A Conceptual Framework for Computer Architecture*

S.S. REDDI

W. W. Gaertner Research, Inc., 1492 High Ridge Road, Stamford, Connecticut 06903

E.A. FEUSTEL

Laboratory for Computer Science and Engineering,
Department of Electrical Engineering, Rice University, Houston, Texas 77005

The purpose of this paper is to describe the concepts, definitions, and ideas of computer architecture and to suggest that architecture can be viewed as composed of three components: physical organization; control and flow of information; and representation, interpretation and transformation of information. This framework can accommodate diverse architectural concepts such as array processing, microprogramming, stack processing and tagged architecture. Architectures of some existing machines are considered and methods of associating architectural concepts with the components are established. Architecture design problems and trade-offs are discussed in terms of the proposed framework.

Keywords and Phrases: computer architecture, framework, composition of architecture, information flow, physical organization, unification of diverse architectural concepts.

CR Categories: 6.0, 6.20, 6.22, 6.29.

INTRODUCTION

Computer architecture is receiving, and will continue to receive special attention as novel architectures differing from the classic von Neumann organization emerge as viable approaches to the problem of increasing the computational speeds and cost-effectiveness of computer systems. Computers such as the CDC 6600, CDC STAR-100, TI ASC, Burroughs B6700, Goodyear STARAN and CRAY-1 are convincing arguments that architecture plays a prominent role in deciding computer system performance and in achieving faster computational speeds than has been previously possible. In the literature there is a multitude of proposals as to how computer architecture can be defined and how an architect's job can be described. Unfortunately, most of these proposed concepts touch only different facets of computer architecture and do not encompass the complete spectrum of architectures. In this paper we present a conceptual viewpoint that allows a coherent and unified treatment of computer architecture. We believe that computer architecture can be viewed as composed of 1) physical organization; 2) control and flow of information; and 3) representation, interpretation and transformation of information, and we develop a framework for architecture based on this viewpoint. We consider some existing com-

* This work was supported by NSF Grant GJ 36471, and was performed while the first author was at Rice University.

CONTENTS

puter architectures and show how to associate architectural concepts and innovations with these three components. We then develop architectural concepts that a system architect can use and verify that these concepts can be accommodated within our framework. Finally, we indicate how our hypothesis can lead to the concept of dynamic system architecture.

EXISTING DEFINITIONS AND INTERPRETATIONS OF COMPUTER ARCHITECTURE

We first consider the various definitions and interpretations of a computer architect's job and of computer architecture as presented in the literature. According to Brooks [8]: "The computer architect designs the external specifications, gross data flow, and gross sequencing of a system. He is, like the building architect, the user's advocate. He must balance the conflicting demands of engineer (cost, speed), programmer (function, ease of use) and marketing (function, speed, cost) to yield the machine of greatest true value to the user. . . ." Foster in *Computer architecture* [17] introduces the architect as follows: "The computer architect in turn is unconcerned with the insides of an adder or a shift register. His job is to assemble the units turned out by the logical designer into a useful, flexible tool that is called a computer." Beizer [6] describes the architect's job as ". . . the design of a hardware/software complex, subject to realistic technical, economic, operational and social constraints such that it 1) works, 2) is optimum and 3) survives." He summarizes the architect's role by stating that "it is synthetical, catalytic and translative. His design is a synthesis of the substances of subordinate disciplines." Abrams and Stein [1] explain the architect's duties: "The job of the computer system architect is to develop an overall concept of a machine—what it can do and how that solves the problem for which the machine is intended. Just as an architect who designs houses must consider utility, appearance, and compatibility with the neighborhood, so must the computer designer balance requirements, user interface, and costs to make a viable design."

Several explicit definitions of architecture also can be found. The term "Architecture" is used by Amdahl, et al., [2] in introducing the IBM System/360 "to describe the attributes of a system as seen by the 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." Foster [17] explains: "The field of computer architecture, or 'the art of designing a machine that will be a pleasure to work with' is only gradually receiving the recognition it deserves. This art, one cannot call it a science, is one step more abstract than that of a logical designer, which in turn is abstracted from the study of electronic circuits." Finally Foster [16] also suggests: "Computer Architecture is the profession of adopting present day technology to the solution of current computing problems and of dreaming about the future of the field in such a way as to influence it for the better."

A FRAMEWORK FOR COMPUTER ARCHITECTURE

The disparities in existing definitions and interpretations of computer architecture are directly traceable to its multifaceted nature. Since computer architecture can be viewed from different perspectives, each individual forms his own notion and interpretation. A FORTRAN user may not perceive significant architectural differences between IBM and CDC computers other than word and core size. On the other hand a system programmer can easily distinguish the architectures of these computers in terms of their operation codes, address formation, and input/output. However, he will find it difficult to distinguish an IBM 370/125 from an IBM 370/155, or a CDC 6600 from a CDC 7600, and will be unable to perceive hardware parallelism that may exist in the central processing unit. Thus architectural details can be transparent or visible depending on the viewer and his level of perception.

Our hypothesis is conditioned by the above observations, as we compare and contrast architectures and architectural features in terms of the following three components. For a valid comparison we should choose commensurate levels, i.e., user level, system programmer level.

1) Physical organization;
2) control and flow of information; and
3) representation, interpretation and transformation of information.

Let us consider some examples. The ILLIAC IV and a conventional von Neumann organization have different architectures; they differ in their physical organization and the way they handle control and flow of information. Tagged architecture [12] differs from von Neumann architecture in representation and interpretation of information. The term "transformation" in the third component refers to the situation where representation and interpretation of information are changed dynamically by the system for user convenience, system efficiency, or for implementing a software system to support privacy and protection. This kind of transformation is to be distinguished from the algorithmic transformation on data undertaken in solving specific problems.

SOME EXISTING COMPUTER ARCHITECTURES

In this section we consider some existing computer systems and explain their architectural features using our three components. These explanations are sufficiently detailed so that the reader may become familiar with the process of associating architectural concepts with the components. In addition we consider microprogramming, an established architectural feature, to reinforce our arguments regarding the validity of our hypothesis.

Physical Organization

Technological advances achieved in the past decade enabled architects to propose many innovative *physical organizations* for computer systems, some of which have already been realized in practice. Three computer systems, the CDC 6600, ILLIAC IV and TI ASC, are studied to show the different organizations that can be conceived. The CDC 6600 is an example of distributed computing and employs an organization which exploits functional parallelism. The ILLIAC IV and TI ASC use array and pipeline organizations respectively for enhanced performance. All three systems depart from the conventional von Neumann machine in their physical system organization.

CDC 6600 Central Processor Organization: The CDC 6600 [32] organization shown in Figure 1 has ten independent functional

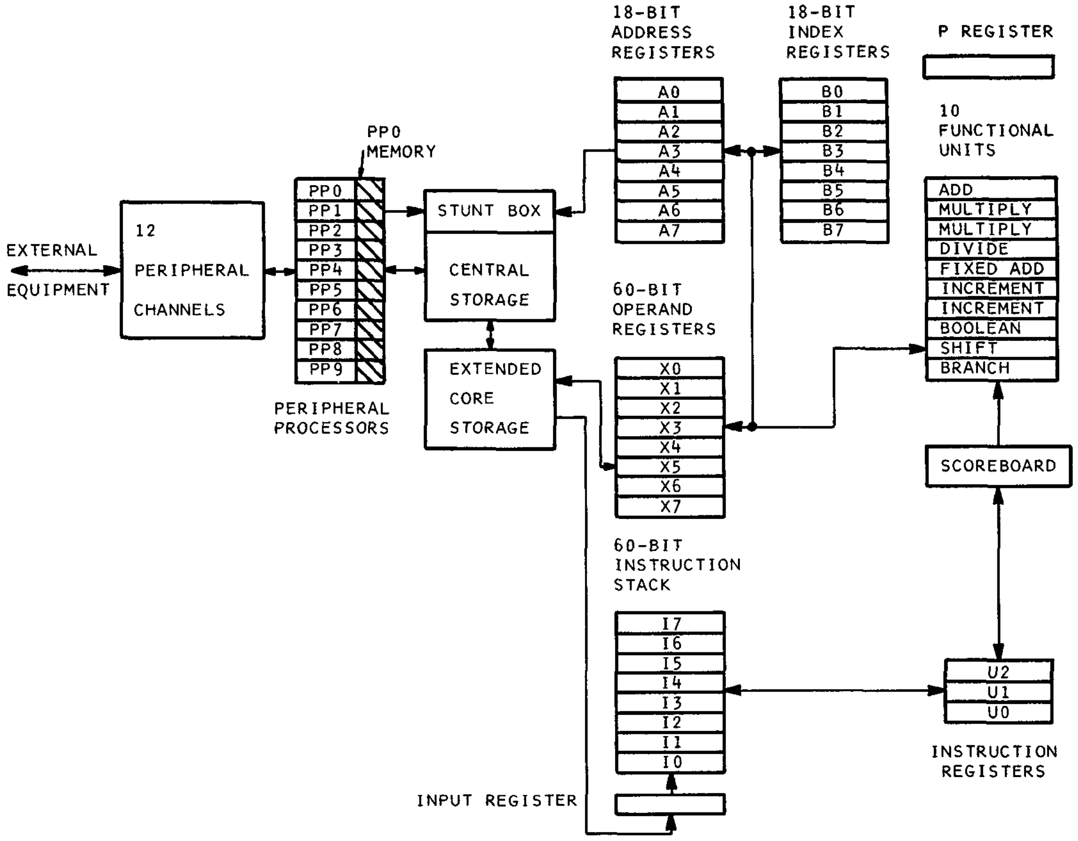


Figure 1. Block diagram of the CDC 6600 computer system, an example of distributed computing architecture.

units, twenty-four data and address registers, an instruction stack and a scoreboard in its central processor. The ten functional units, which are independent of each other, are capable of simultaneous operation. The functional units are specialized to perform operations such as floating add, floating multiply, floating divide, shift, fixed add, etc. The twenty-four registers are divided into data (X_i), address (A_i) and index (B_i) groups; each group consists of eight registers. Except for data registers which are sixty bits long, all registers are eighteen bits long. Data registers X_1 through X_5 are used as "read" registers and X_6 and X_7 as "store" registers. There is a "partner" relationship between X_i and A_i registers. An operand is fetched into X_i, $i = 1$ to 5, from memory by loading its addresses into A_i; similarly an operand is stored from X_i, $i = 6, 7$, into the memory location specified by A_i. The index registers used are to modify the addresses in the address registers. (For instance, there is an instruction which adds A_j and B_k and transfers the result to A_i). The index registers are also used to store fixed-point integers and manipulate floating-point exponents. Registers A_0 and X_0 are used to reference the extended core storage. The partner relationship permits concurrent calculation of arithmetic and address information. All arithmetic computations are performed on operands from the twenty-four registers with results returned to these registers.

