MICROCONTROLLER DESIGN AND CONCEPTS

By

Victor L. Vargas Garcia

A Project Report submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE in ELECTRIC ENGINEERING

UNIVERSITY OF PUERTO RICO

MAYAGUEZ CAMPUS

2004

Approved by:

Fernando Vega, Ph.D. Graduate Committee Member	Date
Jaime Arbona, Ph.D. Graduate Committee Member	Date
Rogelio Palomera, Ph.D. Graduate Committee Advisor	Date
Marco A. Arocha Ordoñez, Ph.D. Representative of Graduate Studies	Date
J. Ortiz Alvarez, Ph.D. Chairperson of the Department	Date
José A. Mari Mutt, Ph.D. Chairperson of Graduate Studies	Date

Abstract

Microcontroller Design and Concepts

By

Victor L. Vargas Garcia

A method for microcontroller design was developed. A basic data path configuration capable of processing the microcontroller basic instruction set was developed first. Based on this configuration, a four-bit microcontroller was developed from its most basic instruction set to the most complex one.

Through the design process, the microcontroller hardware evolves into a complex one as more instructions are added to the basic instruction set. More hardware is added in parallel to the basic data path configuration to make the execution of more complex instructions possible.

As a result it is expected that readers become familiar with the fundamental microcontroller concepts and operations. Design steps, implementation and testing of all the microcontroller development circuits are shown graphically and explained in detail. Finally designers will have a basic guide to develop their own microcontroller using this work procedure.

COMPENDIO

Microcontroller Design and Concepts

By

Victor L. Vargas García

Un método para el diseño de microcontroladores fue desarrollado partiendo de una configuración básica para el camino de datos que permite la ejecución del grupo más sencillo de instrucciones para un microcontrolador. Usando ésta configuración básica, un microcontrolador de 4 bits fue desarrollado desde sus instrucciones más básicas hasta las más complejas.

A medida que el proceso de desarrollo y evolución del microcontrolador se lleva a cabo, instrucciones más complejas se van sumando al conjunto de instrucciones básicas del microcontrolador, añadiendo circuitos en paralelo al circuito básico que forma el camino de datos que permiten que éstas nuevas instrucciones se puedan ejecutar.

Como resultado el lector tendrá una guia y una idea más clara sobre los fundamentos básicos de los microcontroladores, su funcionamiento y su arquitectura. Se mostrarán de una manera grafica y explicada en detalle, los pasos de diseño, implementación y prueba de los circuitos usados en el desarrollo de microcontroladores. Finalmente los diseñadores tendrán una guia básica para desarrollar su propio microncontrolador usando el procedimiento descrito en este trabajo.

To my Lord, my family and all my friends, for all the love and support. Without any of you this work would not have been possible.

To Dr. Rogelio Palomera, who has been not only an excellent professor, but more important, he has been a good friend.

Acknowledgements

Special thanks to Dr. Rogelio Palomera García for being my advisor, for his helpful comments, guidance and suggestions and above all, for being patient and good friend. To my parents Hector Vargas and Zenaida García for their unconditional support and help. To Mirelly Castro for being an excellent friend and for her support in difficult times.

Thanks to Dr. Jaime Arbona for his advice and valuable knowledge in electrical engineering. Thanks to Jorge Justiniano Magraner and Hector Santos for their patience, time and for being great persons and good friends.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Microcontrollers and microprocessors are the most used devices in electronic equipment. Modern technology demands from any engineer, a basic microcontroller or microprocessor knowledge. The basic difference between them is that microprocessors can be configured for the amount of memory and the input / output system used. The microcontroller has all the computing system (I/O system and memory) built in it. Designer's judgment determines which one should be used.

The emphasis of this work will be in the CPU; other important microcontroller parts such as the memory, the I/O system, microcontroller and microprocessor layout, fabrication process and technology are beyond the scope of this work. Design performance parameters like speed, power dissipation, wiring, packing, and transistor sizing are also beyond the scope of this work [8]. Microprocessor Assembly programming is not covered either.

1.1 Justification

The motivation for this work comes after the author took the Computer Architecture undergraduate course. The author realizes that microcontroller design could be an opportunity to summarize and apply most of the electronic engineering basic and advanced courses. Basic circuit analysis, basic electronic course, digital logic circuits and advanced digital design are some of the electrical engineering courses used in this work.

Another motivation for this work lies in the author's desire to present the student the microcontroller concepts, design and operation, as quick and clear as possible. For many years literature has been published regarding microcontroller and digital design. Techniques, methods, and procedures have been published, but most of them are usually explained using a symbolic or algorithmic approach. Some examples of this kind of approach can be found on "The Intel Microprocessors 808X,Pentium and Pentium Pro"[22] , "Computer Organization and Design The Hardware / Software Interface"[21], Embedded Systems and Computer Architecture"[19],Computer Organization and Architecture Principles of Structure and Function [20].

Although this work can serve as a quick reference for people with some microcontroller basic knowledge, it was developed specially for people that have not been exposed to microcontrollers or are exploring the field for the first time. After students understand the basic microcontroller concepts, they can go by their own in the field exploring other design concepts and alternatives.

To grasp the basic concepts at the starting stage, students feel more comfortable when they see the theoretical materialization, simulation and execution of hardware circuits, instead of large equations, diagrams, algorithms and symbols that most of the microcontroller information sources offer. The hardware implementation of every concept is what makes this work useful for beginners to learn and understand microcontroller concepts.

One of the main features of this work lies in the fact that it follows a series of steps and makes emphasis on the most important points in each and everyone of those steps. Beginners just have to follow those steps in order to design and simulate their own microcontroller. This work illustrates the design, simulation, testing, and implementation of all microcontroller circuits in each step. Through the whole process the student will appreciate the complete microcontroller evolution and transformation from zero to a functional unit.

Practice is the key for success in any career. This method provides mechanisms to change some of the microcontroller parts without affecting others. It makes emphasis on modularization. Through the whole process, modules of each part are designed and can be changed individually without affecting the entire system. This allows experimentation and circuit changes to examine what happens.

One possible application of this work is that students can transform the microcontroller schematic into HDL code and download it to an FPGA for prototype simulation. This way, the students increase their understanding of microcontroller concepts and operation, with hands-on experience; they can examine how the instruction execution is and how the microcontroller circuits work in every instruction. Also multiple versions of one microcontroller can be developed with slight changes, allowing students to observe the effect of those changes in each design and simulate each prototype on FPGA. This work provides a mechanism for students to train easier, faster and get more practice in microcontroller design.

A weak point of this method is that it does not achieve an efficient implementation. Performance is not the main point of this work; just delivering to the student the most important microcontroller concepts. In chapter two we find information regarding to microcontroller performance. The focus of this work is in the methodology, not in the computational capabilities and features of the microcontroller.

Besides its educational approach, another important point is that this method provides a mechanism to design a microcontroller that can be simulated, as said before, on FPGA, but also can be used on real applications. In other words, users making slight changes can produce a different microcontroller for new applications as needed. Users do not have to buy a new microcontroller but try a different one using this method. Of course this is convenient for experimentation or academic purposes only, not for applications where performance is the critical point.

Modern microcontroller costs are relatively low, and are very useful for many applications but sometimes there are situations that are better handled with specially designed microcontrollers for specific applications. For example, a designer may want to build and control his/her own personal robot, with a specific instruction set. Designers can find in the market some inexpensive microcontrollers that suit design requirements. But those popular microcontrollers perhaps are for general use, but probably lacking features that designers would be looking for.

It is important to remember that those popular microcontrollers in the market today are not designed for specific needs; some are for general purpose and others are for specific applications. Then, sometimes designers invest huge amounts of time and effort designing and programming assembly routine codes in order to achieve the required microcontroller performance, as to take full control of their robot. Designing a microcontroller for specific needs allows designers to minimize the programming complexity and enhance designers system's performance.

Designers also should keep in mind that microcontroller programming is as important as the microcontroller hardware design. Although it is not the intention of this work to discuss the microcontroller programming, this work illustrates the instruction execution of the microcontroller. This helps a lot when we are trying to understand the basic concepts of assembly programming like the addressing modes, clock cycles, and operands.

The quality of the microprogramming is what makes it possible to transform the complex circuits of the microcontroller into something useful. One of the main motivations for this work will be that inexperienced designers will not only gain an insight of microcontroller design and operation, but also, designers will get a better understanding of the microcontroller assembly programming.

1.2 Research Objective

The main idea of this work is to develop a systematic and straightforward procedure that allows students to understand microcontrollers design and operation. Inexperienced designers should be able to design their own microcontrollers from scratch using this procedure. This work assumes that the student has a basic knowledge of circuit analysis and digital logic circuits.

1.3 Simulations

There are many simulation tools that can be used for microcontroller design. Hardware Description Language (HDL) programming and graphic simulators are the main development tools used in the microcontroller design market. The computer tool used in this work is the graphical simulator Logic Works. Logic Works was chosen because the focus of this work is for beginners in the microcontroller field. Logic Works brings to the student an easy and complete visualization of the circuits and their operation. One of the main features of this work consists in its illustrative techniques and Logic Works results useful for these purposes.

HDL is convenient for large size circuits and then its code can be downloaded into an FPGA for device prototype testing. But its programming nature does not result useful for people trying for the first time to grasp the microcontroller concepts. Users face a double challenge because they are trying to understand the basic principles of the microcontroller operation and at the same time they are trying to learn the programming rules and techniques of HDL code in order to execute the circuit simulation. Logic Works allows users to graphically understand what happens inside the microcontroller during its execution and then, schematics can be transformed into an HDL code and downloaded into an FPGA for further prototype simulation.

1.4 Work Organization

Basic theory about microprocessors, its basic concepts and applications, performance factors and a comparison between microcontrollers and microprocessors are discussed in Chapter 2. The third chapter discusses the digital circuits available for the microcontroller HDL code prototype, the microcontroller implementation alternatives and programming. The fourth chapter describes in detail each of the microcontroller design steps used in this work and the most important points to keep in mind. Chapter 5 has an example of the microcontroller design process described in chapter four. In this chapter the microcontroller instruction set, architecture, basic circuits and the evolution of the data path as new instructions are developed are described in detail. Chapter 6 presents the control unit design. A detailed description of each instruction is given in chapter 7. Chapter 8 presents the conclusions of this work.

Chapter 2. Theory and Applications

2.1 Microcontroller Applications

The microcontroller is one of the most important electronic devices on which modern technology is based on. Microcontroller uses are endless; from toys to microwaves, ovens, TV sets, computers, printers, cars and so on.

Digital circuits become larger and larger as more functions need to be executed. In modern digital world, most individual digital circuit components are sold in a single chip. Those individual chips need power and space to operate. When the circuit becomes huge, the traditional logic design approach is not the best option and microcontrollers become convenient. Microcontrollers are basically sequential machines because their operation depends on their current status and its inputs. Their power lies in the fact that the hardwire configuration allows its operation to be changed depending on programming. It is not required to use additional logic circuits if the operation is changed.

2.2 The Processor and the Microntroller Concepts

Data are words, numbers and graphics that describe people, events, things and ideas. It becomes information when used as the basis for initiating some actions or to take decisions. Data is represented by binary expressions when used in the digital world.

A binary number system is a numeric system that has only two different digits: 1 and 0 (binary); and any of these is called a bit. Data are represented by finite permutation of bits. These combinations are called words. A collection of hardware devices that manipulate binary expressions to process information is called a processor [1].

The processor manipulates binary numbers following an algorithm, which determines the way in which the instructions are processed by the hardware inside the processor, how data begins to be processed and where it is finished. An instruction code in the instruction format indicates to the system which algorithm to perform. This specific algorithm represents the specific instruction to be executed. The following are the principal processor components [1].

1) Arithmetic Logic Unit (ALU): is a combinational logic network that performs the mathematical and logical operations of the processor.
2) Registers: hold the data operated on, between clock cycles for processing.
3) Control Unit: a synchronous sequential logic network that controls all the hardware in the digital system. This unit decodes the instruction,generates the proper sequence of control signals, and activate and deactivates the corresponding hardware units in the system to achieve the right processing according to the instruction.
4) The clock: a periodic pulse waveform that synchronizes all the elements in the system. Every clock cycle represents a state of the system. This means that in every clock cycle the system will have specific hardware control lines that are going to be on or off. The system clock speed depends on the response speed of the circuit elements when data passes through them.

Although these components are the most important ones, they are not alone. A big difference exists between identifying all those main elements and putting them together to work. Digital Logic, gates, multiplexers and other important circuits are necessary for processing support or to solve implementation problems, avoid signal conflicts and so on. Memory (circuit where data and instructions are stored) and input / output circuit interface (computer system used to pass data to and from the central processing unit) are necessary circuits for the microprocessor implementation.

Any hardware involved in data transfer into or out of the processor is considered separate from the processor. Processor only refers to the hardware that manipulates data. When a processor is capable of performing arithmetic operations, logical operations, load and store operations, branching operation and input-output operations, it is called a “general purpose processor”. When it is integrated in a single IC it is called a microprocessor.

A personal computer is usually a connection of components that contain many microprocessors. The motherboard contains the main microcoprocessor, but other microcoprocessors or microcontrollers are also involved. The keyboard, the disk drive interface, the display monitor interface, and the printer are some of the components that may contain their own microcontrollers. Therefore, a personal computer system is a collection of many microcontrollers controlled by a main microprocessor.

2.3 Microcontroller Performance Factors

Microcontroller performance can be defined in terms of speed, size, power, cost, design time and manufacture cost. Each depends on concepts beyond the scope of this work. The main factor determines the microcontroller performance [9] are its architecture, design features and manufacture process. Thus the microcontroller performance depends on designers' judgment at the design stage.

The architecture features determine the remaining microcontroller characteristics. The architecture depends on the microcontroller application. Different applications differ in features and data processing requirements. The Von Neumann architecture and the Harvard architecture [3] are the two main architectures used in microcontroller design. The Harvard architecture is the most popular nowadays. Von Neumann architecture main characteristic is that it uses one main memory where data and instructions are stored. Only one system bus is used for control, data transfer, processing and addressing. Harvard architecture consists of two different and independent memories in which one contains instructions and the other one contains data. Both have their own data bus systems for control, data transfer, processing and addressing. Both memories can be accessed simultaneously.

After the architecture has been defined the design process will be ruled by it. The hardware implementation will process the data by the architecture definition. Every part of the microcontroller hardware has many variables that can be configured to set its operation. Examples of these variables are the chip area and the distance between its components, the chip power dissipation, wiring effects, chip speed, manufacture materials, and packing. Each and every one of those variables is a field of study by itself, but they are beyond the scope of this work [8].

The Architecture and the hardware implementation features transform an idea into a circuit with specific characteristics. Computer simulation allows designers to verify that circuits work as required. When specification constrains and performance requirements are met, it is time for testing and manufacture. Design aspects defined by the architecture determines which manufacture process will be used. Manufacture processes have advantages and disadvantages and they can differ in equipment cost and technology.

2.4 The General Purpose Microcontroller

Microcontrollers execute different kind of instructions. The instructions for a general-purpose microcontroller can be:

1) Arithmetic Instructions.
2) Logic Instructions.
3) Data transfer Instructions.
4) Jump Instructions.
5) Miscellaneous Instructions.

Some microcontrollers are designed to specialize their execution in one or more of those classifications. Those are special purpose microcontrollers. Those basic instructions are combined to perform more complex instructions and the power and speed of execution of the microcontroller allows those instructions to execute complex tasks. Instructions are executed in such a way that an operation is achieved and different operations are used for different applications.

Special purpose microcontrollers are designed for an application where using a general-purpose microcontroller is not the best option. Usually those applications require repetitive execution of one or more instructions, which can be implemented in software or hardware. Hardware instruction implementation allows faster execution and reduces program size. Examples of special microcontrollers can be found on camcorders, digital cameras, automobiles and so on.

2.5 Comparing Microcontrollers and Microprocessors

The microprocessor is an integrated circuit composed by the Control Unit, Arithmetic Logic Unit, Registers and Digital circuit support. The microprocessor uses its data bus pins, address bus pins, and control lines pins to allow connection to other circuits to configure the entire system. The main characteristic of the microprocessor is that it is an open system, which means that its configuration is variable, and can be adapted to many different applications. A block diagram of a microprocessor is shown in figure 2.1.

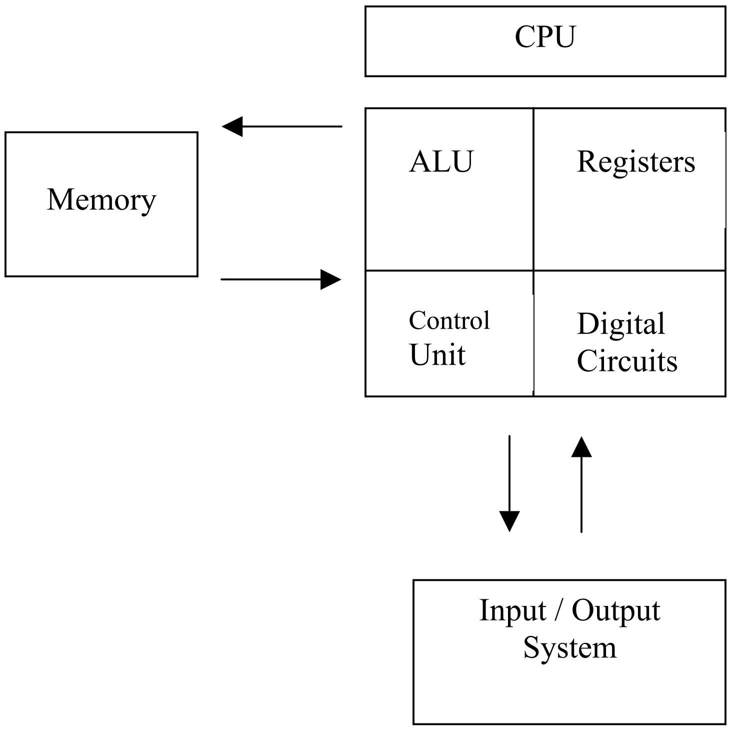


Figure 2.1 The Microprocessor Configuration

The microcontroller is a closed system. In the microcontroller all parts that can be configured in the microprocessor are fixed in the same chip. A block diagram of a microcontroller is shown in figure 2.2. Just the lines that control the peripherals are the ones that go outside the chip. This characteristic makes microcontrollers suitable for specific applications or for general use.

The microcontroller applications range is narrower than the microprocessor's range. The reason is that microcontrollers have all their computing system integrated on the same chip. This reduces the available space inside the microcontroller to include components that the microprocessor have externally like memory and I/O system.

This means that a microprocessor can be used for microcontroller applications but microcontrollers cannot always be used for most microprocessor applications. Microcontrollers are preferred when the application is defined and specific. In those situations where important system modifications are needed or applications are not specialized a microprocessor is more convenient.

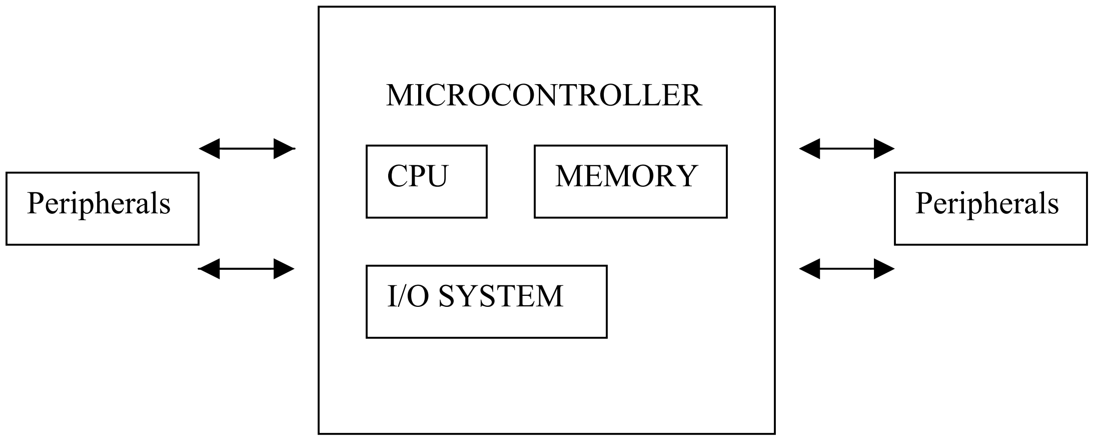


Figure 2.2 The Microcontroller Configuration

Chapter 3 Microcontroller Implementation and Operation

3.1 Implementation Alternatives

Traditionally, digital design was a manual process of designing and capturing circuits using schematic entry tools [2]. The increase in size and complexity of hardware has forced designers to discus new methods and tools for digital design.

Hardware description languages (HDL) and synthesis, have substituted the more traditional schematic process of simulation. This is because HDL allows simulating circuits with hundreds of elements in a relative short period of time. Some of the new tools for HDL simulation are electronic equipment containing Application-Specific Integrated Circuits (ASICs), or Field-Programmable Gate-Arrays (FPGAs).

The introduction of industry standards for hardware description languages and commercially available synthesis tools has helped establish this revolutionary design methodology. Some advantages are:

- Increased productivity yields shorter development cycles with more product features and reduced time to market,
- Reduced Non-Recurring Engineering (NRE) costs,
- Design reuse is enabled,
- Increased flexibility to design changes,
- Faster exploration of alternative architectures
- Faster exploration of alternative technology libraries,
- Enables use of synthesis to rapidly sweep the design space of area and timing, and to automatically generate testable circuits,
- Better and easier design auditing and verification.

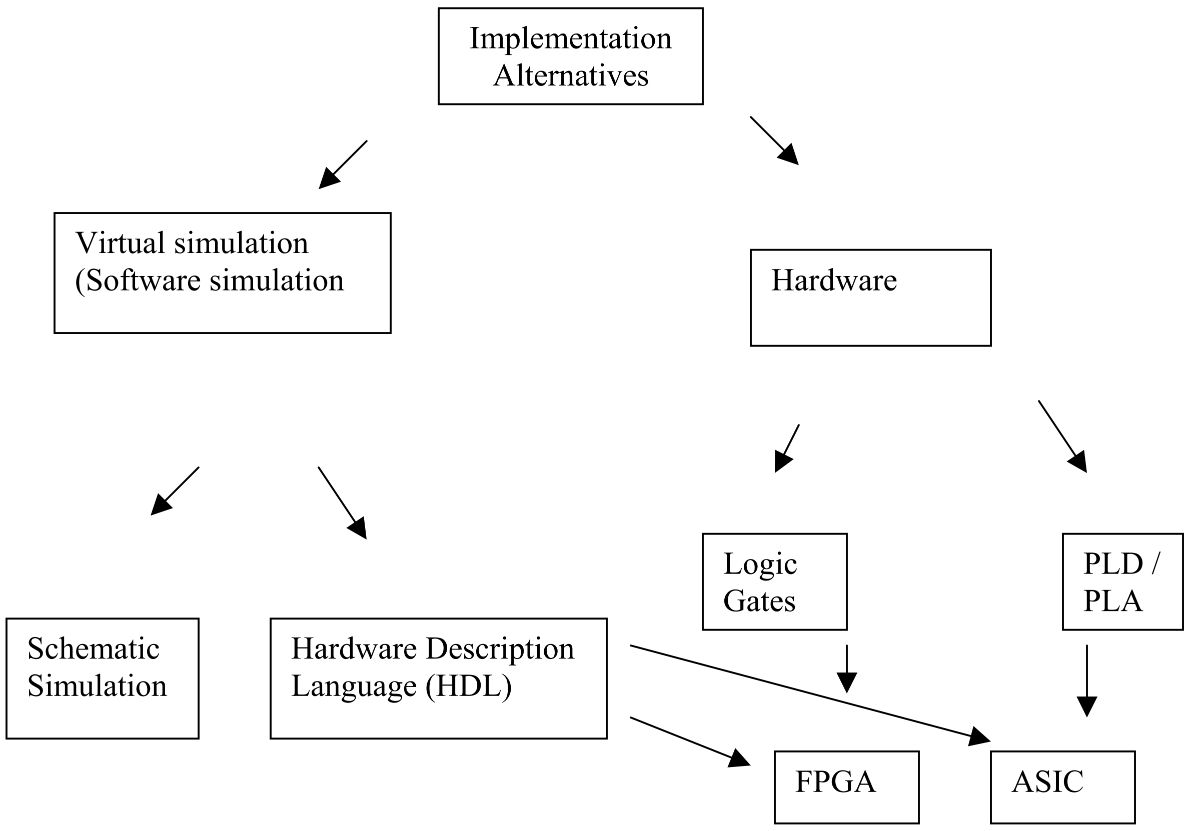

Implementation Alternatives
Virtual simulation (Software simulation
Hardware
Schematic Simulation
Hardware Description Language (HDL)
Logic Gates
PLD / PLA
FPGA
ASIC

Figure 3.1 Microcontroller Implementation Alternatives

Figure 3.1 illustrates the alternatives of hardware implementation available. Modern designs are characterized by their increase in size and complexity, circuit simulation is one of the most important steps in circuit design. Circuit simulation and hardware prototype implementation saves time and money because they allow designers to verify that the implemented digital design works as required.

Software simulation previews the circuit behavior. It serves as a mechanism to verify accurately the principal circuit characteristics and to ensure its design requirements. Hardware implementation, in contrast with software simulation, is a physical prototype configuration that serves to physically simulate the circuit behavior. Note that hardware implementation requires software simulation through HDL. Its advantage lies in the fact that circuits can be tested interacting with other real physical circuits before they are fabricated.

Standard "off-the-shelf" integrated circuits have a fixed functional operation defined by the chip manufacturer. Contrary to this, both ASIC and FPGAs are types of integrated circuit whose function is not fixed by the manufacturer. The designer for a particular application defines the function. An ASIC requires a final manufacturing process to customize its operation while an FPGA does not.

ASICs

An Application-Specific Integrated Circuit is a device that is partially manufactured by an ASIC vendor in generic form. This initial manufacturing process is the most complex, time consuming, and expensive stage of the total manufacturing process. The result is silicon chips with an array of unconnected transistors. The final manufacturing process of connecting the transistors together is then completed when a chip designer has a specific design to implement using ASIC. An ASIC vendor can usually do this in a couple of weeks and is known as the turn around time. One problem is that it is a physical realization, which means that if there are mistakes during the HDL simulation and are not corrected, its physical implementation will have the errors also and there are no mechanism to correct it once it is fabricated. There are two categories of ASIC devices: Gate Arrays and Standard Cells.

Gate Arrays

There are two types of gate array; a channeled gate array and a channel-less gate array. A channeled gate array is manufactured with single or double rows of basic cells across the silicon. A basic cell consists of a number of transistors. The channels between the rows of cells are used for interconnecting the basic cells during the final customization process. A channel-less gate array is manufactured with a "sea" of basic cells across the silicon and there are no dedicated channels for interconnections. Gate arrays contain from a few thousand equivalent gates to hundreds of thousands of equivalent gates. Due to the limited routing space on channeled gate arrays, typically only 70% to 90% of the total number of available gates can be used.

The library of cells provided by a gate array vendor will contain: primitive logic gates, registers, hard-macros and soft-macros. Hard-macros and soft-macros are usually of MSI and LSI complexity, such as multiplexers, comparators and counters. The manufacturer in terms of cell primitives defines hard macros. By comparison, the designer, for example, characterizes soft-macros by specifying the width of a particular counter.

Standard Cell

Standard cell devices do not have the concept of a basic cell and no components are prefabricated on the silicon chip. The manufacturer creates custom masks for every stage of the device's process and silicon is utilized much more efficiently than for gate arrays.

FPGAs

The Field-Programmable Gate Array is a completely manufactured device, but remains design independent. Each FPGA vendor manufactures devices to a proprietary architecture. However, the architecture will include a number of programmable logic blocks that are connected to programmable switching matrices. To configure a device for a particular functional operation these switching matrices are programmed to route signals between the individual logic blocks.

PLD and PLA

The Programmable Logic Device (PLD) is essentially a grid of programmable conductors that form rows and columns with fusible link at each cross point. PLD are classified according to their architecture, which is basically the functional arrangement of internal elements that give a device its unique characteristic. The Programmable Logic Array (PLA) is a device with programmable AND and OR arrays.

3.2 Hardware Description Languages (HDLs)

A Hardware Description Language (HDL) is a software programming language used to model the intended operation of a piece of hardware. There are two aspects of hardware description that HDL facilitates: true Abstract Behavior Modeling and Hardware Structure Modeling.

The Abstract Behavior Modeling is a declarative hardware description language in order to facilitate the abstract description of hardware behavior for specification purposes. The Hardware Structure Modeling is a hardware structure that can be modeled in a hardware description language irrespective of the design behavior. The hardware behavior may be modeled and represented at various levels of abstraction during the design process. Higher-level models describe the operation of hardware abstractly, while lower level models include more detail, such as inferred hardware structure [23].

3.3 Tradeoffs in Microcontroller Design

Is it necessary to use a special purpose microcontroller or a general purpose one can be used? That is an important question that must be answered before attempting to implement a microcontroller. In addition to having the basic instruction set, special purpose microcontrollers usually have instructions specialized to perform specific tasks. Those microcontrollers include in their design, special hardware that is used for execution and calculation support to execute instructions in their specific applications.

The application determines the microcontroller operation, and the operation is executed with specific instructions. Then, the real deal in the design process consists in making tradeoffs between designing more powerful and complex instructions that reduce the programming code, or as another alternative, the operation can be implemented in hardware to save the time-consuming programming of certain tasks and achieve faster execution.

Should an operation be implemented in hardware or software? Is it worth? The answers to those questions depend on many factors like design requirements, available budget, technology used and so on. Hardware instructions implementation result in faster executions but increase design cost. Software implemented operations save hardware and costs but increase the instruction execution time and the programming complexity. There are not defined rules. Designers have to make their choices based on design constrains and available resources to produce the best system performance at the lower cost.

3.4 The Microcontroller Programming

Commonly, every processor is designed with one purpose and has its own instruction set. The microcontroller architecture determine how powerful the instruction set is and how many clock cycles it takes to execute its instructions. As the instructions are more powerful, the microcontroller programming usually becomes more complex but shorter and more tasks are done per clock cycle.

Microcontroller programming is usually done in assembly language. This is because this is a low level programming language. Instruction in this low level programming language are directly related to the machine code, the ones and zeroes or high and low voltage combinations necessary to control all the hardware inside the microcontroller to process data. One advantage of assembly language is that allows the programmer to control some internal process like selecting specific registers that normally cannot be done using a high level programming language.

Each microcontroller has its own assembly language code, so the assembler is specific to the microcontroller. High level programming languages, on the contrary,are independent of the processor. The compiler and other tools are transparent to the programmer, do the translation to the respective processor used by the computing system.

Commercial microcontrollers are very often sold embedded in the so-called evaluation cards. These system boards contain additional hardware and connectors to facilitate applications and programming. The programmer can design the assembly program and download it to the microcontroller easily.

3.5 The Microcontroller Operation

Summarizing, the microcontroller operation consists in three steps:

- Fetch process; the fetch process consists in retrieving one instruction from memory and load it in the Instruction Register.
- Decoding; once the instruction is in the Instruction Register, the control unit receives the operational code from it. The control unit decodes the operational code to identify the instruction to be executed.
- Executing; after the control unit identify the instruction, it start a series of microcontroller hardware signal activations. To carryout the execution process some of the circuit elements must be on and off in each clock cycle. The control unit ensures that the necessary elements are on and off in each clock cycle to accomplish the instruction execution.

Basically the CPU addresses a memory location, obtains (fetches) a program instruction that is stored there, and carries out (executes) the instruction. After completing one instruction, the CPU moves on to the next one. This fetch and execute process is repeated until all of the instructions in a specific program are done. The fetch process clock cycle depend on the Instruction Register size (and i.e. the instruction word) and the number of bits of the data bus. For example if the IR size is eighteen bits and the data path is four bits, then five clock cycles will be needed for the fetch process. The memory size will determine how many instructions can be stored in it and indeed the program size that can be stored.

3.5.1 The Program Counter

To indicate the memory address to retrieve the instruction a special register is used. This register is the Program Counter. The PC holds the address of the memory location where the next instruction is located. The PC input ports are connected to the data bus; in this way the ALU connected also to the data bus increment the PC to the next memory location. The PC output port is connected to the memory address port to identify the required memory location where the instruction is.

3.6 FLAGS

Flags are also called conditional codes. Condition codes are bits set by the CPU hardware as the result of operations. Usually condition codes are collected into one or more registers called flag register. Flags are very useful because they can be used as parameters to make decisions. For example, a microcontroller application can check the flag register to see if the result of one subtraction operation is zero, then, using this information the microcontroller can take decisions to execute other instructions.

Chapter 4 The Microcontroller Design Steps

This chapter describes the steps used in the microcontroller design example of this work. Each step has important points that designers should keep in mind in them. Those points guide the user through the whole design process.

4.1 Methodology Steps

The steps are enumerated in table 4.1

STEP	**DESCRIPTION**
STEP I	Justification
STEP II	Operations Definition
STEP III	Instruction Set Definition
STEP IV	Architecture Definition
STEP V	Arithmetic Logic Unit (design and implementation)
STEP VI	The Register File
STEP VII	The Instruction Register
STEP VIII	Data Path for data processing and Control Signal Table
STEP IX	The PC, Jump and data transfer instructions
STEP X	The Control Unit

Table 4.1 Methodology Steps

The description of each one is given next.

4.2 Steps Description

STEP I: Justification

Designers should first analyze the situation and decide if a microcontroller is needed for the application. The following are some questions that could guide designers at the implementation decision stage.

- What is the application? Application is a computer program or set of programs designed for a particular type of real world job.
- Can the application be implemented with logic circuits? The answer to this question is obviously yes. But, what will be the resulting circuit size? Is it affordable?
- What could be the microcontroller implementation advantage? The importance of microcontroller lies on the fact that it has hardwired circuits that change their operation using programming. Designers should analyze if the amount of different applications justify the use of a microcontroller or if the use of individual operational circuits is more convenient.
- What are the advantages or disadvantages of using a microcontroller in terms of efficiency, time, design complexity and cost? Analyses of tradeoffs are necessary to answer those questions. Budget and design requirements analyses are necessary to decide if a microcontroller use is convenient or not. Sometimes the use of a microcontroller results in a waste of hardware resources. In other situations the microcontroller use results in the less expensive option. There are situations in which programming is avoided using logic circuit, but this choice could result in larger, expensive and more complex circuits.
- Is a microcontroller result in the best option? How many different operations will be used? How many times one operation is executed? Is it better to use individual circuits for every operation or using a microcontroller is more efficient? Do Individual circuits have faster response than the microcontroller? Is this difference in time response needed for the application? Is the microcontroller programming complexity worth instead of using individual circuits? What tasks are done routinely?

