**Georg Institute of Technology**

**Office of Contract Administration**

**Project Administration Data Sheet**

**Project No.:** G-36-609

**Sponsor:** European Research Institute of Ireland

**Type Agreement:** Contract No. 2021

**Award Period:** From 2/28/83 To 4/15/83

**Sponsor Amount:**
- Estimated: $
- Funded: $

**Title:** Investigate and Classify Various Types of Computer Architecture

---

**Administrative Data**

1. **Sponsor Technical Contact:**
   - John W. Burdette (OCA Contact)
   - Mr. William Donegan
   - Administrative Manager
   - European Research Institute of Ireland
   - Plassey, Castletroy, Limerick, Republic of Ireland

2. **Defense Priority Rating:** N/A

**Military Security Classification:** N/A

**Restrictions**

See Attached Supplemental Information Sheet for Additional Requirements.

**Travel:** Foreign travel must have prior approval — Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

**Equipment:** Title vests with None proposed

---

**Copies To:**

- Project Director
- Research Administrative Network
- Research Property Management
- Accounting
- Procurement/EES Supply Services
- Research Security Services
- Reports Coordinator (OCA)
- Research Communications (2)
- GTRI
- Library
- Project File
- Other I. Newton
Date 9/13/83

Project Title: Investigate and Classify Various Types of Computer Architecture

Project No: G-36-609

Project Director: Dr. Pin-Yee Chen

Sponsor: European Research Institute of Ireland

Effective Termination Date: 4/15/83

Clearance of Accounting Charges: 4/15/83

Grant/Contract Closeout Actions Remaining:

- [X] Final Invoice and Accounting Documents
- [ ] Final Fiscal Report
- [ ] Final Report of Inventions
- [ ] Govt. Property Inventory & Related Certificate
- [ ] Classified Material Certificate
- [ ] Other

Assign to: ICS

(School/ Laboratory)

COPIES TO:

Research Property Management
Accounting
Procurement/EES Supply Services

Research Security Services
Reports Coordinator (OCA)
Legal Services (OCA)
Library

EES Public Relations (2)
Computer Input
Project File
Other PIN-YEE CHEN
I. INTRODUCTION

In recent years, we have experiencing the advancement of hardware technology. As the cost of hardware keeps decreasing, it makes the design of computer system more cosy-effective. Hence various types of computer architectures have been proposed in recent years. Not only because hardware is less expensive, but because people knows more about applications and the computer systems, there is no shortage in the field of innovation of the design of computer architectures. Therefore, it is essential to establish a classification scheme to describe those computer architecture and structure features which are identical or similar and, furthermore, to develop a basis for the development of a design methodology as well as providing a framework for historic research. From the user's point of view, the classification scheme helps in the selection of a type of systems suitable for certain applications. This in fact imposes an effort in thinking about the dimensions of interests in Computer Architecture.

Classifications have been performed in various fields. By scientists, it is used to present knowledge and to discover order. So far the most prominent example is the taxonomy of organism in biology.

Its goal is building a classification and naming system, updating it on account of all discovered facts [Beav62]. Plan and animal kingdom were further systematically considered by Linnaeus (1767-1778). He introduced a Latin, bionomial nomenclature. The influence of Linnaeus is still felt today. Since then, much literature and many organizations are devoted to taxonomy and classification in botany, zoology, medicine, archeology and libraries.

In computer science, several classification schemes have been suggested (will be discussed in Section II). In contrast to the classification in
various other domains, such as biology, a classification of computer architecture is not only useful in analysis, but also in synthesis.

In analysis, the classification supports the top down analysis of a machine. The appropriate concept level can be addressed. The partial or exhaustive comparison of various machines, on account of all concepts and parameters, is simplified yet accurately. A specific concept can be studied, for its form and occurrence in various computer architectures. Furthermore, it gives a simplified picture for a layman to understand a machine in a short time.

