served as chairman of the Workshop on Education and Computer Architecture in 1973. He has published a number of papers in technical journals and holds one patent.

C. Gordon Bell is vice president of engineering for Digital Equipment Corporation.

On leave as professor of electrical engineering and computer science at Carnegie-Mellon University, he was previously manager of Computer Design for Digital from 1960-1966. During that time he was responsible for DEC's PDP-4, -5, and -6 computers. He consulted for DEC in 1966-1972 while working at CMU on various computers and products including the PDP-11.

His work in the computer field covers computer architecture, modularity of design, multiprocessors, and applications. His publications include *Computer Structures* (McGraw Hill) and *Designing Computers and Digital Systems* (Digital Press) and several papers.

In addition to his industrial interests, Bell has served as a member of three COSINE committees of the National Academy of Sciences on computer engineering education, and has served the National Science Foundation, Office of Computing Activities. He is a department editor for *CACM*, and is a Fellow of the IEEE.

Frederick P. Brooks, Jr. is Kenan professor and chairman of the Computer Science Department at the University of North Carolina in Chapel Hill. He is best known as the "father of the IBM System/360," having served as project manager for its development and later as manager of the Operating System/360 software project during its design phase. Earlier, he was an architect of the IBM Stretch and Harvest computers.

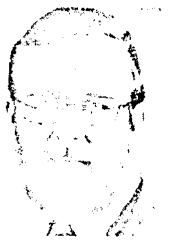
At Chapel Hill, Dr. Brooks has participated in the establishment and guiding of the Triangle Universities Computation Center and the North Carolina Educational Computing Service. He has published *Automatic Data Processing*, the *System/360 Edition of Automatic Data Processing*, and chapters in several other books. His most recent work is *The Mythical Man-Month: Essays on Software Engineering*.

Michael J. Flynn is a founder and vice-president of Palyn Associates and is now the senior consultant for the corporation. He is also a professor of electrical engineering at Stanford University.

He joined IBM in 1955 and, over a 10-year tenure there, worked in the areas of circuit development and computer organization and design. At IBM he was a designer and manager of prototype versions of the IBM 7090 and 7090II. Later, as manager, he was responsible for design and development of the 360 Model 91 Central Processor.

Dr. Flynn joined the faculty of Northwestern University in 1966 and later was professor of computer science at The Johns Hopkins University.

A vice-president of the IEEE Computer Society, Flynn has served as chairman of the IEEE Technical Committee on Computer Architecture and as associate editor of the *IEEE Transactions on Computers*. He has published over thirty papers in technical journals and holds two patents. He received his BSEE from Manhattan College, his MSEE from Syracuse University, and his Ph.D from Purdue University.



Samuel H. Fuller is an associate professor of computer science and electrical engineering at Carnegie-Mellon University. His research interests include topics in computer architecture and the performance evaluation of computer systems. He is currently involved in the measurement and evaluation of C.mmp, a multi-miniprocessor computer system, and the design of new multiprocessors based on the emerging microcomputer technology.

Dr. Fuller received the BSE degree in electrical engineering from the University of Michigan in 1968, the MS and Ph.D degrees from Stanford University in 1969 and 1972, respectively.

He is the author of *Analysis of Drum and Disk Storage Units* (Springer-Verlag), and a co-author of *Introduction to Computer Architecture* (SRA). Dr. Fuller is an editor of the Computer Systems Department of *CACM* and is a member of the ACM and the IEEE Computer Society. He is also a member of Tau Beta Pi, Sigma Xi, Phi Kappa Phi, and Eta Kappa Nu.

Herbert Hellerman has been a professor at the State University of New York at Binghamton since 1969. Earlier, he served on the electrical engineering faculties at Syracuse University and the University of Delaware. Dr. Hellerman was with the IBM Yorktown Research and System Development Divisions for 10 years where he worked on the architecture and performance of multiprogrammed, timeshared, and multiprocessor systems. He is the author of several books. His current computer work centers on hardware/software performance, operating systems and computer system education.

He received his undergraduate degree at Purdue University and the Ph.D from Syracuse University.

Bibliography

We have tried to identify a minimum set of references which, in our opinion, represent the best path through the material in the course of study. These references have been marked with asterisks. The remaining references have been selected because they simplify and broaden the path established by the first set.

[ACM68] "Curriculum 68," *CACM*, 11, 3, March 1968: pp. 151-197.

*[AMD64] Amdahl, G. M., G. A. Blaauw, and F. P. Brooks, "Architecture of the IBM System/360," *IBM Journal of R&D*, 8, 2, April 1964: pp. 87-101.

[ANA73] Anagnostopoulos, P., G. M. Stabler, and A. van Dam, "Computer Architecture and Instruction Set Design," *AFIPS Proc. NCC*, 42, 1973: pp. 519-529.

[ANC69] Anacker, W., and C. P. Wang, "Evaluation of Computing Systems With Memory Hierarchies," *IEEE Trans. on Computers*, EC-16, 6, Dec. 1967: pp. 764-773.

*[AND67] Anderson, D. W., F. J. Sparacio and R. M. Tomasulo, "The IBM System/360 Model 91: Machine Philosophy and Instruction Handling," *IBM Journal of R&D*, 11, 1, January 1967: pp. 8-24.

*[ARD66] Arden, B. W., B. A. Galler, T. C. O'Brien and F. H. Westervelt, "Program and Addressing Structure in a Time Sharing Environment," *JACM*, 13, 1, Jan. 1966: pp. 1-16.