STEP II: The Operation Definition

After a careful study of the application, the next step consists in defining the amount of different operations required for the application.
One computer operation is defined as the calculation executed by a single machine code instruction [8]. It is also the mathematical or logical way of producing a result from one or more operands.

- What are the application operation requirements? Are those operations complex or simple? How many different operations does the application have? Do designers need a new microcontroller to execute one operation or can they use an existing one? If they use an existing one, does it execute the instruction as required in terms of clock cycle, power and speed?
- Is it more convenient to divide those operations in more simple tasks or not? Depending on the application and design requirements this could or could not be possible. Can the microcontroller with its instruction set, execute those individual and simple tasks, or a new one is needed?
- Can those tasks be executed using more than one instruction, or is one instruction enough? The answer to this question lies on the characteristics of every microcontroller instruction and depends on the amount of tasks covered by the instruction.

STEP III: The Instruction Set Definition

The instruction set should contain those instructions that the application requires. Tasks executed, amount of hardware used and clock cycles are very important parameters of an instruction. One instruction is defined as a program statement that has been changed into machine code. The CPU can understand the statement and execute it [8].

- How powerful is the instruction? The term powerful means that many tasks can be executed. This however may result in more hardware or more clock cycles per instruction.
- How many instructions are required to perform the operation? This will be determined by the power of the instruction set. The more powerful the instruction set is, fewer instructions are needed per operation.
- What kind of instructions does every microcontroller must have? Every microcontroller must have at least; logic, arithmetic, branch and data transfer instructions.
- How many complex tasks can be executed using the simplest instruction set? The basic instruction set can be combined to execute complex tasks. For example, a multiplication operation can be executed with successive execution of the addition instruction.
- What instructions should be implemented in hardware and which ones in software and why? Instructions frequently executed must be implemented in hardware. This saves programming time and size, allowing faster instruction execution. Software instructions are used depending on the application.

STEP IV: The Architecture Definition

The Computer architecture refers to the basic ideas and principles in which a computer system is based on [8].

- **The instruction operation**.

  The first task must always be to specify each instruction operation. After designers identify the instruction set, they must document: the instruction's name, as well as operands and execution in symbols for each one.

- **The microcontroller bit number**.

  The microcontroller bit number refers to the size of the group of bits processed during instruction execution. Sometimes choosing the number of bits is as simple as analyzing the required bits for the application. In other cases there are applications in which more than certain amount of bits results unnecessary. Using more than the necessary bits may result in excessive hardware use and an increase in the circuit size, cost and power consumption.

- **The instruction format.**

  The instruction format specifies the order of the instruction parameters in the instruction word. Those parameters include the operational code, registers used, and additional necessary data for the instruction execution.

- **The instruction format organization**

  The instruction word parameters can be organized as designers want. In this work the operational code will be at the left most side, next are the registers used during the operation and finally the additional data used for the instruction execution.

- **The Operational Code (Opcode).**

  The number of instructions decides the necessary bits for the operational code. The operational code identifies each instruction with a unique code for its execution.

- **Addressing modes**

  The addressing modes decide the amount of registers used for data processing. The addressing modes used during the instruction execution decides if more bits have to be used to address the data or not and this affects the size of the instruction word.

- **Bits used for the Register File.**

  The number of registers used in the Register File determines how many address bits in the instruction word are required to address one specific location in it.

- **Number of data buses.**

  The number of data buses in use determines the amount of data processed per clock cycle. Using more than one data bus can save clock cycles per instructions, but increases the data path and control unit circuit complexity.

- **Control Line Bus:** In this work the control lines will be connected to the control unit.
- **Address Bus:** Depending on design requirements the address bus is not necessary if the address bits can be transferred using the data bus. A dual role requires additional hardware.
- **I/O Handling:** Will the I/O ports be memory mapped or handled separately. Memory mapped ports do not require special I/O instructions.

STEP V: The Arithmetic Logic Unit

In step V, the goal is to design the Arithmetic Logic Unit circuit. The ALU is the CPU component where mathematical and logical operations are executed.

- **ALU components**

  The individual circuits that execute all the arithmetic and logical operations are joined together as one unit to compose the Arithmetic Logic Unit.

- **Testing**

  Testing is a very important task in this step. Designers must ensure that every individual circuit in the ALU correctly does every calculation.

STEP VI: The Register File

A register is a small high-speed memory circuit that holds binary data [8]. In This step, the Register File is developed. The Register File is a group of registers used to store data during the instruction execution. It is an important element because data needs to be stored between clock cycles for further processing.

- **Implementation alternatives**

  The number of data buses in the microcontroller determines the Register File design. Sometimes more than one data bus is used to accept and release data simultaneously in one clock cycle. Designers must decide how many data buses will be used in the microcontroller because the Register File will use the same number.

- **The number of registers for the application**

  This is an important design parameter because it affects not only The Register File size but also the Instruction Register size because the IR has bits dedicated for the Register File address. Designers must select the number of necessary registers to hold data in each instruction clock cycle.

STEP VII: The Instruction Register (IR)

The Instruction Register holds the instruction word that will be executed. It is designed at this stage because the numbers of instructions, registers used and the architecture have been defined. The IR is connected to the control unit, the Register File and the data path.

- **Implementation alternative:** The IR implementation consists of a register or a group of registers that holds the instruction word.

- **Size:** It will be easier if the size is equal to the word size because then, the instruction word holds all the required information for the instruction execution. The memory output is connected to the IR to load every single program instruction line. The IR does not have to be the same size of the data bus because it just transfers data and does not contain any other information about the instruction.

STEP VIII: Data Path

The microcontroller data path is the configuration of all the circuits used for data processing. Some key points are very important in this step. It is implemented at this stage because all the necessary circuits have been designed.

- **Layout**
  Designers must be creative and use strategic thinking to make the best circuit arrangement in order to achieve the instruction execution using the minimum amount of hardware and clock cycles.
- **Clock cycles**
  The Register File plays an important role in the number of clock cycles per instruction. More data can be processed at the same time depending on the amount of the Register File input and output ports. Also, another important element is the number of additional registers in the data path used to hold data between clock cycles. This can make a difference in the number of clock cycles per instruction if designers know how to use them.

STEP IX: The PC, Jump and data transfer instructions

The instructions developed at this stage use the existing data path hardware and additional necessary circuits added in it for instruction execution.

- Those instructions need additional circuit support because some of them make decisions between clock cycles. Those circuits are used only when their instructions are executed. It is very important to test those circuits before using them for support. Another reason for using additional hardware is that more than one task per clock cycle is executed in those instructions.
- Block diagram to show the added elements. It is convenient to show the added elements to the data path to see its transformation into a more complex one.
- The Program counter. The program counter was introduced in section 5.3.1 and is developed at this step. This step presents the PC implementation and interconnection in the microcontroller circuit.

STEP X: The Control Unit

The control unit is the CPU section that decodes program instructions and controls their execution. It takes control of every circuit signal in the microcontroller, activating or deactivating those signals in each clock cycle. The signal activation and deactivation per clock cycle make possible the flow of data through all data path circuits. The circuit arrangement determines the amount of processed data in each clock cycle. Then, as more data is processed per clock cycle fewer of them are needed. The developing method used in this work requires that designers "run" by hand every single instruction and take notes of which circuit signals are activated and deactivated per clock cycle.

- **Timer**
  The timer is a counter that goes from zero to seven and is used to specify each instruction clock cycle.
- **Operational Code Decoder**
  This element receives one specific instruction code and release one signal that indicates the microcontroller to execute it.
- **Control Unit Encoder**
  The Control Unit Encoder receives input signals from the opcode decoder and from the timer. The Control Unit Encoder activates the corresponding circuit signals that have to be active in the specified instruction in every clock cycle.
- **Implementation Alternatives**
  The preceding explanation of the control unit operation is implemented using logic circuits for the control unit encoder and the opcode decoder. There is another way of implementation that consists in the use of one ROM that has all the signal activation and deactivation per clock cycle. The control unit implementing this approach uses the opcode to identify the instruction location in ROM. Each line code in ROM represents each instruction clock cycle and the code in every line just controls (activates or deactivates) all the data path circuit signals.

Chapter 5 The Microcontroller Design Example

5.1 STEP I and II The Microcontroller Justification and Operations

The purpose of this chapter is to provide the reader an example of the methodology described in chapter 4. Step I, and II will not be developed in this example because our intention is to show the design and implementation of one general-purpose microcontroller.

5.2 STEP III: The Instruction Set

The choice of microcontrollers instruction set is not standardized due to designers and customers preferences. The microcontroller instructions are classified according to their operation. Table 5.1 presents the basic instruction set for the microcontroller of this work. In this table the transfer notation is used to show the instruction results. Here $A \leftarrow B + C$ for example means that the contents of A is substituted by the result of B + C. Those instructions were selected to show the reader an example of the most common instructions used in microcontrollers.

NAME	**MNEMONIC**	**ADDRESSING MODES**	OPERANDS	**TRANSFER NOTATION**
ARITHMETIC INSTRUCTION SET				
ADDITION	ADD	Register	B, C	A ←(B +C)
SUBSTRACT	SUB	Register	B, C	A ←(B - C)
INMEDIATE ADDITION	ADDI	Immediate	B, DATA	A ←(B +DATA)
INMEDIATE SUBSTRACTION	SUBI	Immediate	B, DATA	A ←(B -DATA)

LOGIC INSTRUCTION SET				
AND	AND	Register	B, C	A ←(B • C)
OR	OR	Register	B, C	A ←(B (+) C)
INMEDIATE AND	ANDI	Immediate	B, DATA	A ←(B •DATA)
INMEDIATE OR	ORI	Immediate	B, DATA	A ←(B (+) DATA)
SHIFT RIGHT	SHR	Register	n	(1n) Bi+n B j+n B k+n ← BiBjBkBl
ARITHMETIC SHIFT RIGHT	SHRA	Register	n	(nBi) Bi+n B j+n B k+n ← BiBjBkBl
CIRCULAR SHIFT	SHC	Register	n	(nBl) Bi+n B j+n B k+n ← BiBjBkBl
SHIFT LEFT	SHL	Register	n	Bj-n Bk-n B1-n (1n) ← BiBjBkBl
NOT	NOT	Register	B	-(B) ← B
DATA TRANSFER				
LOAD	LD A, M	Register	A, M	A ← M
STORE	STR M, A	Register	A, M	M ← A
BRANCH				
UNCONDITION AL JUMP	UNCJMP	Immediate	Last 4 bits	PC ← (LAST 4 BITS)
JUMP IF CONDITION	BRNCH	Register	Address	IF CONDITION IS TRUE: PC ← (ADDRESS)
MISCELLANOUS				

DATA TRANSFER (IN)	IN	Register	ADDRESS	ADDRESS ← DATA
DATA TRANSFER (OUT)	OUT	Register	ADDRESS	PORT ← ADDRESS
READ PSW	RDPSW	Register	ADDRESS	ADDRESS ← PSW

Table 5.1 The Microcontroller Instruction Set

5.3 STEP IV: The Microcontroller Architecture Definition

The Architectural design steps include:

A) The Instruction Set.
B) The number of used bits to represent data (4, 8, 16,32 or 64 bits).
C) Instruction Format and addressing modes.
D) Number of data buses.
E) The instruction execution algorithm (the best arrangement of the hardware to process the software).
F) Clock cycles per instruction.
G) Input / Output mechanisms.

The computer organization must be specially designed to implement a particular architectural specification. The microcontroller task is to execute each and every instruction it receives. This means that each instruction reflects the architecture in use by the microcontroller. After the selection of the desired instructions for the microcontroller, the next step consists in specifying the rest of the architecture.

a) The Instruction Set

Step II defines the instruction set for the microcontroller.

b) Number of microcontroller bits

Because this work is focused on beginners, the number of bits used for this microcontroller will be four. Four-bit microcontrollers are simpler for design and implement. The same techniques used here for this four- bit microcontroller can be used for eight-bit or sixteen-bit microcontrollers.

c) The Instruction Format

After the basic architectural aspects have been defined, the instruction word can be defined. Each instruction word has a group of bits that identifies its specific code. The group of bits used for this code is called the instruction operational code or opcode. This work uses 20 instructions, so, the minimum number of bits for the opcode decoder is 5, because 2^4 = 16, while 2^5 = 32, enough to assign each instruction a specific code.

The Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					Ra			Rb			Rc			Different uses			

Figure 5.1 The Instruction Format

There are no standard rules for the order and meaning of the different groups of bits that compose the instruction word. That depends on designers' judgment and system architecture. The standard for this microcontroller will be the following; accordance to figure 5.1.

1) Bits 17-13 stand for the opcode. Those bits specify the instruction that will be executed.
2) Bits 12 to 10 labeled as Ra, specify the register file address location to store the processed data or the one that has been transferred from memory.
3) Bits 9 to 7 labeled as Rb, represent the register file address location of one instruction operand.
4) Bits 6 to 4 labeled as Rc, represent the register file address location of one instruction operand.
5) Bits 0 to 3 are used depending on the operation. For example, all the instructions that use the immediate addressing mode need a value directly from the instruction word. The value in those instructions is stored in those last 4 bits.

d) The number of data buses

The number of data buses in the system will be just one. Although one microcontroller with more than one data bus could be more efficient, the number of signal activations will be higher per clock cycle. This will result in a more complex control unit and for simplicity purposes the microcontroller of this work have just one data bus.

Architecture design steps; E) Data Path arrangement, F) Clock cycles per instruction, and G) Input / Output mechanisms will be specified at the same instruction design moment.

5.4 STEP V: The Arithmetic Logic Unit

The Arithmetic Logic Unit is one of the most fundamental CPU components. The techniques used in this work for the ALU design consist first in designing all its individual circuits and connecting them in parallel, as illustrated in figure 5.2. In this figure, each block "operation I" stands for an operation associated to an instruction and executed by the ALU. The block has its output connected to a tristate buffer (See figure 5.3) [3]. The signals controlling the tristate buffer operation come from the IR depending on the opcode. We illustrate now the operation blocks.

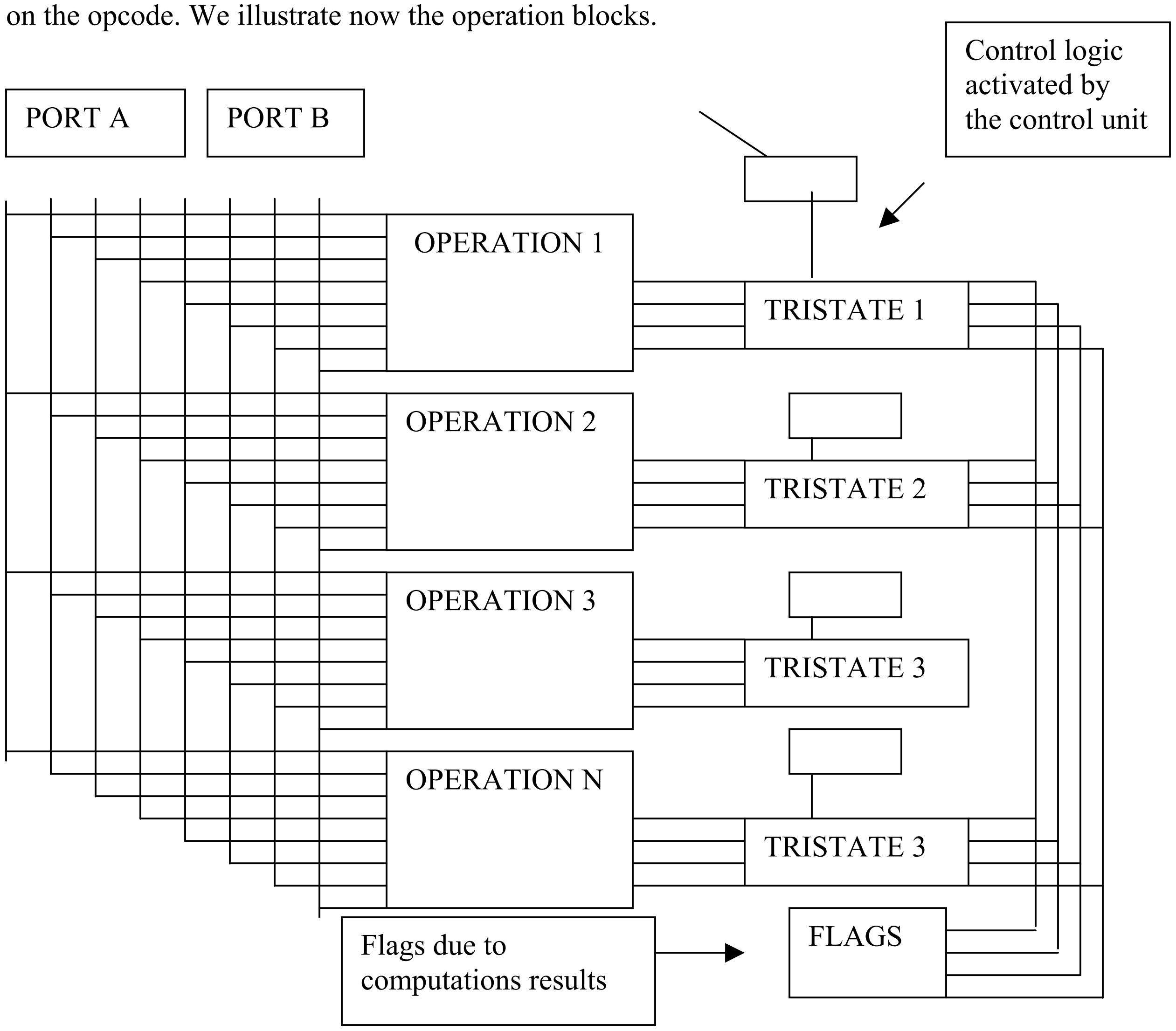


Figure 5.2 The ALU Structure

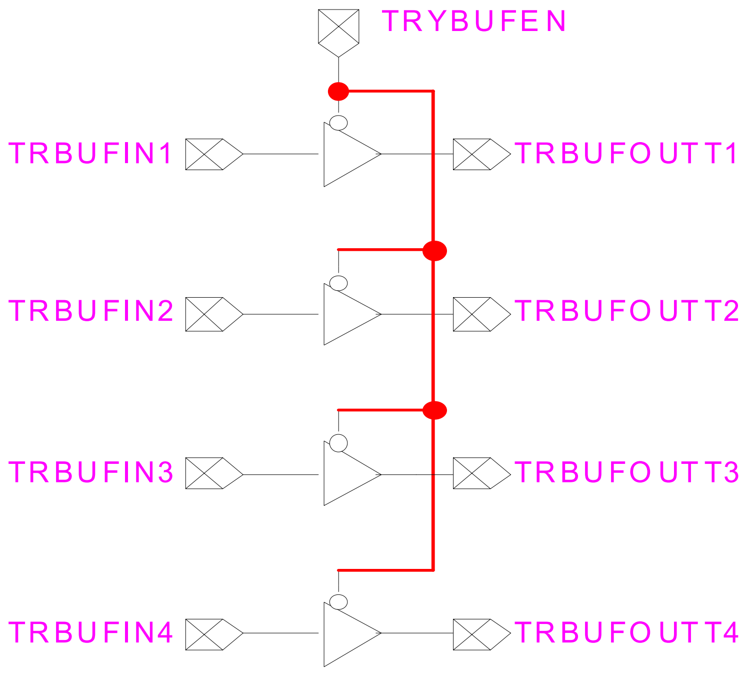


Figure 5.3 Tristate Buffer Implementation Circuit

5.4.1 The Adder and Subtractor

The adder can be designed for example using the carry ripple connection as illustrated by figure 5.4 [3]. After selecting the adder we have to do some testing as illustrated in figure 5.5. We proceed similarly with the subtractor tested as illustrated in figure 5.6.

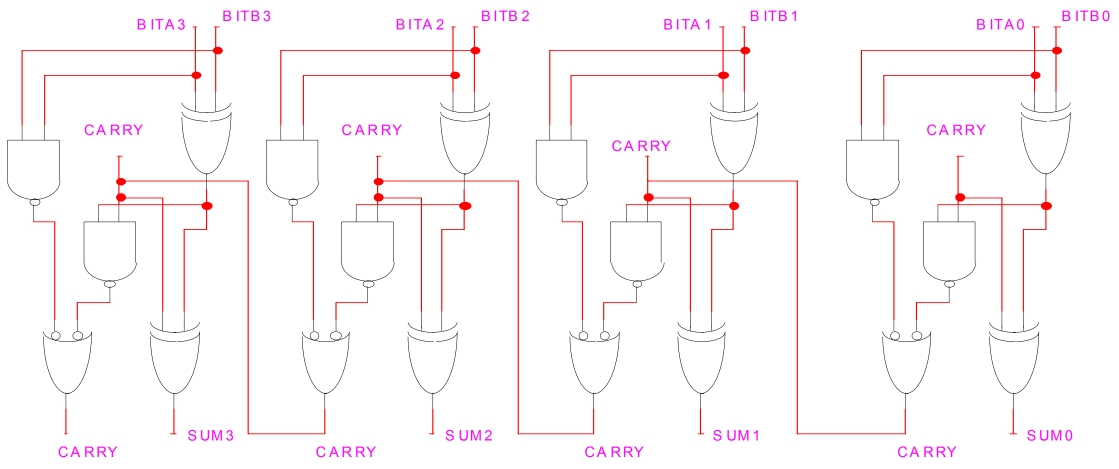


Figure 5.4 Adder Example

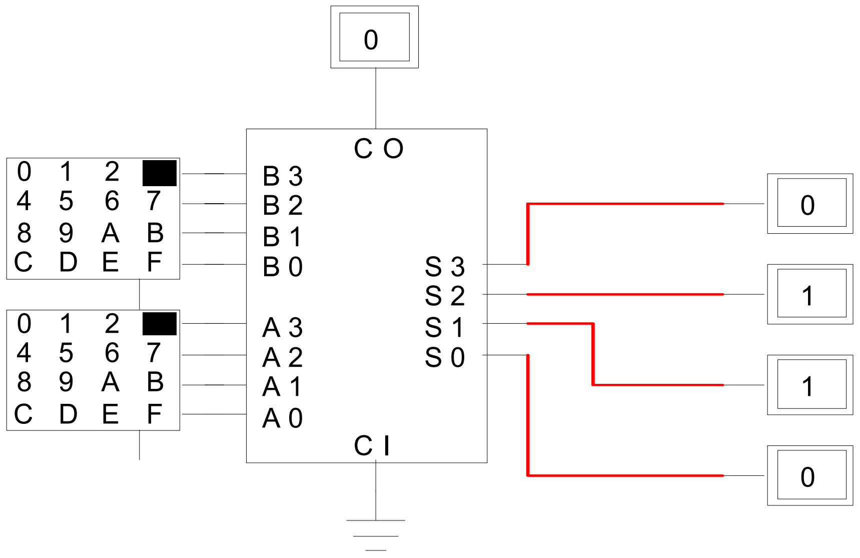


Figure 5.5 Example of Adder Circuit Testing

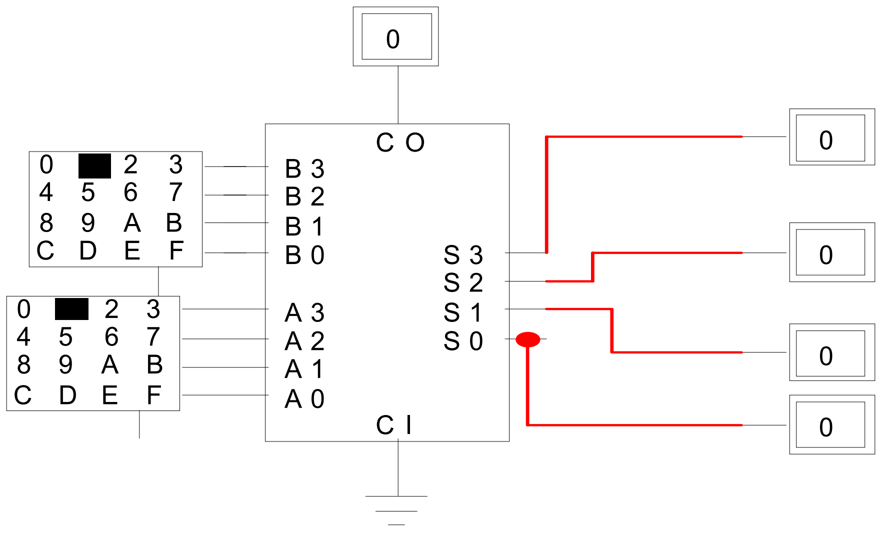


Figure 5.6 The Subtractor Circuit Implementation

5.4.2 Logical Bit wise operations: AND, OR, NOT.

The Bit wise logic functions takes words and bit by bit perform the corresponding function. These blocks can be done with parallel connections of gates as shown in figure 5.7 for the AND block. The OR and NOT operation blocks are equally designed.

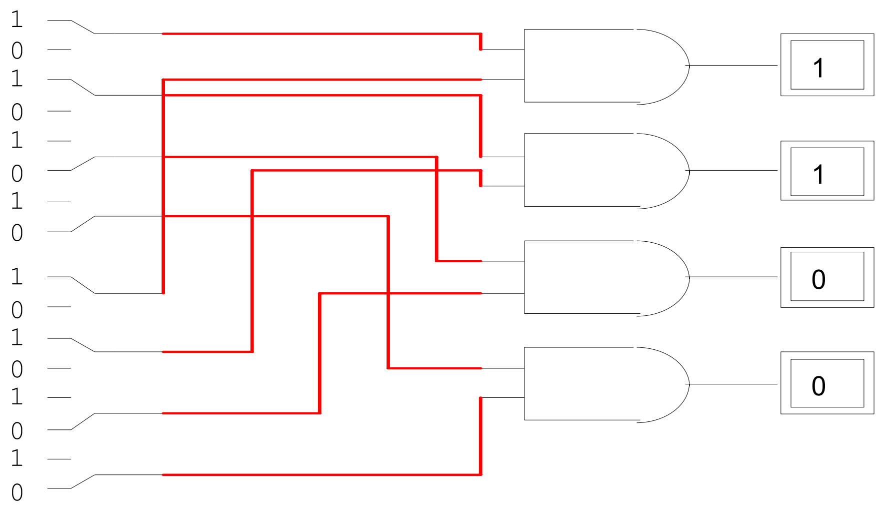


Figure 5.7 AND Circuit Implementation

5.4.3 Shift Right

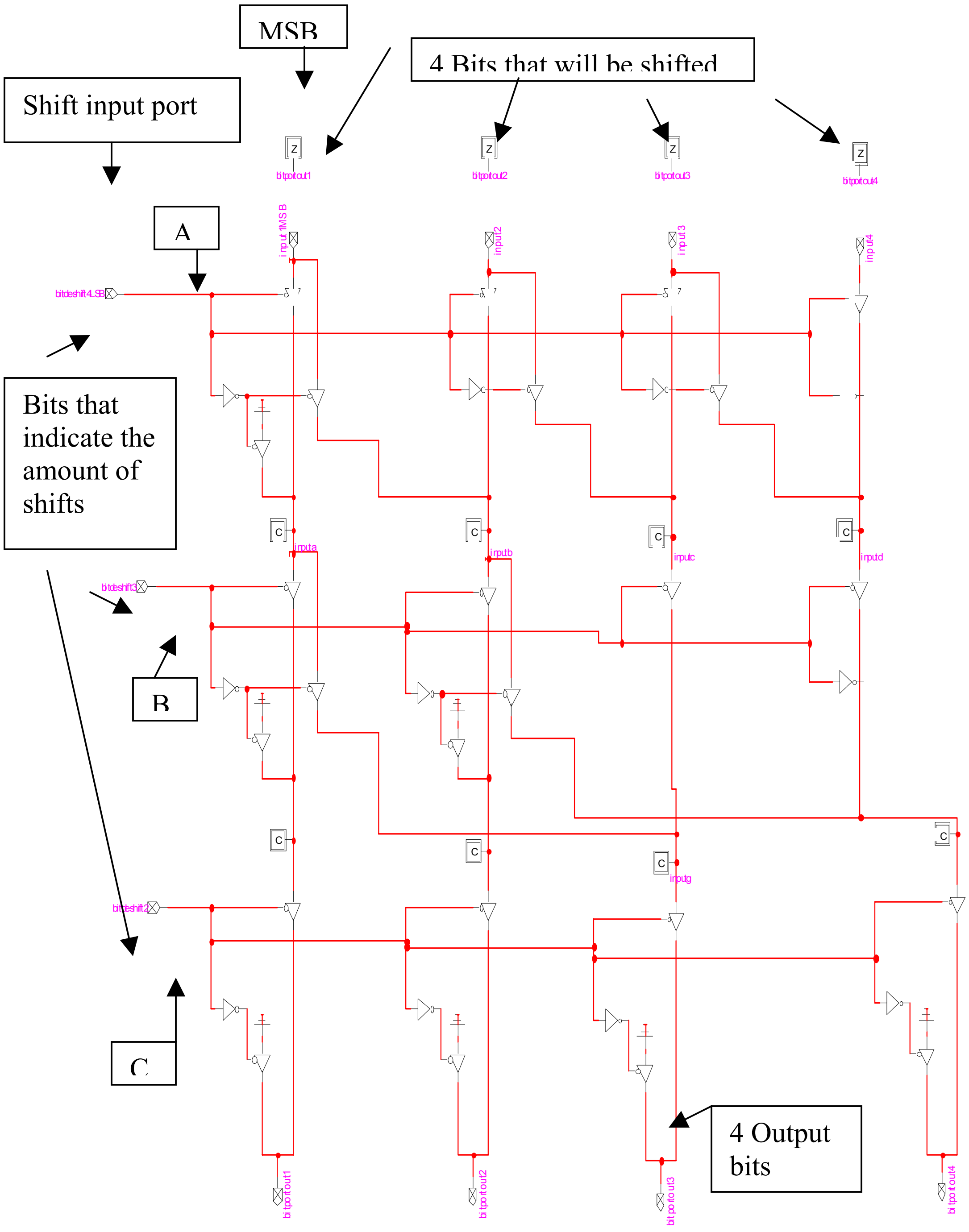

MSB
4 Bits that will be shifted
Shift input port
A
Bits that indicate the amount of shifts
B
C
4 Output bits

Figure 5.8 The Shift Right Implementation Circuit

In figure 5.8 shows the circuit used to execute the Shift Right instruction. This operation takes one string of binary bits and makes the specified shift places to the right, replacing the vacant places with zeroes. For example, atwo place shift to the string 1111 results in 0011. The circuit is composed by the processing hardware for the binary number that will get the shift places.

In figure 5.8, the string CBA specifies the times that shifting takes place. Thus, 001 will cause one shift to the right (1 x 2^0 = 1). Two places to the right (1 x 2 ^1 = 2) and so on. Notice that the maximum number of shifts is 4, since the data has 4 bits, so C = 1 yields a string of 0's. Figure 5.9 illustrates the mechanism used for the shifting decision.

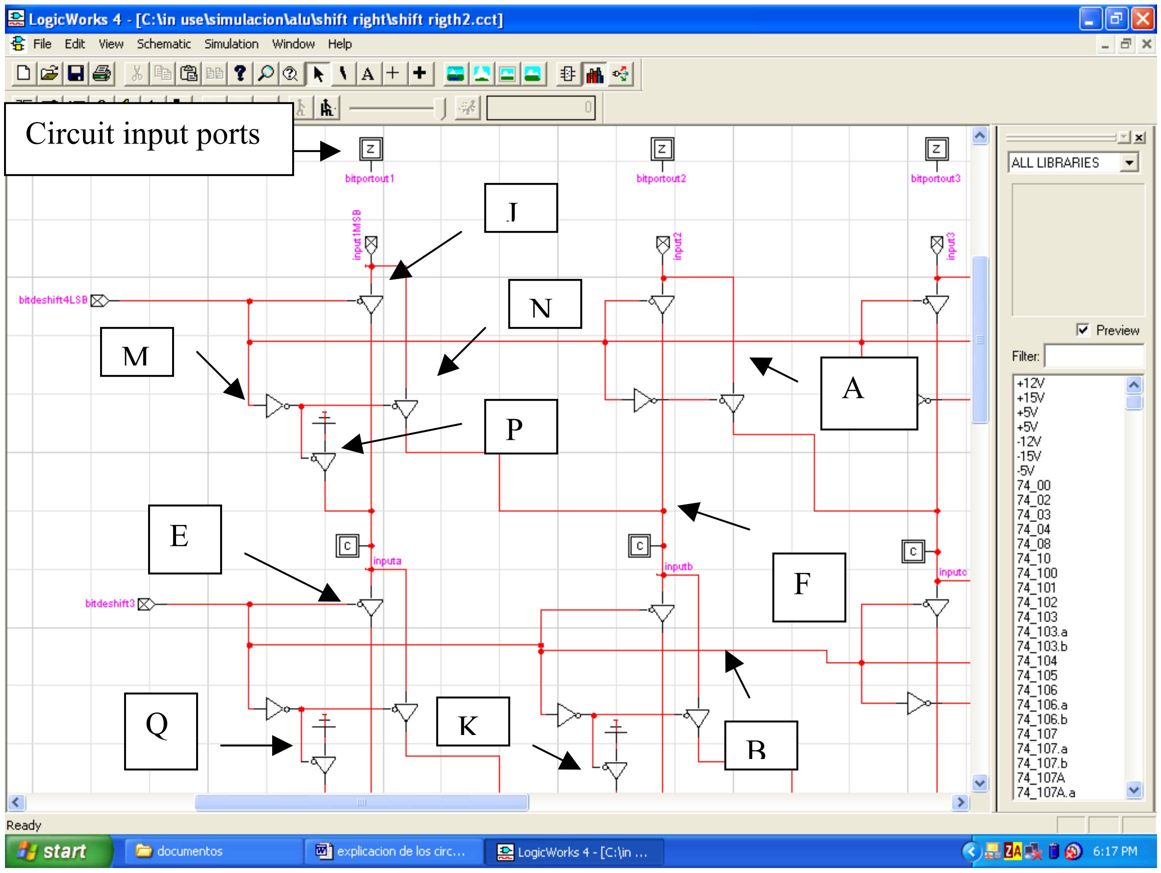


Figure 5.9 The Shift Right Instruction Mechanism

Shift Right function mechanism

Once the binary number that will be shifted is in the circuit input port, bits that specify the amount of shift has to be present also in the circuit shift input ports. Depending on the number used to indicate the amount of shifts, the first stage with the letter A (see figure 5.9) will be a zero or one. If it is zero, that zero will activate the tristate buffer with the letter A, and it will allow the data in the input port to pass to the next stage directly to the tristate buffer with the letter E.