In synthesis, the requirements of the users can be systematically addressed by the classification. It clarifies the relationship of decisions. New ideas and parameters can be introduced.

However, computer architecture is still a very young field. The advancement of VLSI technology and the understanding of computational structure may require significant changes of the current classification scheme in the next decade. It also may be argued that the exhaustiveness of a classification is a hindrance for further invention, especially for a young dynamic field like computer architecture. In a classification related concepts are presented in an orthogonal fashion, which may cause some problem in some cases where sibling concepts and parameters are not completely mutual orthogonal.

Furthermore, the classification of computer architecture can be approached from quite different directions. Thus the different views of architecture lead to different concepts to be classified. When we review existing classification schemes in Section II, we can see the diverse and different emphasis on various parts of computer architecture.

In Section III we will propose a classification scheme.

This justification of the proposed scheme is that it should serve as a
base for the evaluation of the possible alternatives; and it should be useful in classifying structures and concepts which will emerge in the next years, and be of use to the designers of these structures. A further justification of the scheme is that it should help in the establishment of a systematic design methodology and the elements of the classification scheme can be composed and decomposed by operations which are suitable for the purposes of the computer architect.

II. Review of Existing Classification Schemes

Computer systems are in a constant evolutionary state. Therefore, it is not surprising that an adequate classification cannot be found at this time. Moreover, existing classification schemes differ in the information on which they are based.

II.1 Flynn's Classification

In 1966, Flynn proposed a classification which is still widely used. As it is well known, based on the instruction streams and data streams, Flynn partitions systems into four categories by looking at the four pairs derived from the product of:

\[(\text{Single, Multiple}) \times (\text{Instruction, Data})\]

(i.e. SISD, MISD, SIMD, MIMD)

The SISD systems correspond to conventional uniprocessors. MISD class is empty, SIMD to array processors controlled by a single control unit (e.g. ILLIAC IV) and MIMD to multiprocessors with various control units.

Such a broad cut in the myriad of organizations can not be fully satisfying. Some want to redefine the meaning of instruction and/or data streams. For example, it has been proposed [ReMc82] to interpret the single or multiple
data stream on a per processor basis. Then a vector processor of the CRAY-1 type would be an SIMD machine, a multiprocessor with conventional uniprocessing elements (e.g. C.mmp) would be an MISD system and a multiprocessor with pipelined processors (e.g. S-1) would be MIMD. We do not subscribe to this interpretation because, first we have a hard-time classifying pipeline processors and do not consider them as "pure" SIMD systems, and second, we do not really know were to place multifunctional unit machines (pipelined or not) in Flynn's classification. In the literature many authors restrict themselves to the classes SISD, SIMD, and MIMD. A further difficulty occurs if a computer system contains both parallelism and pipelining.

Furthermore, a computer may be classified to a different class depending on which part of the system you are looking at. For instance, CDC 6600's is a MIMD class, but its CPU is definitely not. Categorizing pipeline processors is a challenge. At what level do we define pipelining? If we consider that a buffer holding a prefetched instruction is an instance of pipelining, then most current (micro)processors are pipelined. If we look at the pipelining of arithmetic operations, then we might want to distinguish between local pipelining within a functional unit (as in the multiplier-divider of the IBM 360/91), unifunction pipelines which accept new operands at every cycle (as in the adder of the IBM 360/91, the functional units of the CDC 7600, and the CRAY-1 and CYBER 205 scalar units), and multifunction pipelines with associated set-up and flush times (as e.g., in the TI ASC, CDC STAR-100 and vector units in the CYBER 205). Finally, we might want to restrict the term pipeline processor to those architectures which include a vector instruction set (e.g., TIASC, CDC STAR-100, CRAY-1, CYBER 205). But in this latter case, as it has been painfully realized by some manufacturers, a fast scalar processor must also be included to have a viable architecture. In the following, we will try and avoid the term pipeline processor and speak instead of pipelined
funtional units (for the second case described above) and of vector processors if a vector instruction set- and associated hardware - is present (a weakness of this terminology is that a highly overlapped processor such as the Amdahl 470 is not distinguishable from an Intel 8080).

