# DESIGN AND ANALYSIS OF DISTRIBUTED REAL-TIME SYSTEMS

PAUL J. FORTIER

Intertext Publications, Inc.
McGraw-Hill, Inc. New York, NY

*Library of Congress Catalog Card Number 85-60356*



10 9 8 7 6 5 4 3 2 1

ISBN 0-07-021619-3

Intertext Publications, Inc.
McGraw-Hill Book Company
1221 Avenue of the Americas
New York, NY 10020

# Acknowledgments

The realization of this book is due to more than just the efforts of the author. In order to put the record straight I wish to express my appreciation to the following individuals. First, I wish to thank all my instructors throughout my formative and present period of instruction for instilling in me the zest for knowledge and the seeds of thought. They truly made this effort possible. Additionally, I wish to give special thanks to Robert Charette of Softec for his many enlightening discussions on various portions of this text, to Thomas Conrad for his inputs in the area of software engineering, and to Daniel Juttelstad for his blue-sky discussions and constant prodding. A very special level of gratitude is due to Lloyd Watts for putting my handwritten notes into usable form, and to William Giallo for the artwork which appears throughout the book. Finally, I want to thank my wife, Kathleen, for her crucial efforts at clarifying and organizing my original illustrations, and for her undying support during the preparation of this manuscript. Finally, I want to thank my children, Daniel, Brian, and Nicole, for their patience with me while I constantly neglected them during the writing of this book.

Paul J. Fortier
Newport, Rhode Island
April, 1985

# Table of Contents

CHAPTER ONE

# Introduction

The design of computer systems, whether distributed or otherwise, is made up of the selection and design of the basic building blocks which comprise the system (such as memories, CPUs, I/O devices, networks, software, etc.), the combination of these elements into a system architecture, and the evaluation of whether the collection of components constitutes a successful computer system. This evaluation can be done before the computer is built via analytical simulation models, during design time via modeling and testing to verify that the initial specification or expectations are being met or through user acceptance via rigorous use and feedback once the system has been delivered.

Each one of these evaluation techniques has merit from a designer's viewpoint. At one extreme they allow total flexibility to build anything wanted and see what happens afterwards (the architect's Heaven). That is, one can wait until the system is delivered and used to see if it truly is an operable design and implementation. At the other extreme, rigorous a priori studies will drive the actual design to the point where the architect has little to do once the initial testing is done. That is, the major building blocks would be selected based on user needs and the design would be derived directly from the basic building blocks.

The above extremes represent the art of computer design versus the science of computer design (Figure 1.1).

These design techniques can be accomplished ad hoc. That means, as you build a computer, the build-a-little, test-a-little approach can be used, or the project can be based upon structured, computer-aided design tools to give an operable and efficient design before the building of hardware and expending of capital is begun.

Based on the above, the goal of this text is to introduce the reader to the concepts for design and performance evaluation of Distributed Computer Systems. We will start with the elementary notion of the nature of a computer. This discussion will deal with the basic components of the classic von Neumann machine, followed by introductions of concepts for other classes of computers. It will include special purpose machines such as signal processors, array processors, backend computers, data flow machines, etc. This will lead into a discussion of what constitutes a distributed computer, what its capabili-

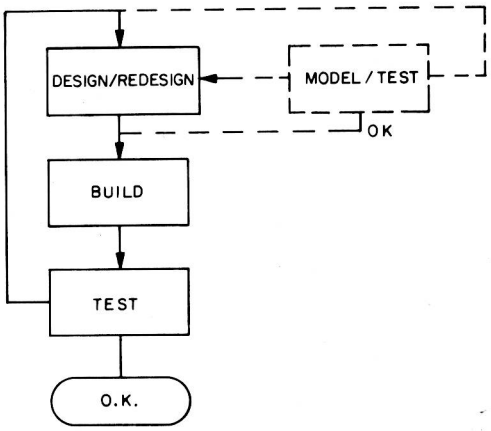


**FIGURE 1.1 Approach to system design.**

ties are, and what problems arise from distribution not seen in the central computer models and how they differ.

Following this the reader will be introduced to the major components which comprise distributed systems, namely networks, operating systems, and data base management systems, ending with discussions for performance evaluation and computer-aided design.

Network architecture design will deal with the description, classification, and design of protocols and hardware to allow a collection of computers or devices to converse over some geographic distance. This distance can be widely separated, as seen in ARPANET, or more tightly coupled as in a local network, as seen in a real-time control application such as paper mills, nuclear plants, ships, planes, etc.