The inverter with the letter M will receive a 0 that will change to 1, causing that the tristate buffers with letters N and P be deactivated and do not allow the flow of data through them. If the bit at stage A is 1, the tristate buffer with the letter J will be deactivated and will stop the flow of data through it. The inverter with the letter M will receive a 1 that will turn into a 0, this 0 will activate the tristate buffers N and P. The tristate buffer N will be responsible for the shifting process. This tristate buffer N is connected to the most significant bit and when activated, it allow the MSB to pass to the node labeled F as the second bit. The tristate buffer P will ensure that the vacant place is filled with a 0. Then the second bit of shift in the second stage labeled as B, will use the same mechanism to make further movements to the right of the new string of bits processed in the first stage.

The process can be similarly followed. Figure 5.10 shows a testing for the shift right operation. Recall that shifting to the right can be interpreted as dividing an unsigned number by 2.

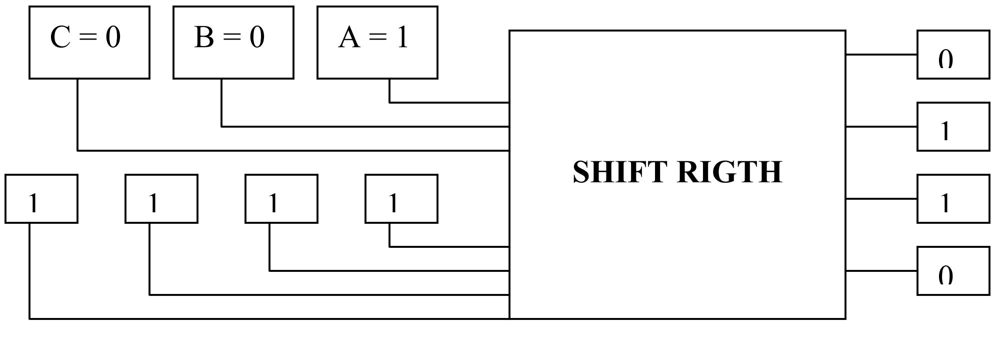


Figure 5.10 The Shift Right Circuit Testing: shifting 1100 once

5.4.4 **Arithmetic Shift**

The Arithmetic Shift instruction works basically in the same way as the shift right, but with a slight change. This change consists in that now, the grounds that fill the resulting vacancies are changed by a direct connection with the first bit of the number that will be shifted. This is said to be an arithmetic shift because the vacancy will be filled with the most significant bit of the number that will be shifted, thereby maintaining the sign bit. Figure 5.11 shows the circuit and figure 5.12 a testing for string 1100.

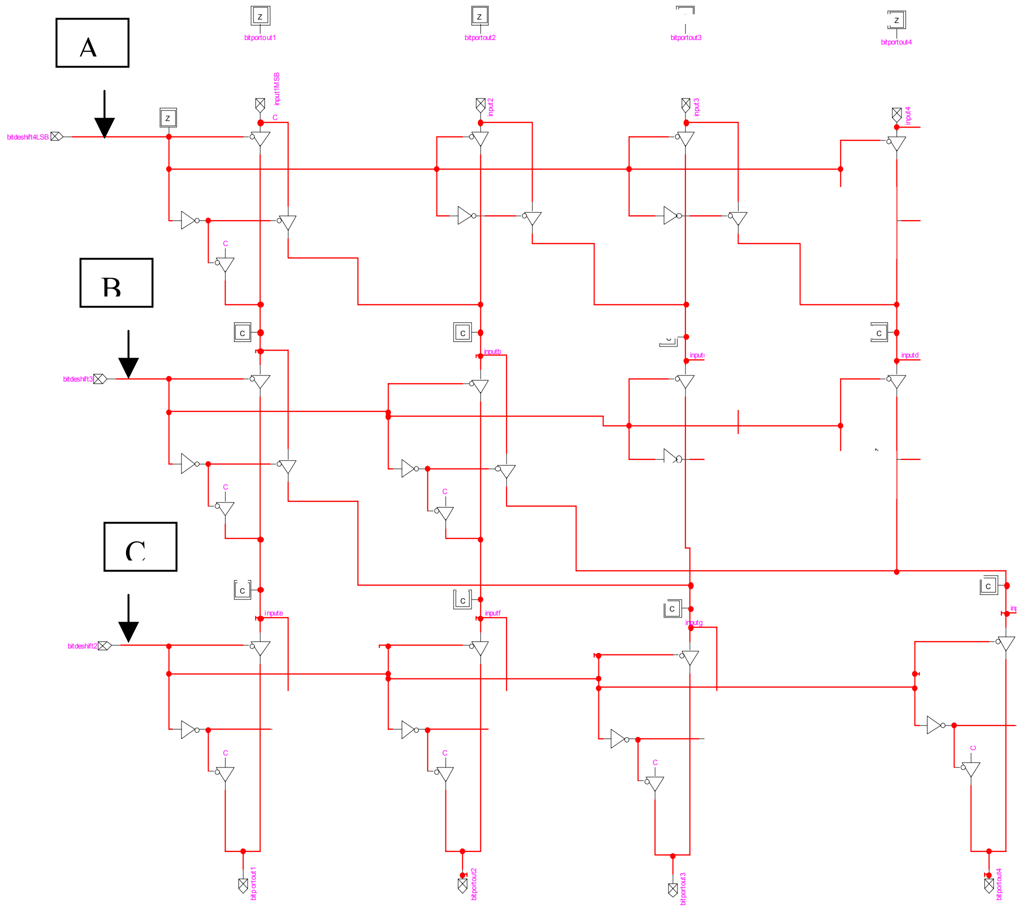


Figure 5.11 The Arithmetic Shift Implementation Circuit

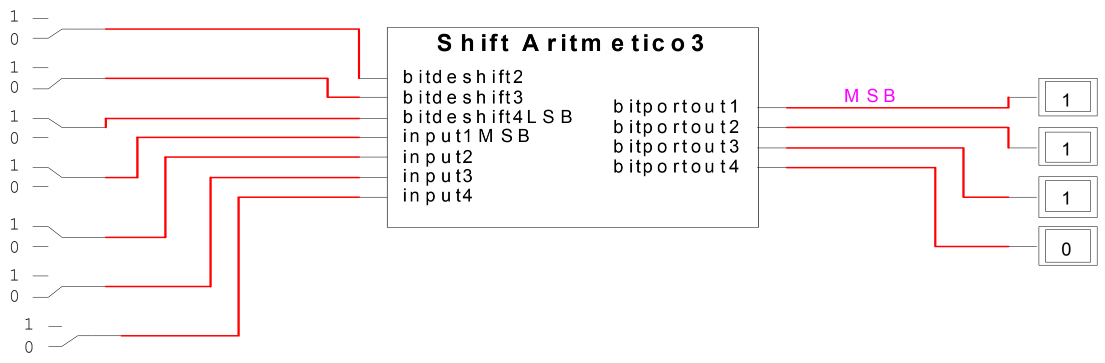


Figure 5.12 Arithmetic Shift Circuit Testing: shifting 1100 once

5.4.5 Circular Shift

The circular shift operation consists in circular permutations. The basic skeleton for the circuit is similar to that of the shift right, as illustrated in figure 5.13. The main difference is that the tristates originally connected to ground are now connected to one of the input bits. In figure 5.13 the boxed labels stand for the same input bit connections. The circuit was tested as always.

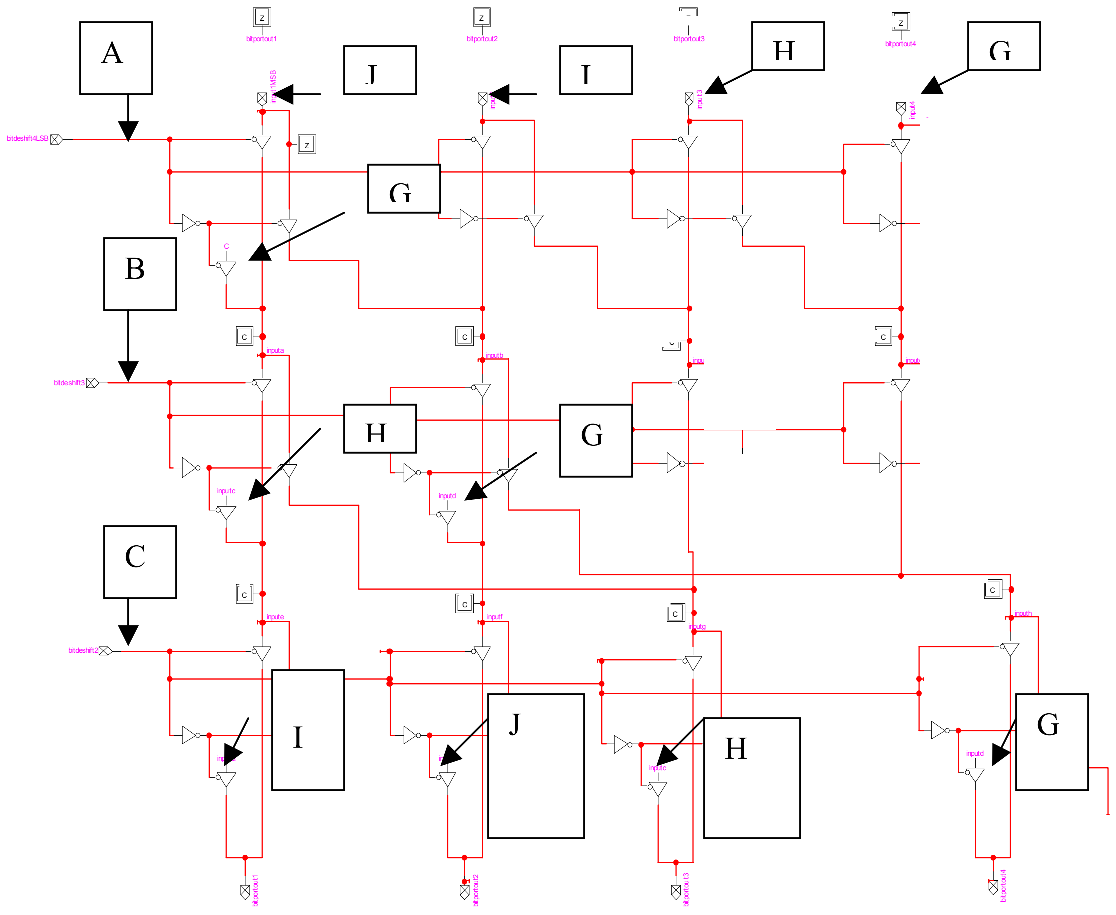


Figure 5.13 Circular Shift Circuit Implementation

5.4.6 Shift Left

The shift left circuit works in exactly the same way as the shift right, but the circuit configuration now makes the movement to the left. Figure 5.14 shows the circuit implementation.

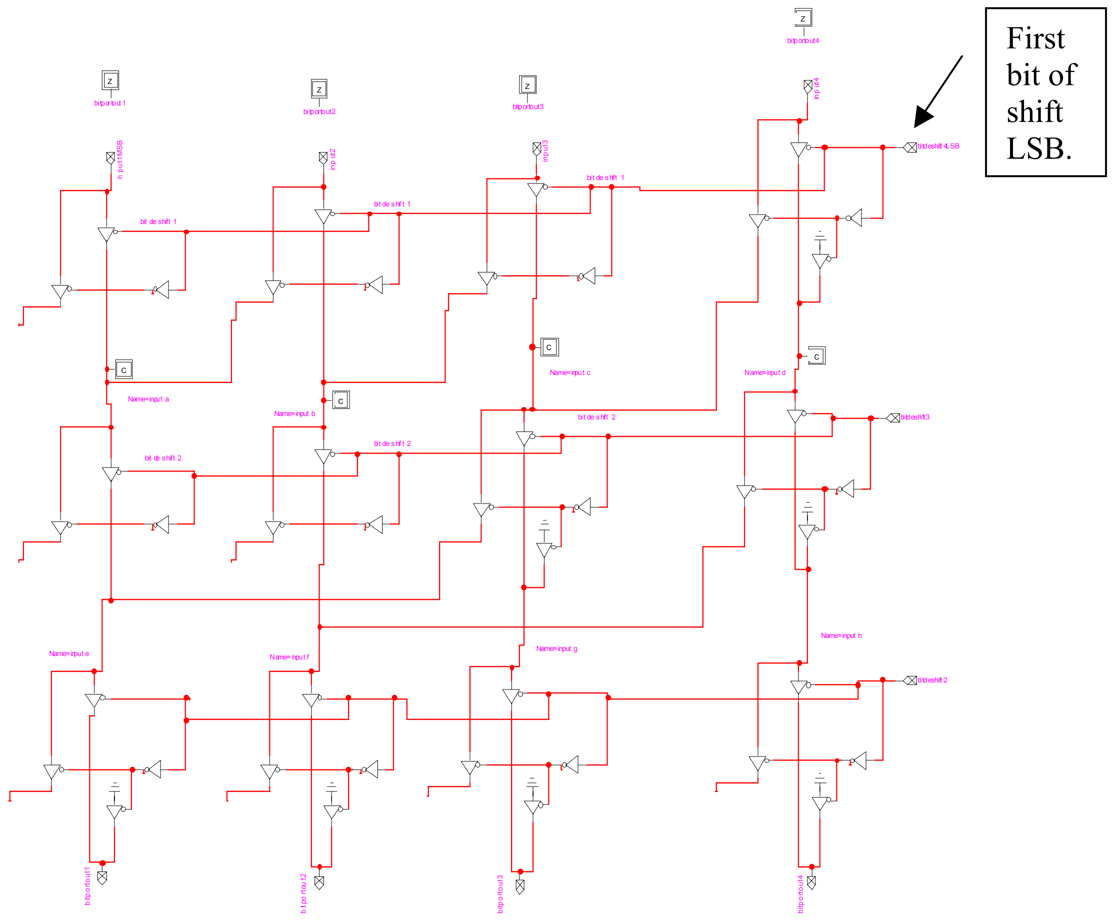


Figure 5.14 Shift Left Implementation Circuit

5.4.7 Final Arithmetic Logic Unit implementation

Once all the basic instructions circuits are designed and implemented individually, the ALU can be implemented adding flags. The data bus that feed the individual circuits is the same data bus that the microcontroller uses to transfer data between its components. All mathematical and logical calculations are executed at the same time, but only the desired calculation will be the one released to the ALU output port by means of the tristate buffer activated.

Figure 5.15 to 5.18 illustrate how the ALU circuits are connected. Caution should be taken with the significance of the input and output bits of every circuit. Mistakes can lead to miscalculations and continue through the rest of the instruction execution.

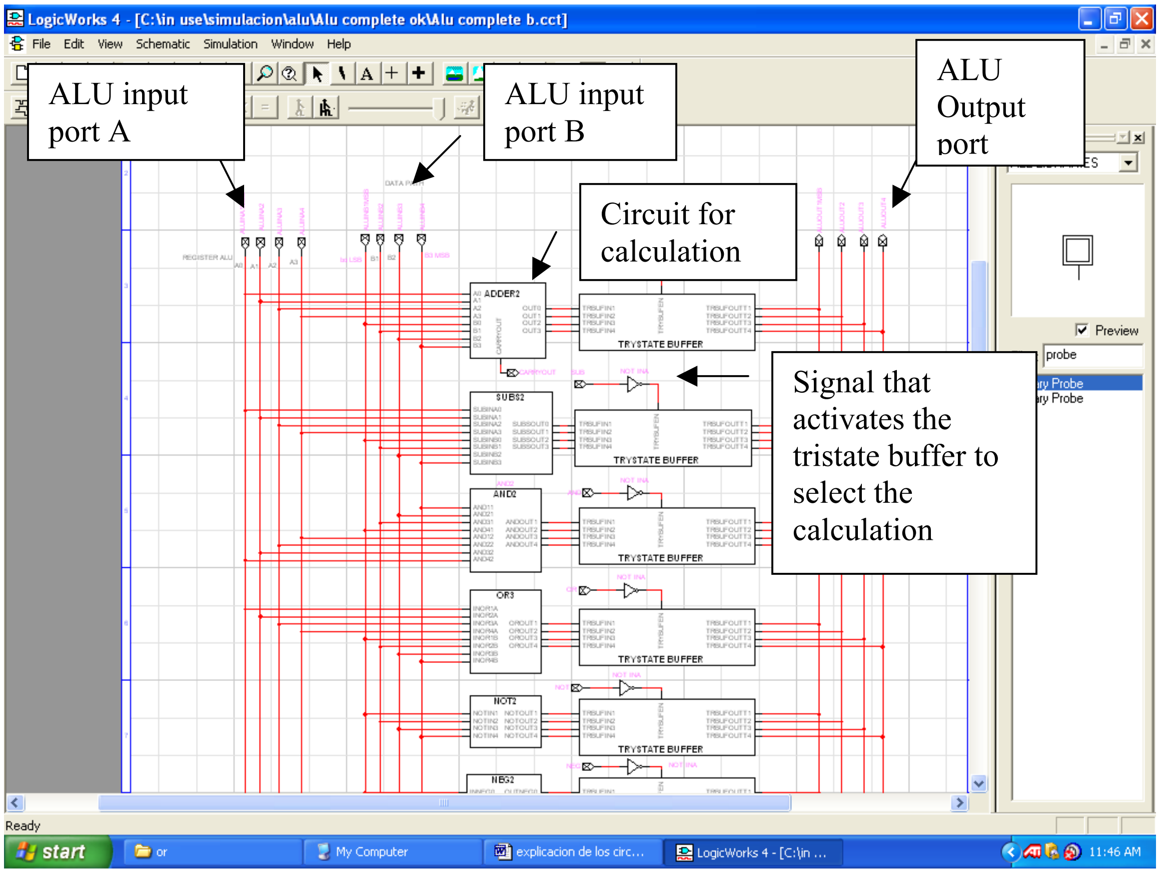

LogicWorks 4 - [C:\in use\simulacion\alu\Alu complete ok\Alu complete b.cct]
File Edit View Schematic Simulation Window Help
ALU input port A
ALU input port B
ALU Output port
Circuit for calculation
Signal that activates the tristate buffer to select the calculation
ADDER2
TRYSTATE BUFFER
SUBS2
AND2
OR3
NOT2
NEG2
Preview
probe
Ready
start
My Computer
explicacion de los circ...
LogicWorks 4 - [C:\in ...
11:46 AM

Figure 5.15 The Arithmetic Logic Unit Implementation (top view)

.

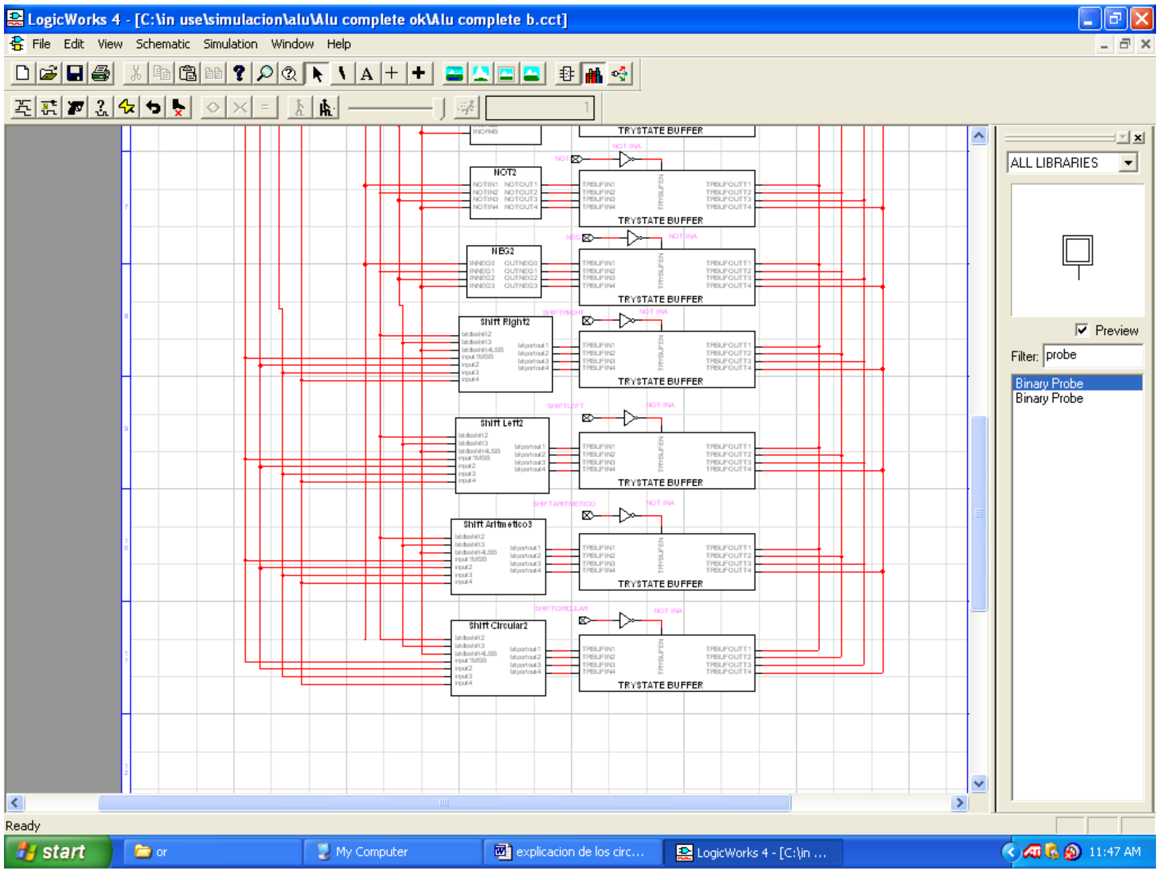
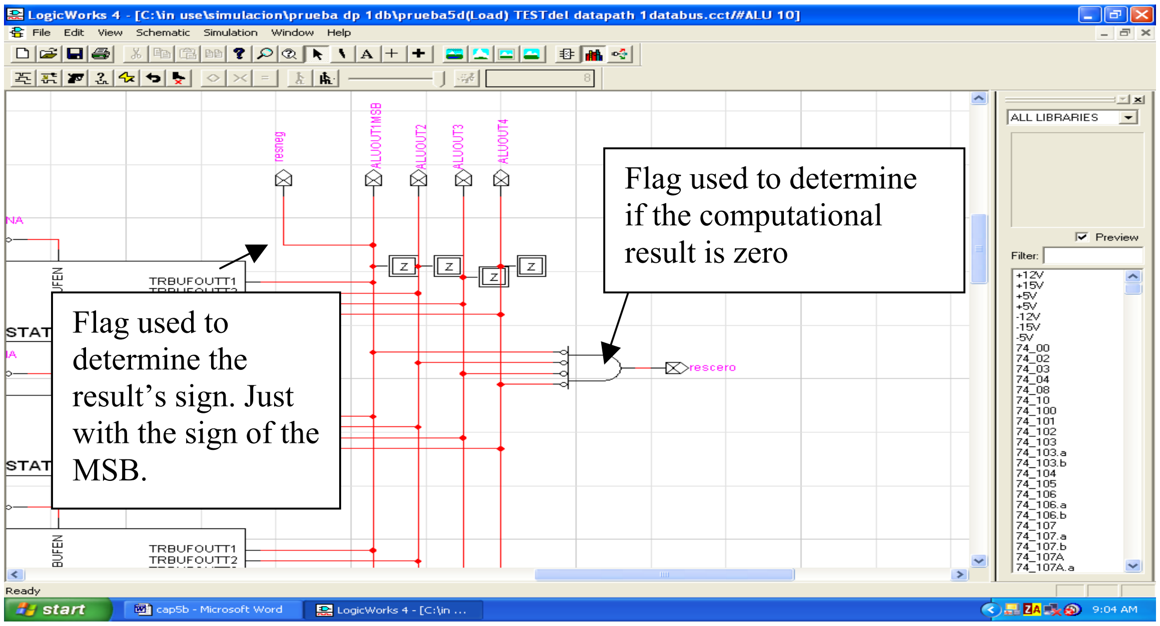

LogicWorks 4 - [C:\in use\simulacion\alu\Alu complete ok\Alu complete b.cct]
File Edit View Schematic Simulation Window Help
NOT2
NEG2
Shift Right2
Shift Left2
Shift Aritmetico3
Shift Circular2
TRYSTATE BUFFER
ALL LIBRARIES
Preview
Filter: probe
Binary Probe
Binary Probe
Ready
start
My Computer
explicacion de los circ...
LogicWorks 4 - [C:\in ...
11:47 AM

Figure 5.16 The Arithmetic Logic Unit Implementation (bottom view)


LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#ALU 10]
File Edit View Schematic Simulation Window Help
resneg
ALUOUT1MSB
ALUOUT2
ALUOUT3
ALUOUT4
TRBUFOUTT1
TRBUFOUTT2
rescero
ALL LIBRARIES
Preview
Filter:
Ready
start
capSb - Microsoft Word
LogicWorks 4 - [C:\in ...
9:04 AM

Flag used to determine if the computational result is zero

Flag used to determine the result’s sign. Just with the sign of the MSB.

Figure 5.17 The ALU Flags Hardware

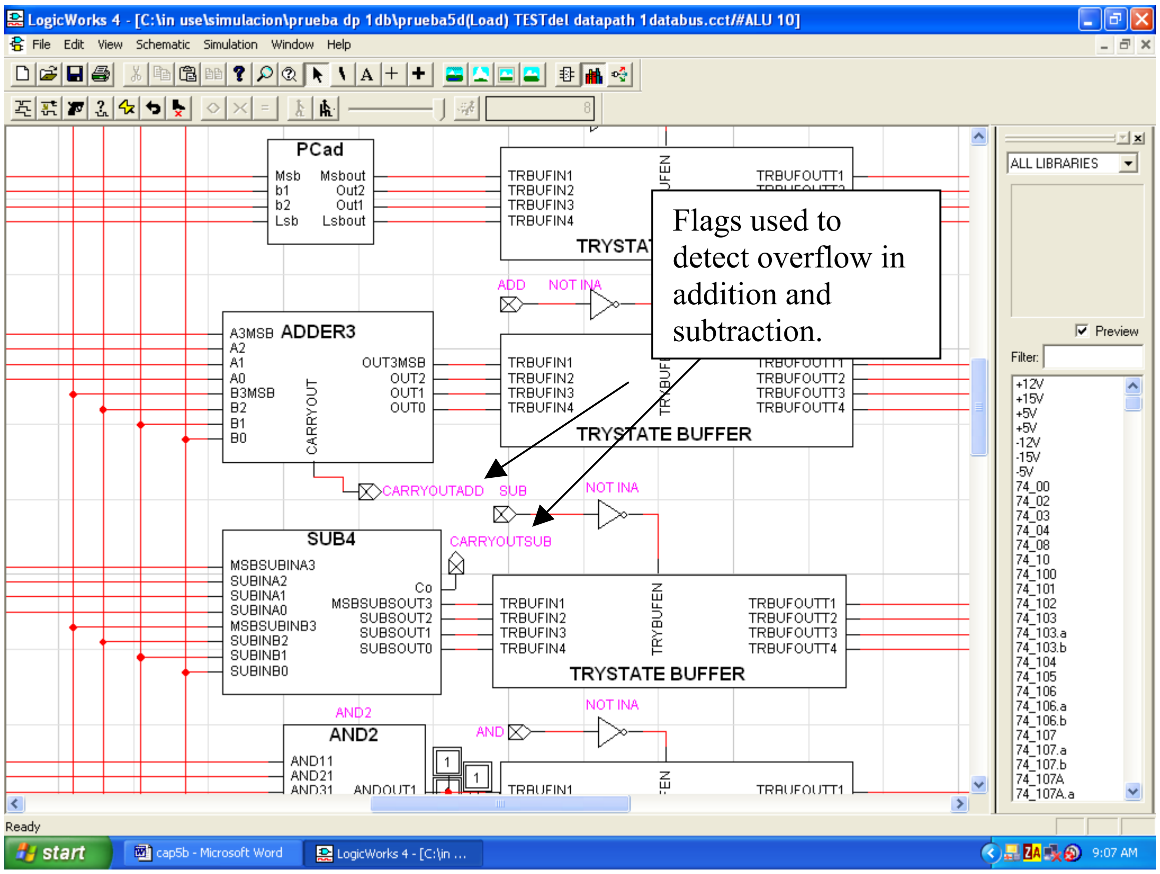

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#ALU 10]
File Edit View Schematic Simulation Window Help
PCad
Msb b1 b2 Lsb
Msbout Out2 Out1 Lsbout
TRBUFIN1 TRBUFIN2 TRBUFIN3 TRBUFIN4
TRBUFOUTT1 TRBUFOUTT2 TRBUFOUTT3 TRBUFOUTT4
TRYSTATE BUFFER
Flags used to detect overflow in addition and subtraction.
ADD
NOT INA
ADDER3
A3MSB A2 A1 A0 B3MSB B2 B1 B0
OUT3MSB OUT2 OUT1 OUT0
CARRYOUT
CARRYOUTADD
SUB
CARRYOUTSUB
SUB4
MSBSUBINA3 SUBINA2 SUBINA1 SUBINA0 MSBSUBINB3 SUBINB2 SUBINB1 SUBINB0
Co MSBSUBSOUT3 SUBSOUT2 SUBSOUT1 SUBSOUT0
TRYBUFEN
AND2
AND11 AND21 AND31
ANDOUT1
AND
ALL LIBRARIES
Preview
Filter:
+12V +15V +5V +5V -12V -15V -5V 74_00 74_02 74_03 74_04 74_08 74_10 74_100 74_101 74_102 74_103 74_103.a 74_103.b 74_104 74_105 74_106 74_106.a 74_106.b 74_107 74_107.a 74_107.b 74_107A 74_107A.a
Ready
start
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
9:07 AM

Figure 5.18 Flags Used for Overflow

Figure 5.19 illustrates the final implementation of the ALU. As a test, the ALU receives 1111 in data port A and 1111 in data port B; the subtraction operation is executed leading as a result 0000 in the output data port and the corresponding flag is activated.

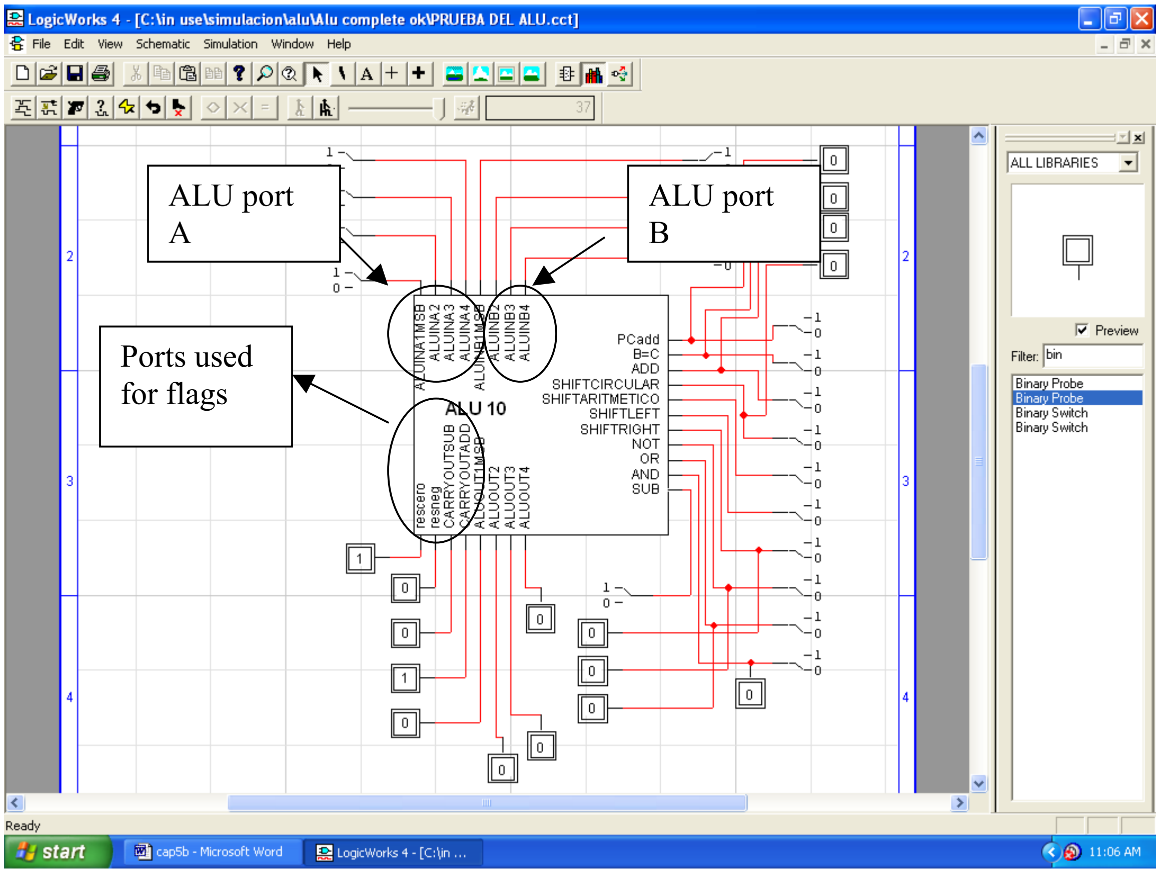


Figure 5.19 The Arithmetic Logic Unit Testing (top view)

5.5 STEP VI The Register File

The Register File stores data retrieved from memory input port resulting from operations. All temporary data used by the microcontroller to perform its operations is also stored in the register file. The Register File structure design consists of three stages: The register selection stage, the input stage and the output stage.

5.5.1 The Register File Selection Stage

This stage is shown in figure 5.20. The instruction word identifies three parameters: Ra, Rb and Rc. Each of these parameters, when referring to registers, are actually addresses that identify a register from the register file. Since Ra, Rb and Rc as shown in figure 5.1 have three bits, the register file has 8 registers.