It should be noted that this confusion about pipelining is widespread. This issue must have been debated by the authors of the Computing Reviews classification since the term Pipeline Processors appears under two different headings: Single Data Stream Architectures and Multiple Data Stream Architectures. We can even be more open minded and within the framework of Flynn's classification we could argue that vector processors fit under the MISD class (a single datum - a vector - passing through several stations - instructions - simultaneously), or SIMD (a single instruction - a pipe - working on several elements of a vector simultaneously), or MIMD (a combination of the two previous interpretations.)

II.2 Kuck's Classification [Kuck80]

Instead of using data stream in Flynn's classification, Kuck uses

\[(\text{Single, Multiple}) \times (\text{Instruction, Execution})\]

He distinguishes single instruction stream from multiple instruction streams simply on the basis of the number of "programs" being executed at once, where a "program" is assumed to need a single instruction location register in some control unit for its execution. Thus, two subroutines being executed simultaneously for one user, count as two instruction streams. The distinction between single and multiple execution streams is based on the ability of the global control unit to sequence one or more operation types, respectively, at once. For main memory and processor operations, an operation type corresponds roughly to what has traditionally been specified by a single operation code.
For example, store, fetch, fixed-point addition, floating-point operation, left shift, logical compares, and so on, are single operation types.

According to Kuck's classification, a typical uniprocessor computer is a SISE system. For parallel and pipeline computer systems, a single instruction stream leads to the execution of a single operation type on many data elements, so they are SISE machines. An example for a SIAME machine is in a multifunctional processor (e.g., the CDC 6600 CPU) a MAISE machine can be represented by a uniprocessor with instruction level multiprogramming (e.g., the CDC 6600 FPUs). A typical multiprocessor computer system is a MIME machine.

Under this scheme, parallel and pipeline systems cannot be distinguished from the conventional uniprocessor. However, it is desirable to distinguish these systems from standard uniprocessors. Kuck introduces the ideas of scalar and array instruction streams, and scalar and array execution streams. Thus, ILLIAC IV has scalar type instructions with an array execution stream, whereas Burroughs BSP, the TI ASC, CRAY-1, and CDC CYBER-205 have array instruction streams with array execution streams. The difference is that the ASC, BSP, CRAY-1, and CYBER-205 have instructions that refer to whole vectors at once (with a base, limit, and increment for indexing them), whereas ILLIAC's instructions are more or less indistinguishable from those of a traditional uniprocessor.

To extend the above notion, ILLIAC IV is called a SISSEA (Single instruction, scalar; single execution, array) system, and TI TIASC, Burroughs BSP, CRAY-1, and CYBER-205 SIASEA (single instruction, array; single execution, array) systems. hence, there are a total of 16 types of systems, and this provides a reasonably good taxonomy for a wide range of machine organizations (see Figure 2).

The main theme in Kuck's classification is to categorize machines based
on their global control unit organizations and to realize that system capacity can be strongly related to these categories. Clearly, array execution machines have higher processor capacity than scalar execution systems of the same clock speed. Array instruction stream machines generally have higher control unit capacity in that more simultaneity of control unit operation is possible; hence, instruction handling is faster.

II.3 Feng's Classification [Feng72]

Based on the number of bits which are processed in parallel in a word and the number of words which are processed in parallel, Feng can construct a two-dimensional graph with the ordinate as the number of words processed in parallel and the abscissa as the word length (normally 12, 16, 24, 32, 48, 60 or 64). Hence, a computer structure is represented by a point in a plane (Fig.3) and the ordinate can be determined by the number of processors. For example, C.mmp which contains 16 PDP-11's with word-length 16-bits is represented by (16,16). The ordinate can also be determined by the number of arithmetic and logical units in an array processor. Thus, ILLIAC IV is represented by (64,64).