Instructions are loaded into the processing unit under the control of P register (see Figure 1). The instruction stack contains eight 60-bit registers capable of holding up to 32 instructions. Instructions are accessed from memory and stored in a last in, first out (LIFO) manner in the instruction stack. When the program executes a branch to one of the instructions stored in the stack, that instruction is directly fetched from the stack rather than from memory. In this way it is possible to cut down the number of memory accesses in the case of loops that can be accommodated in the stack. Instructions pass from the instruction stack to three instruction registers U_0, U_1 and U_2 whose contents are interpreted and decoded by the scoreboard. The scoreboard examines the instruction to be executed and determines whether the functional unit and registers needed for the execution of the instruction are available; if they are available the instruction is initiated. Otherwise, the instruction is held until they are available. The scoreboard maintains the status of each central register and functional unit so that it can decide when an instruction can be issued. The scoreboard provides interlocks between instructions so that though parallelism in instructions is exploited, precedences among instructions as specified by the programmer are still preserved.

The ILLIAC IV System: This system [5] employs an array structure for organizing its processors. It consists of 256 processing elements arranged into four quadrants. Only one quadrant has been constructed because of economic considerations.

A quadrant consists of a control unit and sixty four processing elements. Processing element i is connected to processing elements $i + 1 \pmod{64}$, $i - 1 \pmod{64}$, $i + 8 \pmod{64}$ and $i - 8 \pmod{64}$. The control unit fetches instructions from a processing element memory, decodes them, and issues control pulses to the processing elements for execution. It broadcasts memory addresses and data words when they are common to all processors. An instruction can be either a control or processing unit instruction. The former directs operations local to the control unit whereas the latter controls the execution of the processing units. The control unit is designed to overlap the executions of the two different instruction types.

The processing element consists of the processing element memory (2048 64-bit words), arithmetic and logic circuitry, and registers. Each element executes the common instruction issued by the control unit on data stored in its memory. An element will not execute the common instruction if its enableflip-flop is not set. In other words all the enabled elements execute the same instruction issued by the control unit. Digital Equipment PDP-10 computers perform the executive control of system operation, I/O processing and supervision, and compilation of the ILLIAC IV programs.

The TI ASC System: A principal feature of the Texas Instruments Advanced Scientific Computer (ASC) [31] is the central processor's four pipelines (Figure 2). If computational requirements do not warrant the capacity of four pipelined arithmetic units, one or two pipeline CPs are available. The CP pipeline consists of the instruction processing unit (IPU), the memory buffer unit (MBU), and the arithmetic unit (AU). The IPU supplies a continuous stream of instructions for execution by the other units of the pipeline. It fetches and decodes instructions, accesses and stores operands, and handles branch conditions. The MBU provides an interface between the central memory and AU by supplying a continuous stream of operands to the AU and storing results in the memory. The AU performs arithmetic operations and is pipelined as shown in Figure 2. Depending on the mode of arithmetic operation, the pipeline is structured to divert operand flow along the dotted or solid lines. The central processor is a vector as well as scalar processor and has vector instructions at machine level.

CONTROL AND FLOW OF INFORMATION

Information flow and its control in a computer play a prominent role in deciding the computer architecture. This component encompasses the aspects of control mechanisms and schemes used for controling and directing the system's information flow. Note the interdependence between the control and flow elements. Sometimes control initiates information flow (as in conventional syn-

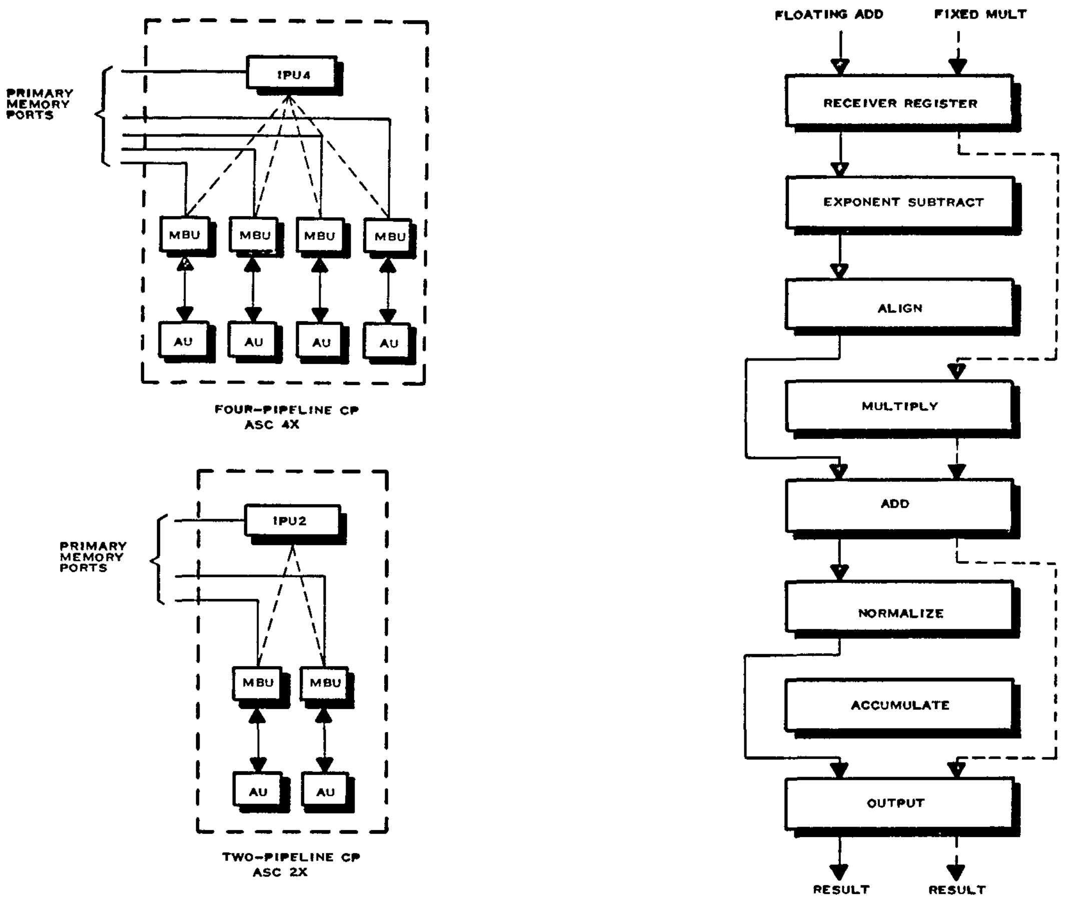


Figure 2. Processing and arithmetic pipelines of the TI/ASC system. (Reprinted, by permission of Texas Instruments Inc., from *A description of the advanced scientific computer system*).

chronous computers where control pulses are issued periodically to direct information flow) and sometimes information flow initiates control (as in interrupt and data driven schemes); thus the control element may be isolated from or incorporated in information flow. Control may be centralized (as in most computer systems) or decentralized (as in a network of microprocessors).

Alteration of information flow in a traditional computer leads to novel and interesting architectures. A familiar example is the Burroughs B6700, a stack processor. In the following we describe briefly the B6700 processor and stack organization and indicate how information flow is affected by the stack.

The Burroughs B6700: An integrated hardware-software approach is taken in designing the B6700. The system is programmed using high level languages (e.g., ALGOL, COBOL and FORTRAN) and does not offer the user a traditional machine level language. The system hardware is designed to handle block structured languages and procedures efficiently by means of a stack mechanism and display registers. One learns to appreciate the system architecture more when one becomes familiar with the aspects of system implementation associated with block structured languages, such as accessing global and local variables, address formation for accessing procedure segments, and nesting of blocks.