One register is selected by means of a decoder 3x 8 (device I in figure 5.20). Q0 activates register 0, Q1 activates register 1 and so on. The selection of S2 S1 and S0 given by the equation Sj = Raj (ACTRADB1) + Rb (ACTRADB1) + Rc (ACTRADB1), where Raj is bit j of Ra, and ACTRADB1 is a signal from the control unit to use Ra. At a certain moment, the control unit will activate one and only one of ACTRADB1 signals to indicate which register is assigned to Ra, Rb or Rc of the instruction word.

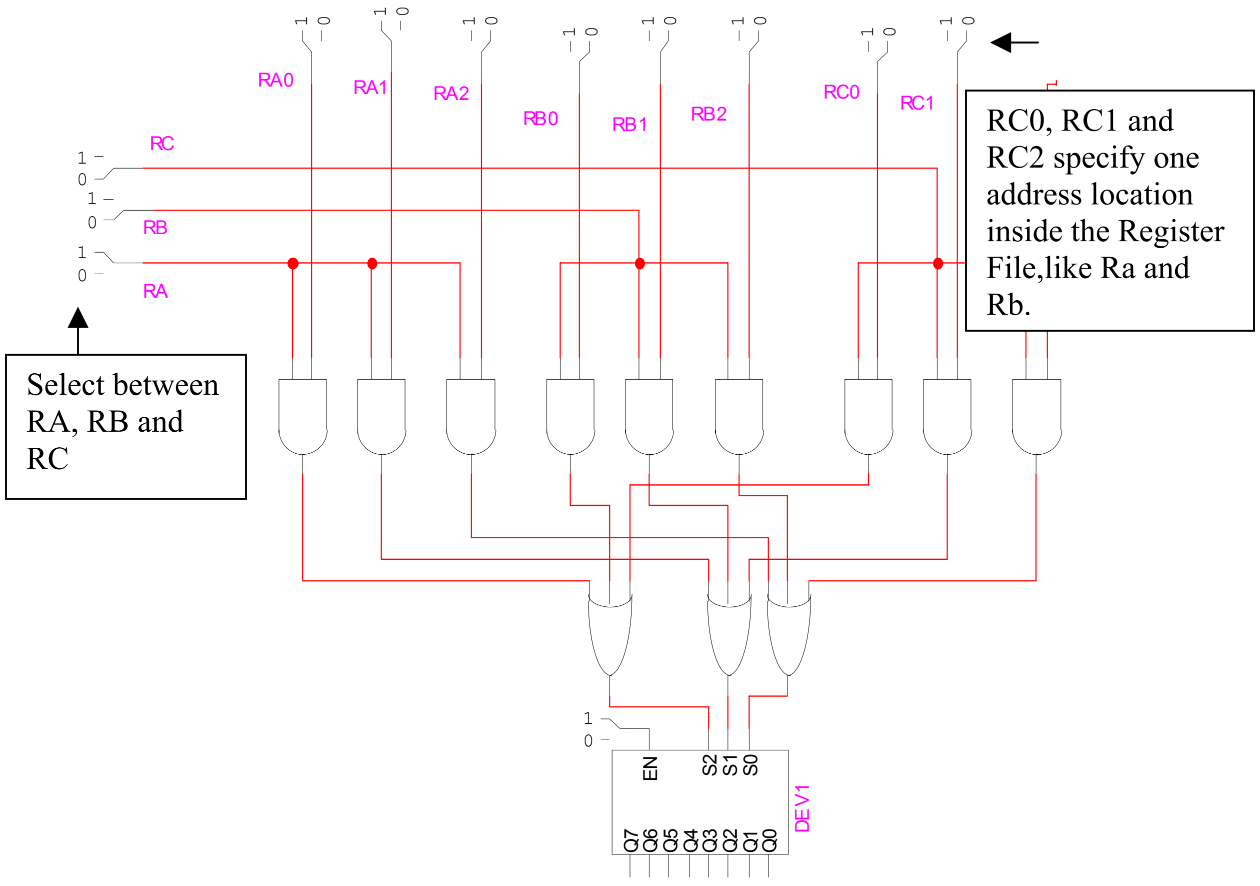

RA0
RA1
RA2
RB0
RB1
RB2
RC0
RC1
RC
RB
RA
RC0, RC1 and RC2 specify one address location inside the Register File,like Ra and Rb.
Select between RA, RB and RC
EN
S2
S1
S0
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
DEV1

Figure 5.20 The Register File Selection Stage

5.5.2 The register file input stage

Figure 5.21 illustrates the input stage for register j (j = 0,1,2…7) of the register file.

- The register inputs are connected to the data bus.
- Each register clock is activated with the following equation CK = (READ)(Qj), where Qj comes from decoder selection.

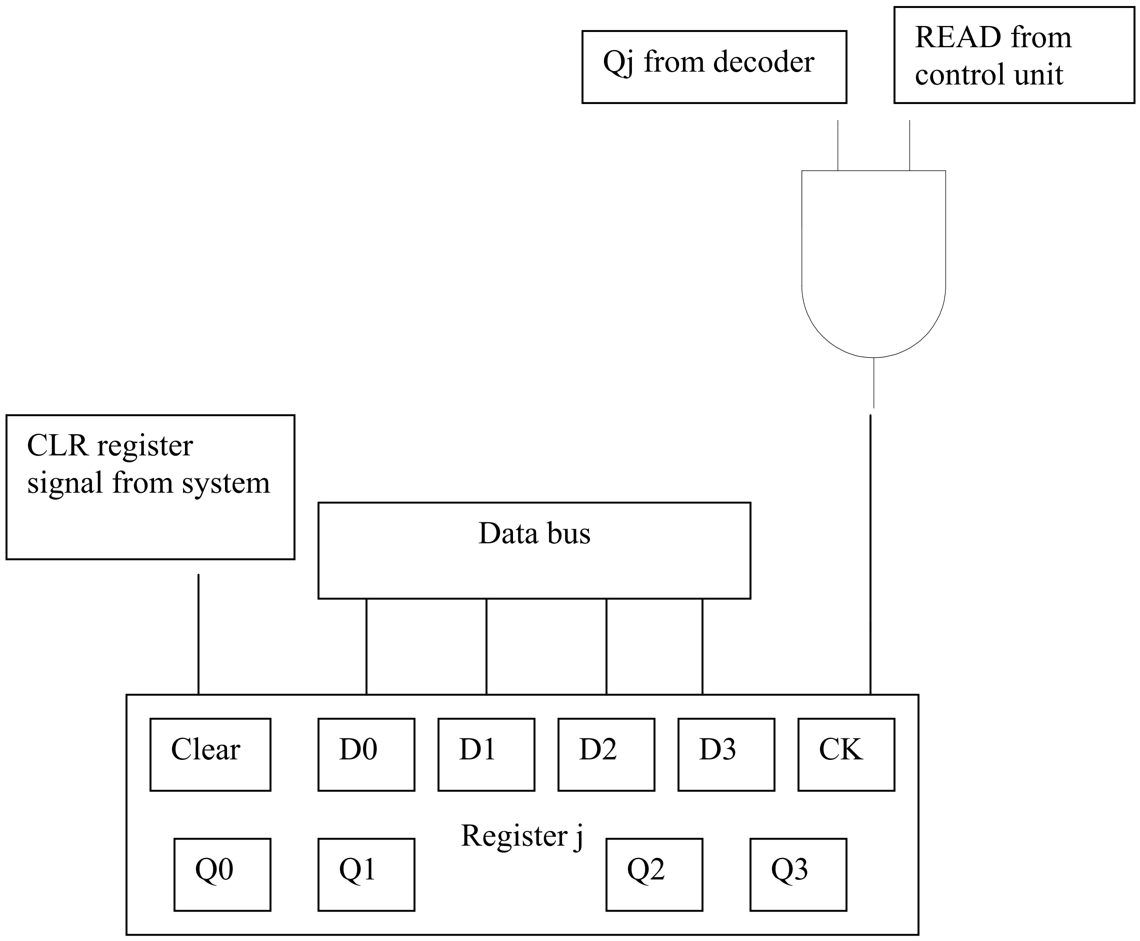


Figure 5.21 Module Rj of the register file: Input Stage

Read is a pulse generated by the control unit. All eight registers are connected similarly. Only the connection to the decoder changes for each case. Since the register will store the data only after a "CLK pulse", and CLK = Qj(from decoder). Read pulse, only one register will store the data.

5.5.3 The Register file output stage

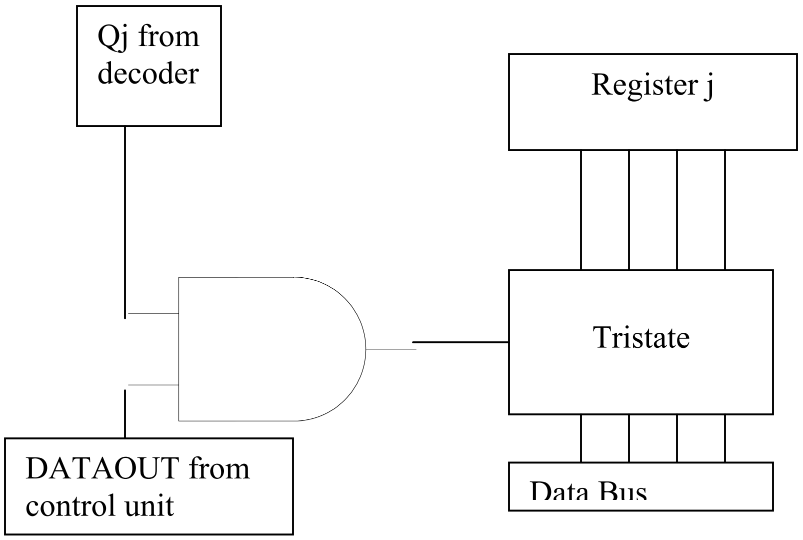


Figure 5.22 Module Rj of the register file; output stage

The register's outputs are connected to the data bus via tristate buffers. For register Rj, the tristate is activated by Qj from the decoder at the selection stage and a signal from the control unit requiring the data out. The basic module is shown in figure 5.22.

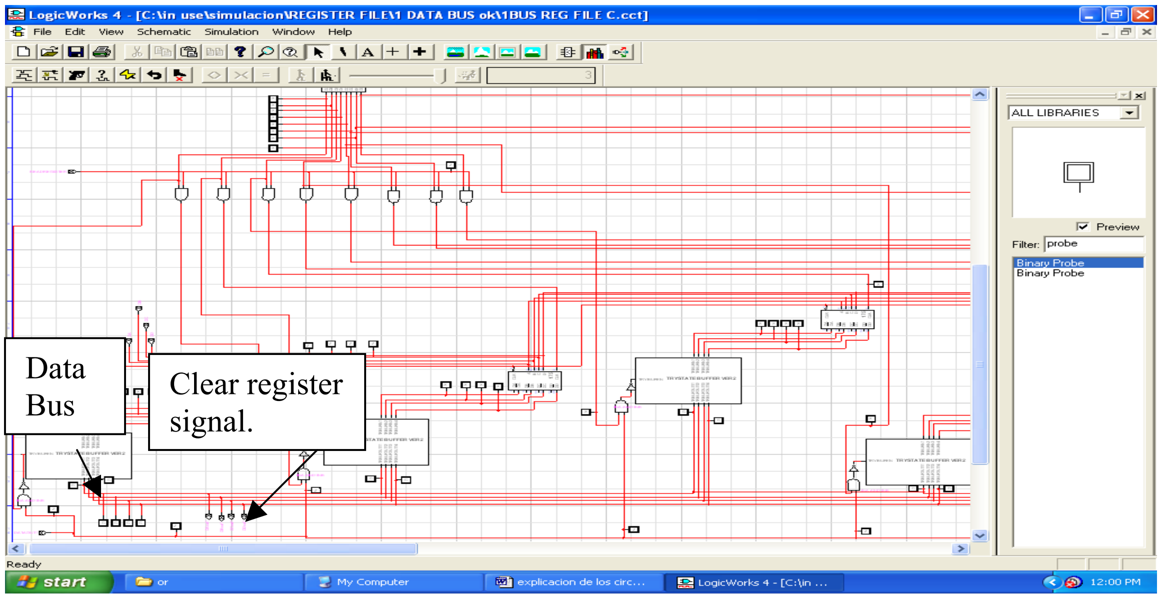


Figure 5.23 The Register File input Stage

5.5.4 Register File Implementation

A partial view of Logic Works schematic for the register file is shown in figure 5.23.Figure 5.24 illustrates the Register File testing. Register File input ports QA, QB, QC and QD will be connected to the Arithmetic Logic Unit output port to store the processed result from ALU. This figure presents an example of the Register File function mechanism. The address of Ra is 0000. In order to use the address of Ra, the Register File signal for Ra must be activated; this is the label B. In order to store data from the data bus; the READ REGISTER signal must be activated. To release the data specified by the address of Ra to the output port, the DATA OUT signal must be activated. The CLRL signal labeled with the letter E is used to erase any data in any register.

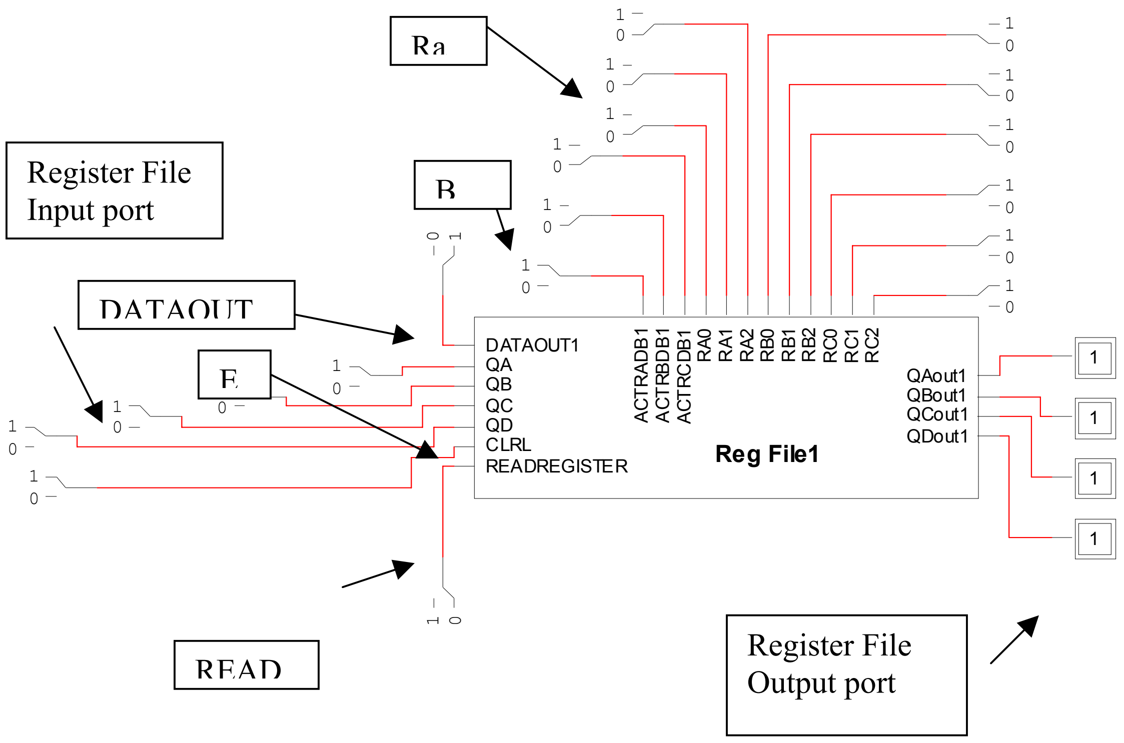


Figure 5.24 The Register File Testing

5.6 STEP VII: The Instruction Register

The Instruction Register is the register that holds the instruction word for execution. The IR is connected to the Register File and the Control Unit (discussed later). Note that from the instruction format (Figure 5.1) this register has to be 18 bits long und thus uses 18 flip-flops. It has two control lines, one to read the data and the other one to clear the register.

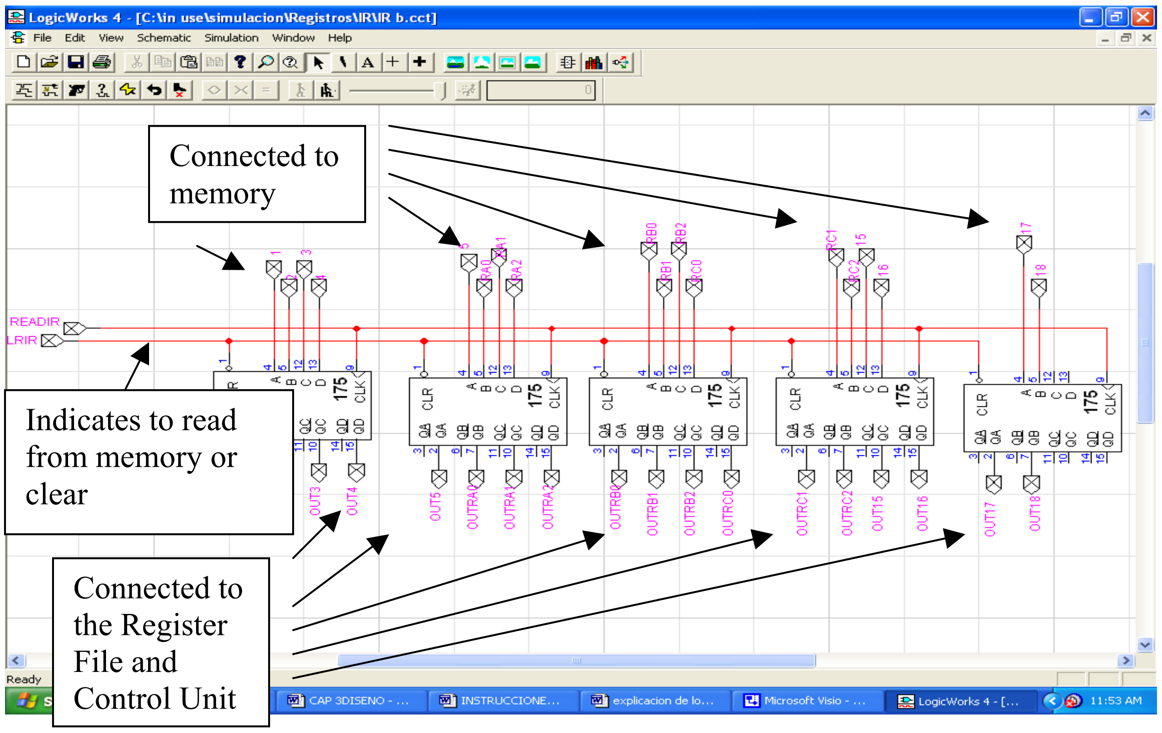

LogicWorks 4 - [C:\in use\simulacion\Registros\IR\IR b.cct]
Connected to memory
Indicates to read from memory or clear
Connected to the Register File and Control Unit
READIR
LRIR
175
CLR
CLK
OUT3
OUT4
OUT5
OUTRA0
OUTRA1
OUTRA2
OUTRB0
OUTRB1
OUTRB2
OUTRC0
OUTRC1
OUTRC2
OUT15
OUT16
OUT17
OUT18
Ready

Figure 5.25 The Instruction Register Implementation

5.7 STEP VIII: The Data Path

5.7.1 Basic Data Path

In order to make useful all the elements already discussed it is necessary to provide a path for communication between them to transfer data from one to another. Figure 5.26 illustrates the interconnection of the elements already discussed; they form the simplest microcontroller data path for this work. In this figure the control signals from the control unit are not shown. This data path can perform the basic microcontroller instructions and will be used as the basis to develop more complex instructions. As more complex instructions are added, this data path undergoes an evolution into a more complex one, adding more hardware in parallel to this configuration.

To test the feasibility of basic instructions this data path can process data provided by switches as shown in figure 5.27. Switches can be used to store values in the Register File. The address lines of Ra, Rb and Rc are connected from the IR to the Register File to access the data. The Register File output port is connected to the Arithmetic Logic Unit input ports to perform the logic and mathematical operations. The ALU output port is connected to the Register File input port to store results.

In figure 5.26 one register is added to the ALU port A. This is because this is a one data bus microcontroller and one value must be stored in that register in order to use the next clock cycle to put the second operand in the ALU port B and then execute the instruction with both operands. Another register is used at the ALU output port to hold results between clock cycles. Finally the ALU output port is connected to the Register File to store results.

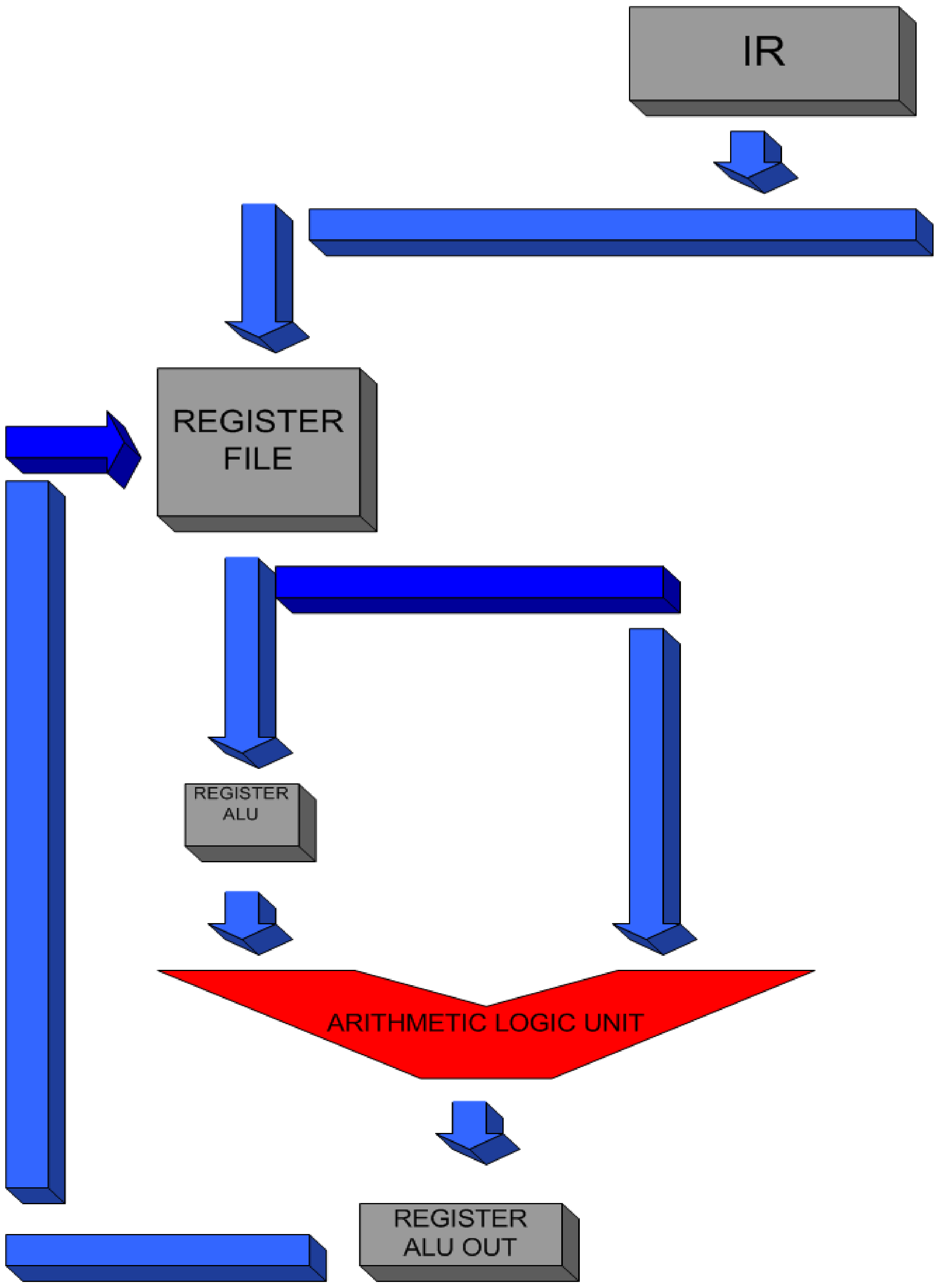
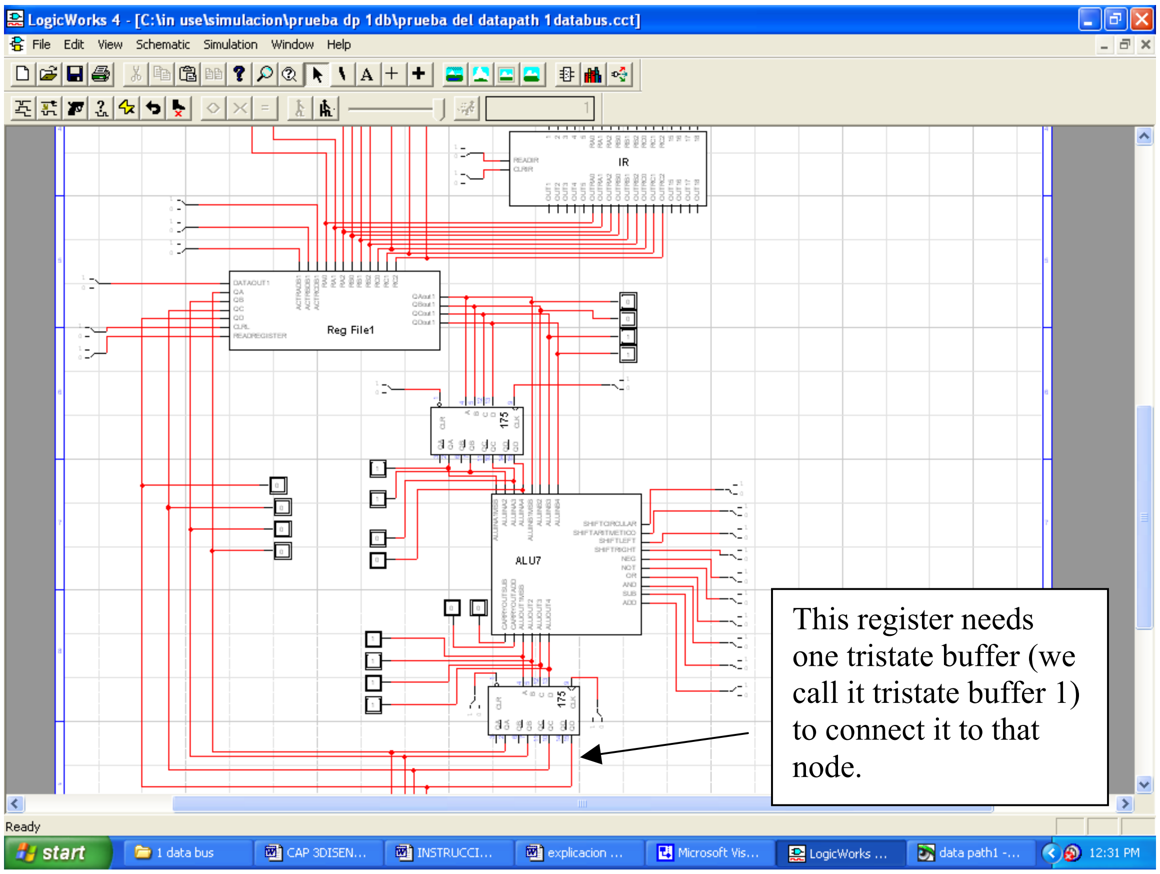


Figure 5.26 The Resulting Microcontroller Data Path


LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba del datapath 1databus.cct]
File Edit View Schematic Simulation Window Help
IR
Reg File1
175
ALU7
175
Ready
start
12:31 PM

This register needs one tristate buffer (we call it tristate buffer 1) to connect it to that node.

Figure 5.27 The Basic Microcontroller Data Path

5.7.2 Data Path with Immediate Operations

At this point, when adding new hardware to implement new new instructions, there are some details that should be taken care of, in particular:

1) For the new hardware:

- Control signals
- Instruction Register related logic
- Connection to buses and other blocks

2) Overall issues such

    - Signal conflict
    - Delays

The data path is next modified to include other arithmetic and logic operations using the ALU namely, the immediate addressing mode operands.

The immediate values are put in bits 0 – 3 of the instruction register.The data path modification consists in making a connection between those immediate values in the Instruction Register and the ALU port B. But the connection cannot be done directly because the values in the Register File can cause conflict with those in the data path. To solve this problem a tristate buffer is used to isolate the data in the Register File from those in the data bus as shown in figure 5.28. New parts added in the data path are identified with lines. The Logic Works Schematic is shown in figure 5.29.

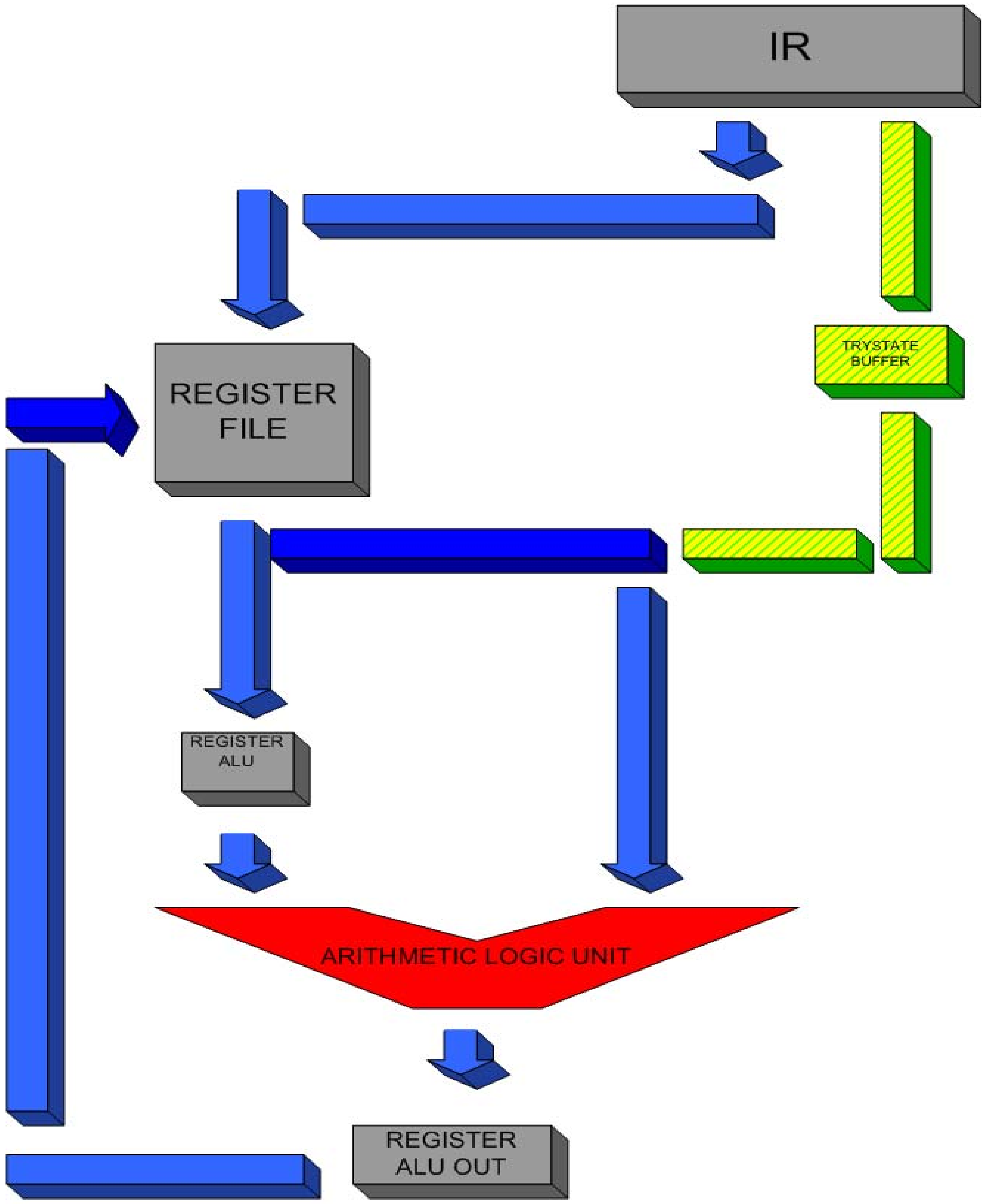


Figure 5.28 Added Elements for Immediate Instructions Execution

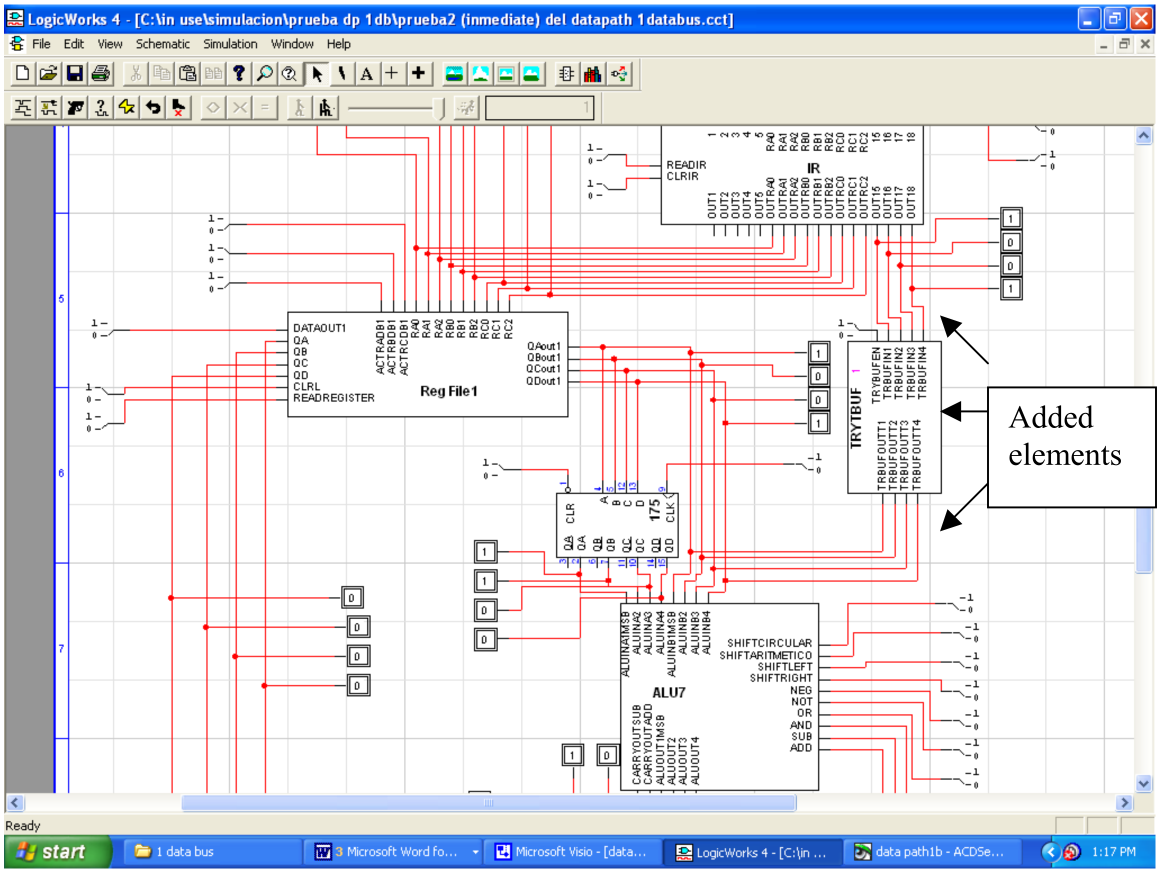

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba2 (inmediate) del datapath 1databus.cct]
File Edit View Schematic Simulation Window Help
IR
Reg File1
TRYTBUF
175
ALU7
Added elements
Ready
start
1:17 PM

Figure 5.29 The Immediate Instructions Circuit Implementation

5.7.3 Shift Operation

The next implemented instruction is the Shift instruction. All Shifts instructions use in their instruction format a group of bits called count. Those bits determine if the shift will be executed with the count bits or with data in the Register File. This suggests that a combinational circuit needs to be added to the data path to perform this logic decision. This circuit is known as “Count Decoder”. It has to be connected to the count bits in the IR because it will use those bits to take its decision and is discussed later. The circuit is shown in figure 5.30.