Therefore, Feng's classification does not allow to distinguish between multiprocessors like C.mmp and array processors. It also does not distinguish between autonomous processors which execute programs and ALU's which execute operations, i.e., it does not distinguish between processing levels.

Similarly Feng's classification scheme fails to represent the pipeline structure at the program level at PEPE. PEPE is characterized as (32,16) and the fact that each set of data (up to 288, each representing a flying object) is processed successively in three different ways is not represented.

II.4 Classification of Multiprocessing
Enslow [Ensl74] defines that a multiprocessor system has to satisfy the following conditions:

1. two or more processors, having access to a common memory, whereby private memory is not excluded
2. shared I/O
3. a single integrated operating system
4. hardware and software interactions at all levels
5. the execution of a job must be possible on different processors
6. hardware interrupts

However, because of more progressive project of computer architecture, two machines may belong to multiprocessor systems but have completely different applications. That is, a multiprocessor system is itself a broad spectrum. Without knowing more detail about its functions, it is of little use to know whether this is a multiprocessor machine or not.

II.5 The Erlanger Classification Scheme (ECS)

Erlanger [Erla77] attempts to cover the full generality of the computer architecture field. The classification is based on the distinction between three processing levels:

1. Program control unit
   Using a program counter and some other registers, and, in most cases, a microprogram device, the PCU interprets a program instruction by instruction.

2. Arithmetic and logic unit
   Each ALU uses the output signals of a microprogram device to execute sequence of micro instructions according to the interpretation
process performed by the PCU.

3. Elementary logic circuit

Each of the microoperations which make up the microoperation set initiates an elementary switching process. The logic circuits belonging to one bit position of all the microoperations are called an ELC.

In his notation, each system is represented by a triple

\[ c = (k,d,w) \]

where:

- \( k \) is the number of program control units;
- \( d \) is the number of ALU's controlled by each PCU;
- \( w \) is the length of information units processed by each ALU.

Pipelining can be expressed at each level with \( k' \) indicating the number (or stage, or level) of pipelining in PCU's, \( d' \) the number of pipelining in ALU's per processor, and \( w' \) the number of pipeline stages in ELC per ALU. Hence, the triple now reads as follows:

\[ t = (kxk', dxd', wxw') \]

All entities are independent of one another. All combinations therefore can appear.

This scheme allows categorization at the system level (e.g., amount of global parallelism, SIMD vs. MIMD organization), can be useful at the ALU or functional unit level, and in addition provides information on the internal pipelining of the ALU's.

Note that the classification schemes we have reviewed so far are concerned with the structure features of processors. Even on this aspect, they are
inadequate to represent a large variety of systems such as non-homogeneous (vector) processors, attached processors, data-flow machines and any special-purpose processor.

II.6 The ACM Computing Reviews Classification Tree

In 1982, ACM Computing Reviews published its new 1982 CR classification system. Under the heading "Computer System Organization," there are five subheadings, they are:

- General
  - Hardware/software interfaces
  - Instruction set design
  - System architectures
  - System specification methodology

- Processor Architectures
  - General
  - Single Data Stream Architectures
  - Multiple Data Stream Architectures (Multiprocessors)
  - Other Architecture Styles
  - Miscellaneous

- Computer-Communication Networks
  - General
  - Network Architecture and Design
  - Network Protocols
  - Network Operations
  - Distributed Systems
  - Local Networks
  - Miscellaneous

- Special-Purpose and Application-based Systems
Fig. 4 shows the detail listing of CR classification.

An initial bone of contention is that architecture in the CR classification is a component of organization while many computer architects will argue that organization is a means to implement an architecture. To us this is a semantic problem of little significance. More important in our view is the separation between Processor Architecture and Special-Purpose and Application-Based Systems in the computer structure area. We agree that functionality is one of the dimensions required for the classification of computer systems. However, the real-time and/or process control processors are less special-purpose and application based than some SIMD machines.