Processor Organization

A block diagram of the processor [10] is shown in Figure 3 [11]. A program word is

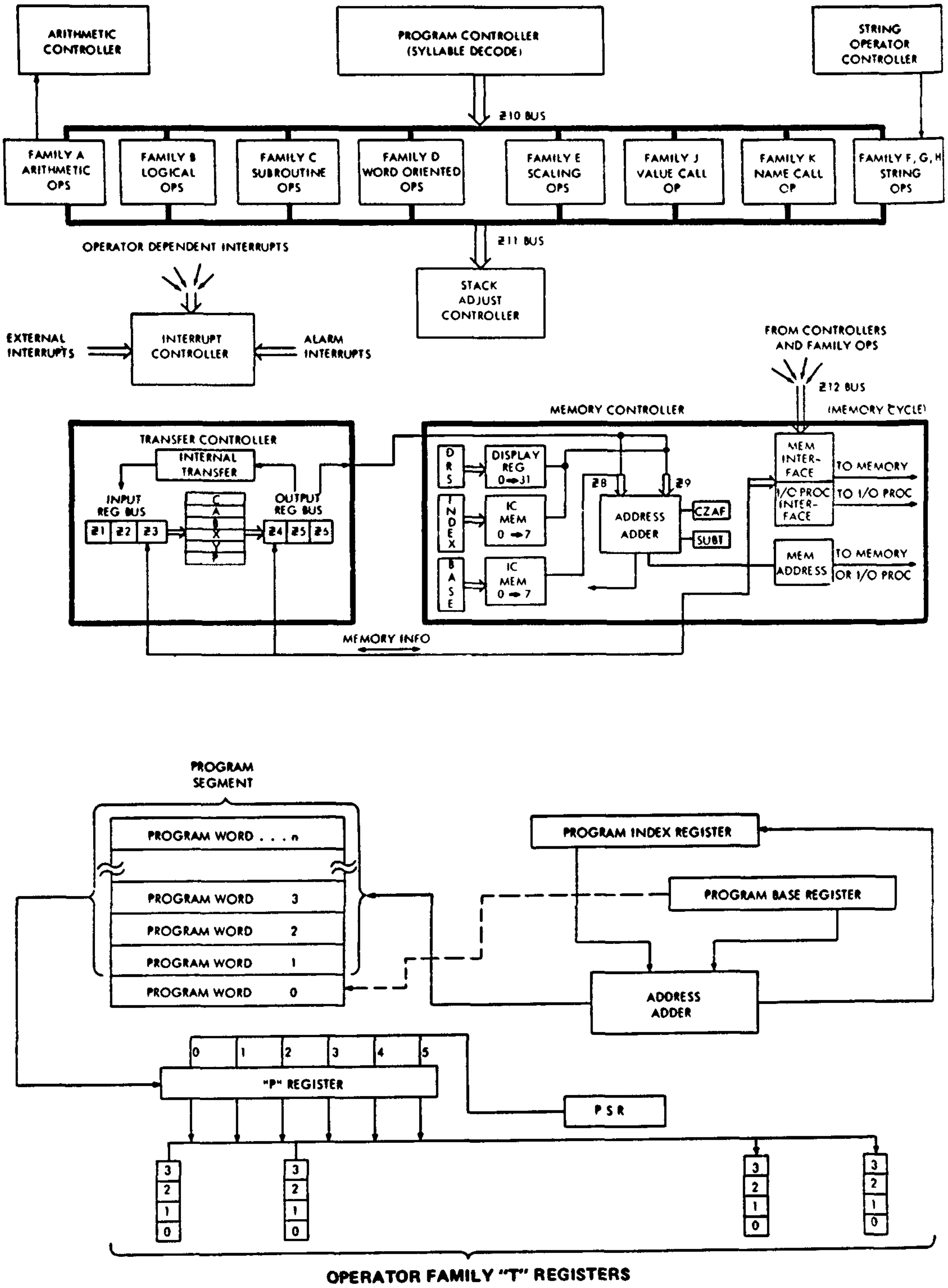


Figure 3. A functional diagram of the Burroughs B6700, a stack processor. (Reprinted, by permission of Burroughs Corporation, from *The B6700 information reference manual 1058633*).

transferred to the *P* register through the memory controler under the control of the program controler. The program controler examines the instruction to be executed and initiates the proper operator family. Related operators are grouped into a single operator family; thus family *A* performs all arithmetic operations, family *B* performs all logic operations, and so on. The arithmetic and string controlers are enabled by their respective family operators; they control and supervise the execution of arithmetic and string operations.

The transfer controler contains registers *A*, *X*, *B* and *Y* necessary for setting up the stack. The interrupt controler provides a method of interrupting the program flow by setting up necessary control words in the stack. The processor takes the appropriate action by examining the control words. The memory controler handles storage requests and contains an adder which is used to generate addresses. The stack controler is responsible for automatic stack adjustment; it manipulates data between main memory and the *A* and *B* registers during the pop-up and push-down operations of the stack.

The machine language operators are composed of eight-bit syllables and each memory word consists of six syllables. The operator syllable pointed to by the program syllable register is decoded, and the operator family specified by the operator syllable is selected to receive the execute signal. The *T* register in each family specifies the operation to be performed in that particular family.

Stack Mechanism

Registers *A* and *B* are used to set up the stack [13] (Figure 4); registers *X* and *Y* are used as extension to *A* and *B* when double precision operands are used. The stack is formed by linking an assigned area of the memory to *A* and *B*. When *A* and *B* are filled up with operands and a third operand is entered into the stack the operand in *B* is pushed into the stack's memory area. Register *S* contains the address of the last word placed into the memory area. The stack memory area of a job is bounded by

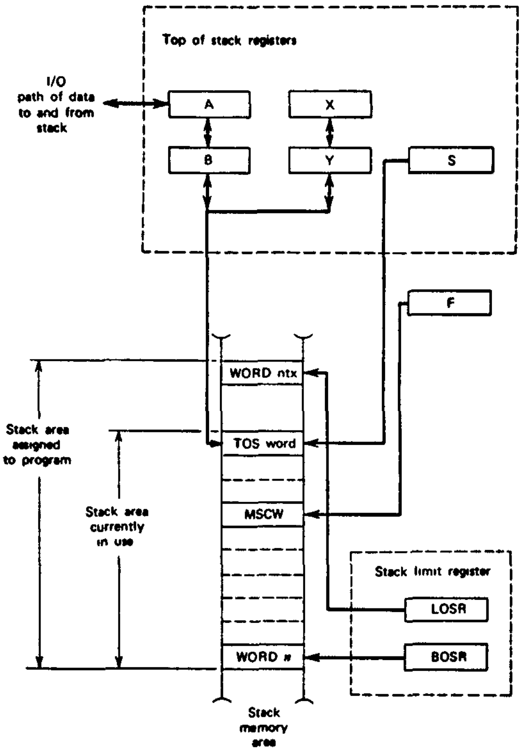


Figure 4. The stack mechanism of the B6700. (Reprinted, by permission of John Wiley and Sons, Inc., from *Multiprocessors and parallel processing*, P.H. Enslow (Ed.).)

the bottom of stack (BOS) and stack limit (SL) registers.

Word formats used in the system are shown in Figure 5 and are determined by the hardware using the leading three tag bits. Data is addressed by means of data descriptors (DD), indirect reference words (IRW) and stuffed indirect reference words (SIRW). Program code is accessed through segment descriptors. In descriptor words the address field contains the absolute address of an array in either main or disc memory depending on whether the presence bit *P* is one or zero. The length field defines the length of the array. IRW and SIRW are used to address data located in the stack memory area of the job.

The stack keeps track of the stack history list and the addressing environment list. The stack history list consists of blocks and procedures contained in the stack and hence is dynamic in nature. The addressing en-

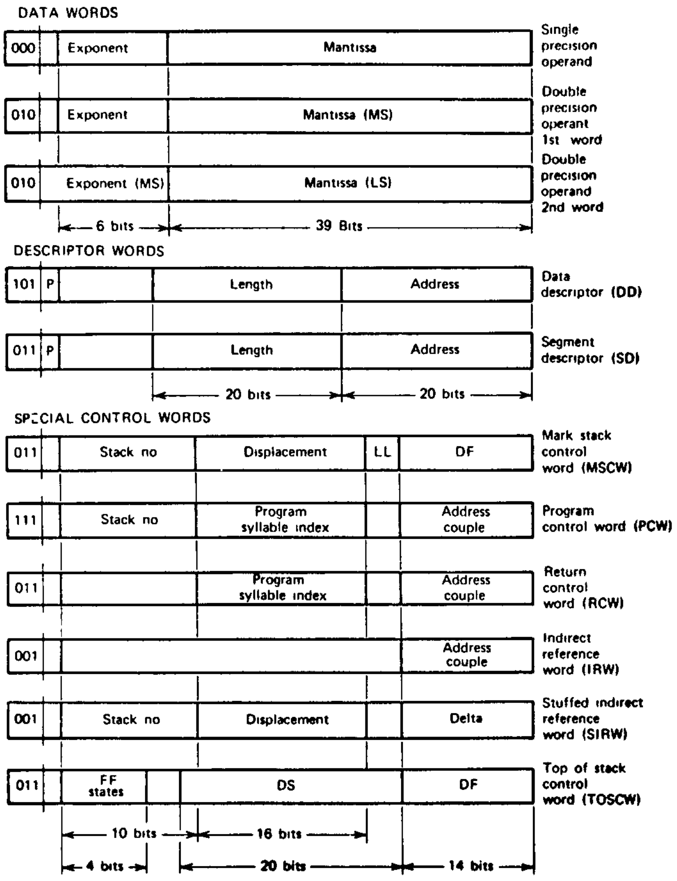
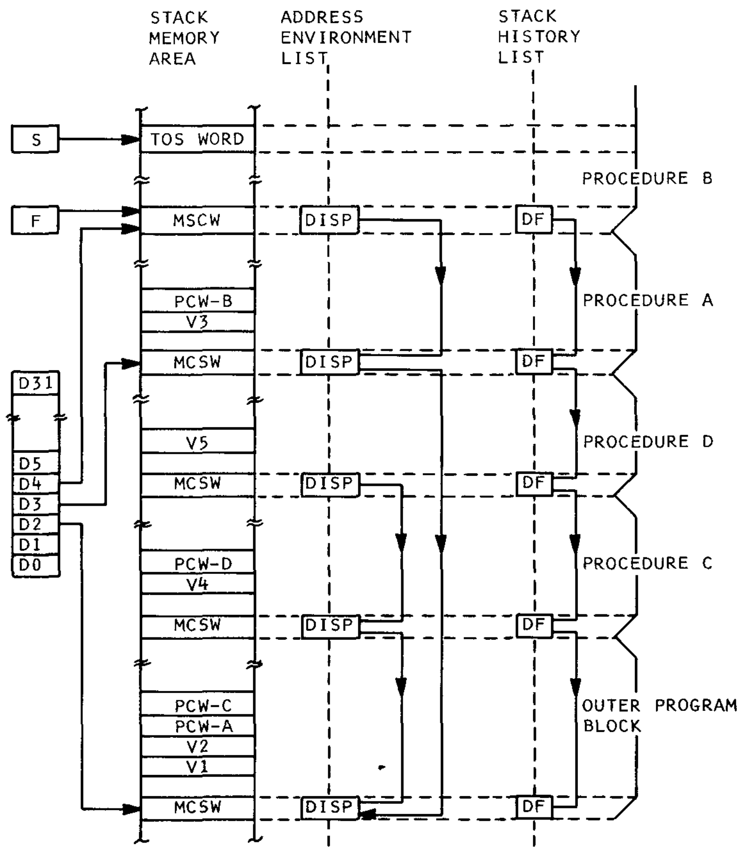