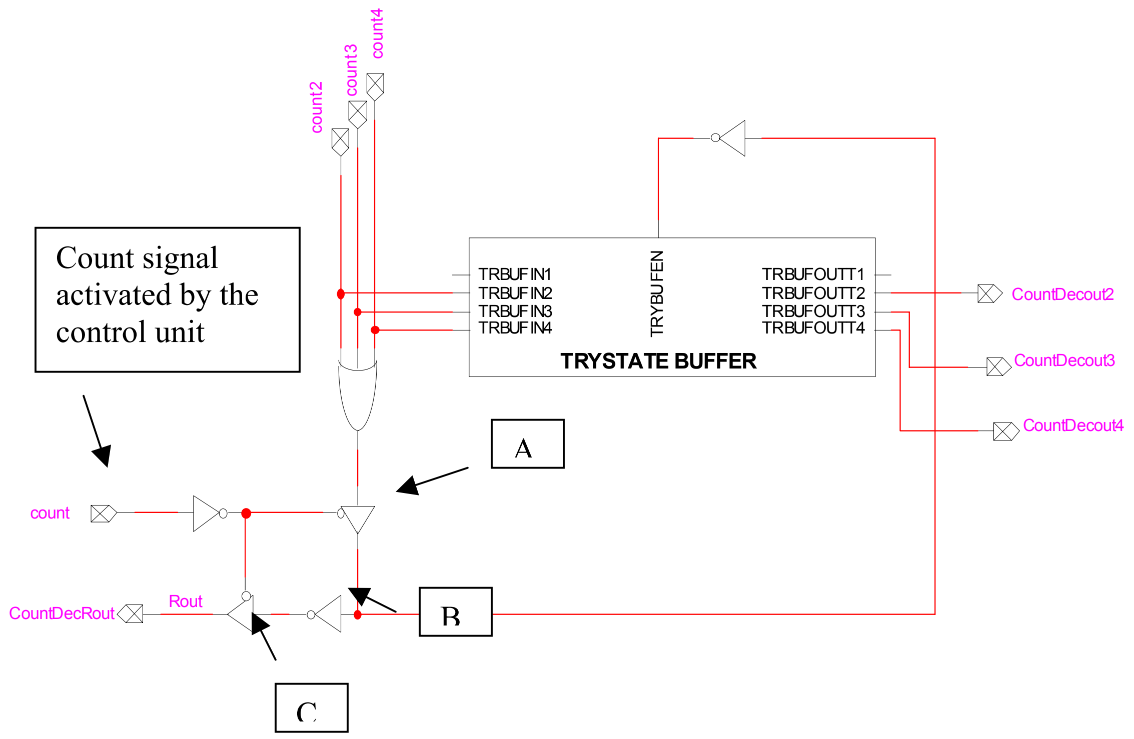


Figure 5.30 The Count Decoder

Now we explain how the decoder function. The count bits are the least significant bits of the instruction word in the shift instruction. When the count signal is activated, the Count Decoder circuit verifies the count bits condition and makes its logical decision. The tristates labeled A and C receive a low from the inverter and are automatically activated. If all the count bits are zero (000), the inverter labeled B will receive a low voltage that turns into a high signal to the Rout port. The count decoder will automatically send a signal to the data out signal port of the Register File to release the data specified by Rc. The tristate buffer in the Count Decoder receives a high signal due to its inverter, but is not activated. If all count bits are not zero the inverter B will receive a high signal that turns into a low signal and the Register file data out signal is not activated. The tritstate buffer holding the count bits inside the count decoder receives a low signal due to its inverter, and releases them to the ALU port B.

The count decoder is connected to the ALU port B via a tristate and the register file through a multiplexer 2 x 1 as illustrated in the modified data path of figure 5.31.

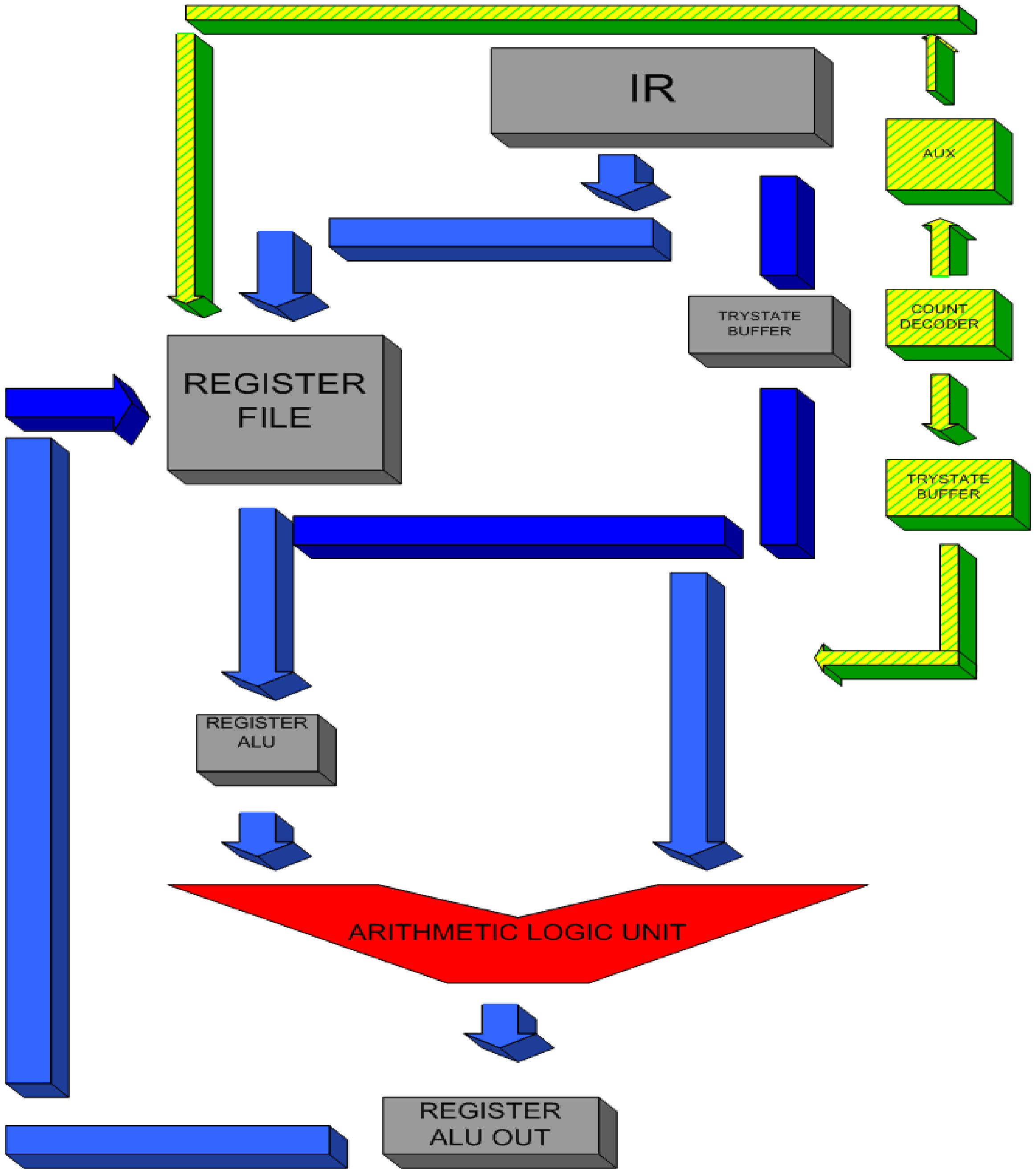


Figure 5.31 Modified Data Path for Shift Instructions

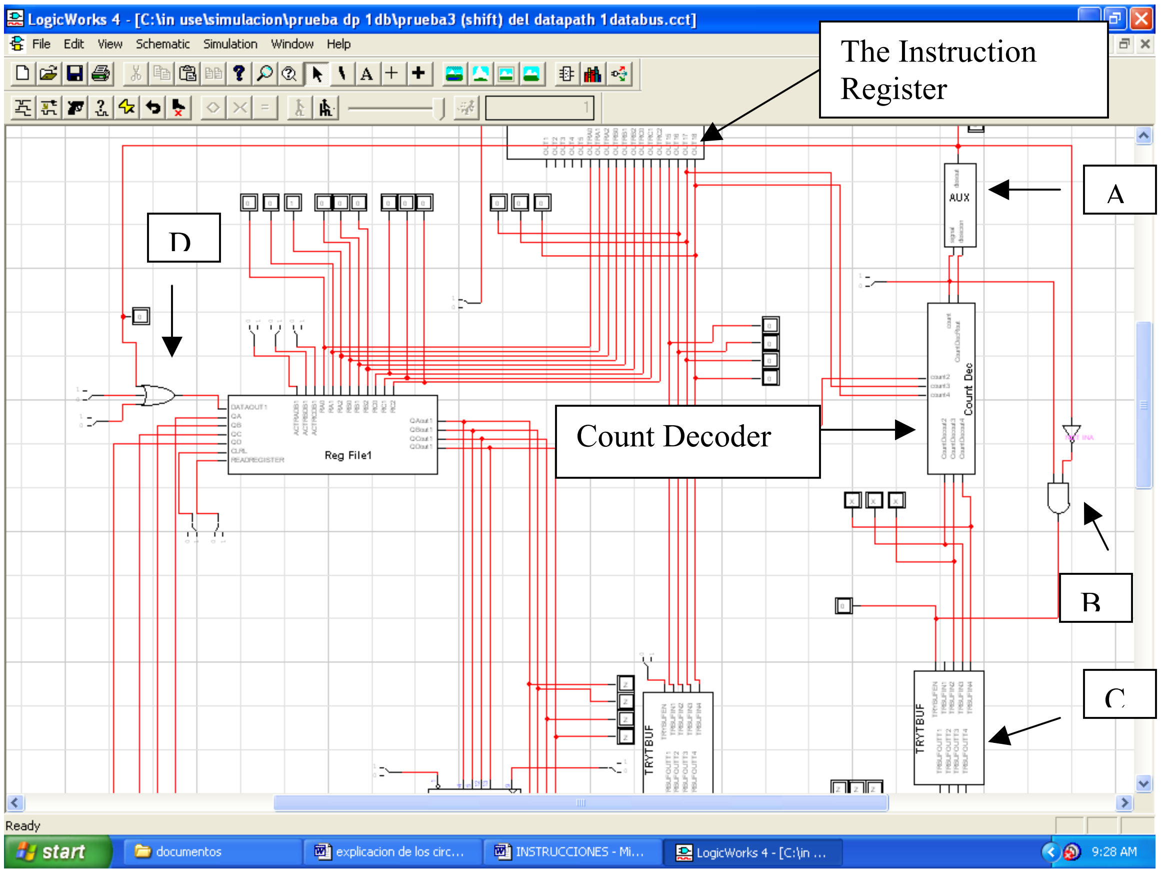

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba3 (shift) del datapath 1databus.cct]
File Edit View Schematic Simulation Window Help
The Instruction Register
A
D
Reg File1
Count Decoder
AUX
Count Dec
B
C
TRYTBUF
Ready
start
documentos
explicacion de los circ...
INSTRUCCIONES - Mi...
LogicWorks 4 - [C:\in ...
9:28 AM

Figure 5.32 Count Decoder Data Path Implementation

The Logic Works schematic is shown in figure 5.32 as stated. The circuit A, is the auxiliary circuit, is really a two to one multiplexer, and it solves some problems at the implementation stage. Note in figure 5.32 that this implementation requires the auxiliary circuit, one tristate buffer labeled C and an additional logic labeled B. The tristate buffer is used to isolate the data from the count decoder to the data bus when not in use.

The logic labeled B is used to ensure that the tristate buffer C is activated only if the count decoder signal is activated and the Rout signal of the count decoder is low. Remember that the Rout signal activates the Register File output port and if this logic is not used, there is a risk to release the data of the count decoder at the same time with the register File to the data bus and result in signal conflict. This logic guarantees that if the count bits are zero just the data in the Register File will be released and that if they are not zero; the count decoder will send them to the data bus, but only one set of data at a time.

The OR labeled D in figure 5.32 is necessary because the control unit like the Count Decoder will need to release data to execute other instructions. Later shall be illustrated that the load instruction requires data to be released and that is why the data out port of the Register File has a three input OR logic gate.

The two to one multiplexer, is shown in figure 5.33. This circuit is activated simultaneously with the count decoder. Its signal port is connected with the activation port of the count decoder. If the count decoder signal is not activated, is 0 , then DESOUT = 0, otherwise it is the ROUT signal from the count decoder. Without this circuit, the Rout signal received by the Register File output port would be a high impedance signal when the count decoder signal is not activated causing some conflicts.

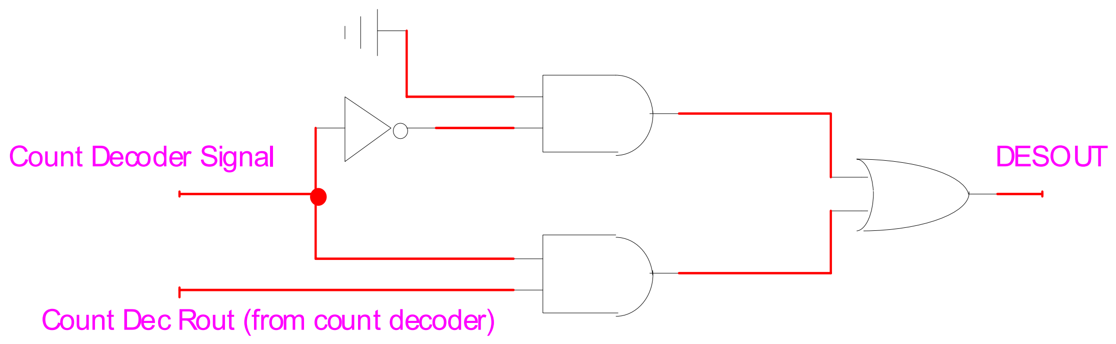


Figure 5.33 Auxiliary Circuit

5.8 STEP IX Jump, Data Transfer Instructions, and the PC

5.8.1 The Branch Instruction

Depending on certain conditions, the execution of a non-continuous program code could be necessary. In programming, this is called a jump. The term “jump” here means that the program counter (register that holds the address of the next instruction) gets an address value that is not consecutive on. To analyze the necessary conditions for one jump a combinational circuit is needed. This circuit is known as the Conditional Jump Decoder (CJD) and will be added to the existing data path for the Branch instructions execution.

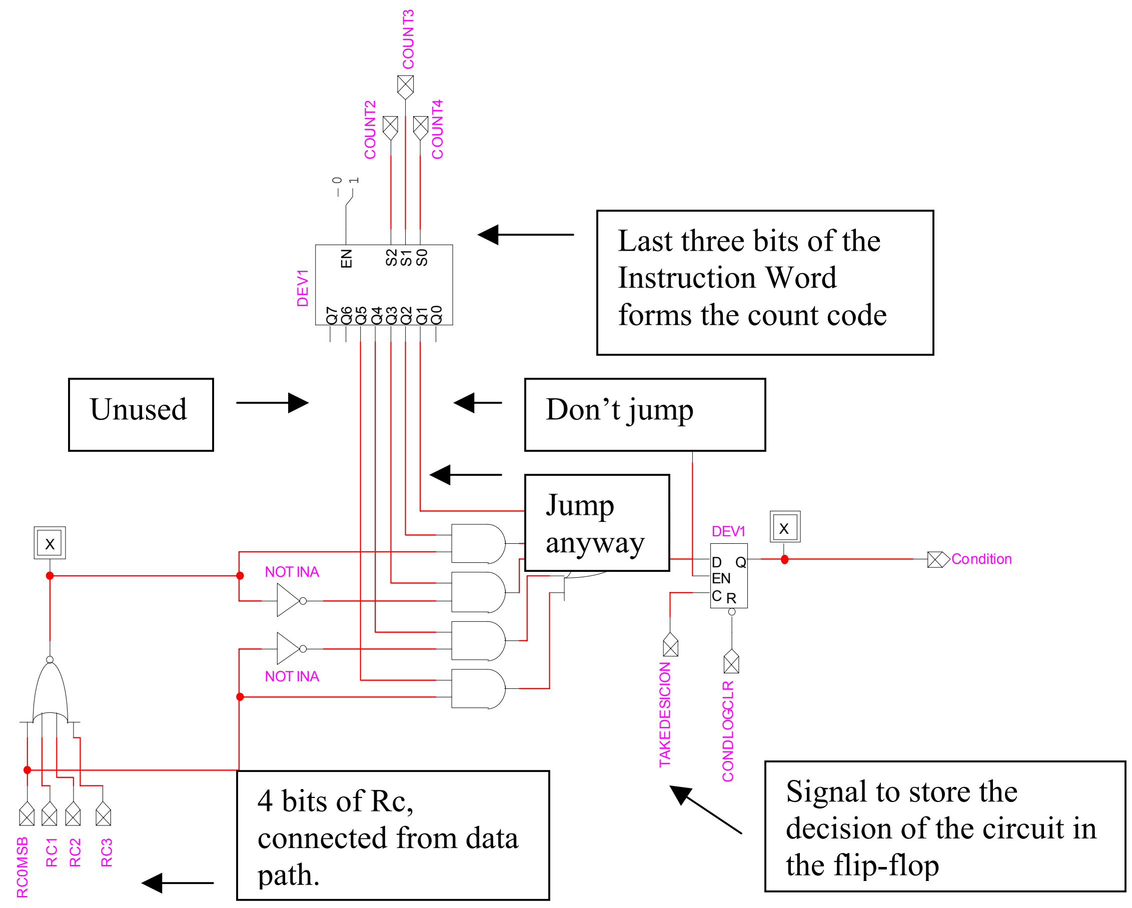

COUNT3
COUNT2
COUNT4
0
1
DEV1
EN
S2
S1
S0
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
Last three bits of the Instruction Word forms the count code
Unused
Don't jump
Jump anyway
X
NOT INA
NOT INA
DEV1
D
Q
EN
C
R
X
Condition
TAKEDESICION
CONDLOGCLR
RC0MSB
RC1
RC2
RC3
4 bits of Rc, connected from data path.
Signal to store the decision of the circuit in the flip-flop

Figure 5.34 The Conditional Jump Decoder

The Conditional Jump Decoder activates or not the Program Counter based on the count bits and the contents of Rc. The count bits will select the decoder's output and depending of which output is selected; additional logic is used to make decisions. The following table 5.2 indicates the Conditional Logic execution accordingly with count and Rc. This instruction was designed to cover the most needed cases. Other way of implementation could be designing each case separately and leave to the designers the decision to choose among all available options. Using this format the programmer just need to specify the parameters of the required jump.

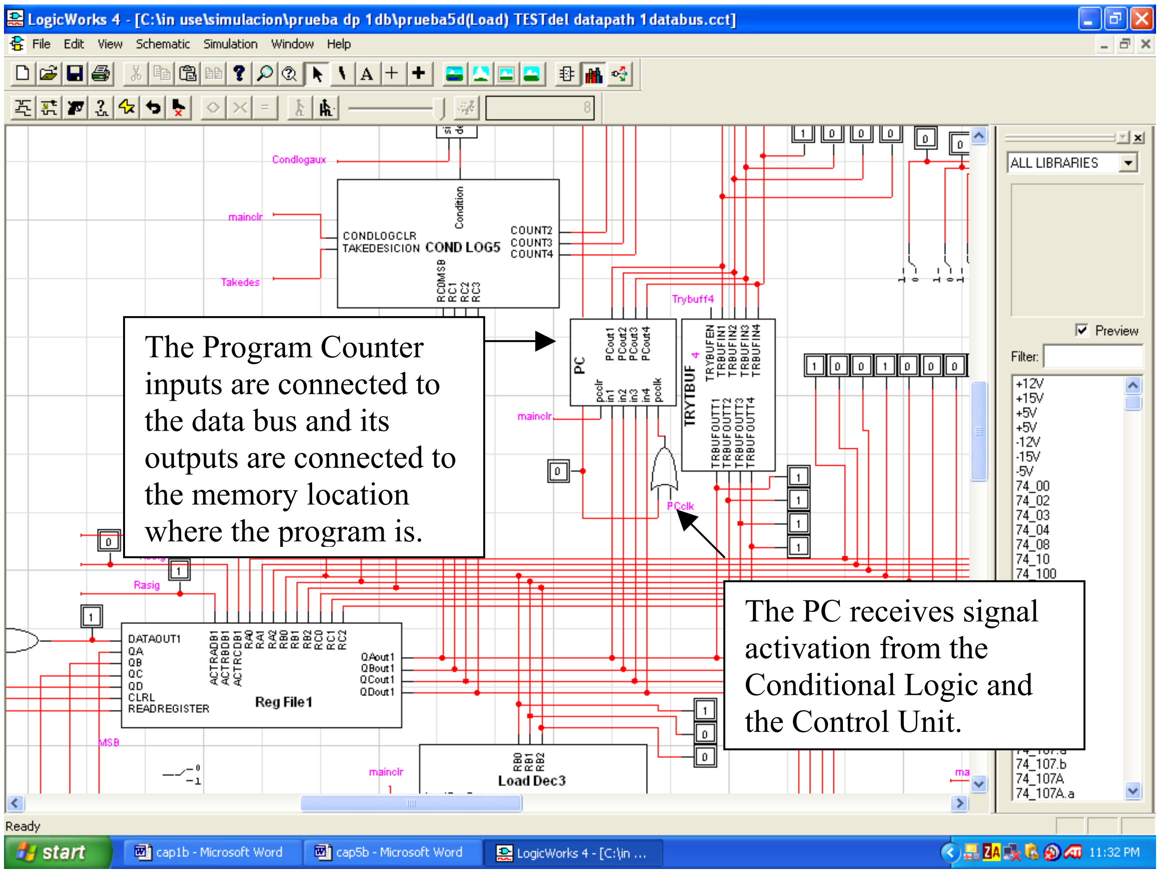
Count code	**Description**	**Task**
000 = 0	Don't jump	In the first six cases the operation will be the same. PC stores what is stored in the register specified by Rb from the data path.
001 = 1	Jump anyway	
010 = 2	Verify the bit address of Rc, if they are not equal to 0 $\rightarrow$	
011 = 3	Verify the bit address of Rc, if they are equal to 0 $\rightarrow$	
100 = 4	Verify the bit address of Rc, if they are zero or grater than 0 $\rightarrow$	
101 = 5	Verify the bit address of Rc, if they are less than 0 $\rightarrow$	
110		Unused
111		Unused

Table 5.2 Conditional Logic Cases

5.8.2 The Program Counter

The program counter is the register that stores the next instruction memory address location. Its inputs are connected to the data bus to receive the next address value from it and its outputs are connected to the memory device where the program is stored. The connections are shown in figure 5.35 and 5.36. The memory device where the microcontroller's program is stored, has address bits that indicate the desired specific program code location. After memory receives the address by the PC, the fetch process begins. It consists in addressing the instruction specified in memory by the PC and loading it to the IR.

The fetch process is discussed at the Control Unit design stage. After the fetch process is complete, the instruction execution begins. The Branch instruction allows the programmer to specify a memory location where instructions are located and execute them and then continue executing the program. In this instruction the register Rb will store the four bits memory address to execute the jump. Figure 5.37 shows the data path with the new boxes included. The implementation is shown in figure 5.38


LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct]
File Edit View Schematic Simulation Window Help
COND LOG5
PC
TRYTBUF
Reg File1
Load Dec3
ALL LIBRARIES
Preview
Filter:
Ready
start
cap1b - Microsoft Word
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
11:32 PM

The Program Counter inputs are connected to the data bus and its outputs are connected to the memory location where the program is.

The PC receives signal activation from the Conditional Logic and the Control Unit.

Figure 5.35 The Program Counter implementation

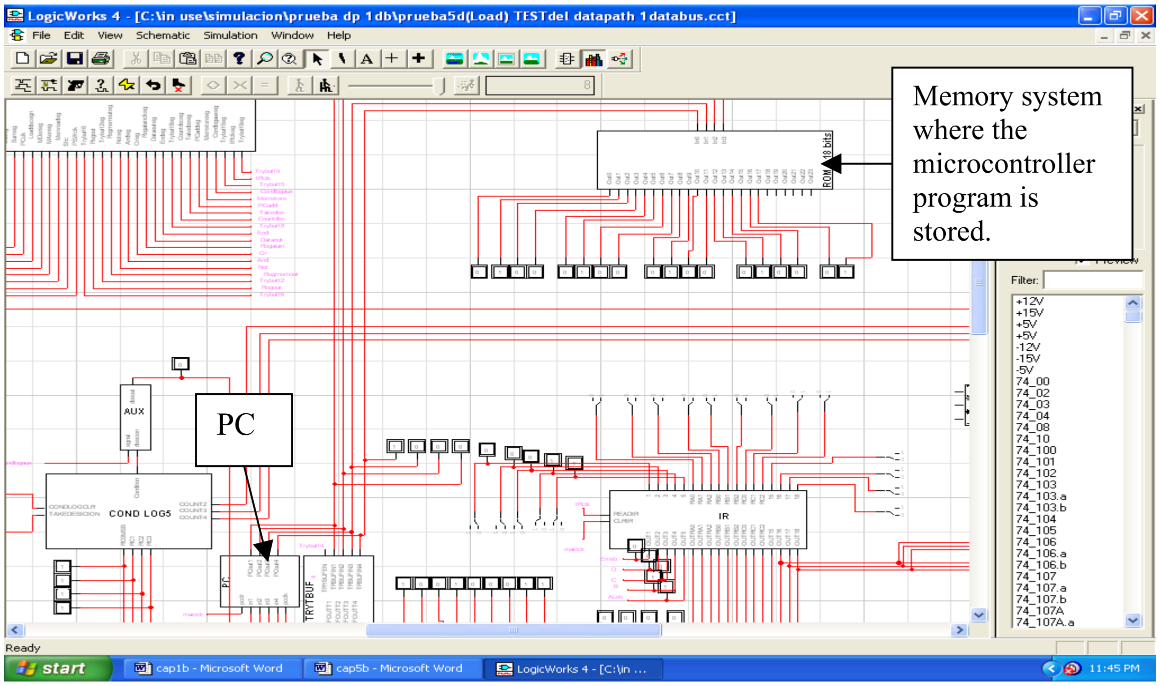


Figure 5.36 Memory ROM implementation for program storage

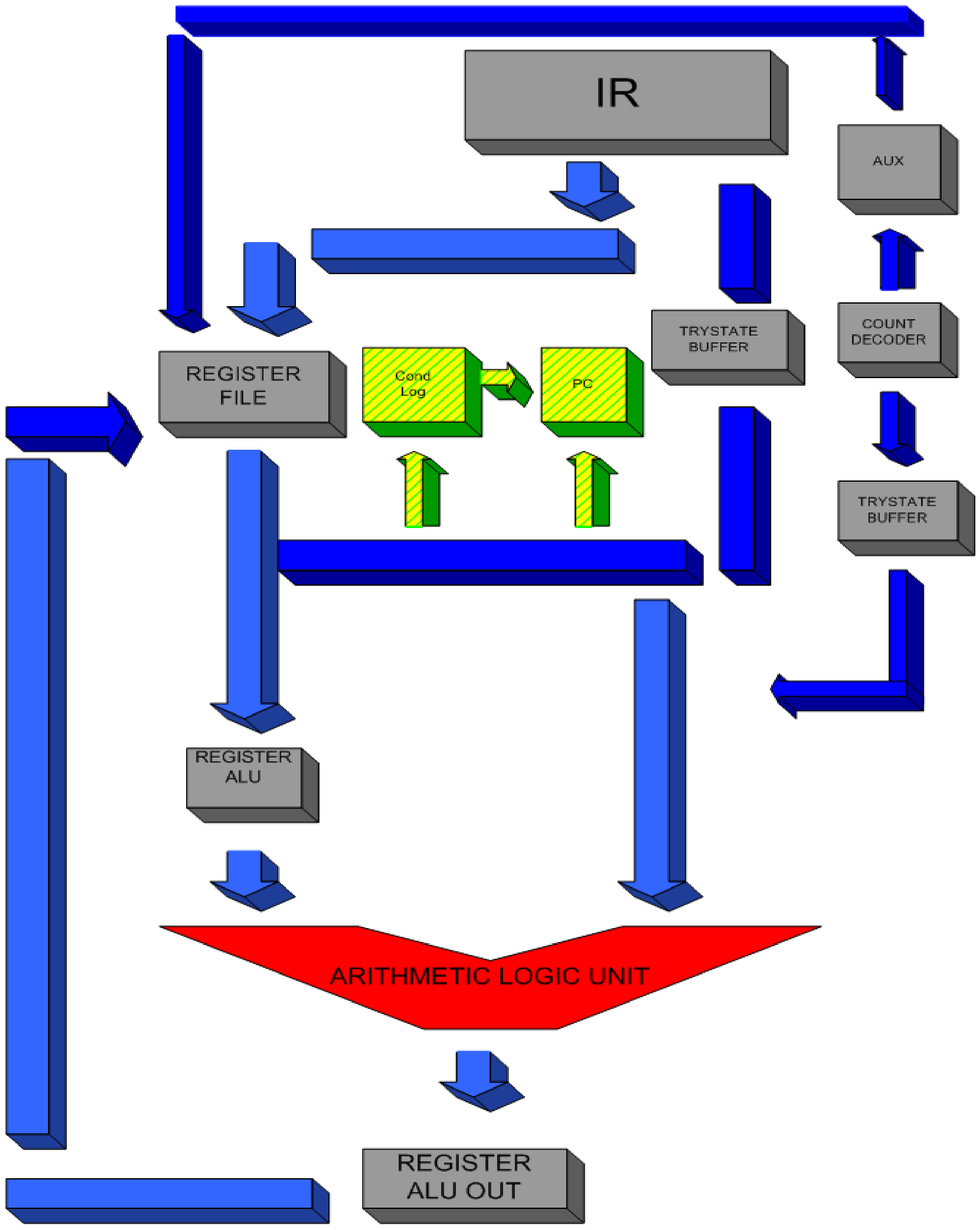


Figure 5.37 Modified Data Path for the Branch Instruction

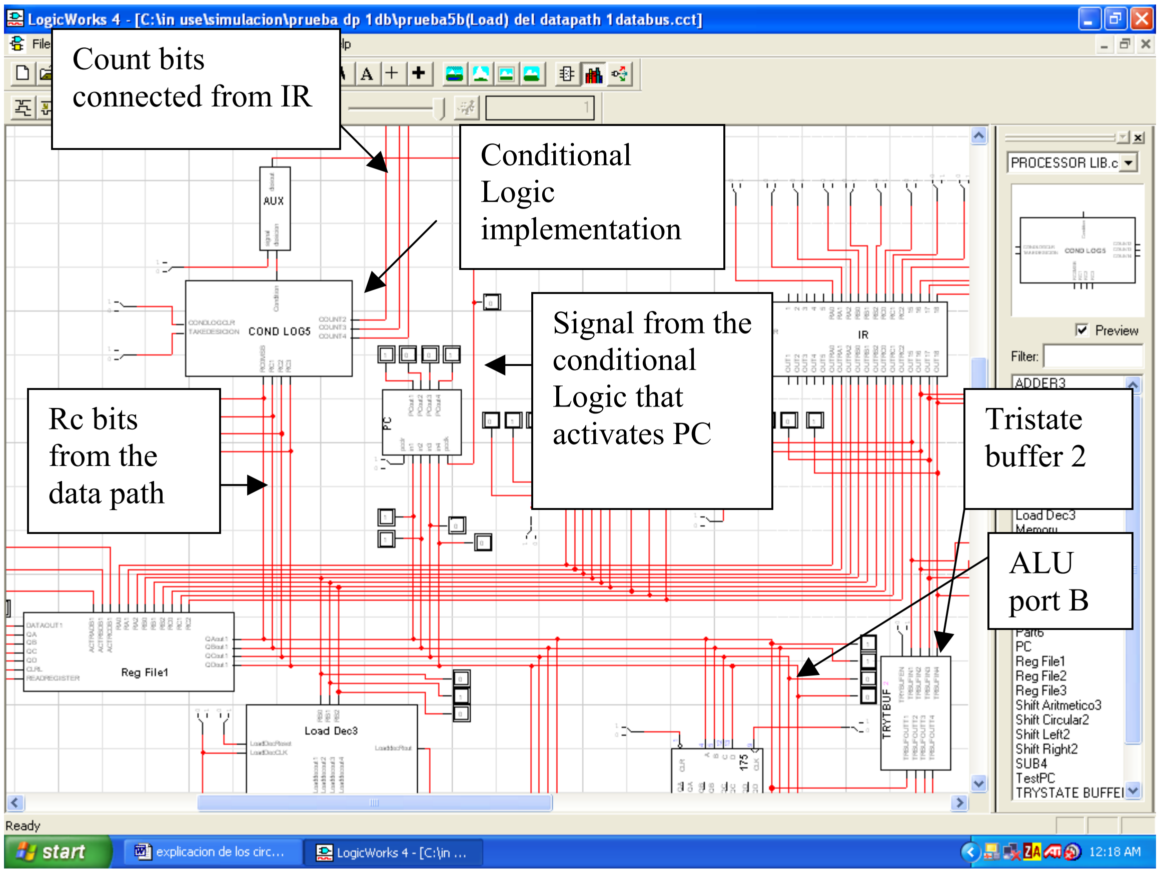

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5b(Load) del datapath 1databus.cct]
Count bits connected from IR
Conditional Logic implementation
Signal from the conditional Logic that activates PC
Rc bits from the data path
Tristate buffer 2
ALU port B
AUX
COND LOGS
IR
Reg File1
Load Dec3
TRYTBUF
PROCESSOR LIB.c
Preview
Filter:
ADDER3
Load Dec3
Memory
PC
Reg File1
Reg File2
Reg File3
Shift Aritmetico3
Shift Circular2
Shift Left2
Shift Right2
SUB4
TestPC
TRYSTATE BUFFER
Ready
start
explicacion de los circ...
LogicWorks 4 - [C:\in ...
12:18 AM

Figure 5.38 The Conditional Logic Implementation

The Conditional Jump Decoder implementation needs also the auxiliary circuit described before for the same Count Decoder reasons.