CR classification scheme is rather simple and general: it distinguishes between uniprocessors (SISD and other attributes which do not add anything to the simple SISD labelling), multiprocessors (almost any system with more than one processor except for data-flow machines), and others (e.g., non-Von Neumann machines such as data-flow, HLL architectures, demand-driven, etc...), to which one could have added - among others - data base machines, attached processors, and aforementioned real-time processors).

There are other viewpoints to look at computer architecture. For instance, a classification of computer controls [Tour79], of data-driven and/or demand-driven computer [Tre178], of data base machines [Song81]. In the following we will briefly discuss two additional important classification schemes.

II.7 Classification of Interconnection Networks
Communication plays important roles in the design of computer architecture. Data has to transfer between processors, processor and memory or I/O efficiently. Feng [Feng81] proposes the following four design decisions to classify an interconnection network:

- Operation mode
  - synchronous
  - asynchronous
- Control strategy
  - centralized control
  - distributed control
- Switching methodology
  - circuit switching
  - packet switching
- Network topology
  - regular
    - static
    - dynamic
  - irregular

Figure 5 shows the topologies of interconnection networks and Figure 6 and 7 show examples of static and dynamic network topologies respectively. The cross product of the set of categories in each design decision forms a quadruple:

\[
\{\text{operation mode}\} \times \{\text{control strategy}\} \times \{\text{switching methodology}\} \\
\times \{\text{network topology}\}
\]

The quadruple represents a space of interconnection networks. Obviously, the cross produce contains some uninteresting cases, but a network designer can obtain a meaningful subspace by exercising a practical view of engineering
technology. However, one can look at the interconnection network from reliability or the capability of reconfiguration and partitioning points of view.

II.8 Classification of the Binding of Programs to Machines

Based on the relationships between language and machine, Fig. 8 can be used to introduce some of the basic approaches to computer architecture.

- Conventional Architecture
  It is represented by path 1 (Fig.8). An extensive compilation process translates a high-level-language program to a low-level machine-language program; the latter is then interpreted by the machine.

- Language-Directed Architecture
  It is represented by Path 2. The source program is compiled into a higher level machine language, which is in turn interpreted by the machine. For instance, Burrough B5500, B6500, B6700, B7600 [HaDe68, Earn80] and th MU5 [IbCa78] are architecture toward ALGOL. Th Intel iAPX 432 is another example with a partial orientation towards Ada [Zajm81]

- Type A High-Level-Language Architecture
  It can be represented by the third path in Figure 8. The architecture is directed toward the language to such an extent that the high-level language can be thought of as the assembly language (symbolic machine language) of the system. That is, there is a one-to-one correspondence between the statement types and operators in the high-level language and the machine operations. Note that although this approach is considered the second non-von Neuman architecture category, there is no concise distinction between this approach and the language-
directed approach. Rather, the distinction is one of degree; one could consider the type A high-level-language architecture as being a highly language-directed architecture.

Examples of architecture in this category include a real-time system for the FLUID language [BrCo73], a FORTRAN machine [MePu65], an Algol-W machine [GuFi76], and an LISP microprocessor [StSu79].

- **Type-B High-Level-Language Architectures**

  This architectural category is also represented by path 3 in Figure 8 and is almost identical to the Type A architecture. The only difference is that the assembly process is performed by software in a type A machine, but it is performed by the machine in the type B approach. That is, the machine assembles the source program and then interprets the assembled program.

  Example of type B machines include APL machine [Robi75] a FORTRAN machine [Bask67], and the SYMBOL machine [RiSm71].

- **Type C High-Level-Language Architecture**

  This category is represented by the bottom path (path 4) in Figure 8. Here the architecture and the language are identical, the architecture is the high-level language.