DATA WORDS
000 | Exponent | Mantissa — Single precision operand
010 | Exponent | Mantissa (MS) — Double precision operant 1st word
010 | Exponent (MS) | Mantissa (LS) — Double precision operand 2nd word
6 bits
39 Bits
DESCRIPTOR WORDS
101 | P | Length | Address — Data descriptor (DD)
011 | P | Length | Address — Segment descriptor (SD)
20 bits
20 bits
SPECIAL CONTROL WORDS
011 | Stack no | Displacement | LL | DF — Mark stack control word (MSCW)
111 | Stack no | Program syllable index | Address couple — Program control word (PCW)
011 | Program syllable index | Address couple — Return control word (RCW)
001 | Address couple — Indirect reference word (IRW)
001 | Stack no | Displacement | Delta — Stuffed indirect reference word (SIRW)
011 | FF states | DS | DF — Top of stack control word (TOSCW)
10 bits
16 bits
4 bits
20 bits
14 bits

Figure 5 Control word and data formats of the B6700.
(Reprinted, by permission of John Wiley and Sons, Inc., from *Multiprocessors and parallel processing*, P. H. Enslow (Ed.).)

vironment list enables the system to access local and global variables relative to the block or procedure presently under execution.

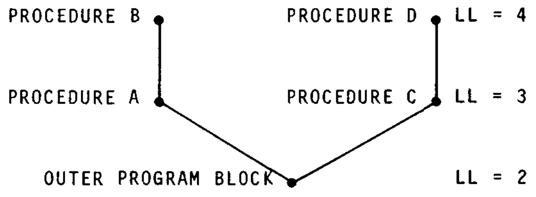
Mark stack control words (*MSCW*) are used to maintain both the stack history and addressing environment lists. Figure 6 shows an example of how an MSCW is entered when a procedure is entered. The parameters and local variables of the procedure are entered following the MSCW and hence are referenced by addressing that is relative to the location of the MSCW. The DF fields of the MSCWs keep track of the stack history whereas the DISP fields maintain the addressing environment list.

The variables are accessed by means of address couples. An address couple consists of the lexicographic level (LL) of the variable and an index value (σ) (Figure 6). When a variable is specified by an address couple, one can deduce from the execution environment and the lexicographic level which block or procedure specifies the variable. The display registers ($D0$ to $D31$) contain the addresses of the MSCWs of the procedures that are linked by the address environment list to the procedure under execution. By adding the index value to the contents of the appropriate display register the absolute address of the variable is generated. As an example let $V1$ be referenced in Procedure B. Since $V1$ is represented by the address couple (2,2) the system obtains the address of the appropriate MSCW from the display register $D2$ and adds 2 to it to obtain the address of $V1$. Note that the display registers must always



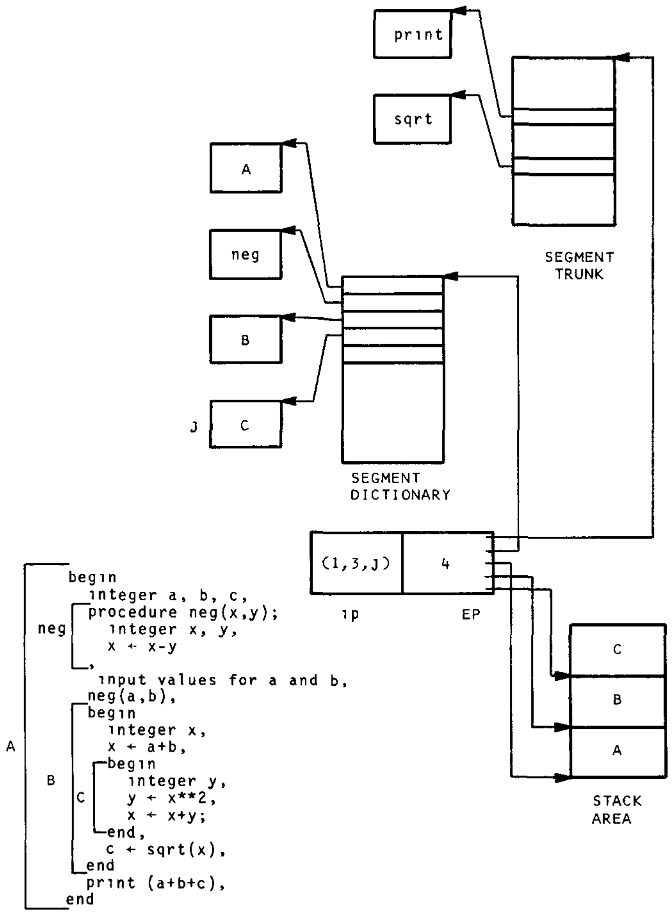
```
┌begin──────────────────────Lexicographical Level 2
│  real V1;                 LL = 2, δ = 2
│  real V2;                 LL = 2, δ = 3
│  procedure A;             LL = 2, δ = 4
│  ┌begin───────────────────Lexicographical Level 3
│  │  real V3,              LL = 3, δ = 2
│  │  procedure B;          LL = 3, δ = 3
│  │  ┌begin────────────────Lexicographical Level 4
│  │  │  V3 ← 3;
│  │  │  V1 ← 3,
│  │  └end,
│  │  B
│  └end,
│  procedure C,             LL = 2, δ = 5
│  ┌begin                   Lexicographical Level 3
│  │  real V4;              LL = 3, δ = 2
│  │  procedure D,          LL = 3, δ = 3
│  │  ┌begin                Lexicographical Level 4
│  │  │  real V5;           LL = 4, δ = 2
│  │  │  V4 ← 4;
│  │  │  V5 ← 5;
│  │  │  A;
│  │  │  V2 ← V4;
│  │  └end;
│  │  D;
│  └end;
│  C;
└end:
```

An ALGOL Program with Lexicographical Levels Indicated.



Addressing Environment Tree for the ALGOL Program.

Figure 6. An example illustrating stack processing capabilities of the B6700. (Adapted, by permission of John Wiley and Sons, Inc., from *Multiprocessors and parallel processing*, P. H. Enslow (Ed.).)



```
begin
  integer a, b, c,
  procedure neg(x,y);
    integer x, y,
    x ← x-y
  ,
  input values for a and b,
  neg(a,b),
  begin
    integer x,
    x ← a+b,
    begin
      integer y,
      y ← x**2,
      x ← x+y;
    end,
    c ← sqrt(x),
  end
  print (a+b+c),
end
```

Figure 7. Program representation in the B6700.

point to the address environment list and have to be changed upon procedure exit or entry to the correct MSCWs.

The stack mechanism also provides the capability of handling several active stacks by organizing them as a tree. It is beyond the scope of the present paper to go into these details. Interested readers should refer to Organick [27].

Representation, Interpretation and Transformation of Information

The information processing capabilities of a computer system depend to a large extent on how the system interprets and represents the information. In this section we consider different types of representation, interpretation, and transformation of information that a computer may use.

Program Representation in the Burroughs B6700: Consider the ALGOL program shown in Figure 7. The system keeps track of the program segments by means of the segment dictionary. The instruction pointer (*IP*) and Environment Pointer (*EP*) provide the physical location of the instruction in the program as well as the environment under which the instruction will be executed. (We discussed in the preceding section how the display registers which constitute the EP provide environmental information and pointers are set up in the stack area for the execution record of the program). The *IP* is a three-tuple of the form either (l, i, j) or $(0, i, j)$. (l, i, j) indicates that the instruction resides in the *j*th location of the *i*th segment in the segment dictionary. (*0, i, j*) means that a "system intrinsic" (i.e., system routine) is addressed. The *j*th location of the *i*th segment in the stack trunk which contains supervisory code segments and system tables is accessed. In the example *print* and *sqrt* are system intrinsics. The segment descriptors indicate whether the segments are in the main memory and how to locate the segments on the disc if they are not in the main memory.

This kind of program representation leads to efficient memory utilization. Since the stack trunk, stack segment and stack dictionary compactly represent the program status and its code requirements, the working sets tend to be small. Also the representation leads to sharing of programs and data. Though two programs may share a common procedure segment and hence have the same instruction pointers, their *EP*s are different. For further discussion of the B6700 and its system description see [27].

Representation and Interpretation of Instructions

Instructions determine the information flow and reflect the structure and capabilities of the system. One of the principal duties of the computer architect is to develop a comprehensive instruction set which is simple to use but exploits the system resources to their fullest extent.

An instruction can be of zero-address, one-address or multi-address format. Zero address instructions are used in computers with stack processing where the use of operands on the top of the stack is implied by the instruction. Single- and multi-address formats are used in computer systems modeled after von Neumann systems. Richards [29] gives an excellent discussion of the various address formats.