The next section will describe operating systems comprised of a collection of Software/Firmware/Hardware responsible to provide a usable environment for user tasks to run while not interfering with one another.

Following this section a chapter dealing with data base management is provided. A data base management system provides the interface and control to global data in a unified fashion to all users. This chapter will describe the basics of DBMS followed by the issues in the design and structure of distributed data base management systems. Once this concept has been adequately covered, the text will introduce the reader to performance evaluation of computer systems via analytical models and simulation leading to discussion of design aids for users in automating the design of distributed systems.

## Architecture and History of Early Computers

In this section we examine the architecture of the basic von Neumann machine and describe the historical growth and development of computers from

the mid-1940s when von Neumann wrote his paper [379] dealing with the organization of the stored information computer, to the present status of distributed computers.

The first computer grew out of the need of the military to have a mechanism to quickly compute ballistic missile trajectories. The outgrowth of these early studies was ENIAC, completed in 1946 at the Moore School of Engineering under the direction of Eckert and Mauchly. It consisted of approximately 1,500 relays and 18,000 vacuum tubes and had the capability of 5,000 add/subtract instructions per second. This early computer had hardwired programs and required rewiring to reprogram. This system was used for approximately 10 years before being retired.

Modern stored program computers appeared in 1944. The idea for this class of computers is attributed to von Neumann. His first stored program computer was the EDVAC, which was never built. Wilkes, who studied with von Neumann, built the first stored program computer, the EDSAC, in 1949 in the United Kingdom. It had a prime memory of 1,024 words and secondary memory of 4,600 words. It was the first computer to use a memory hierarchy.

At the same time this was occurring, von Neumann, Goldstein, et al were working on the TAS or von Neumann machine at Princeton. This machine became the basis for all modern computers. The architecture has not changed drastically over the years, other than for technology insertion. Due to its importance, a brief description of it is included herewith. A more detailed study will be included in future chapters.

The basic von Neumann machine is comprised of five major building blocks or components (Figure 1.2) as described below.

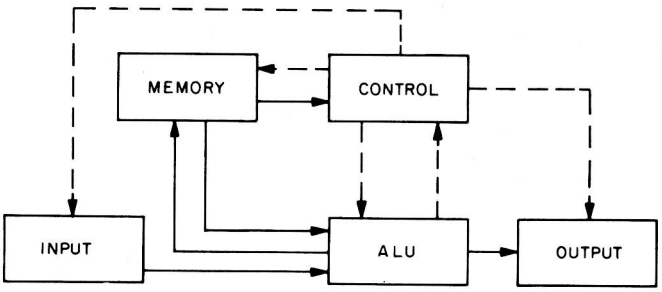


**FIGURE 1.2 Von Neumann/IAS architecture.**

## INPUT

The input unit functions as the means upon which a human can put information (programs/data) into the computer to be acted on.

A myriad of devices have been built to date to describe these functions, as will be seen in Chapter 2.

### OUTPUT

The output unit functions as the means upon which the computer provides the requisite outputs as described to it via the user's stored program (input) commands.

### MEMORY

The memory unit functions as the medium which is utilized in storing instructions (programs), intermediate data, and final results of computations.

### ALU

The CPU (arithmetic logic unit) is the functional unit which performs the instructions (arithmetic, logic, and redistribution operations) on supplied data.

### CONTROL

The control unit interprets instructions supplied from memory and provides the needed logic signals to devices to execute the interpreted instructions.

The von Neumann architecture operates in a very structured manner. That is, it operates under a cycle called the read/execute cycle. This cycle, in general, performs as follows:

1. The control unit fetches an instruction from memory based on a pointer which is incremented through a program.
2. The control unit then decodes (interprets) the instruction into its actions.
3. Based on this decoding, the instruction is now executed. Example: fetch operand, store operand, get input, perform addition, etc.
4. Once step three is completed, control reverts to step one.

Von Neumann's original paper describes in great detail much of what is seen in today's computers. From this start computers went through various phases, but they still greatly resemble their early ancestors.

The mid- to late-1940s saw great growth in research, at various institutions, into the design of IAS (von Neumann) style machines.

During the time that the IAS computers were being built, the first real-time response computer was being built at M.I.T. It was completed in 1951. This type of computer is significant because it allowed computers to be used in fast response situations or human life critical situations such as real-time industrial simulations, air traffic control, process control, etc.