  Examples of such architectures include an Algol machine [Hayn77], APL[ThMy70], Basic[Yama80], Fortran[ChCH80], Pascal[WaLi80], and LISP[Will78] machines.

### III. Hierarchical Approach for Classifying Computer Architecture

Basically, a computer system is consisted of two parts: computer structure and computational structure. Computer structure is referred to the hardware structure and the rules applied to these hardware resources. Com-
putational structure is constituted by a set of abstract attributes specifying
the type and structure of the information components of the machine, their
representation in the machine, and the operations the machine is capable of
performing on them. Furthermore, the relationships between programs (OS com-
piler, or applications) and machine will also be addressed.

Using only computer structural features to classify computer architec-
tures cannot distinguish such features that are belonged to computational
structures. For instance, consider Computer A and B. Both of A and B are
multiprocessor systems which are identical according to computer structure
classification. However, computer A is employed tagged storage and reduced
instruction set while computer B is used in a conventional way. The use of A
and B could be quite different even though they have the same computer struc-
tural features.

Note that while the computational structure to some extent implies com-
puter structural solutions, the inverse is not true; that is, there exist no
one-to-one relation between computer and computational structures.
Consequently, computer structure alone in principle are not sufficient to
characterize or taxonimize a computer architecture. A meaningful classifica-
tion therefore must characterize all significant computational and computer
structural features. Such a characterization should leave out insignificant
implementational details, i.e., should be abstract. Yet it is expandable for
new concepts and ideas.

The set of definitions presented in the following exhibits a hierar-
chical approach to the classification of computer architecture.
0. Computer Architecture

1. Computer Structure

2. Hardware Resources

3. Processor structure

4. Single processor

5. Conventional processor (Von Neumann architecture)

5. Pipelined processor

5. Multifunctional units

4. Multiprocessors

5. Array and vector processor

5. Associative processor

5. Parallel processor

5. Pipelined processor

5. Multiple-instruction stream, multiple data stream processor (MIMD)

5. Multicomputer system

4. Other Architecture Styles

5. Adaptable processor

5. Capability architecture

5. Data-flow architecture

5. High-level language architecture

5. Stack-oriented processor

3. Arithmetic and Logic Structures

4. Functional Units

5. Adder/subtractor

5. Multiplication unit

5. Floating-point operation

5. Logic unit (shift, AND, ...
4. Design Styles
   5. Serial
   5. Parallel
   5. Pipeline
   5. Chaining

4. Rounding, Reliability, Testing and Fault-Tolerance
   5. Diagnostics
   5. Error checking
   5. Rounding schemes
   5. Redundant design
   5. Test generation

3. Memory Structure

4. Memory Organization
   5. Associative memories
   5. Cache memories
   5. Interleaved memories
   5. Mass storage
   5. Primary memory
   5. Virtual memory
   5. Parallel memories

4. Access Methods
   5. Sequential access
   5. Random access
   5. Parallel access

4. Reliability, Testing and Fault-Tolerance
   5. Diagnostics
   5. Error-checking
   5. Redundant design
5. Test generation

2. Interconnection Structures

3. Functions
   4. Interprocessor communication
   4. Intraprocessor communication
   4. Processor-memory communication
   4. Computer-computer communication

3. Operation Mode
   4. Synchronous
   4. Asynchronous

3. Control strategy
   4. Centralized control
   4. Distributed control

3. Switching Methodology
   4. Circuit switching
   4. Packet switching

3. Network Topology
   4. Regular
   5. Static
   5. Dynamic
   4. Irregular

3. Fault-Tolerance and Diagnosis
   4. Diagnosis
   4. Error-checking
   4. Degree of fault-tolerance

2. Control Structures

3. Control Mode
   4. Synchronous
4. Asynchronous

3. Control strategy
   4. centralized control
   4. distributed control

3. Control Design Styles
   4. Hardwired control
   4. Miroprogrammed control
   4. Writable control store

2. Instruction Level Structures

3. Types of Instruction Set
   4. 0 - address instructions
   4. 1 - address instructions
   4. 2 - address instructions
   4. 3 - address instructions
   4. Mixed type instructions
   4. Data-flow instructions
   4. Miscellaneous