Addressing of operands in an instruction can be indexed or indirect. When the addressing is indexed, the addresses of the operands are formed by adding or subtracting the contents of "index" registers to the address parts of the instruction. When the addressing mode is indirect, then the address of the operand can be found in the location specified by the address part of the instruction. The instruction repertory of a computer system includes instructions which allow jump operations and which handle subroutine operations.

The instruction set can be significantly affected by the system architecture. For instance, consider tagged architecture [12] which differentiates integer operands from floating point operands at machine level. In this case it is not necessary to have separate instructions for floating point add and integer add. When an add instruction is issued, the system, by examining the operands involved, decides whether it is a floating point or integer add. Another example of the effect of architecture on the instruction set is the stack processor (e.g., the B6700).

Representation and Interpretation of Data

Knuth [25] explains data as: "representation in a precise, formalized language of some facts or concepts, often numeric or alphabetic values, in a manner which can be manipulated by a computational method." Inevitably, when data represents mathematical concepts or real life situations, relationships between the data elements are bound to exist. Since data is a representation, the precision to which the representation should be expressed becomes a factor. A properly conceived computer system should cope with the problems of relational and precision representations of data.

Floating point numbers can be represented in the IBM 370 system series in short (32 bits), long (64 bits) or extended precision (128 bits) form. The system uses the byte as its basic storage unit which consists of 8 bits. When a floating point instruction like ADR (add long floating point) is to be executed, the system fetches 8 consecutive bytes into the central processing unit (CPU) where the left-most (or the first) byte is the one addressed in the instruction. The system recognizes the operands as floating point, integer, etc., by examining the instruction rather than the operands.

In the Burroughs B6500/7500 organization, data words are distinguished as single precision (48 bits) or double precision (96 bits) operands by attaching 3 tag bits to every word (51 bits). Data may be referenced as an operand (without any qualifications), and the processor knows by examining the tag bits whether the operand is single or double precision. For example, when a command is issued to store an operand on the top of the stack, the word specified by the operand's address is fetched and examined. If the operand is double precision, then the next word is also fetched and stored in the stack. The system recognizes the type of operand, e.g., integer, or floating point integer, or extended precision by examining the instruction. Tag bits also distinguish data words from the program code. Hence, when a job attempts either to execute data as part of the program or to modify the program, an interrupt is issued.

Transformation (or Dynamic Representation) of Information

The representation of information can be static or dynamic. However, a computer may be used to determine dynamically the changes in the representation of information that are needed for user convenience, system efficiency, and privacy. Programs are usually represented at the user level in high level languages. The representation of programs is then changed to machine level languages for execution. Changes of representation are performed for the convenience of the user. (Note that this concept makes it possible to distinguish between systems which use compilers and interpreters for program execution.) Further examples of this type of representational changes include automatic transformation of ASCII characters to EBCDIC by the system.

System efficiency may dictate that different representations of information be used in different situations. Sparse matrices (matrices whose elements are mostly zero) can be economically stored by means of binary patterns and lists of nonzero values. The use of ones in binary patterns indicates that their corresponding matrix elements are nonzero. The values of these elements are obtained by choosing the appropriate values from the list. An economy in storage results because binary patterns can usually be compacted and stored as binary words. When the matrices are not sparse it may be efficient to store them in major order fashion in rows or columns. Another example of where representations may be changed is in the storing and accessing of data elements. The elements may be stored and accessed by a table look-up or by hash coding techniques depending on the application.

Privacy considerations may also warrant making changes in the representations of information. Consider the system where a common data bank has to be shared by different users and each user is authorized to access only some portions of the data. The system may encode the data supplied by the user and store the encoded data in the bank. The conversions of encoding or privacy transformation are performed to ensure that only authorized users can gain access to a data set. When the user supplies the proper identification the system decodes and presents the requested data to him. Privacy transformations are discussed in [20].

Physical Organization and Control and Information Flow

It is sometimes necessary to create new control paths among the physical resources of a computer system to exploit the parallelism that is present in hardware and programs and to increase the system's performance. In such cases both the physical and control elements contribute to the desired objective. A typical application of this approach may be seen in the overlapped operation of I/O and processor computation found in most contemporary computer systems. Sometimes information flow is controled to exhibit to the user a machine architecture that is not real. An example of this is the compatibility feature found in the IBM 370 series. We examine selected architectural features of the IBM 370 series and the CDC 6600 computers and indicate how these features can be explained as a combination of physical organization and control of information flow.

IBM System/370 Series. Let us now

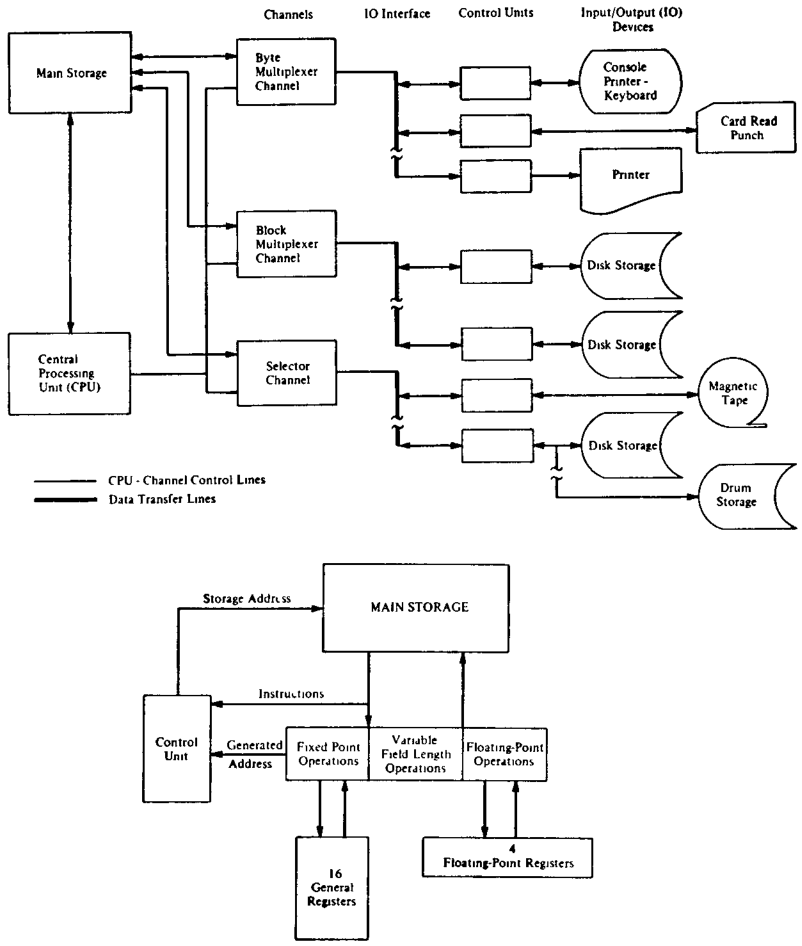


Figure 8. Architecture of the IBM System/370.
(Reprinted, by permission of IBM Corporation, from Harry Katzan, Jr., *Computer organization and the System/370.*)

consider some salient architectural features of the IBM 370 series, [24] the main feature of which is the compatibility between the various models in the series. Though the models vary in their designs, performance indices, and prices, they exhibit the same architecture to the user. This is accomplished by providing the same instruction set and employing microprogramming in all but the largest models. An advantage of this feature is that the user is offered models of varying performance indices and storage capacities. If the user finds that a more (less) powerful system is needed than the one currently being used, he can switch to a higher (lower) numbered model without reprogramming. A disadvantage is that the models lose some of their performance in maintaining compatibility. Small models use much of their memory in providing compatible software and are burdened by sophisticated I/O features suitable for larger models. Large models can be operated more efficiently if the downward compatibility feature is not required [13].

Basic CPU Organization

The user sees the architecture of the system as in Figure 8. The central processor consists of sixteen general registers and four floating point registers. The general registers can be used to hold operands or as index and base address registers. The floating point

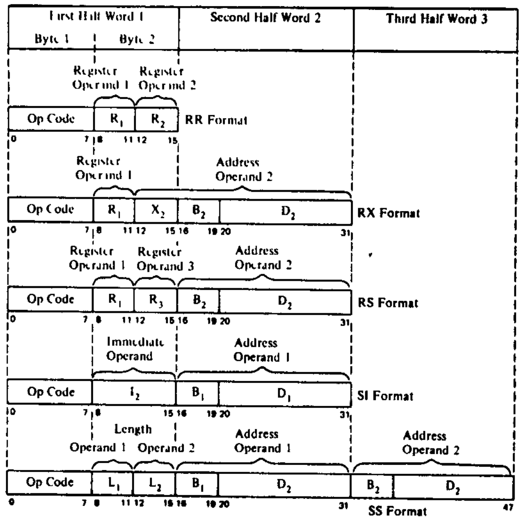


Figure 9. Instruction formats of the IBM/370. (Reprinted, by permission of IBM Corporation, from Harry Katzan, Jr., *Computer organization and the System/370.*)

registers hold floating point quantities. The registers reduce the number of memory accesses for data by storing temporary operands; this reduction in memory accesses in turn reduces conflicts for memory by the I/O and CPU units. The system has a program status word (PSW) register whose contents indicate the status of the program under execution; this enables the system to handle interrupts and multiprogramming.

Each model has a different engineering design. For instance, the simplest model 125 does not provide any hardware adders whereas model 165 has an address adder, a parallel adder, and serial adder. The instruction formats for the system are shown in Figure 9. They specify the contents of registers and/or memory locations as operands. The instruction execution appears to the user as sequential; however high performance models employ overlapping of instruction and operand fetching with instruction execution, and prefetching instructions along both paths of a branch. The system hardware cannot recognize structured operands (e.g., vectors and matrices) and it is up to the programmer to make the system recognize such operands by programming.