The 1950s saw the birth of the computer industry. The first successful machine was the UNIVAC I delivered in 1951 to the census bureau. Its major architectural advance was in its use of magnetic tapes for inputting and outputting and storage of large amounts of data.

Following these early machines were subsequent design modifications With each new computer some level of architecture was either refined or upgraded via technology insertion.

Following this and other earlier successes, the industry took off. The next big leap for the industry came with the arrival of core memories in the mid-1950s. This event allowed designers to replace the inefficient electrostatic storage tubes with a new, more reliable, survivable storage medium which required less space, weight, and power than its predecessor. Along with this new memory came an advance in multilevel storage systems, that is, the use of magnetic tape and disk (drum) as secondary storage. At the same time, to alleviate the burden of programming in machine languages, efforts were begun to develop various high level languages. The most noteworthy, based on its impact on the industry, would have to be the development by Bachus and his coworkers of FORTRAN during the period 1954-1957. The influence of this language is still felt throughout the industry.

The next phase in the history of computers came in the late 1950s to mid-1960s and was due to the use of transistors. These devices allowed many manufacturers to supply more computing power for the dollar.

The trend at this point was to continually develop larger and more sophisticated machines such as the CDC 6600, and the Burroughs 5000 to name a few. These machines were designed to be used with high-level languages and included sophisticated operating systems.

As time went on, the trend began to shift from large scientific computers to general purpose computers. The emergence of Digital Equipment Corporation (DEC) with its line of minicomputers and the advance of MSI (medium-scale integrated circuits) followed by LSI (large-scale integration) and presently VLSI (very large-scale integration) have aided computer designers by allowing them to provide more and more capability for the same or less cost than previous designers. This is a phenomenon not seen in other industries. These led to many markets being developed such as personal computers, microprocessors, minicomputers, super computers, etc.

Along with these strides in architectural features software also progressed through many phases. Operating systems grew from the primitive capability of one user on the system at a time (Figure 1.3), through the batch environment where job turnaround time was reduced, then on to multiprocessing where the CPU, utilizing its operating system, swaps jobs in and out, giving all users the feel of running the computer by themselves.

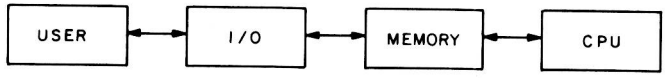


**FIGURE 1.3 Single-user system, c. 1952.**

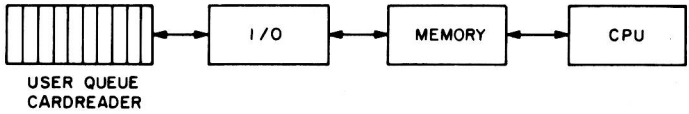


**FIGURE 1.4 Batch processing, c. 1958.**

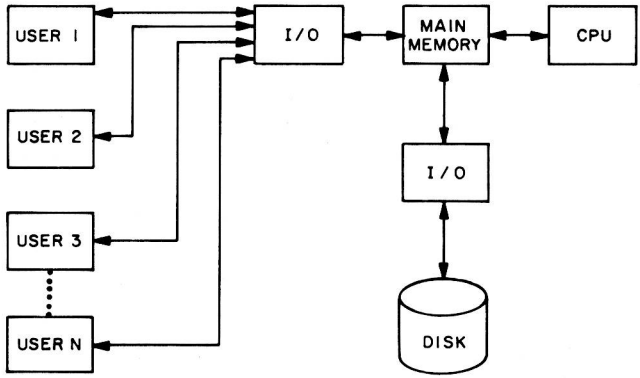


**FIGURE 1.5 Time-sharing computer (absolute address), c. 1962.**

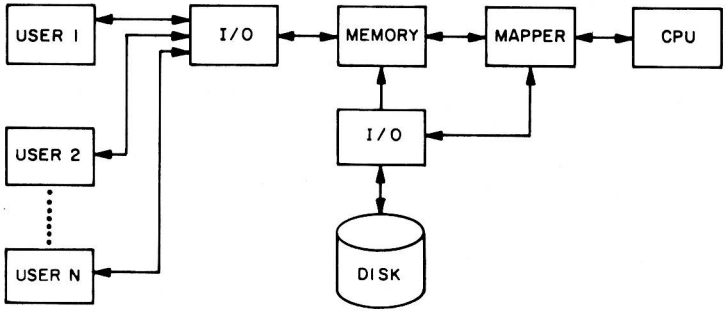


**FIGURE 1.6 Time sharing with virtual storage hierarchy, c. 1972.**

The early stand-alone concept of the von Neumann machine, where one user used the computer while others waited until he was done, was an inefficient use of an expensive resource (Figure 1.3). This concept disappeared and progressed, based on user need, into batch systems where the inefficiency of users' jobs being loaded and unloaded once completed was resolved. In a batch environment many users' jobs were bundled together and fed into the machine and outputted once completed. This required only one setup and breakdown between much larger jobs. This aided system efficiency, but still did not remove all the inefficiency. If a job acquired the CPU and had long