3. Design Philosophy
   4. Reduced instruction set concept
   4. Complex instruction set concept
   4. Stack-oriented processor working state
   4. Register-register processor working state
   4. Memory-memory processor working state

1. Computational Structure

2. Control Structure

3. Resource Scheduling
   4. Resource Allocation
5. Processor allocation
5. Memory management
6. segmentation
6. paging
6. memory mapping
6. addressing schemes

4. Communication and Synchronization
5. Synchronization mechanisms
5. Communication protocols

3. Execution Control

4. Single Resource Control
5. Sequencing
5. Branching
5. Repeated sequencing
5. Concurrent sequencing

4. Multiple Resource Control
5. Mode of Operation
6. centralized
6. distributed

5. Synchronization Rule
6. master-slave
6. P-V semaphores
6. data-flow
6. autonomous
6. coroutine

2. Information Structure

3. Function Set

4. Access Function
5. Access types
   6. scalar access
   6. set access
   6. structure access

5. Function types
   6. Read function
   6. Write function

5. Addressing Mode
   6. absolute addressing
   6. abstract addressing
   6. capability addressing
   6. content addressing
   6. object addressing

4. Binding Function
   5. Single assignment
   5. Eventual value assignment
   5. Multiple value assignment
   5. Shared value assignment

4. Arithmetic and Logic Function
   5. Arithmetic operation
   5. Logic operation
   5. Relational operation

4. Decoding Function
   5. Instruction decoding
   5. Descriptor decoding
   5. Security decoding

4. Test and Set Function
4. Conversion Function
   5. Object and/or representation conversion

3. Object Type

4. Basic Type
   5. Instruction
   5. Integer
   5. Real
   5. Floating-point
   5. BCD number
   5. Boolean
   5. Descriptor
   5. Pointer (Address)
   5. Identifier
   5. Capability
   5. Structure
   5. Record
   5. Type tag

4. Set Type
   5. Homogeneous set type
   5. Heterogeneous set type
   5. Domain

3. Object Representation

4. Basic object
   5. Generic representation
   5. Self-identifying representation
   5. Self-descriptive representation

4. Data Structure Object
   5. Linear ordering
5. Relational structure
5. Graph
5. Partition
4. Set-type Object
5. Descriptor
5. Set of basic object representation

2. Program Structure
3. Data Mechanism
  4. By literals
  4. By value
  4. By reference
3. Control Mechanism
  4. Sequential
  4. Parallel
  4. Recursive
3. Compilation Process
  4. Assembler
  4. Interpreter
  4. Directly executed
IV Conclusion

We have reviewed existing classifications of computer systems. Those attempting to cover the whole spectrum of computer architectures have been found to be too general. Those which intentionally restrict themselves to specific types of architectures or devices are useful but by nature insufficient. We introduced two important categories in computer architecture: computer structure and computational structure. The proposed approach is an attempt to show that not only hardware structure of the system is important, but the operational structure is also important. In order to utilize the system efficiently, hardware structure and computational structure should be matched. Computational structure is dynamic, it will show different operational structures with different applications. However, hardware structure is static in general, that is, it can not be changed, hence for some application the hardware structure can not match with the operational structure. Therefore, the performance of the computer system is unsatisfactory for some applications. People begin to realize the problem and dynamic structure of computer system is starting to emerge to address this problem [Kaka79].

We do not claim the proposed classification is the only scheme or "complete" scheme. As we noted in Section 1, computer architecture is a young and dynamic field, it is almost impossible to cover or contain such a field. A classification is merely directed to an abstraction and simplification and pointed out the important elements under considerations.
REFERENCES


575-587.