I/O Handling

The responsibility of I/O handling is shared by the control unit of the CPU and I/O channels. The channels have their own registers and are capable of performing data transfers between memory and I/O devices. The CPU specifies in its I/O commands where the channels can find their commands in the main storage. The I/O requests for memory are given top priority (i.e., the cycle stealing technique is used). A significant feature of the I/O organization is that the user can configure the system by attaching or removing I/O devices.

Memory Organization

There is no one fixed memory organization for all the models. Models 155 and 165 provide 4K buffer storage systems in addition to their main storage units. The buffer and main storage are organized into rows and columns. At the intersection of each column and row there is a block of 32 bytes, i.e., the storages are partitioned into blocks of 32 bytes and each block can be specified by a row and column. (A byte consists of 8 bits.) An address array maintains the addresses of the elements in the buffer. When the CPU makes a storage reference, the address array is consulted to determine whether the referenced element is in the buffer. If the element is present it is sent to the CPU; otherwise the element is fetched from the main storage and dispatched to the CPU. Then the block containing the element is stored in the buffer in the following manner: Blocks are transferred from the main storage to the buffer columnwise, i.e., a block in column *i* of the main memory is transferred to column *i* of the buffer. The block that is to be stored in the buffer replaces the least recently used block in its column. Model 165 also uses interleaving for its memory organization.

The storage system provides a protection feature which can be used in multiprogramming. The feature is implemented by dividing the main store into blocks (of 2048 bytes) and assigning storage keys to the blocks. Each active program has a protection key associated with it. Usually the operating system assigns the protection keys to the programs. A program can store in a block only when the protection key of the program matches the storage key of the block or the protection key is zero. The storage operation is inhibited and an alarm signal is given if the keys do not match and the protection key is not zero. The storage key has an extra bit which protects fetch operation. If the bit is zero, only store operation is protected. Otherwise both store and fetch are protected.

CDC 6600 Memory Organization. The CDC 6600 memory hierarchy consists of a fast central storage and slow extended core storage (ECS) [32]. The central storage of 131,072 60-bit words is composed of 32 independent banks. The banks are interleaved to provide high block transfer rates. The computer has two cycles, major (1000 nanoseconds (nsec) and minor (100 nsec). The storage read and store cycle take one major cycle, whereas transferring a data word through the storage distribution system takes one minor cycle. There is a mechanism called the stunt box which examines the requests and directs information flow in and out of the central storage. When the stunt box accepts a new access request it decides whether the bank requested is busy or free; if the bank is free the read and store cycle is initiated. If the bank is busy, the address requested is circulated within the stunt box. The stunt box can hold three circulating addresses and each circulation takes 300 nsec. Top priority is given to the addresses in circulation for access to the storage. Because of the circulation time (300 nsec) and the major cycle time (1000 nsec) the mechanism prevents permanent recirculation of any request. In case of consecutive requests to the same bank the requests are satisfied after at most two major cycles.

The stunt box is also responsible for attaching priorities to requests coming from the central processor unit and peripheral processing units. It prevents the situation where in the recirculating addresses read and write requests are made to the same storage location. This is because of the stunt box's out-of-order recirculation properties. It also checks whether the requested address violates bounds. The storage distribution system is responsible for transferring requests and data to and from the central storage.

The secondary storage consists of 15,744 488-bit word core memory. The 488-bit words (8 of which are parity bits) are disassembled to 60-bit words used in the central storage. The CPU can transfer any number of 60-bit words between the central store and ECS by simple commands. The major advantage of the ECS is that it can transfer blocks of information at a rate of 60 million bits/second. It may be directly addressed but at a considerably slower rate. Thus, its principal use is as a high speed buffer.

CDC 6600 I/O Handling. Ten peripheral processing units (PPU) handle I/O activities. Each PPU consists of four registers and a storage unit of 4096 12-bit words. The processors are arranged in the form of a "barrel". The barrel has ten positions and each position is occupied by a PPU. There is one position called "slot" which is capable of accessing and utilizing arithmetic and logic hardware; the ten PPU's share the slot by circulating the contents of their four registers. When a PPU is in the slot it stays there for 100 nsec and uses the arithmetic and logic hardware to execute the program stored in its storage unit. A PPU instruction requires one or more steps of execution with each step taking 1000 nsec. It can be noted that the central storage read and store cycle takes 1000 nsec which is the time interval between consecutive sharing of the slot by any PPU. The PPU can transfer data between peripheral devices and main memory and supervise the operation of the devices. The PPU have the capability of establishing paths to I/O devices through twelve peripheral channels. A PPU can interrupt the operation of the central processor by means of an exchange jump. When an exchange jump is issued, the CPU makes an exchange between the contents of its 24 registers and the contents of the "exchange package" which starts at a location in the central storage specified by the PPU. The exchange package consists of 16 words and specifies the new contents of the 24 central registers. Once the exchange is made the CPU starts on the new program specified by the program address register (note this register is one of the central registers). A PPU is also capable of monitoring the CPU by transferring the contents of the CPU program address register to one of its registers. The CPU can also initiate the exchange jump.

Physical Organization, Control of Information Flow and Representation and Interpretation of Information

Now we consider the architectural feature of microprogramming which can only be explained when all three components of architecture are used. In the literature this feature is usually associated with system architecture. The reason for this association is that microprogramming is able to present to the user an architecture that is not a real machine architecture.

Microprogramming: Husson [21] proposes the following definition: "Microprogramming is a technique for designing and implementing the control function of a data processing system as a sequence of control signals, to interpret fixed or dynamically changeable data processing functions. These control signals, organized on a word basis and stored in a fixed or dynamically changeable control memory, represent the states of the signals which control the flow of information between the executing functions and the orderly transition between these signal states."

A basic microprogramming scheme [33] is shown in Figure 10. Register I contains an address which is decoded by the decoder (D). The horizontal line in the read only memory (ROM) that corresponds to the address is activated and issues signals. The signals under Matrix *A* control the data paths of the arithmetic units, registers, etc., of the computer system. The signals of Matrix *B* specify the next address to be decoded and are forwarded to Register II. Conditional jumps can be handled as shown at *X*. A flip-flop whose state can be controled

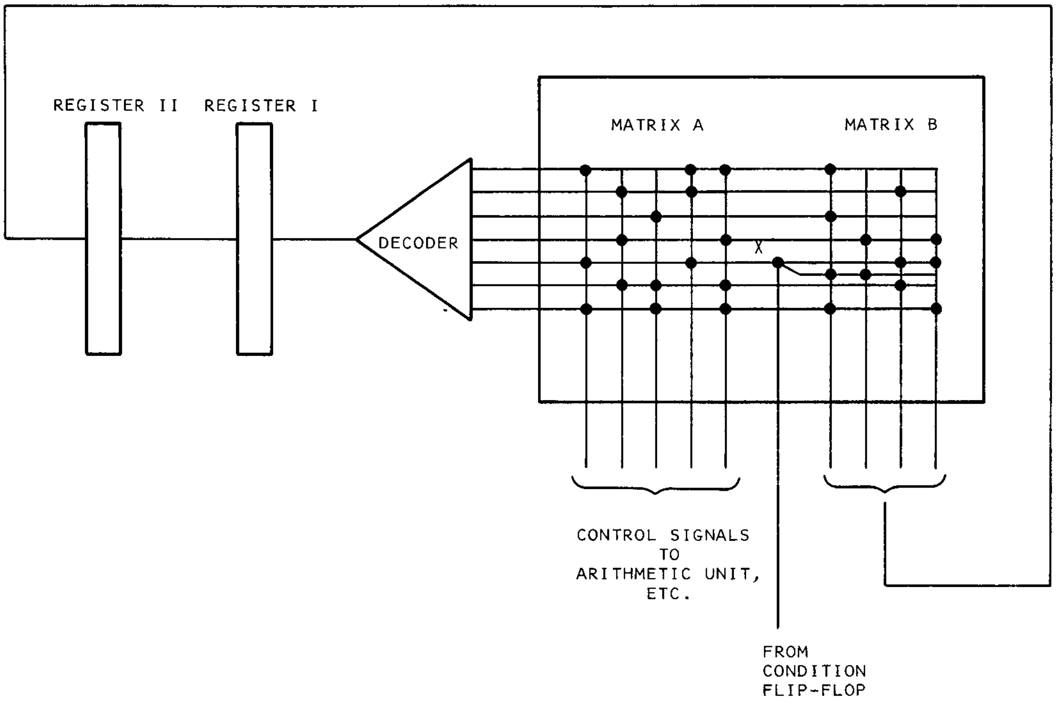


Figure 10. A simple elementary microprogramming scheme.

by the previous orders issued by Matrix *A* decides which of the two lines in Matrix *B* is to be energized.

The signals issued when any line of the ROM is activated form a microorder. The format used for microorders can be either horizontal or vertical depending on how the orders are interpreted. In a horizontal format, each signal under Matrix *A* directly controls a gated data path. In a vertical format, the signals are organized into fields and each field controls the operations of a particular section (like an adder) of the computer system. In this format, encoding of signals is performed and hence horizontally formatted microorders are usually longer than vertically formatted ones. Vertical format microorders sometimes resemble machine language instructions in that they have operand and address fields. Maximal parallelism at hardware level can be exploited by using horizontal format microorders, but generating these orders can be cumbersome and time consuming.

Microprogramming has been used in widely differing contexts. For its applications the interested reader should refer to Flynn and Rosin [15]. Present day large systems like the CDC 6600/7600 and IBM 360/195 do not use microprogramming for their control units. It appears that microprogramming is used in practice, not for its systematic implementation of the control section, but for its ability to offer emulation capabilities. It is interesting however, to note that microprogramming is used to implement the control of the streaming unit of the CDC STAR 100 [23].