5.8.3 Load and Store

Load and Store, work basically in the same way, the obvious difference is that one write to memory and the other read from memory. In those instructions mathematical manipulations are done to calculate the data memory address location. Those instructions works with Rb address bits. Based on the values of Rb the address bits 0000 will be loaded to the data bus or the data at Rb will be released to the data bus and in either case added to Mc. Mc are the last four bits of the instruction word and represent the desired memory address location. Mc set the initial memory address location and Rb locates one specific position from Mc. A combinational logic circuit to decide if it loads Rb or 0000 is needed and it is called The Load decoder, see figure 5.39.

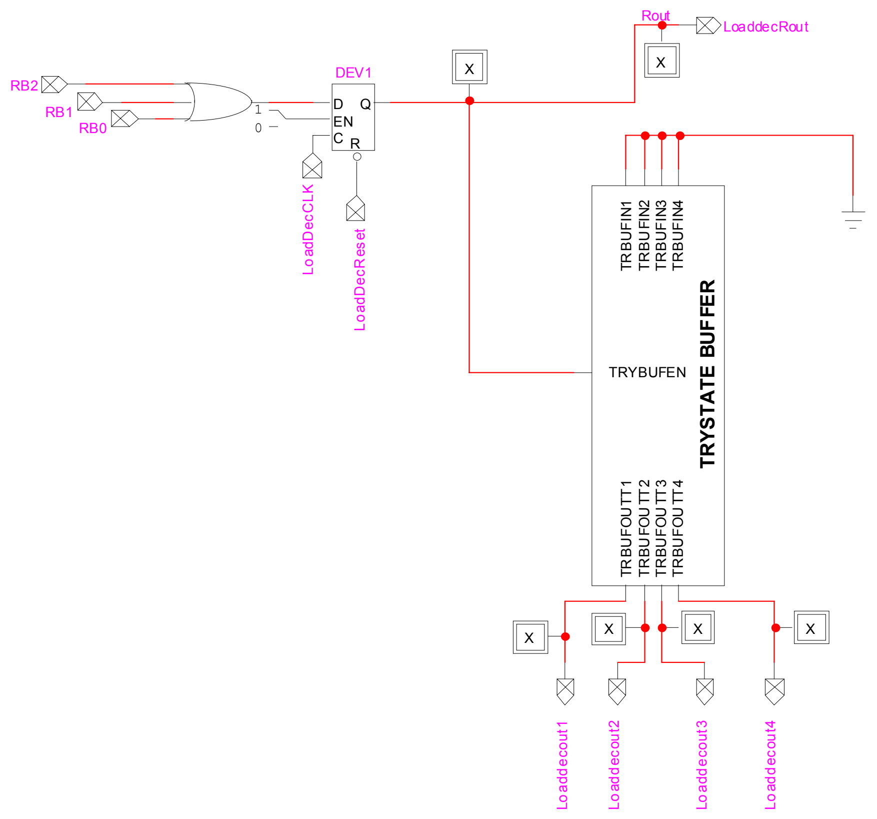


Figure 5.39 The Load Decoder Circuit

Before the Load Decoder activation, the Register File must activate the address bits Rb. The Load decoder verifies all the bits of Rb. If all are zero, when the flip flop inside the Load Decoder gets the OR decision, the Load Decoder Rout signal will be low (this signal releases the data of the Register File) and the tristate buffer will be activated charging 0000 to the data bus. If all the Rb bits are not zero the Rout signal will be high and release the data specified by Rb in the Register File to the data bus.

The load Decoder implementation in figure 5.40 requires a connection with the Rb bits in the Instruction Register. A tristate buffer is connected to the Load Decoder output. Its control logic ensures that no conflict occurs when data is released to the data bus. The auxiliary circuit ensures that the Register File output port receives a zero when the Load Decoder is not activated. The AND gate that controls the tristate buffer sends a high signal only when the load decoder signal is activated and the Load Decoder decision is zero. In this way the system ensures that no conflict between data occurs in the data path at any moment.

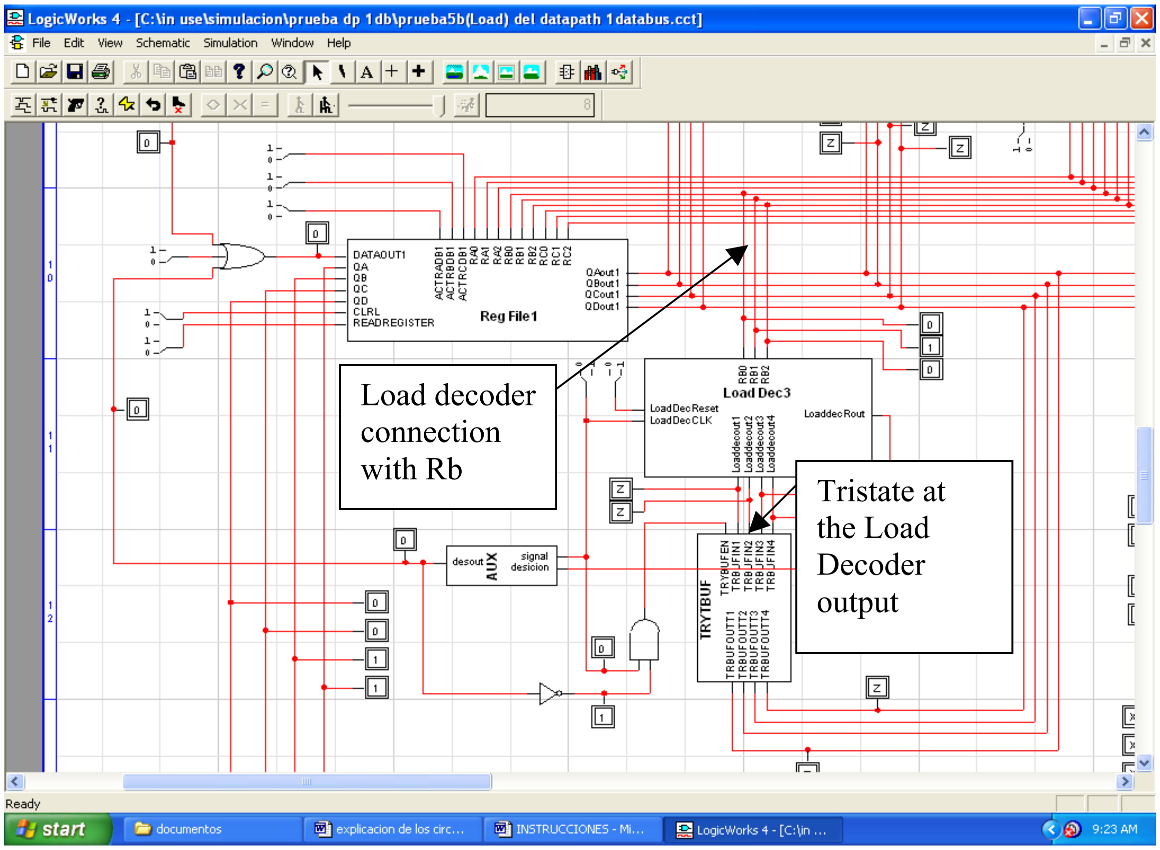

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5b(Load) del datapath 1databus.cct]
Reg File1
Load Dec3
TRYTBUF
AUX
Load decoder connection with Rb
Tristate at the Load Decoder output
Ready

Figure 5.40 The Load Decoder Implementation

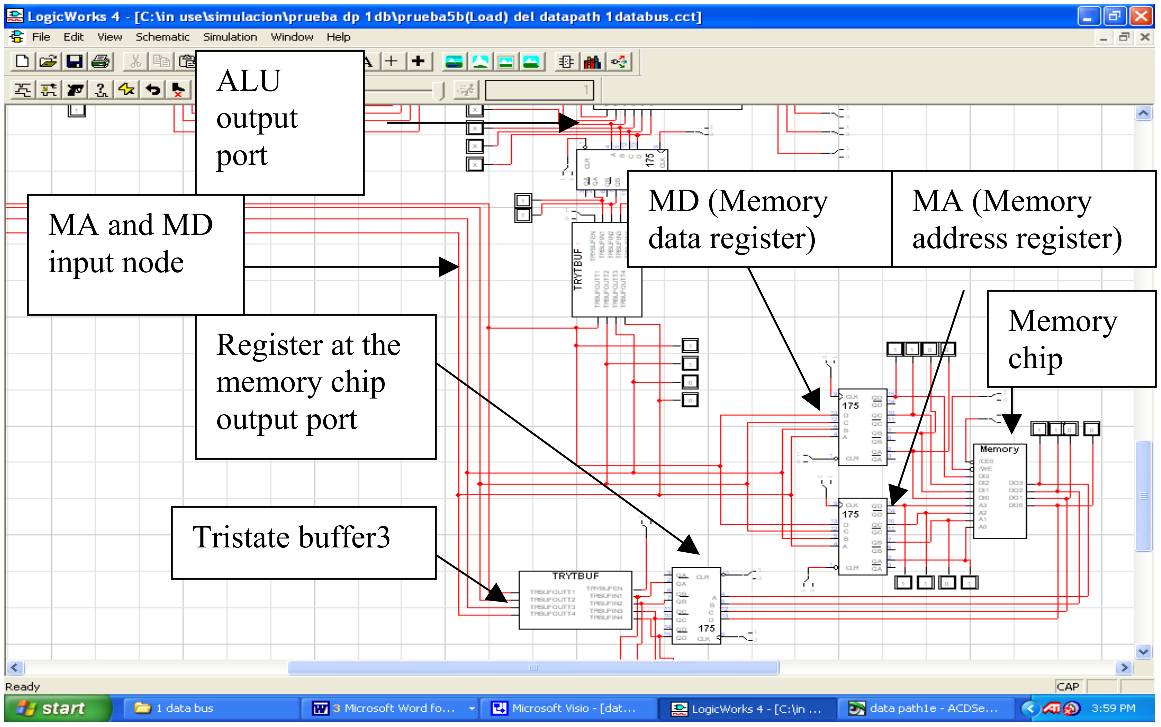

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5b(Load) del datapath 1databus.cct]
ALU output port
MA and MD input node
MD (Memory data register)
MA (Memory address register)
Memory chip
Register at the memory chip output port
Tristate buffer3
TRYTBUF
Memory
175
Ready

Figure 5.41 Memory Implementation for Load and Store

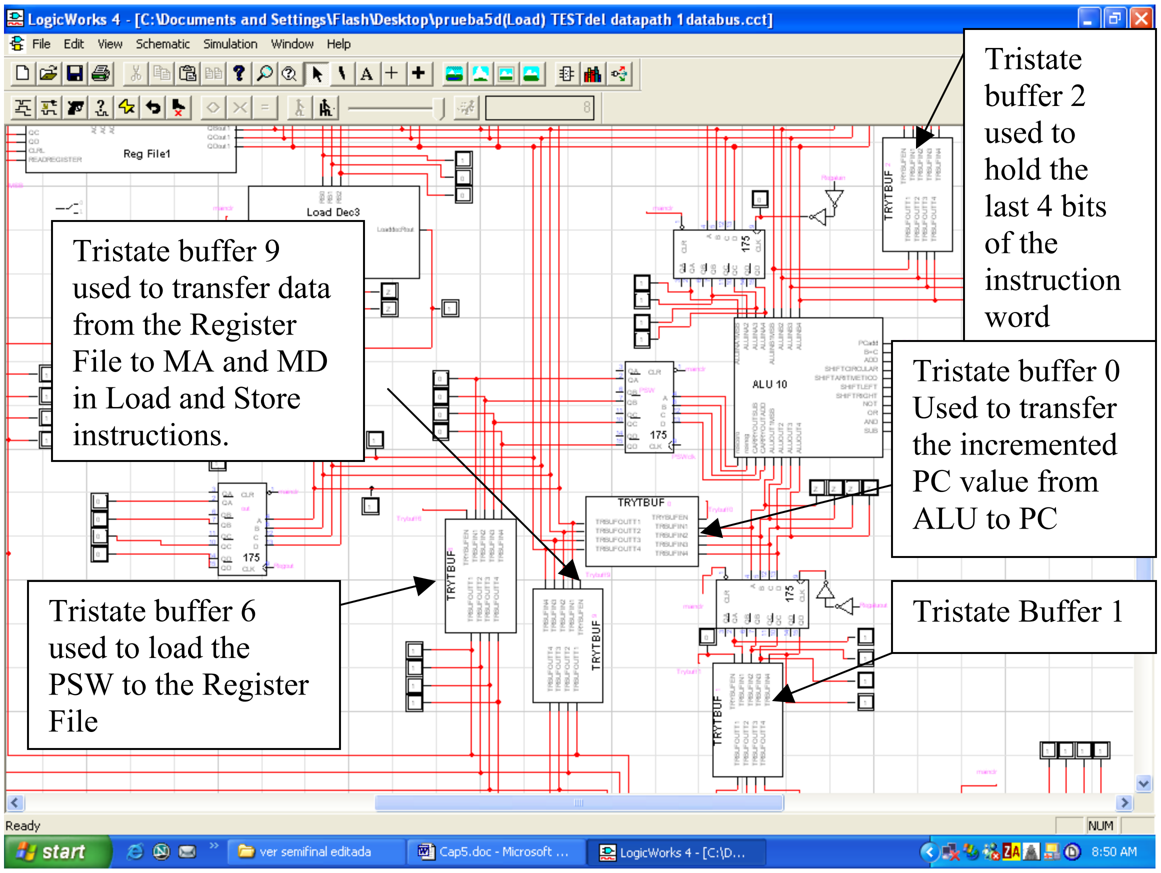

LogicWorks 4 - [C:\Documents and Settings\Flash\Desktop\prueba5d(Load) TESTdel datapath 1databus.cct]
File Edit View Schematic Simulation Window Help
Reg File1
Load Dec3
ALU 10
TRYTBUF
175
Tristate buffer 2 used to hold the last 4 bits of the instruction word
Tristate buffer 9 used to transfer data from the Register File to MA and MD in Load and Store instructions.
Tristate buffer 0 Used to transfer the incremented PC value from ALU to PC
Tristate buffer 6 used to load the PSW to the Register File
Tristate Buffer 1
Ready
NUM
start
ver semifinal editada
Cap5.doc - Microsoft ...
LogicWorks 4 - [C:\D...
8:50 AM

Figure 5.42 Added elements for Load, Store, Read PSW and Fetch

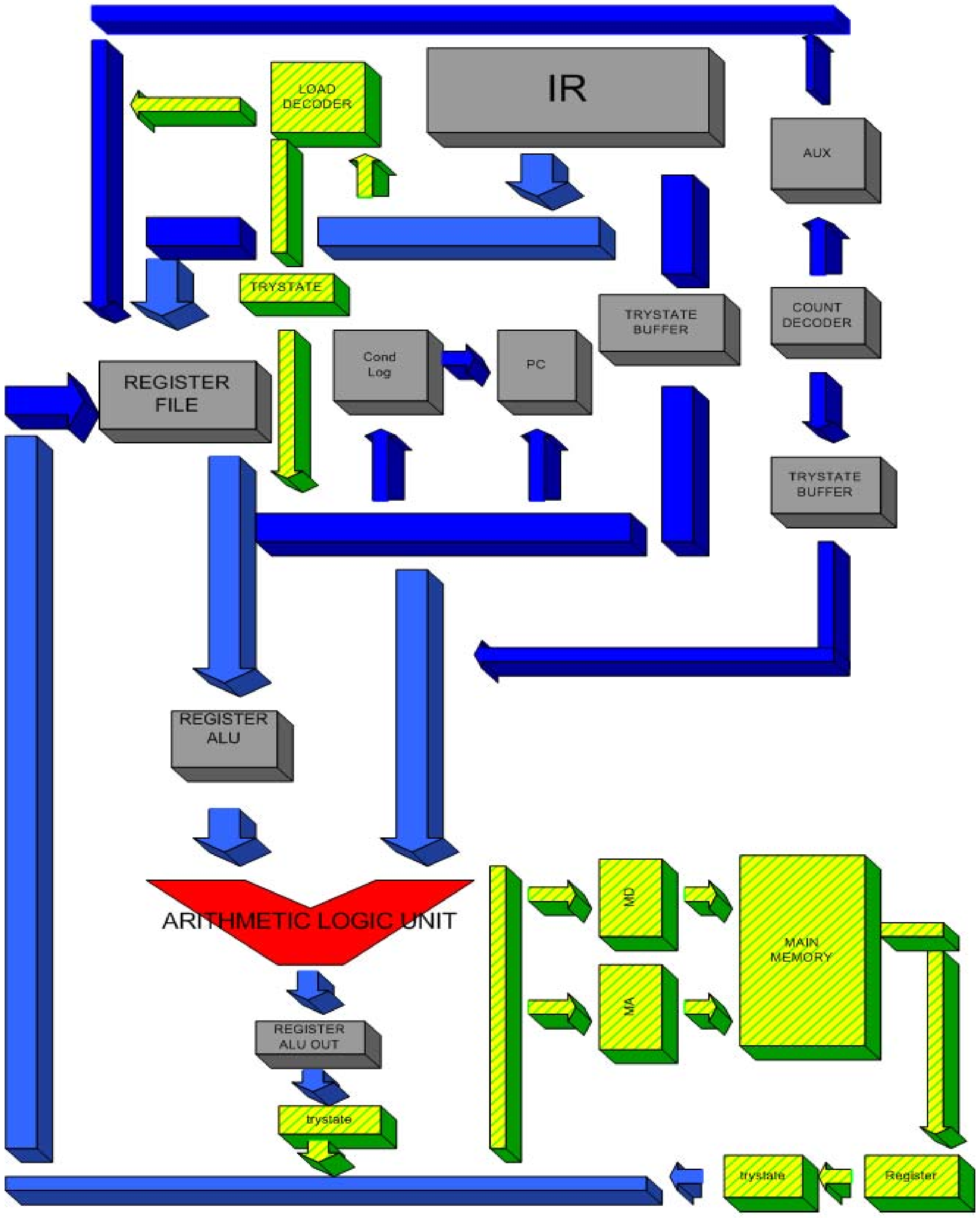


Figure 5.43 Modified Data Path to include Load and Store operations

5.8.4 Miscellaneous Operations

The next figure illustrates the implementation of the IN function. The purpose is to obtain information from outside. That is the reason for using one tristate buffer (tristate buffer 8) connected to the Register File input port.

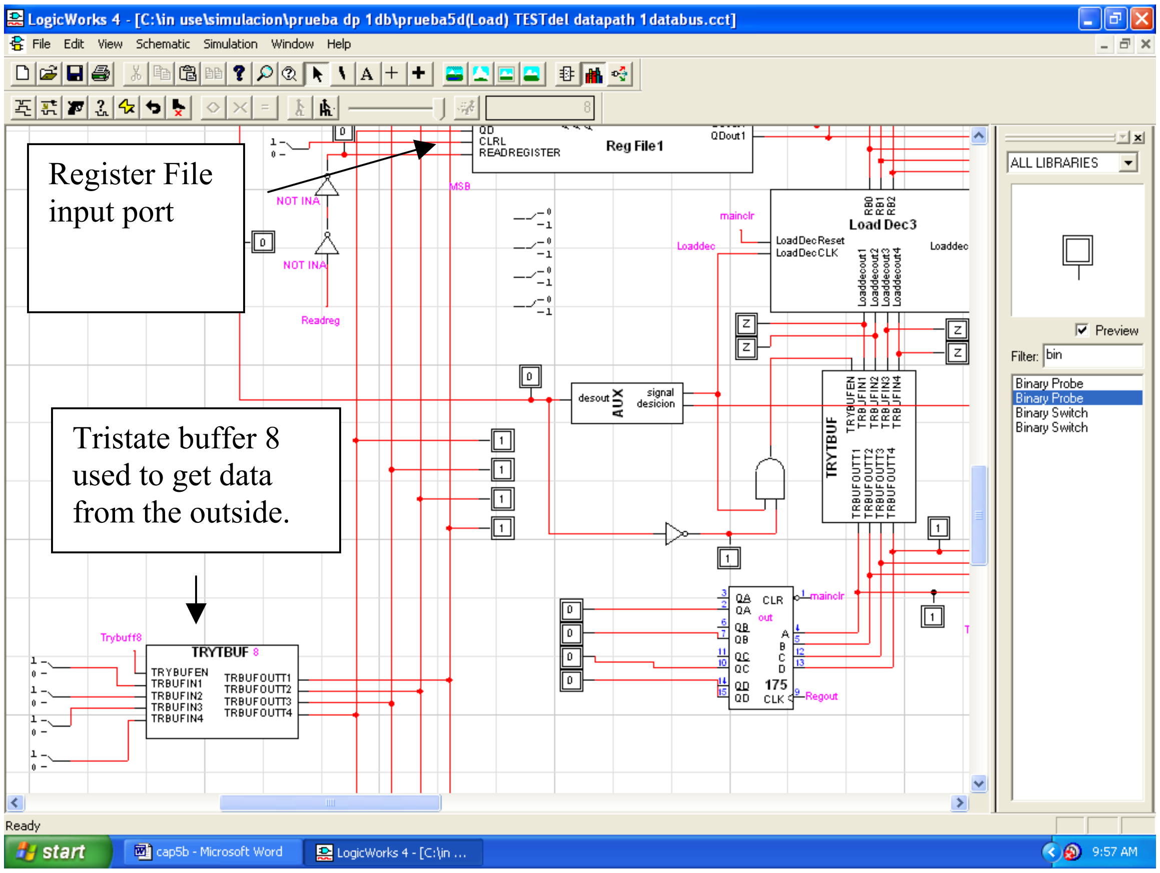


Figure 5.44 (In) Instruction Hardware Implementation

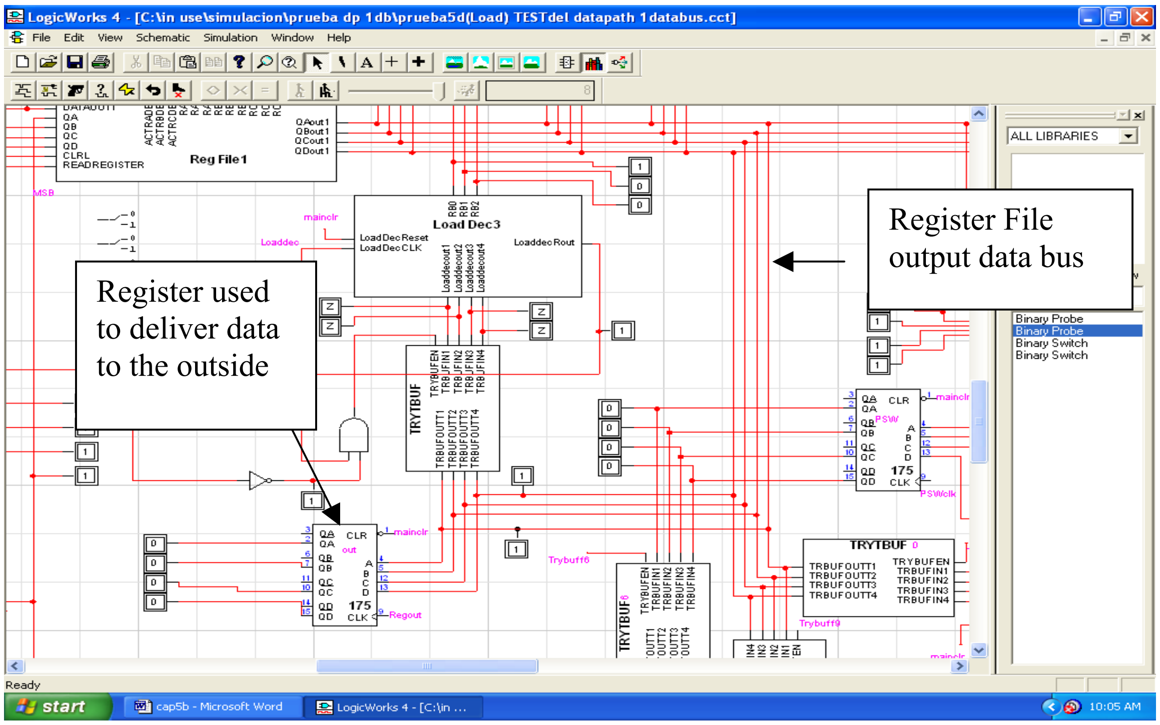

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct]
File Edit View Schematic Simulation Window Help
Register File output data bus
Register used to deliver data to the outside
Reg File1
Load Dec3
TRYTBUF
TRYTBUF
175
175
ALL LIBRARIES
Binary Probe
Binary Probe
Binary Switch
Binary Switch
Ready
start
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
10:05 AM

Figure 5.45 Circuit Implementation for Out Instruction

The instruction implementation consists first in collecting all flags in one register called the Processor Status Word or PSW (see the register labeled A), then connecting it to the Register File input port (label C), and using one tristate buffer (called tristate buffer 6).

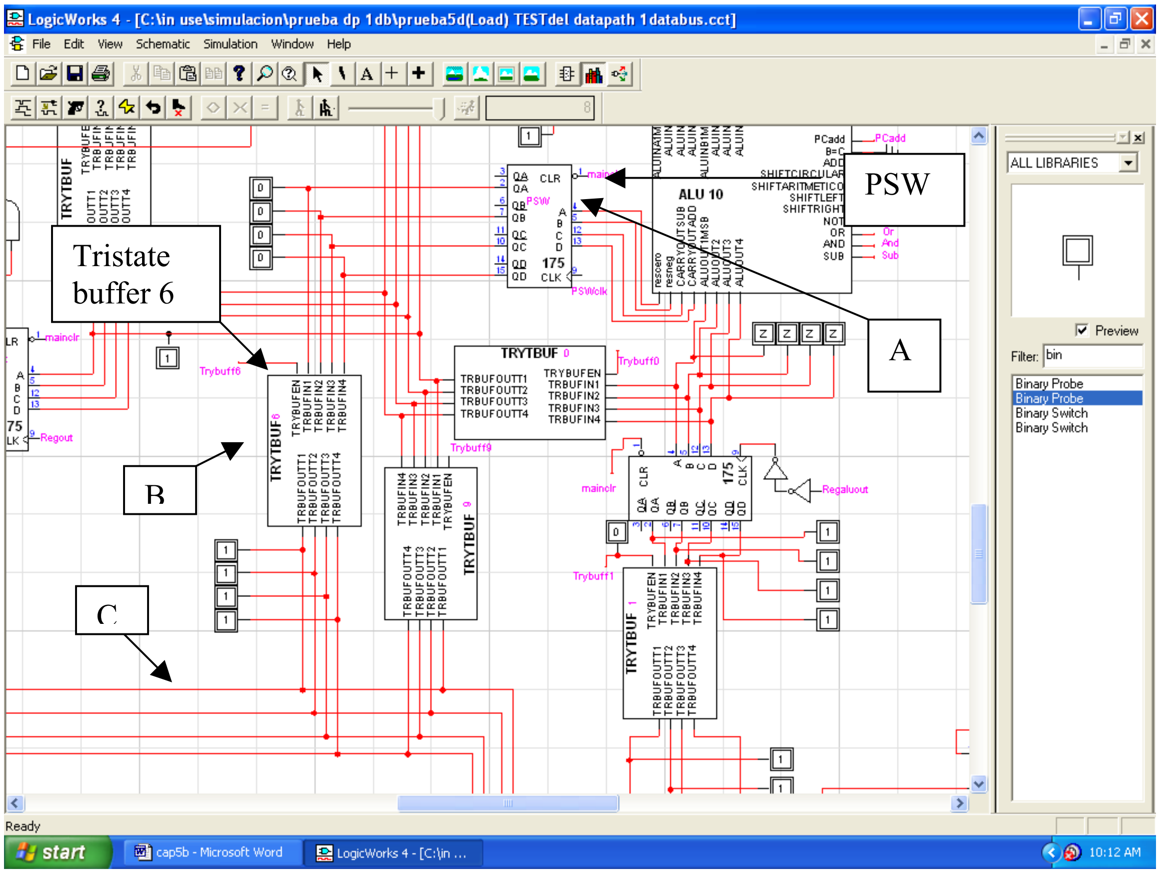


Figure 5.46 Read PSW Instruction Hardware

Cap 6

THE CONTROL UNIT

The step X is described in a separate chapter because this unit is essentially a sequential circuit. The control unit is the final stage for the microcontroller development in this work. The control unit takes control of signal activation of microcontroller circuits in each clock cycle. The next figure illustrates the control unit configuration.

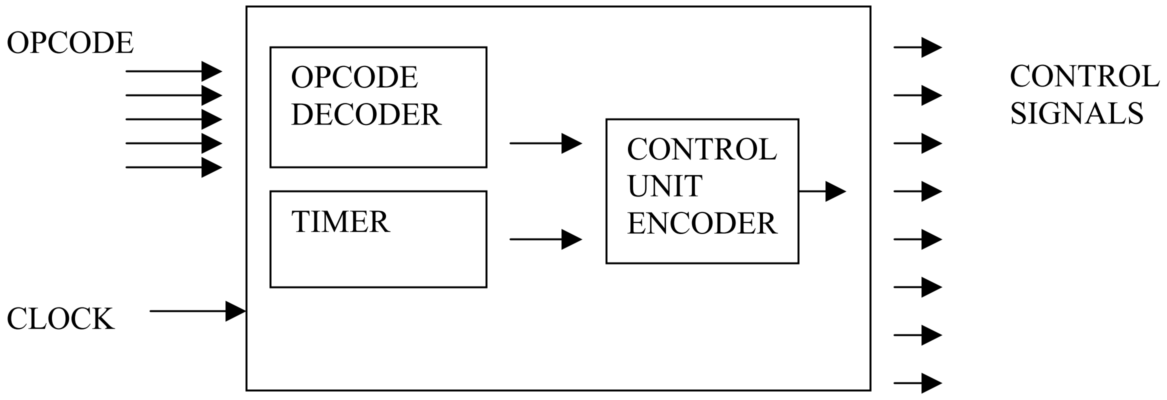


Figure 6.1 The Control Unit Implementation

6.1 The Fetch Process

Before starting the Control Unit discussion, something must be said about the fetch process. The fetch process consists in loading one memory address value in the PC, and delivering it to the memory device address port to obtain a specific microcontroller programming code. All the preceding instruction discussion left two clock cycles for the fetch execution. The author knew from the beginning how many clock cycles were needed for the fetch process, making an educated guess of the following:

a) The data bus size.
b) The amount of memory used to store the program.
c) Program Counter size. - The PC does not have to be the same size of the data bus; but the address bus. This means that you must use more than one clock cycle just only to fill the PC with the new address value.
d) The Existing data path circuitry. - It must provide the necessary circuits to ensure that the PC is incremented in every instruction execution, and that no signal conflict occurs.
e) The fetch process. - Designers must ensure that PC is incremented in each instruction, but they must decide how the data travels between the microcontroller circuits. One alternative to execute the fetch process and increment the PC could be better than other. There are many possibilities to execute the fetch process and this work provides one possible alternative of it. The reader must use its creative and critical thinking to make the judgment and decide how the fetch process will be carried out.

Designers have to make a trade off between those alternatives and decide the number of the fetch process clock cycles and their data processing route in the data path circuits.
The fetch process used in this work uses just two clock cycles (see figures 6.2 and 6.3):

1) In the first clock cycle the Control Unit activates the Instruction Register read signal to load from memory the instruction word to be executed. Also, the tristate buffer 4 is activated to release the current PC value to the data path. Finally, in the same clock cycle, the register at port A of the ALU is activated to store the current PC value as shown in figure 6.2.
2) In the second clock cycle, the Control Unit activates the add PC signal as shown in figure 6.3 from the Arithmetic Logic Unit to increase the current PC value by one. The tristate buffer 0 at the ALU output port is activated to deliver the

incremented PC value to the data path. Finally in that same clock cycle, the Program counter clock is activated to load the incremented value to the PC.

At the fetch process designing stage, a new function is needed in the ALU. The fetch process needs one circuit that increment the PC by one. We just add one adder to the ALU circuit that takes the ALU port A data and add one to it. The figures 6.4 and 6.5 illustrate this implementation in the ALU. Observe that it is just one adder and is connected in the same way as the other elements. One of the advantages of the technique used in this work is that it allows users to add circuit elements without making significant design changes to the entire system.