idle loops for I/O, etc. the CPU would remain idle when this occurred (Figure 1.4). The next generation of operating systems removed this inefficiency by allowing many users to share the CPU at one time via interactive I/O devices. The idea here was to give each process a chunk (quantum) of CPU time to operate in. If the quantum was well selected, then users would view the computer as one dedicated to their task. The problem with this setup was that the system still required users to have great knowledge of the system structure and addressing to be able to locate their programs in physical memory (Figure 1.5).

The present operating systems supply a level of multiprocessing to users along with a virtual memory environment. That is, many users can use the computer simultaneously while utilizing a memory space much larger than the physical space. The operating system in concert with compilers, linkers, and loaders provides mechanisms to handle user virtual addresses and map them to physical hardware (Figure 1.6).

The trends in the future are clearly based on the past. Vendors will strive for smaller, faster and cheaper computers which supply the same, if not more, computing power.

This trend will aid in the development of low-cost collections of computers combined into a single system (distributed processing).

These networks will be used by personal computer users' resource-sharing networks (Figure 1.7) as well as industrial users' local area networks (Figure 1.8) performing a myriad of user tasks.

The remainder of this book will discuss the basic hardware blocks of a computer and how these are combined into collections of computers to deliver increased capability to users along with discussions of how to select the proper capabilities in support of user requirements.

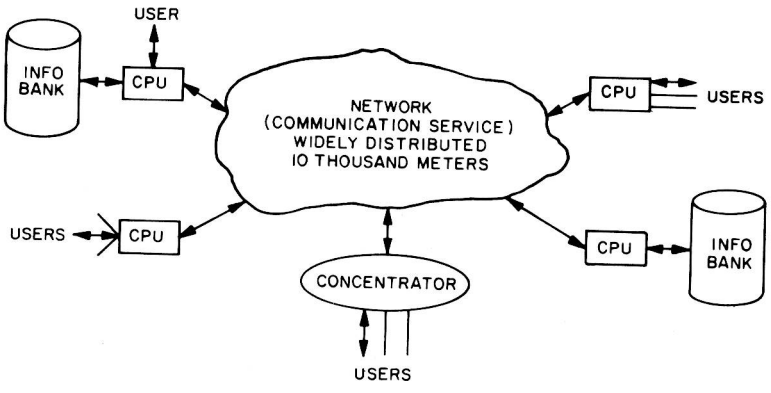


**FIGURE 1.7 Resource networks.**

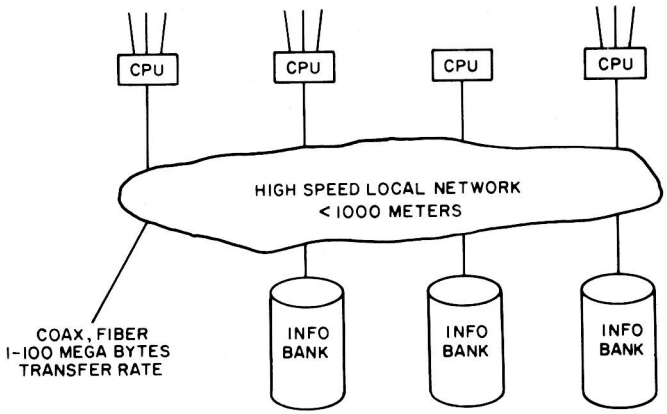


**FIGURE 1.8 Local area network.**

CHAPTER TWO

# The Basics of Computer Architecture

All modern digital computers consist of an interconnection of central processing units (CPU), memories, and input/output devices connected together by communication buses and controlled via the control unit (Figure 2.1). This chapter will discuss and introduce the architecture of these four components and the method by which they are interconnected. Also presented will be a high-level vista of various devices built from these basic building blocks, namely associative processors, array processors, signal processors, pipeline processors and multiprocessors. (Figure 2.2).