ARCHITECTURAL CONCEPTS AND CONSIDERATIONS

In this section we discuss the advantages and disadvantages of some architectural concepts. At first view they may appear to be totally unrelated to each other; however a little thought will reveal that each of these concepts can be categorized under one or a combination of the three components of architecture. Thus, a framework based on our proposal that architecture is composed of these three components can accommodate these seemingly unrelated and diverse concepts. We conclude by considering some of the problems and trade-offs an architect faces in implementing these concepts and in evolving an architecture.

Array Organization

In this organization identical processors are connected in an array fashion. The ILLIAC IV is a familiar example of this type of organization. The ILLIAC IV operates in a single instruction stream—multiple data stream mode (SIMD) [14], i.e., at any time all the enabled processors execute a single instruction (issued by a single control unit) on different data; the processors that are not enabled do not execute the instruction. However with suitable operating systems, it should be possible for array processors to handle the multiple instruction stream—multiple data stream mode of operation.

The array organization is very effective in exploiting parallelism when the characteristics of the problem to be solved match the physical structure. Matrix operations provide an example of this kind of problem. When all the processors are identical, manufacturing and maintenance are greatly simplified. A disadvantage of the array organization is the poor utilization of resources that may result when the problem structure does not match the physical structure. The failure of a single processing element can hamper the operation of the entire system; a sophisticated system could, however, create new and alternate data paths for continued operation of the system.

Pipeline Organization

This organization consists of functional units arranged in a pipeline where each functional unit handles a particular task. It is often used in commercial computer systems to improve system performance. Examples of this organization include instruction handling in the IBM 360/91 and the arithmetic pipeline units in the TI ASC [31] and CDC STAR [19] systems. It is well suited to handling job streams where all the jobs go through the same processing stages. Most vector operations can, for example, be operated in this manner. Pipeline organization loses its efficiency when some jobs require a processing sequence different from that of the pipeline. Job dependencies adversely affect the job flow and hence the efficiency of this organization. Since the processing of jobs becomes "diffused"—at any instant the pipeline contains jobs at different levels of completion—interrupts and machine malfunctions cannot be handled satisfactorily. For instance, the architecture of IBM 360/91 has to settle for what is referred to as an "imprecise interrupt" [3].

Modular Organization

This organization consists of independent functional units (capable of performing specialized tasks) and/or processors (capable of performing any task). Tasks, when they are ready, are dispatched to the appropriate functional units or processors (usually by the supervisor of the organization). The central processing unit of a CDC 6600 employs a modular organization in which there are ten independent modules. In the SYMBOL system, function modules are dedicated to perform portions of the computing process such as translation, memory control, garbage collection, central processor, or other processes. In contrast to array and pipeline organizations the modular organization usually has a variable structure. The supervisor of the modular organization can, by establishing appropriate data flow paths, simulate any particular structure (e.g., a pipeline or an array).

An advantage of this type of organization is the enhanced performance obtainable by using overlap and distributed function computation. The organization can ensure graceful degradation of performance in case of system component failures. Graceful degradation is achieved by having multiple function modules of the same type; when a module fails, its task can be assigned to another module. On the other hand, the supervisory system for such an organization tends to be complex because it has the additional responsibility of properly dispatching tasks and ensuring correct execution of the program (by preserving task precedences). The lack of structure in this organization can increase the overhead of dispatching tasks. The processing of jobs is "diffused" as in the pipeline organization.

Stack Processing

In this type of processing, information flow between central registers is controled in a such way that a pushdown store (or a stack) is realized [7]. New operands, which are entered into the top register of the stack, cause a "pushdown" action to occur, i.e., the contents of each register move down by one register level. Binary operations can be performed on the top two registers with the result being returned to the top register. The contents of the top register can be stored in main memory. The Burroughs B6500 and English Electric KDF-9 employ stack processing.

The following discussion of advantages and disadvantages of stack processing is based on Brooks [7]. Stack processing minimizes main memory data references when evaluating algebraic expressions. With stack processing, shorter program representation is possible as most operand addresses can be eliminated. It simplifies subroutine management and compilation of source programs, especially those programs with recursive definitions. Stack processing makes it easier to handle block structured languages like ALGOL. However, this type of processing is helpful only if the items that are to be processed can be made to "surface" to the top of the stack. A further disadvantage is that many stacks, such as a stack for control and a stack for data, are often needed for satisfactory operation. When variable length fields are used, stack registers must be of variable length to accommodate the values selected from these fields. This often proves to be difficult to implement.

Virtual Memory

By automatic control of information flow between the main and secondary memories, a system with virtual memory [11] gives the programmer an illusion of operating with a main memory that is larger in capacity than the actual memory. This is accomplished by dividing the address space into blocks of contiguous addresses and storing them in both the main and secondary memories. When the programmer makes a reference to an item not present in the main memory, the computer system automatically transfers the block containing the referred item from the secondary to the primary memory. The new incoming block will displace a resident block according to some fixed rule if the main memory cannot accommodate the new block. When the blocks are of variable size, one has "segmentation;" when they are of fixed size, the situation is referred to as "paging."

The principal advantage of virtual memory is that the user can be indifferent to main memory limitations in his programming. He need not concern himself with the problems of overlays and memory management. The large address space provided by virtual memory also simplifies multiprogramming. On the other hand, efficient utilization of the main memory is not always possible. Paged systems round up storage requests to the nearest integral number of pages and this sometimes causes appreciable loss of the main memory ("fragmentation"). Multiprogrammed systems sometimes exhibit performance degradation which is due to a phenomenon known as "thrashing" [11].

Virtual Machines

By means of hardware and software control of information flow a single computer system presents to the users multiple exact copies of the system. Each user is given the illusion that he has the complete computer system at his disposal. As an example the IBM's VM/370 offers the user a virtual IBM 370 system on which he can run any system/370 or system/360 operating system. The virtual machine, of course, runs several times slower than the real machine. The appearance of multiple copies of the basic machine is handled by the virtual machine monitor which interfaces the user's operating system and the real machine. For details concerning the implementation of the monitor, refer to Madnick and Donovan [26]. An advantage of virtual machines is that the users can run different operating systems on the same real machine at the same time. On the negative side, the virtual machine is several times slower because there is overhead associated with the monitor.

Parallel Processing

In this type of processing, the performance of a computer system is increased by introducing control and data paths among its hardware resources. For our purposes we consider parallel processing at bit and task levels. We follow the model of Shore [30] for bit level processing. Figure 11 shows a system which consists of a data memory (DM), an instruction memory (IM) and a control unit (CU). In the DM, words are stored horizontally. A bit (word) slice is any set of bits exposed by a single vertical (horizontal) cut through the DM. The word slice processing unit (WSPU) can operate on word slices whereas the bit slice processing unit (BSPU) operates on the bit slices. In Shore's terminology, Machine I refers to the system with only word slice processing capabilities, Machine II refers to the system with only bit slice processing capabilities and Machine III has both of the processing capabilities. (There are also Machines IV, V and VI which are best considered at task level.) It is interesting to note that Machine I is a conventional sequential processor and Machine II is a bit serial associative processor. Shore's scheme does not fit Flynn's classification [14]. Shore states: "In terms of a taxonomy introduced by Flynn, it is often stated that Machine II is a single-instruction-stream, multiple-data-stream processor whereas Machine I is not. In fact, they both are. Machine I processes multiple-bit-streams a word slice at a time, whereas Machine II processes multiple-word-streams a bit slice at a time. The myopic association of multiple-data-streams with multiple-word streams is a conceptual error having nothing to do with computing power."

Shore considers the ratio of processing

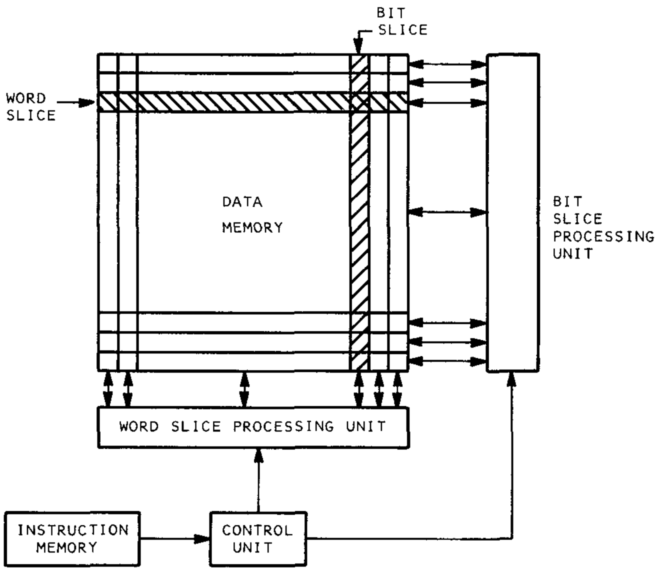


Figure 11. A bit-level processing computer system.

hardware to memory hardware for evaluating the effectiveness of his machines. As can be noted, the ratio also reflects the effect of creating information flow paths by means of physical organization.

At task level there are many approaches to parallel processing. One approach consists of the functional decomposition of tasks and the dispatching to independent functional units which are specialized to execute them (e.g., the CPUs of the CDC 6600 and IBM 360/91). In another approach, the functional or processing units have a fixed operand and control routing structure (like a pipeline or an array) imposed on them. When the system consists of equally capable processing systems, its mode of operation can be characterized by single instruction-multiple data streams (e.g., the ILLIAC IV and PEPE) or multiple instruction-multiple data streams [14].

Central to parallel processing is the problem of recognition of parallelism in programs and task scheduling to achieve maximal concurrency. Extensive work has been done and is continuing in these problem areas. Baer [4] gives a good survey of the work done.

Tagging of Information

Iliffe [22] in his proposal of a basic language machine (BLM) suggests tagging of data and address descriptions for identification at machine level. From the tags associated with data, it is possible to recognize their precision and type (floating or fixed point, etc.). Addresses can be specified by "code words". A code word can specify a block of contiguous words which starts at the location given by the address part of the code word. The length of the block is also specified by the code word. It is possible that a code word can specify a set which consists of code words and data. It can be noted that structural data representations can be easily handled by the code word scheme.