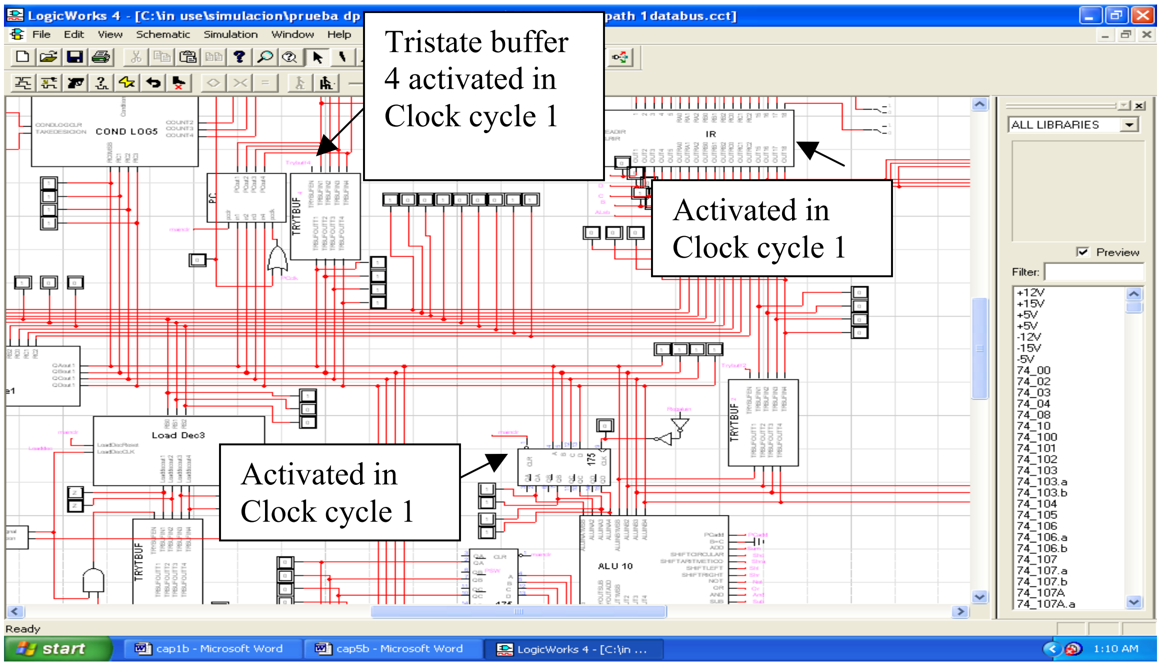

LogicWorks 4 - [C:\in use\simulacion\prueba dp path 1databus.cct]
File Edit View Schematic Simulation Window Help
Tristate buffer 4 activated in Clock cycle 1
Activated in Clock cycle 1
Activated in Clock cycle 1
COND LOGS
IR
TRYTBUF
PC
Load Dec3
ALU 10
ALL LIBRARIES
Preview
Filter:
Ready
start
cap1b - Microsoft Word
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
1:10 AM

Figure 6.2 Fetch Process First Cycle

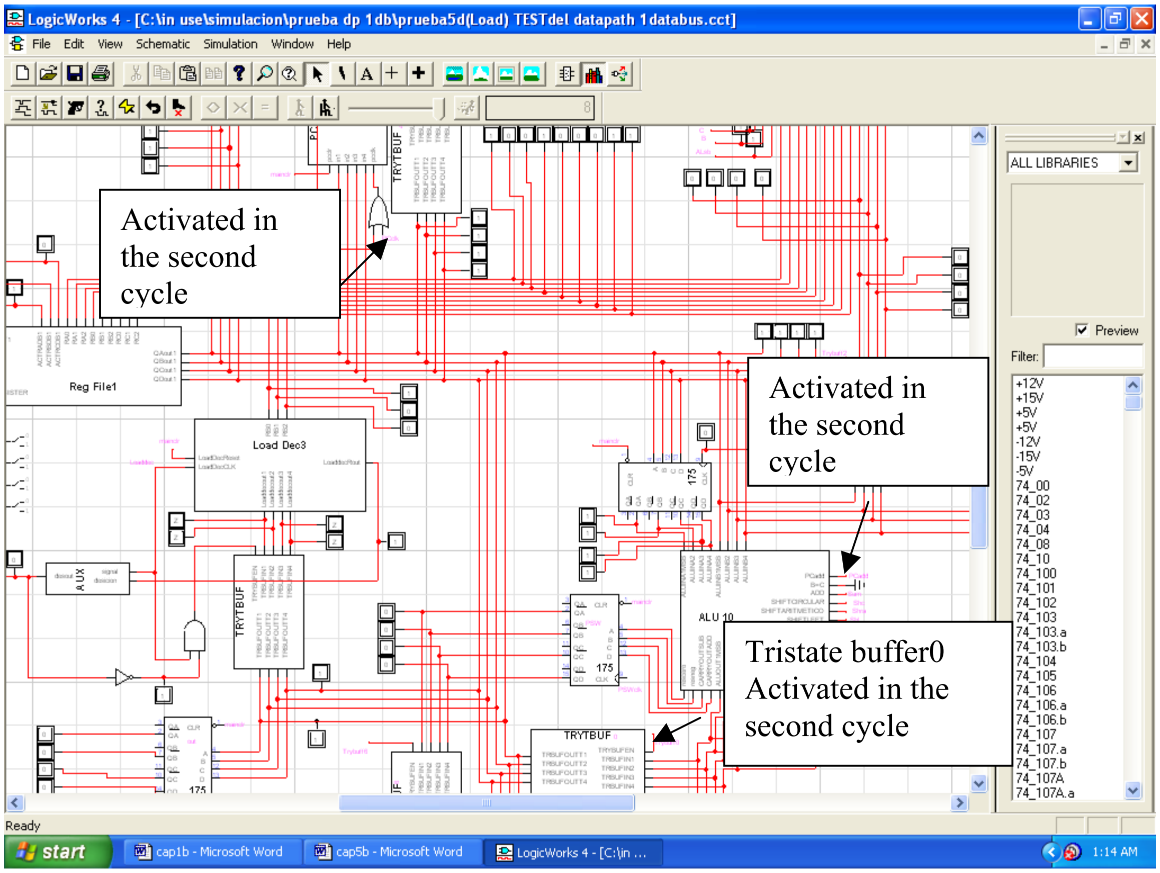

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct]
File Edit View Schematic Simulation Window Help
Activated in the second cycle
Activated in the second cycle
Tristate buffer0 Activated in the second cycle
Reg File1
Load Dec3
TRYTBUF
ALU 10
175
ALL LIBRARIES
Preview
Filter:
Ready
start
cap1b - Microsoft Word
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
1:14 AM

Figure 6.3 Fetch Process Second Cycle

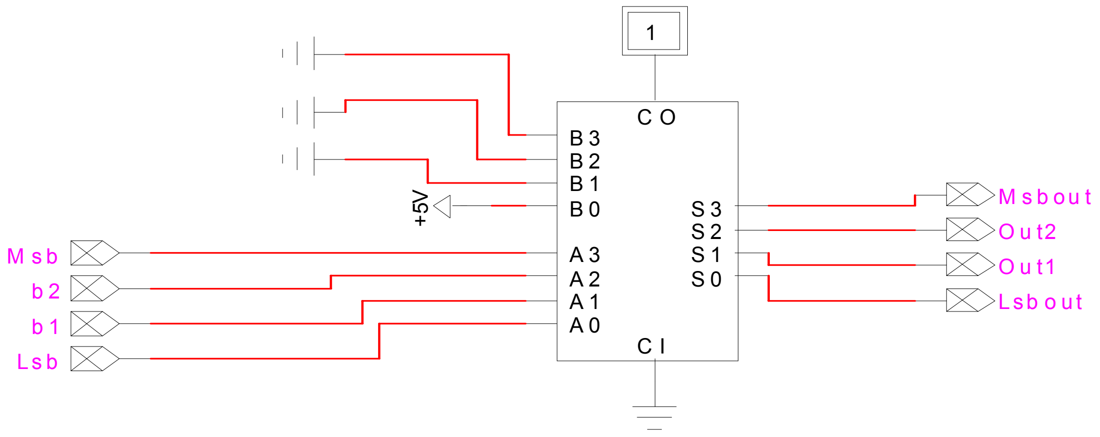


Figure 6.4 PC Incrementer Circuit

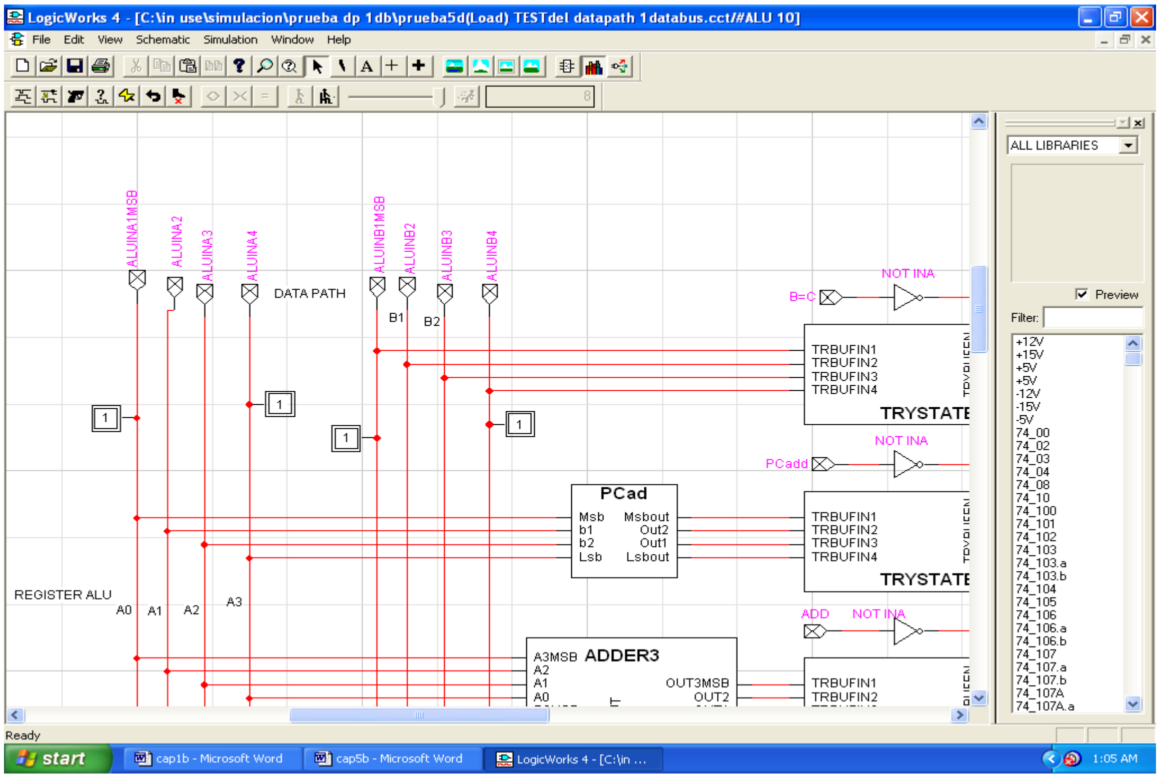


Figure 6.5 PC Incrementer Implementation

6.2 THE CONTROL UNIT ENCODER

The Control Unit Encoder is the hardest stage in the Control Unit design process that is why it is explained first. The Control Unit encoder takes information from the timer and the opcode decoder to activate specific signals in each clock cycle. Then, the first step is to analyze the signal activation per instruction. The following tables show this process.

ADD															
Cycle	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	add	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4
1	X								X						X
2		X									X		X		
3					X			X	X						
4						X		X		X		X			
5				X			X							X	
6			X												

Table 6.1 Add instruction signal activation by clock cycle

SUB															
Cycle	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	Sub	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4
1	X								X						X
2		X									X		X		
3					X			X	X						
4						X		X		X		X			
5				X			X							X	
6			X												

Table 6.2 Sub instruction signal activation by clock cycle

AND															
Cycle	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	And	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4
1	X								X						X
2		X									X		X		
3					X			X	X						
4						X		X		X		X			
5				X			X							X	
6			X												

Table 6.3 AND instruction signal activation by clock cycle

Or															
Cycle	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	Or	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4
1	X								X						X
2		X									X		X		
3					X			X	X						
4						X		X		X		X			
5				X			X							X	
6			X												

Table 6.4 Or instruction signal activation by clock cycle

ADDi																
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	add	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Try Buf 2
1	X								X						X	
2		X									X		X			
3					X			X	X							
4										X		X				X
5				X			X							X		
6			X													

Table 6.5 ADDi instruction signal activation by clock cycle

Subi																
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	sub	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Try Buf 2
1	X								X						X	
2		X									X		X			
3					X			X	X							
4										X		X				X
5				X			X							X		
6			X													

Table 6.6 Subi instruction signal activation by clock cycle

ANDi															
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	add	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Try Buf 2
1	X								X						X	
2		X									X		X			
3					X			X	X							
4										X		X				X
5				X			X							X		
6			X													

Table 6.7 ANDi instruction signal activation by clock cycle

Ori																
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	or	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Try Buf 2
1	X								X						X	
2		X									X		X			
3					X			X	X							
4										X		X				X
5				X			X							X		
6			X													

Table 6.8 Ori instruction signal activation by clock cycle

NOT																
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	not	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	
1	X								X						X	
2		X									X		X			
3						X		X		X		X				
4				X			X							X		
5			X													

Table 6.9 Not instruction signal activation by clock cycle

SHR																
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	shr	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Count Dec
1	X								X						X	
2		X									X		X			
3					X			X	X							
4						X				X		X				X
5				X			X							X		
6			X													

Table 6.10 SHR instruction signal activation by clock cycle

SHRA															
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	shra	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Count Dec
1	X								X						X	
2		X									X		X			
3					X			X	X							
4						X				X		X				X
5				X			X							X		
6			X													

Table 6.11 SHRA instruction signal activation by clock cycle

SHC																
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	shc	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Count Dec
1	X								X						X	
2		X									X		X			
3					X			X	X							
4						X				X		X				X
5				X			X							X		
6			X													

Table 6.12 SHC instruction signal activation by clock cycle

SHL																
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	shl	PC add	Reg ALU Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Count Dec
1	X								X						X	
2		X									X		X			
3					X			X	X							
4						X				X		X				X
5				X			X							X		
6			X													

Table 6.13 SHL instruction signal activation by clock cycle

BRANCH																
Cyc	IR	PC	END	Ra	Rb	Rc	Read Reg	Data Out	Reg Alu IN clk	Cond Log Aux	PC add	Reg Alu Out clk	Try Buf 0	Try Buf 1	Try Buf 4	Cond Log
1	X								X						X	
2		X									X		X			
3						X		X								X
4					X			X		X						
5			X													

Table 6.14 BRANCH instruction signal activation by clock cycle

LOAD									
Cycle	IR	PC	Rb	Reg Alu IN clk	pcadd	Load Decoder	Reg alu out	Try buff 0	Try buff 4
1	X			X					X
2		X			X			X	
3			X	X		X			

Table 6.15 LOAD instruction signal activation by clock cycle

Cycle	End	sum	Reg alu out	Try buff 2	Try buff 1	MAen	Mem read	Reg Mem Out	Ra	Read Reg	Try Buff 5
4		X	X	X							
5					X	X					
6							X	X			
7									X	X	X
8	X										

Table 6.15 LOAD instruction signal activation by clock cycle (cont)

STORE									
Cycle	IR	PC	Rb	Reg Alu IN clk	pcadd	Load Decoder	Reg alu out	Try buff 0	Try buff 4
1	X			X					X
2		X			X			X	
3			X	X		X			

Table 6.16 STORE instruction signal activation by clock cycle

STORE															
Cyc	sum	Reg alu out	Try buff 2	Try buff 1	MAen	Mden	Mem read	Reg Mem Out	Ra	Read Reg	Data Out	Try Buff 5	End	Mem store	Try buff 9
4	X	X	X												
5				X	X										
6						X			X		X				X
7														X	
8													X		

Table 6.16 STORE instruction signal activation by clock cycle (cont)

IN										
Cycle	IR	PC	END	Ra	Read Reg	Reg Alu IN clk	pcadd	Try buff 0	Try buff 4	Try buff 8
1	X					X			X	
2		X					X	X		
3				X	X					X
4			X							

Table 6.17 IN instruction signal activation by clock cycle

OUT										
Cycle	IR	PC	END	RB	Data Out	Reg Alu IN clk	pcadd	Try buff 0	Try buff 4	Reg Out
1	X					X			X	
2		X					X	X		
3				X	X					X
4			X							

Table 6.18 OUT instruction signal activation by clock cycle

READ PSW											
Cycle	IR	PC	END	Ra	Read Reg	Reg Alu IN clk	pcadd	Try buff 0	Try buff 4	Try buff 6	PSW
1	X					X			X		
2		X					X	X			
3				X	X					X	X
4			X								

Table 6.19 READ PSW instruction signal activation by clock cycle

At this stage we have seen the instruction activation per cycle. The next step consists in transforming each signal (each column of those tables) in one specific digital circuit, analyzing per instruction cycle the activated signals. The next step consists in designing digital logic circuits that become asserted when those conditions occur. Note that each row of signal activation is a function of the instruction executed and its cycles.

Ra	
INSTRUCTION	**CYCLE**
1) ADD	5
2) SUB	5
3) AND	5
4) OR	5
5) SUBi	5
6) ADDi	5
7) ANDi	5
8) Ori	5
9) SHR	5
10) SHL	5
11) SHRA	5
12) SHC	5
13) LOAD	7
14) STORE	6
15) IN	3
16) READ PSW	3
17) NOT	4

Table 6.20 Ra signal activation in terms of instructions and cycles

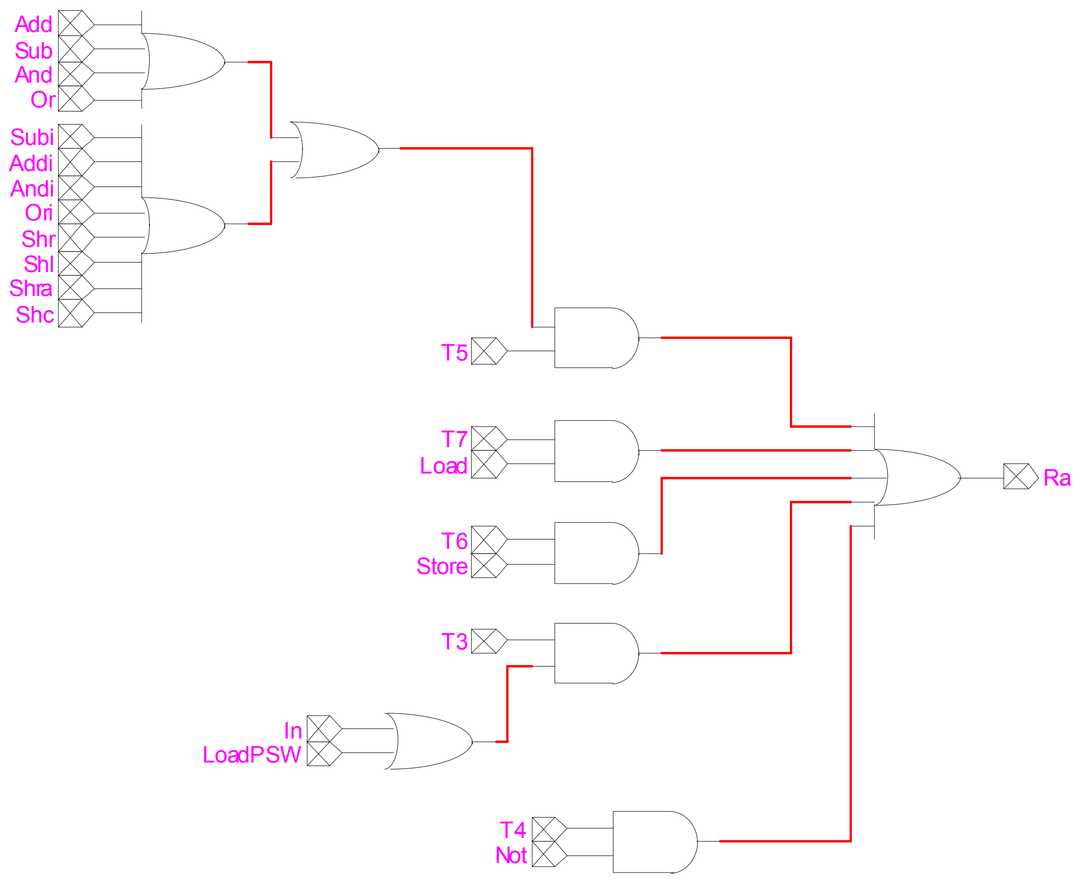


Figure 6.6 Ra signal circuit implementation

Regmemout signal activation

Regmemout	
INSTRUCTION	CYCLE
LOAD	6

Table 6.21 Regmemout signal activation in terms of instructions and cycles

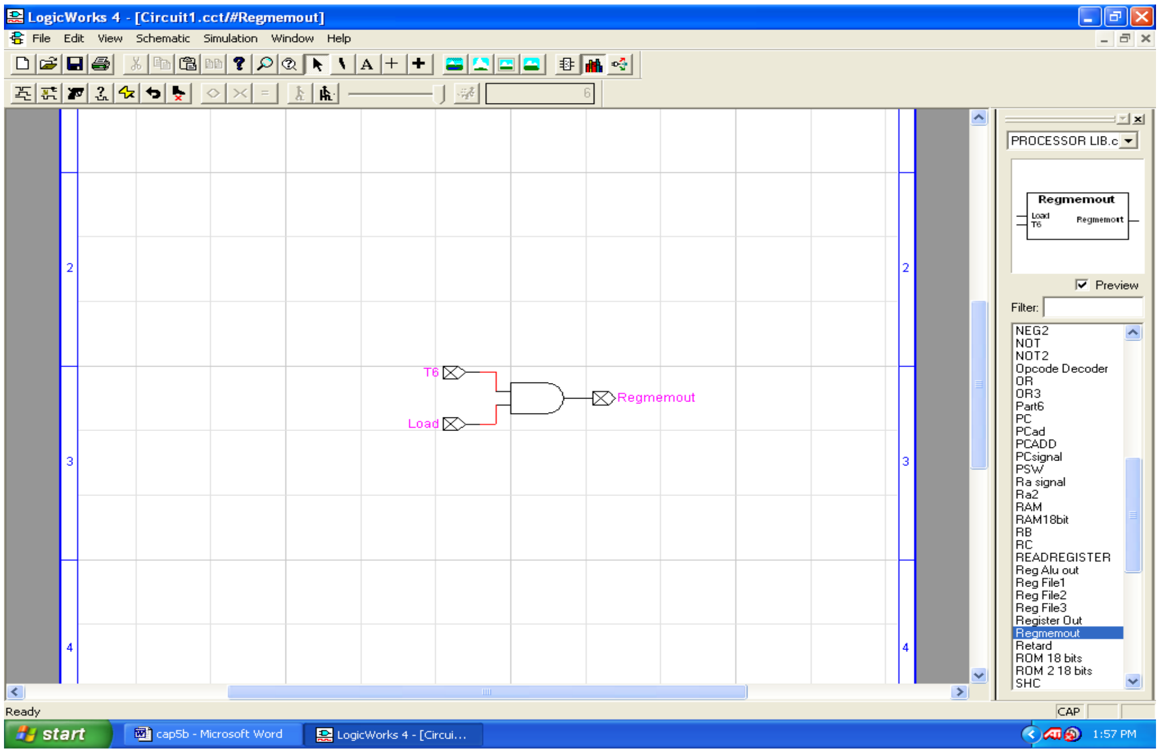


Figure 6.7 Regmemout signal circuit implementation

Reg Alu out	
INSTRUCTION	CYCLE
1) ADD	4
2) SUB	4
3) AND	4
4) OR	4
5) SUBi	4
6) ADDi	4
7) ANDi	4
8) Ori	4

9) SHR	4
10) SHL	4
11) SHRA	4
12) SHC	4
13) LOAD	4
14) STORE	4
15) NOT	3

Table 6.22 Reg Alu out signal activation in terms of instructions and cycles

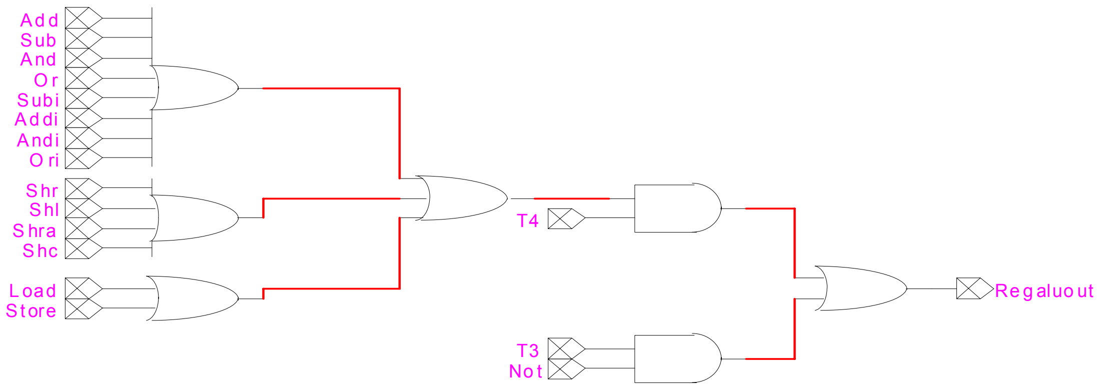


Figure 6.8 Reg Alu out signal circuit implementation

TRISTATE BUFER 0	
INSTRUCTION	CYCLE
1) ADD	2
2) SUB	2
3) AND	2
4) OR	2
5) SUBi	2
6) ADDi	2
7) ANDi	2
8) Ori	2
9) SHR	2
10) SHL	2
11) SHRA	2
12) SHC	2
13) LOAD	2
14) STORE	2
15) IN	2
16) OUT	2
17) READ PSW	2
18) UNC JUMP	2
19) NOT	2
20) BRANCH	2

Table 6.23 Tristate buffer 0 signal activation in terms of instructions and cycles

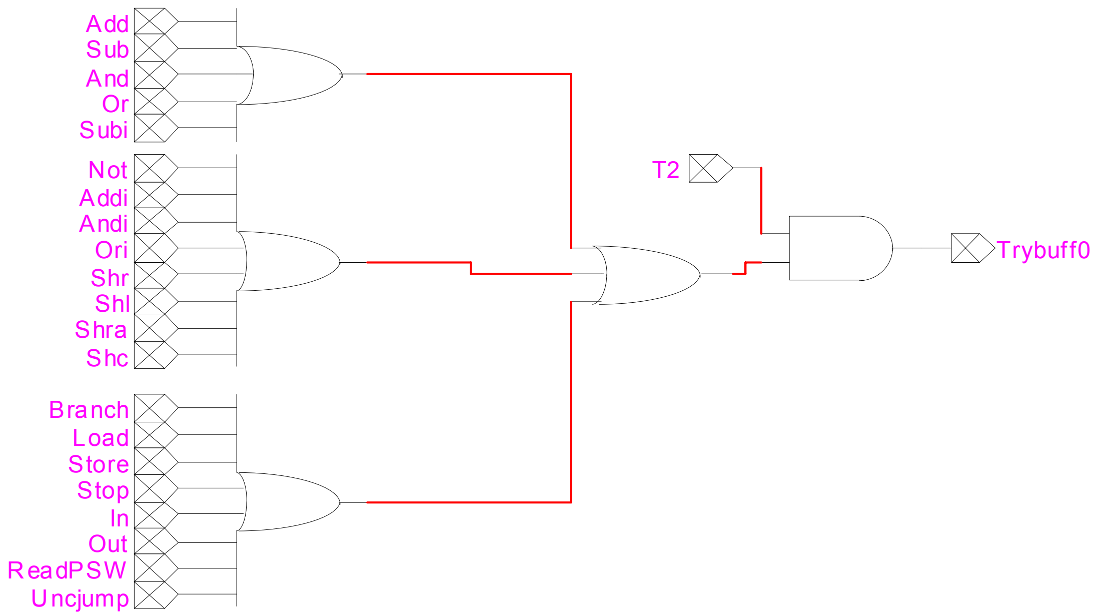


Figure 6.9 Trybuff 0 signal circuit implementation

TRYBUF1	
INSTRUCTION	**CYCLE**
1) ADD	5
2) SUB	5
3) AND	5
4) OR	5
5) SUBi	5
6) ADDi	5
7) ANDi	5
8) Ori	5
9) SHR	5
10) SHL	5
11) SHRA	5
12) SHC	5
13) LOAD	5
14) STORE	5
15) NOT	4

Table 6.24 Tristate buffer 1-signal activation in terms of instructions and cycles

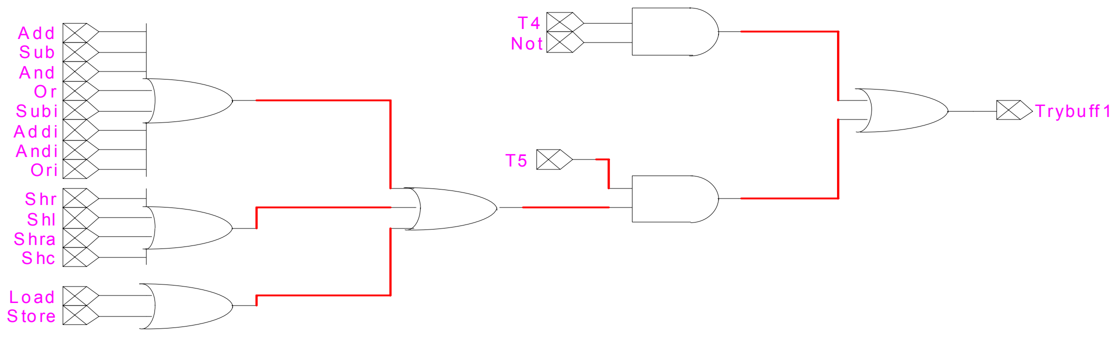


Figure 6.10 Trybuf 1 signal circuit implementation

TRISTATE BUFFER 2	
INSTRUCTION	CYCLE
1) SUBi	4
2) ADDi	4
3) ANDi	4
4) ORi	4
5) LOAD	4
6) STORE	4
7) UNCJUMP	3

Table 6.25 Tristate buffer 2-signal activation in terms of instructions and cycles

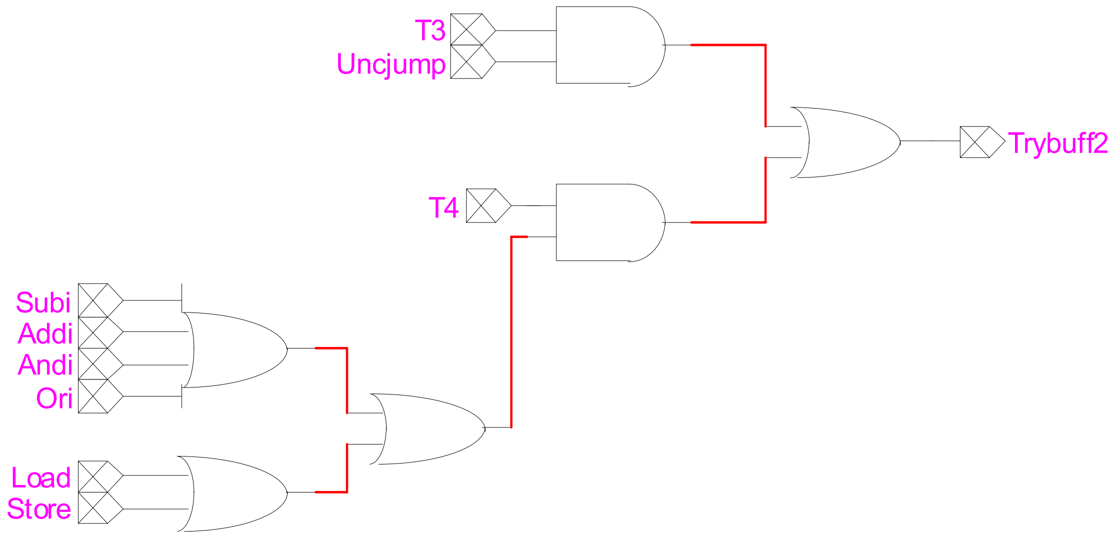


Figure 6.11 Trybuff2 signal circuit implementation

DATA OUT	
INSTRUCTION	**CYCLE**
1) ADD	3,4
2) SUB	3,4
3) AND	3,4
4) OR	3,4
5) SUBi	3
6) ADDi	3
7) ANDi	3
8) Ori	3
9) SHR	3
10) SHL	3
11) SHRA	3
12) SHC	3
13) BRANCH	3,4
14) STORE	6
15) OUT	3
16) NOT	3

Table 6.26 Data out signal activation in terms of instructions and cycles

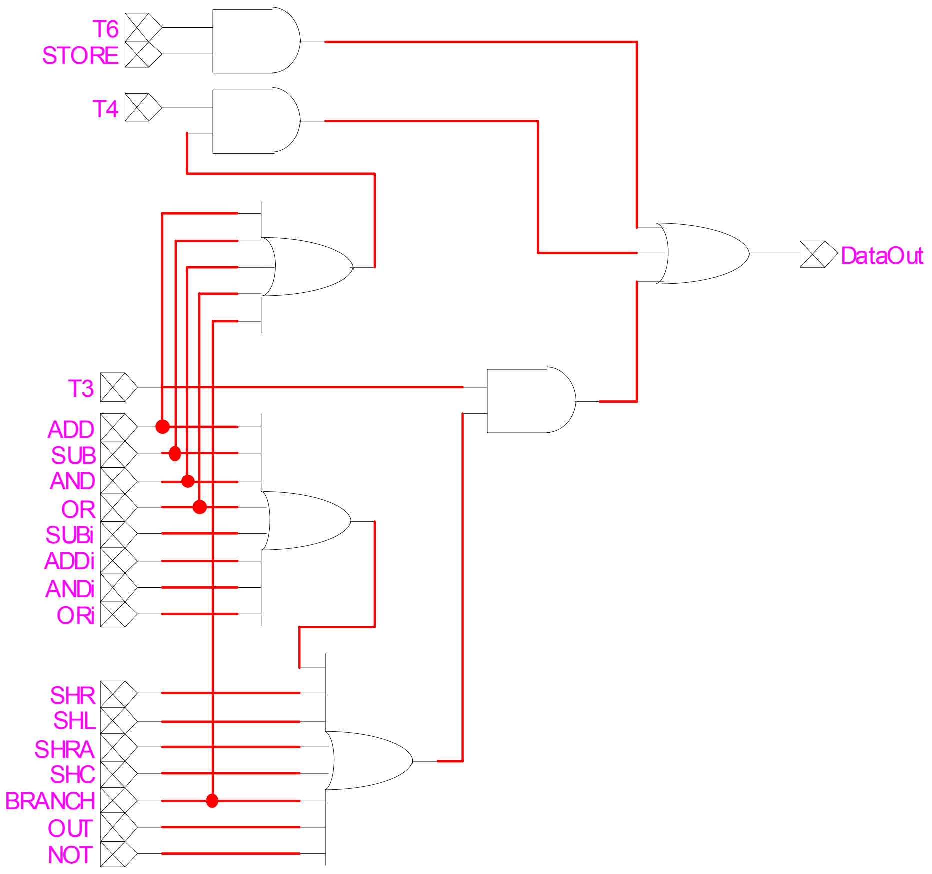


Figure 6.12 Data out signal circuit implementation

REG ALU IN	
INSTRUCTION	CYCLE
1) ADD	1,3
2) SUB	1,3
3) AND	1,3
4) OR	1,3
5) SUBi	1,3
6) ADDi	1,3
7) ANDi	1,3
8) Ori	1,3
9) SHR	1,3
10) SHL	1,3
11) SHRA	1,3
12) SHC	1,3
13) LOAD	1,3
14) STORE	1,3
15) IN	1
16) OUT	1
17) READ PSW	1
18) UNC JUMP	1
19) NOT	1
20) BRANCH	1