[ZAJM81] S. Zeigler et al., "Ada for the Intel 432 Microcomputer", Computer,
Fig. 1: M. Flynn's classification with some examples

Fig. 3: T. Feng's classification with some examples
<table>
<thead>
<tr>
<th>Instruction Streams</th>
<th>Single</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scalar</td>
<td>Array</td>
</tr>
<tr>
<td></td>
<td>SISSES</td>
<td>SISSEA</td>
</tr>
<tr>
<td>Scalar</td>
<td>Babbage von Neumann Uniprocessor</td>
<td>ILLIAC IV (1 quadrant) STARAN IV Simple parallel processors</td>
</tr>
<tr>
<td>Array</td>
<td>SIASES</td>
<td>SIASEA</td>
</tr>
<tr>
<td></td>
<td>Uniprocessor with array control unit</td>
<td>TI ASC (1 pipe) CDC STAR Simple pipeline processor</td>
</tr>
<tr>
<td></td>
<td>MISSES</td>
<td>MISSEA</td>
</tr>
<tr>
<td>Scalar</td>
<td>Uniprocessor with instruction level multiprogramming</td>
<td>ILLIAC IV (4 quadrants) (1 job)</td>
</tr>
<tr>
<td>Array</td>
<td>MIASES</td>
<td>MIASEA</td>
</tr>
<tr>
<td></td>
<td>Uniprocessor with array control unit and instruction level multiprogramming</td>
<td>MISSEA with array instructions</td>
</tr>
</tbody>
</table>

Figure 2 Kuck's Classification Scheme
C. Computer Systems Organization

C.0 GENERAL
Hardware/software interfaces
Instruction set design
System architectures
Systems specification methodology

C.1 PROCESSOR ARCHITECTURES
C.1.0 General
C.1.1 Single Data Stream Architectures
Multiple-instruction-stream, single-data-stream processors
(MISD)
Pipeline processors
Single-instruction-stream, single-data-stream processors
(SISD)
Van Neumann architectures

C.1.2 Multiple Data Stream Architectures
(Multiprocessors)
Array and vector processors
Associative processors
Interconnection architectures (e.g., common bus, multiprocessor memory, crossbar switch)
Multiple-instruction-stream, multiple-data-stream processors (MIMD)
Parallel processors
Pipeline processors
Single-instruction-stream, multiple-data-stream processors
(SIMD)

C.1.3 Other Architecture Styles
Adaptable architectures
Capability architectures
Data-flow architectures
High-level language architectures
Stack-oriented processors

C.1.m Miscellaneous
Analog computers
Hybrid systems

C.2 COMPUTER-COMMUNICATION NETWORKS
C.2.0 General
Security and protection

C.2.1 Network Architecture and Design
Centralized networks
Distributed networks
Network communications
Network topology

C.2.2 Network Protocols
Protocol architecture
Protocol verification

C.2.3 Network Operations
Network management
Network monitoring
Public networks

C.2.4 Distributed Systems
Distributed applications
Distributed databases
Network operating systems

C.2.5 Local Networks
Access schemes
Buses
Rings

C.2.m Miscellaneous

C.3 SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS
Process control systems
Real-time systems

C.4 PERFORMANCE OF SYSTEMS
Design studies
Measurement techniques
Modeling techniques
Performance attributes
Reliability, availability, and serviceability

C.m MISCELLANEOUS

Figure 4 CR Classification Tree
Figure 5. Topologies of interconnection networks.

Figure 6. Examples of dynamic network topologies: (a) single stage; (b-l) multistage; and (i) crossbar.
Figure 6

Examples of multistage and crossbar (1) dynamic network topologies.
Examples of static network topologies: (a) one dimensional; (b-f) two dimensional; and (g-j) three dimensional.
Figure 8 General language/machine relationships.