In the BLM tags are also used to implement "escape actions." Whenever the machine encounters an operand whose tag specifies an escape action, the machine interrupts and follows the appropriate action. Numerical overflows, invalid addresses and unauthorized storage accesses can be handled by escape actions.

Iliffe claims the following advantages for his BLM. The machine can recognize the information structure of a program at machine level; this increases the versatility of the machine. Because of the use of code-words, the linear store structure of a conventional computer system is avoided. This is helpful in multiprogramming storage allocation schemes. Since data are identified by tags, instructions need not specify the data types; a single add instruction is sufficient to specify every add type. This results in a smaller instruction set. Also one can detect "mixed arithmetic errors" (such as addition of a floating point number to an integer) simply by examining the tags. The BLM also has certain disadvantages. In a linear store every item can be addressed directly, whereas extra store accesses may have to be made in data structures composed of code words. Overhead is associated with memory allocation because of the data structure involved. For further discussions of tagged architecture, see Feustel [12].

Emulation

Emulation is a combined hardware-software approach to the process of modeling the physical behavior of one machine on another [21]. A host machine *A* can be made to emulate a target machine *B* with the aid of microprogramming. This means that *A* can interpret and execute the machine program written for *B* by means of microprogramming. As an example, an emulator is available which makes it possible to emulate the IBM 7080 on the IBM 360/65. The emulator considers the machine instruction of the IBM 7080 and performs necessary storage mapping conversions; it interprets and translates the instruction into a 360/65 machine instruction. Then the host machine executes the instruction.

There are many advantages to emulation. When the user changes computer systems he does not have to reprogram if he can emulate his old system on the new one. Emulation leads to compatibility, a principal feature of the IBM 370. A disadvantage of emulation is that it is inherently slow and does not fully utilize the resources of the host machine.

Developing an Architecture

Now we briefly consider some of the problems and trade-offs an architect faces in evolving an architecture. These considerations are discussed within the framework of the three components of architecture. Assume that the architect decides to make the computer system provide the capacity of ten processing units. This decision can be implemented either by replicating processing units (physical organization) or by time multiplexing a single processing unit (control and flow of information). The first approach, which is expensive, provides higher performance and graceful performance degradation in case of processing unit failures. The second is more economical but might require a sophisticated control.

Consider a system in a list processing environment. The architect wishes to provide the capabilities of linked data structures. He may design a conventional system without any list processing capabilities and leave the task of handling data structures to the system programmer (control and flow of information). Alternately, the architect can provide data structure capabilities at machine level itself by making appropriate provisions at the hardware level (representation and interpretation of information). To improve the reliability of data transmission links the architect may either resort to replication and major voting or better technology (physical organization) or incorporate parity bits to the words transmitted (representation and interpretation of information). Similarly, the reliability of adders can be improved by replication or by coding the operands (e.g., AN coding [9]). Thus the architectural problems and decisions involved in implementing an architecture can be viewed in terms of the three components of architecture.

Dynamic Architectures

The computer user is becoming increasingly aware of the effect of architecture on system performance. He realizes that the array organization is ideal for solving relaxation problems, that the pipeline organization is effective in handling matrix and vector operations, and that stack processing makes it easier to compile and execute ALGOL programs. Since no single architecture can satisfy the needs of all users, it has become desirable to have a computer system whose architecture can be defined and varied dynamically.

At present, emulation is the main principle used to offer variable architectures to the user. But emulation is inherently slow and inefficient and would defeat our purpose, which is to speed up computation with dynamic architecture. Using our three component approach to architecture, it is possible to conceive a system with dynamic organization. The user can specify the architecture he needs in terms of the three components, and the system will exhibit this architecture by introducing appropriate changes in its control and data paths and by altering its representation and interpretation of information. The speed requirements dictate that these changes be executed at hardware level. The authors [28] propose a system where it is possible to structure system resources as a pipeline, an array, or in any configuration the user may want. Structuring is accomplished by dynamically establishing bus paths between the resources. Thus the physical element of architecture is 'altered' by suitable control of information flow. Similarly, the other components of architecture can be altered. For instance, information flow can be controled to exhibit a stack or nonstack structure depending on the program environment. By attaching tags to operands and interpreting them dynamically, we can obtain an architecture in which the third component is a variable.

REFERENCES

[1] ABRAMS, M. D.; AND STEIN, P. G. *Computer hardware and software, an interdisciplinary introduction,* Addison-Wesley, Reading, Mass., 1973.

[2] AMDAHL, G. M.; BLAAUW, G. A.; AND BROOKS, F P , JR , "Architecture of the IBM System/360," *IBM J. R & D.* (April 1964), 87-101.

[3] ANDERSON, D. W.; SPARACIO, F. J.; AND TOMASULO, R. M. "The IBM System/360 Model 91: machine philosophy and instruction handling," *IBM J. R. & D.* **11,** 1, (Jan. 1967), 8-24.

[4] BAER, J. L. "A survey of some theoretical aspects of multiprocessing," *Computing Surveys* **5,** 1 (March 1973), 31-80.

[5] BARNES, G. H. et al, "The ILLIAC IV computer," *IEEE Trans. Computers* (August 1968), 746-757

[6] BEIZER, B. *The architecture and engineering of digital computer complexes,* Vols. 1 and 2, Plenum Press, New York, 1971.

[7] BROOKS, F. P., JR., "Recent developments in computer organization," in *Advances in electronic and electron physics,* Vol. 18, Academic Press, New York, 1963, pp. 45-65.

[8] BROOKS, F. P , JR., "The future of computer architecture," in *Proc. IFIP Congress* 65, Vol. 1, Spartan Book Co., Washington, D.C., 1965, pp. 87-91.

[9] BROWN, D. T. "Error detecting and correcting binary codes for arithmetic operations," *IEEE Trans. Electronic Computers* (Sept. 1960), 333-337.

[10] BURROUGHS CORPORATION, *Burroughs B 6700 information processing systems reference manual,* Burroughs Corp., Detroit, Michigan, 1972.

[11] DENNING, P. J. "Virtual memory," *Computing Surveys* **2,** 3 (Sept. 1970), 153-189.

[12] FEUSTEL, E. A. "On the advantages of tagged architecture," *IEEE Trans. Computers* (July 1973), 644-656.

[13] FLORES, I. *Computer organization,* Prentice-Hall, Englewood Cliffs, N.J., 1969.

[14] FLYNN, M. J. "Very high-speed computing systems," in *Proc. of IEEE,* 1966, IEEE, New York, 1966, pp. 1901-1909.

[15] FLYNN, M. J.; AND ROSIN, R. F. "Microprogramming: an introduction and a viewpoint," *IEEE Trans. Computers* (July 1971), 727-731.

[16] FOSTER, C. C. "Computer architecture," *IEEE Trans. Computers,* (March 1972), 19.

[17] FOSTER, C. C *Computer architecture,* Van Nostrand Reinhold Company, New York, 1970.

[18] HAUCK, E. A.; AND DENT, B. A. "Burroughs' B6500/B7500 stack mechanism," in *AFIPS Spring Jt. Computer Conf.,* 1968, Thompson Book Co., Washington, D.C., pp. 245-251.

[19] HINTZ, R. G.; AND TATE, D. P. "Control Data STAR-100 processor design," in *COMPCON 72 Sixth Annual IEEE Comp. Soc. Internatl. Conf.,* IEEE, New York, 1972, pp. 1-4.

[20] HOFFMAN, L. (Ed) *Security and privacy in computer systems,* Melville Publ. Co., Los Angeles, Calif , 1973.

[21] HUSSON, S. S. *Microprogramming: principles and practice,* Prentice-Hall, Englewood Cliffs, N.J , 1970.

[22] ILIFFE, J. K. *Basic machine principles,* (2d Ed.), American Elsevier, New York, 1972.

[23] JONES, L H.; AND MERWIN, R. E. "Trends in microprogramming: a second reading," *IEEE Trans. Computers* (August 1974), 754-759.

[24] KATZAN, H., JR., *Computer organization and the System/370*, Von Nostrand Reinhold Co., New York, 1971.

[25] KNUTH, D E. *The art of computer programming*, Vol. 1, Addison-Wesley, Readıng, Mass., 1968.

[26] MADNICK, S. E.; AND DONOVAN, J. J. *Operatıng systems*, McGraw-Hill, New York, 1974.

[27] ORGANICK, E. I. *Computer system organization, the B5700/B6700 serıes*, Academic Press, New York, 1974.

[28] REDDI, S. S.; AND FEUSTEL, E. A. "An approach to restructurable computer systems," in *Proc. Sagamore Computer Conf.*, 1974, Lecture notes in *Computer scıence*, Vol. 24, Springer Verlag, New York, 1975, 319–337.

[29] RICHARDS, R. K. *Electronic digital systems*, John Wiley & Sons, New York, 1966.

[30] SHORE, J. E "Second thoughts on parallel processing," *Computers and Electrical Engineerıng* (June 1973), 95–109.

[31] TEXAS INSTRUMENTS INC. *A descriptıon of the advanced scıentıfic computer system*, Equipment Group, Texas Instruments, Inc., Austın, Texas, 1973.

[32] THORNTON, J E. *Design of a computer: the CDC 6600*, Scott, Foresman & Co., Glenview, Ill., 1970.

[33] WILKES, M. V.; AND STRINGER, J. B. "Microprogramming and the design of the control cırcuits in an electronic dıgital computer," in *Proc. Cambrıdge Phıl. Soc.*, Part 2, 1953, Cambridge Univ. Press, New York, 1953, pp. 230–238.