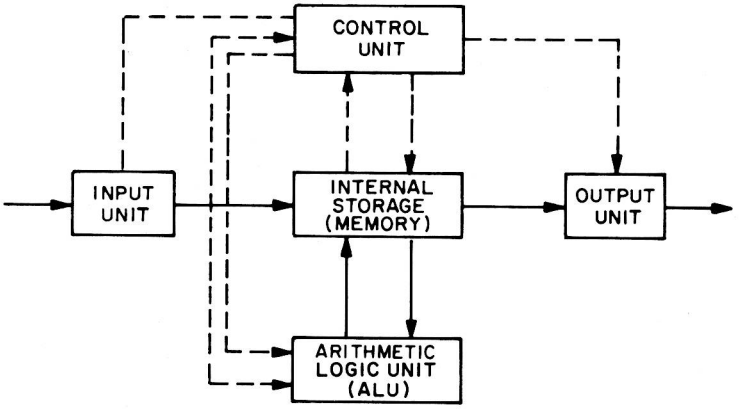


**FIGURE 2.1 Control unit interaction.**

## Arithmetic Logic Unit (ALU)

The ALU is the device in a digital computer which performs the actual work of computation, calculation, and comparisons. This device is composed of a shifter, adder, logical operator circuit, and an accumulator, temporary and conditional registers. This device accepts data from the internal storage memory and acts on it based on the control unit signals being supplied.

Using the supplied data the ALU may be requested to do operations such as add the content of the accumulator (ALU basic resident register for per-

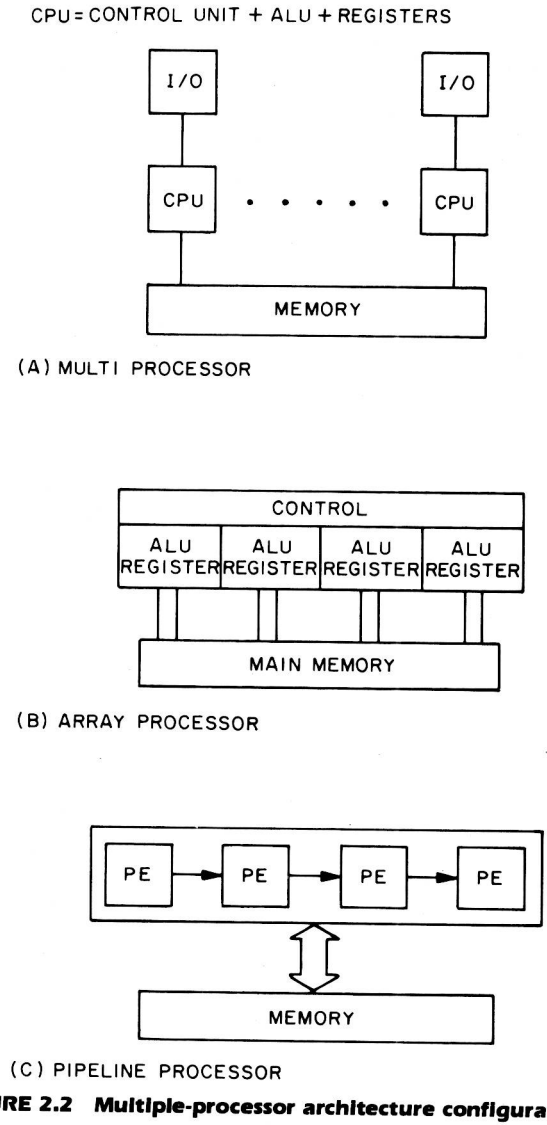


**FIGURE 2.2 Multiple-processor architecture configuration.**

forming operations) with the data being supplied into the temporary register (Figure 2.3) and store the result back into the accumulator (A register).

Other operations that the ALU would be required to perform include: increment contents of the A register and store results back into the A register, neumonic code Inc A (operation code or assembly instruction), decrement the contents of the A register and store back into B register.

Neumonic dec A

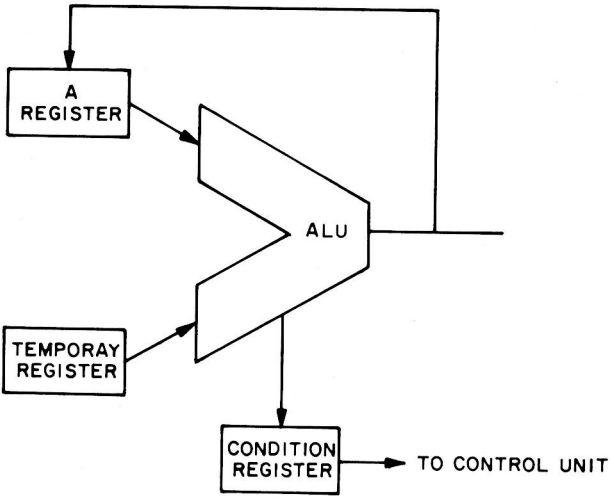


**FIGURE 2.3 Arithmetic logic unit.**

Load accumulator with the contents of memory location X (Figure 2.3) or store A into memory location X.

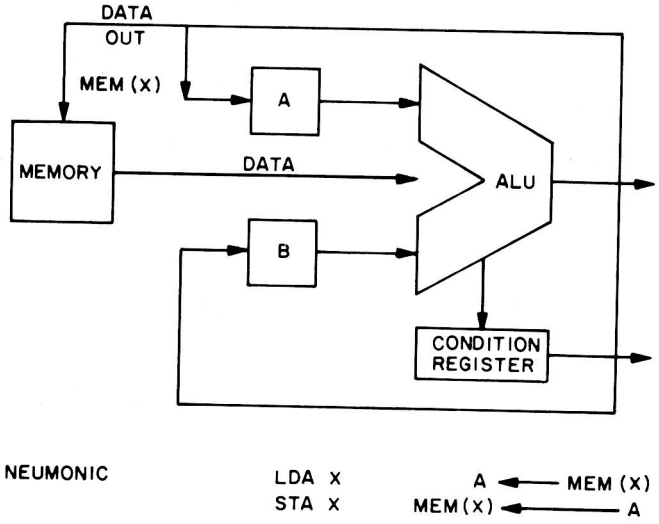


**FIGURE 2.4 Operation of ALU.**

| | | | |
|---|---|---|---|
| Neumonic | LDA | X | A<------MEM[X] |
| | STA | X | MEM[X]<------A |

Load B register with the contents of memory location X or store B register into memory location X.

| | | | |
|---|---|---|---|
| Neumonic | LDB | X | B<------MEM[X] |
| | STB | X | MEM[X]<------B |

Add memory X to register A or add memory to register B.

| | | | |
|---|---|---|---|
| Neumonic | ADA | X | A<-----A+MEM[X] |
| | ADB | X | B<-----B+MEM[X] |

Other operations include arithmetic such as sub B from A, Multiply A times B, and divide A by B.

Multiply, Divide — These require either microprogramming algorithms or extra hardware to perform and usually require many more memory cycles to perform.

The ALU also performs logical operations such as:

compares A with B and outputs results into condition register as well as, and, or, etc. All these operations output a status bit into the condition register.

Outputs of the condition register are then used by the control hardware/software to produce the proper effect on the processor. This action will be described further on in this chapter.

ALUs are comprised of hardware capable of performing the tasks (instructions) listed above. Hardware for these units includes devices such as half adders, full adders, right/left shifters, comparator circuits, and, or, nor, etc. connected by internal buses and controlled via the control unit's control lines.

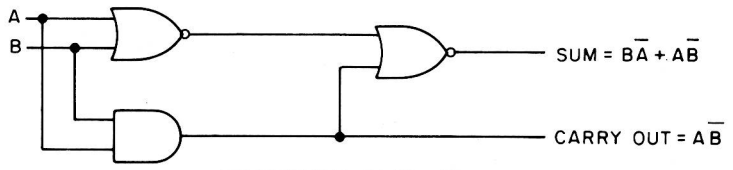


**FIGURE 2.5 Half adder.**

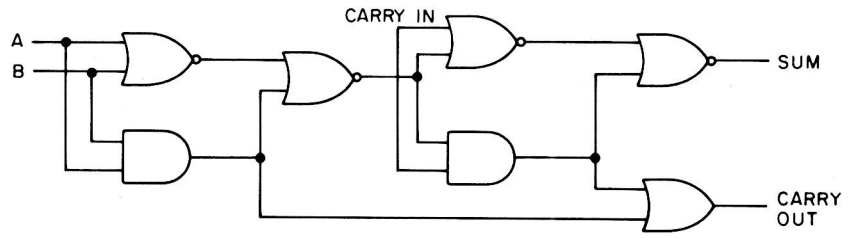


**FIGURE 2.6 Full adder.**

More details of specific hardware and software algorithms for the operations of the ALU can be found in [375], [376], [377], [378].