Table 6.27 Reg Alu in signal activation in terms of instructions and cycles

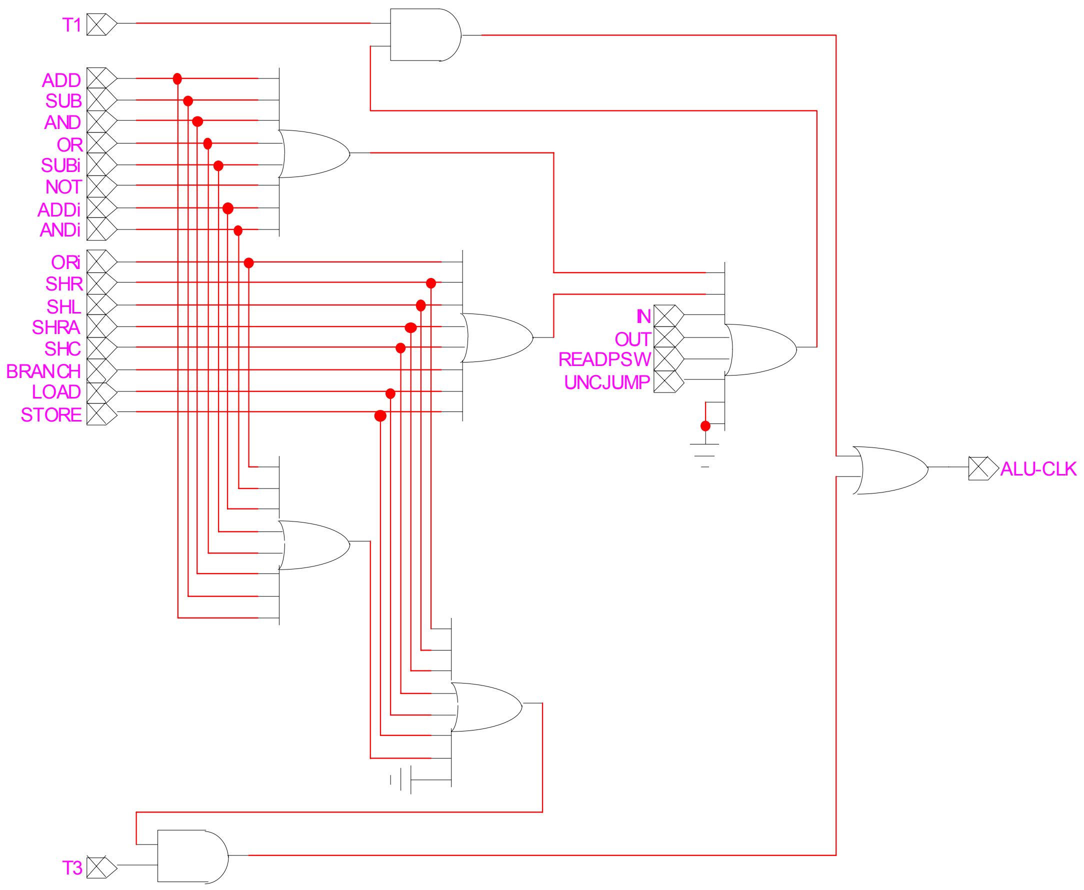


Figure 6.13 ALU clock signal circuit implementation

Rb	
INSTRUCTION	CYCLE
1) ADD	3
2) SUB	3
3) AND	3
4) OR	3
5) SUBi	3
6) ADDi	3
7) ANDi	3
8) Ori	3
9) SHR	3
10) SHL	3
11) SHRA	3
12) SHC	3
13) LOAD	3
14) STORE	3
15) OUT	3
16) BRANCH	4

Table 6.28 Rb signal activation in terms of instructions and cycles

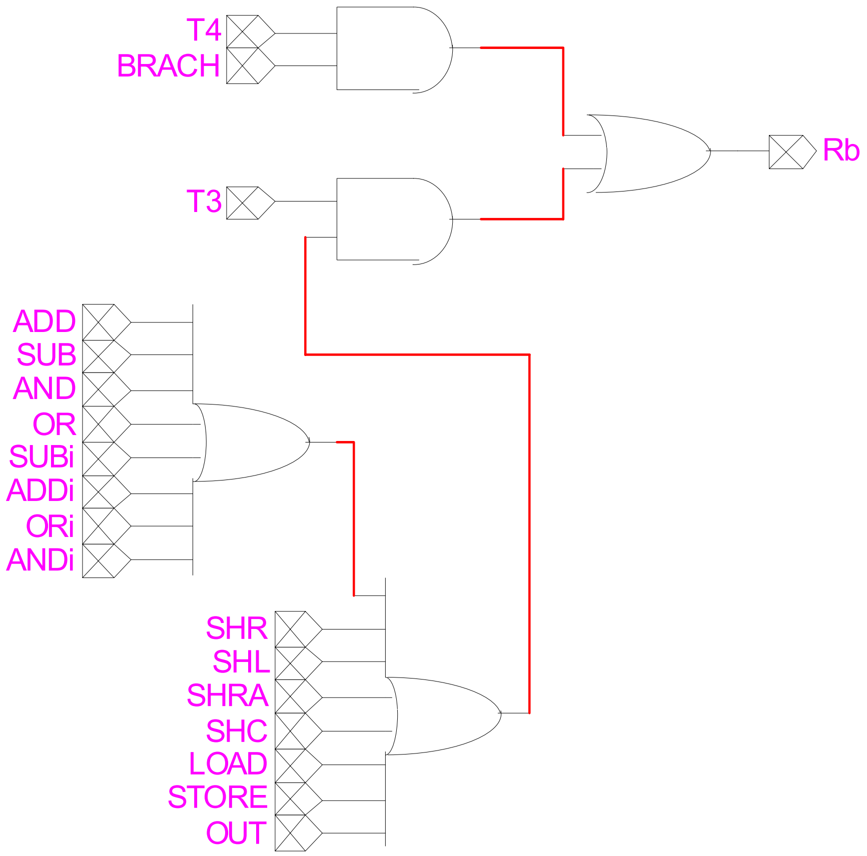


Figure 6.14 Rb clock signal circuit implementation

Rc	
INSTRUCTION	**CYCLE**
1) ADD	4
2) SUB	4
3) AND	4
4) OR	4
5) NOT	3
6) SHR	4
7) SHL	4
8) SHRA	4
9) SHC	4
10) BRANCH	3

Table 6.29 Rc signal activation in terms of instructions and cycles

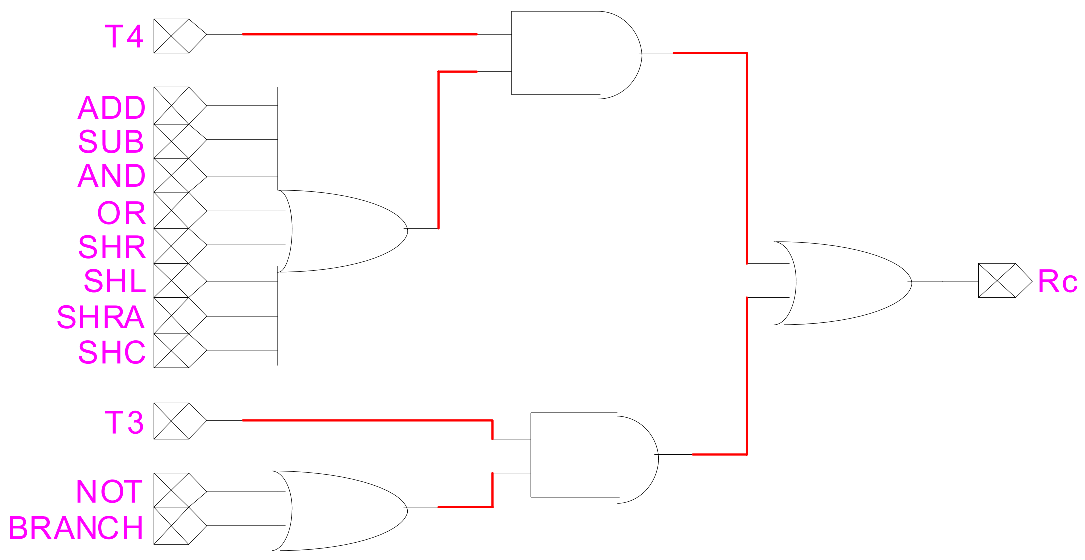


Figure 6.15 Rc signal circuit implementation

READ REGISTER	
INSTRUCTION	**CYCLE**
1) ADD	5
2) SUB	5
3) AND	5
4) OR	5
5) SUBi	5
6) ADDi	5
7) ANDi	5
8) Ori	5
9) SHR	5
10) SHL	5
11) SHRA	5
12) SHC	5
13) LOAD	7
14) IN	3
15) READ PSW	3
16) NOT	4

Table 6.30 Read register signal activation in terms of instructions and cycles

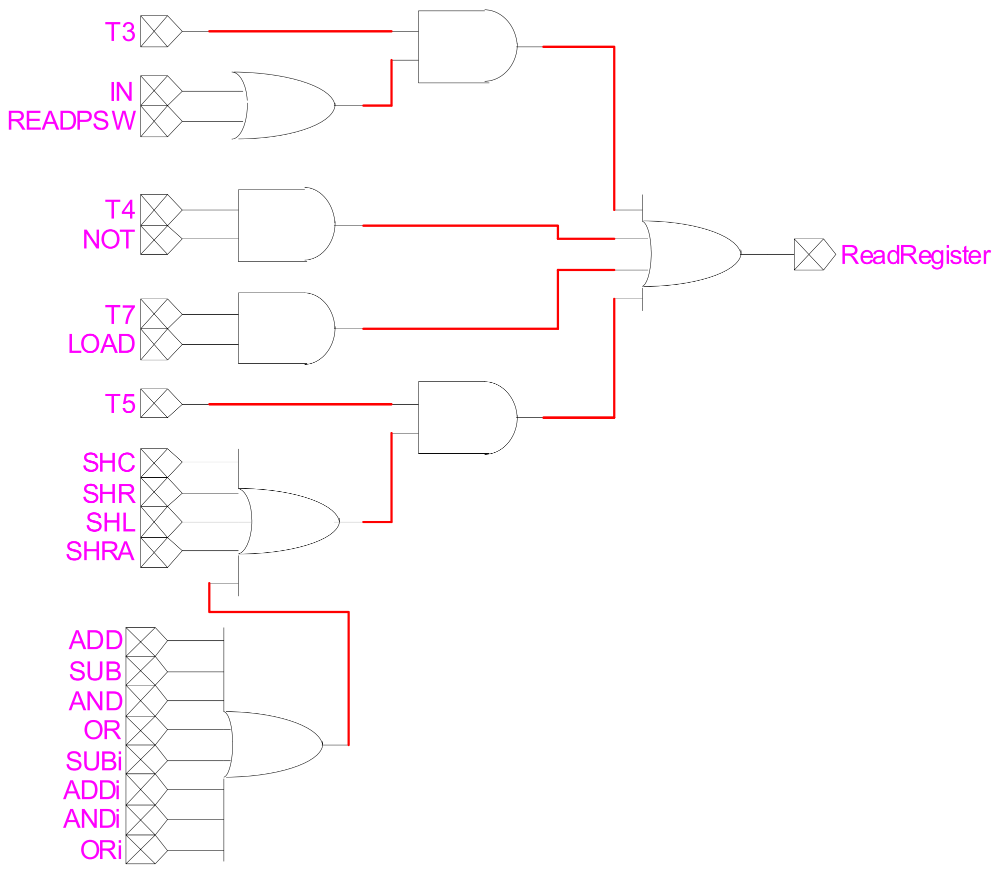


Figure 6.16 Read Register signal circuit implementation

PC SIGNAL ACTIVATION	
INSTRUCTION	CYCLE
1) ADD	2
2) SUB	2
3) AND	2
4) OR	2
5) SUBi	2
6) ADDi	2
7) ANDi	2
8) Ori	2
9) SHR	2
10) SHL	2
11) SHRA	2
12) SHC	2
13) LOAD	2
14) STORE	2
15) IN	2
16) OUT	2
17) READ PSW	2
18) UNC JUMP	2,3
19) NOT	2
20) BRANCH	2

Table 6.31 PC signal activation in terms of instructions and cycles

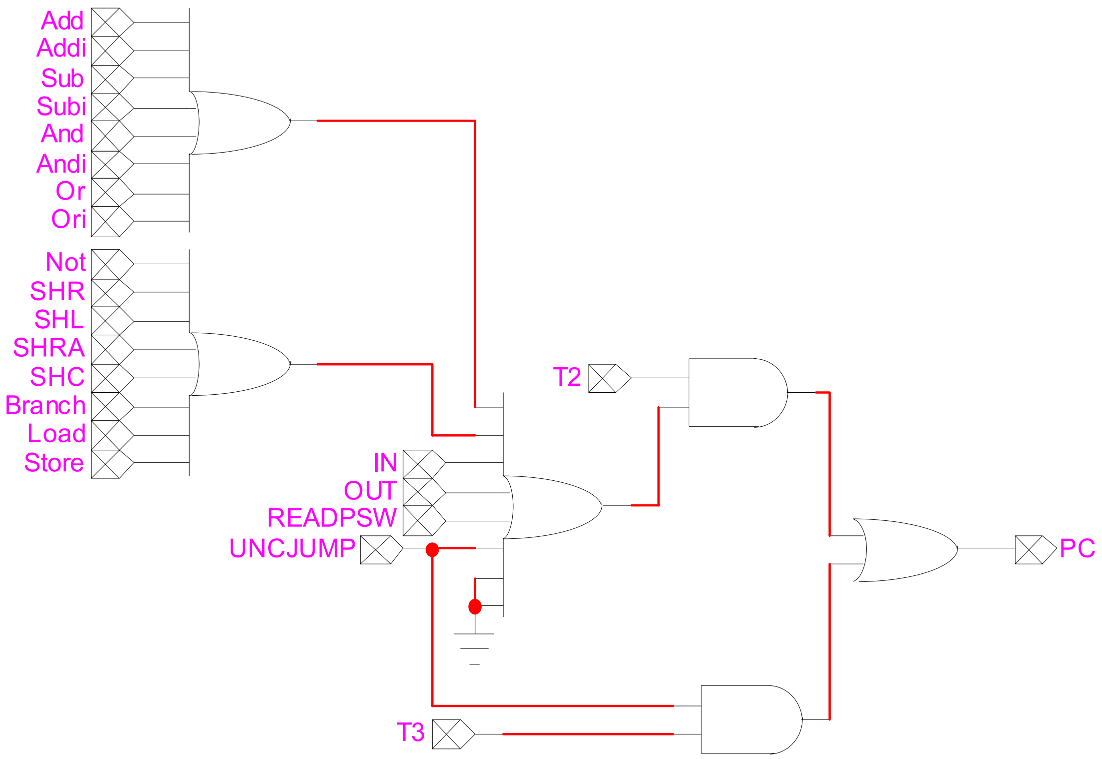


Figure 6.17 PC signal circuit implementation

END SIGNAL ACTIVATION	
INSTRUCTION	CYCLE
1) ADD	6
2) SUB	6
3) AND	6
4) OR	6
5) SUBi	6
6) ADDi	6
7) ANDi	6
8) Ori	6
9) SHR	6
10) SHL	6
11) SHRA	6
12) SHC	6
13) LOAD	8
14) STORE	8
15) IN	4
16) OUT	4
17) READ PSW	4
18) UNC JUMP	4
19) NOT	5
20) BRANCH	5

Table 6.32 END signal activation in terms of instructions and cycles

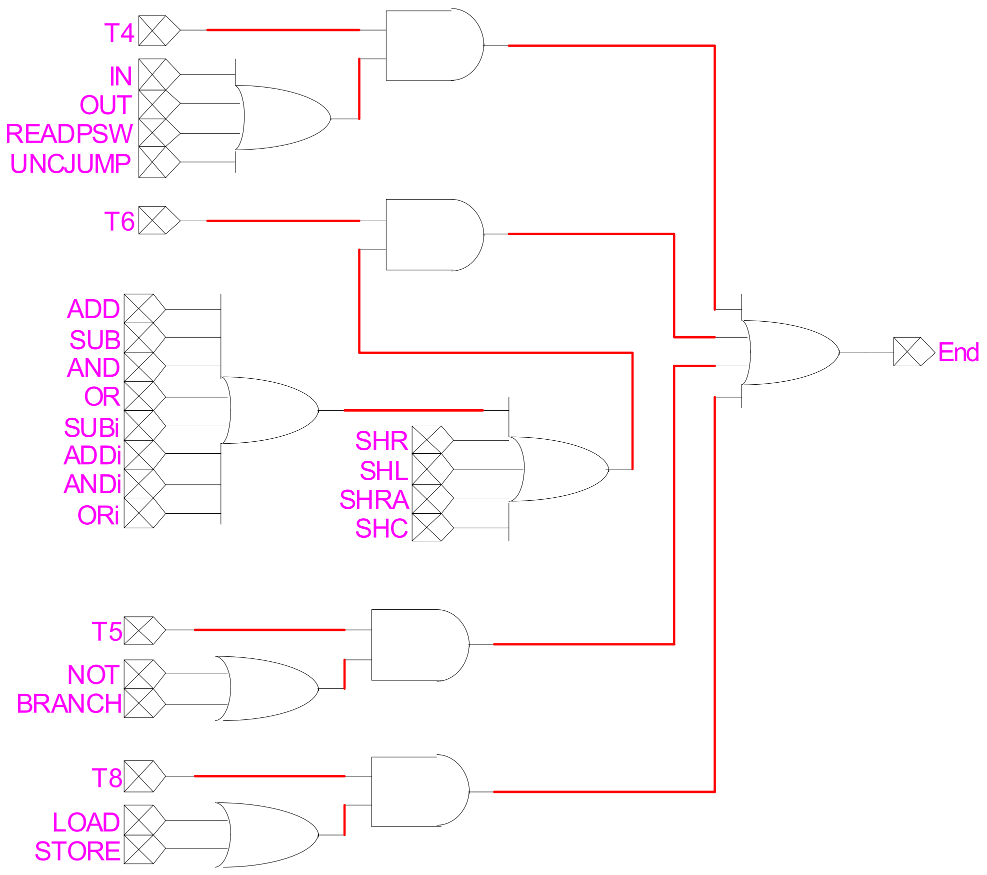


Figure 6.18 END signal circuit implementation

Those circuits already illustrated shall guide the reader to do the same with the rest of the signals. For illustrative purposes, just the remaining implementation circuit signals will be shown, but all of them where obtained using its corresponding signal activation table.

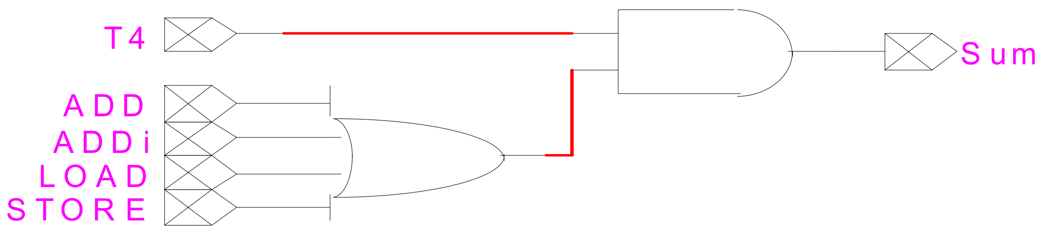


Figure 6.19 SUM signal circuit implementation

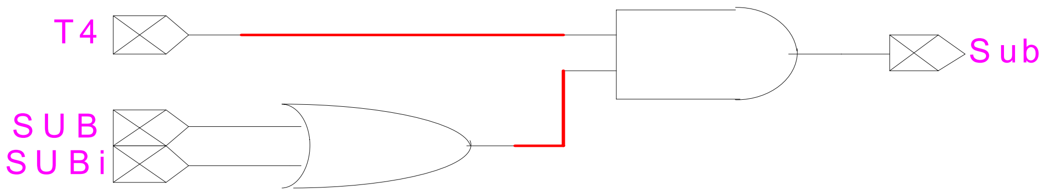


Figure 6.20 Sub signal circuit implementation

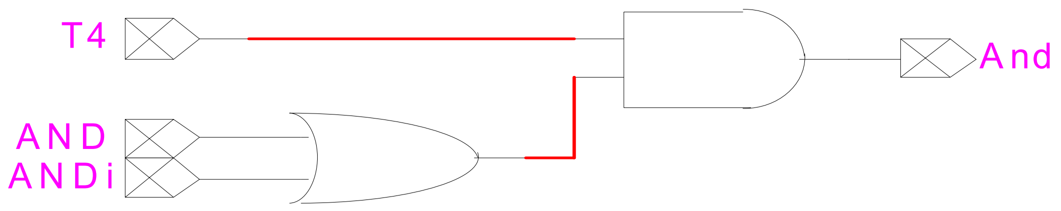


Figure 6.21 And signal circuit implementation

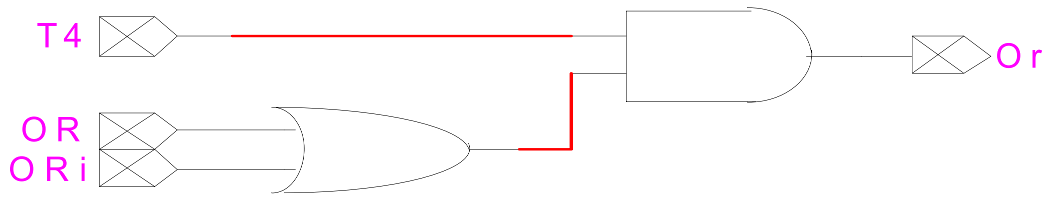


Figure 6.22 Or signal circuit implementation

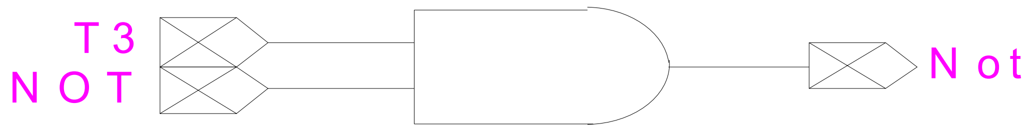


Figure 6.23 Not signal circuit implementation

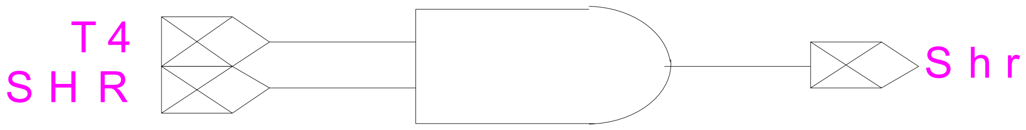


Figure 6.24 Shr signal circuit implementation

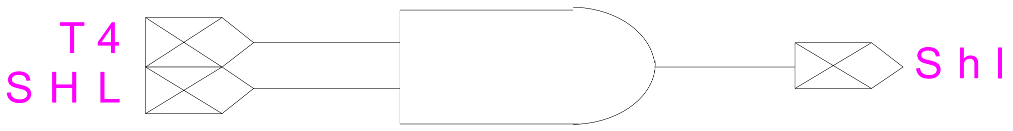


Figure 6.25 Shl signal circuit implementation

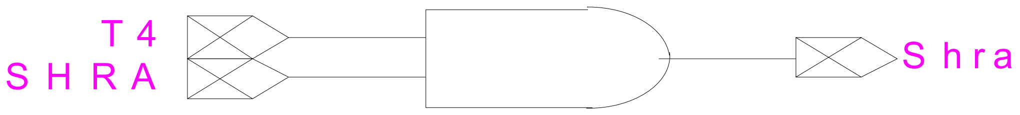


Figure 6.26 Shra signal circuit implementation

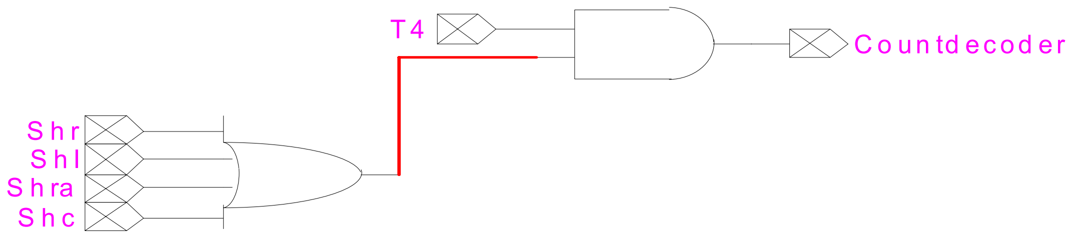


Figure 6.27 Count Decoder signal circuit implementation

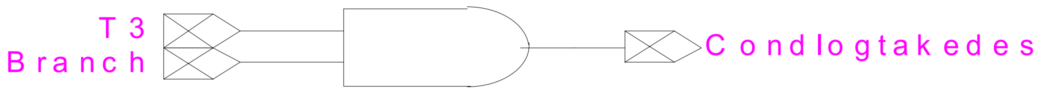


Figure 6.28 Conditional Logic signal circuit implementation

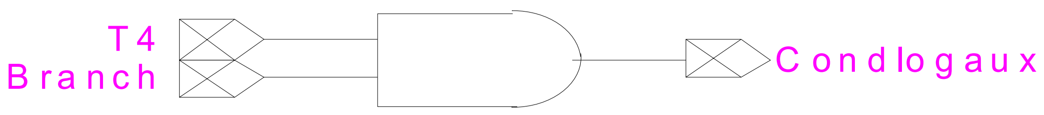


Figure 6.29 Condlogaux signal circuit implementation

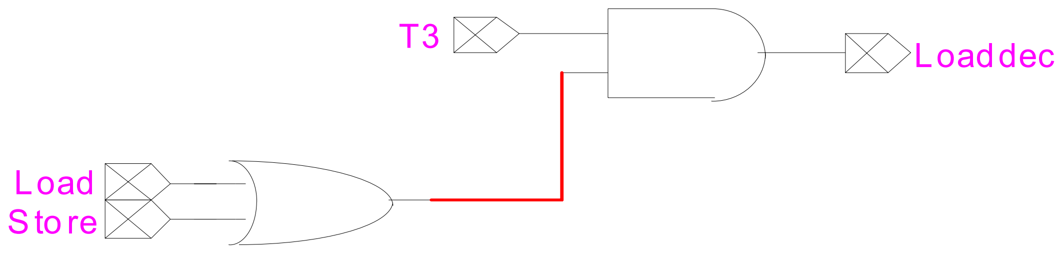


Figure 6.30 Load Decoder signal circuit implementation

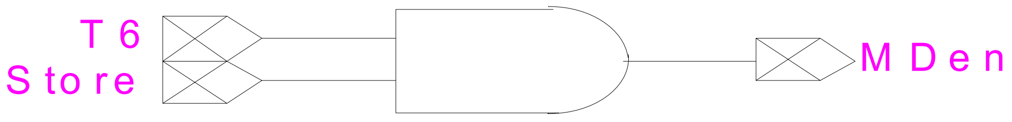


Figure 6.31 MDen signal circuit implementation

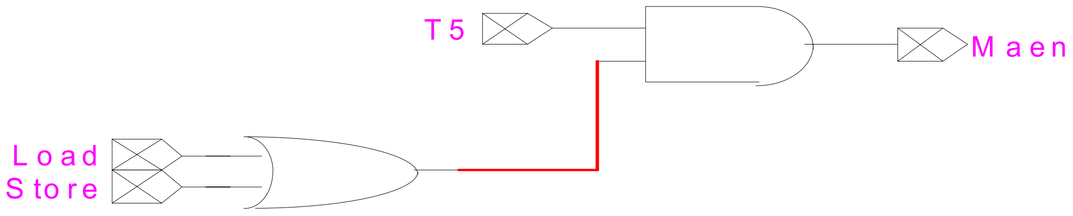


Figure 6.32 MAen signal circuit implementation

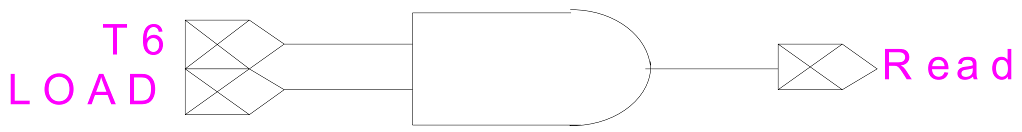


Figure 6.33 Read signal circuit implementation

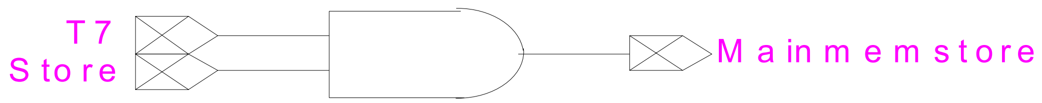


Figure 6.34 Main memory store signal circuit implementation

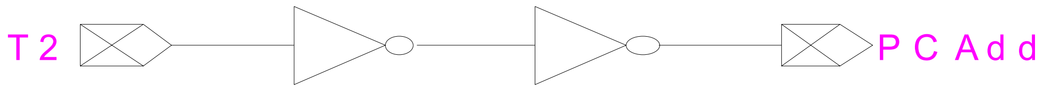


Figure 6.35 PC add signal circuit implementation

REGALUIN	
INSTRUCTION	CYCLE
1) ADD	1
2) SUB	1
3) AND	1
4) OR	1
5) SUBi	1
6) ADDi	1
7) ANDi	1
8) Ori	1
9) SHR	1
10) SHL	1
11) SHRA	1
12) SHC	1
13) LOAD	1
14) STORE	1
15) IN	1
16) OUT	1
17) READ PSW	1
18) UNC JUMP	1
19) NOT	1
20) BRANCH	1

Table 6.33 Regaluin signal activation in terms of instructions and cycles

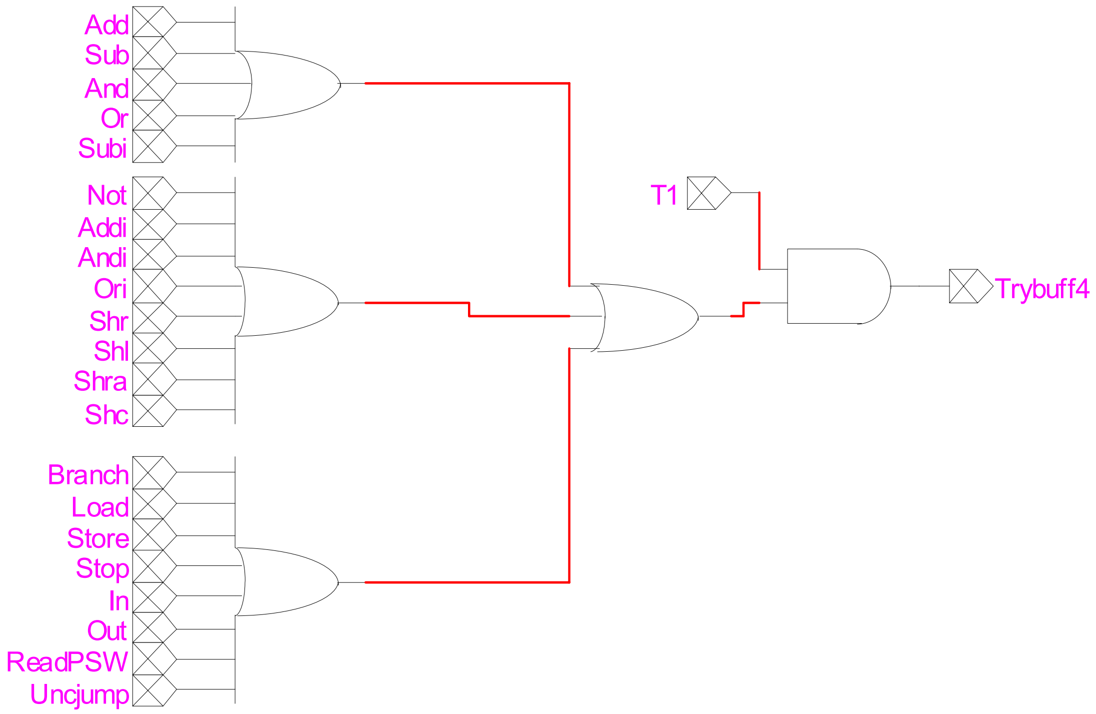


Figure 6.36 Tristate buffer 4-signal circuit implementation

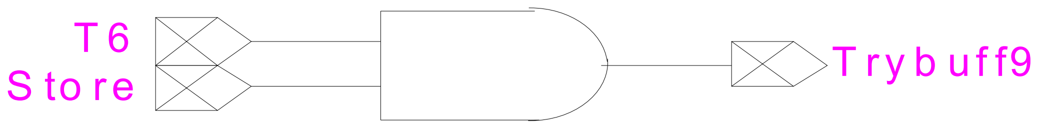


Figure 6.37 Tristate buffer 9-signal circuit implementation

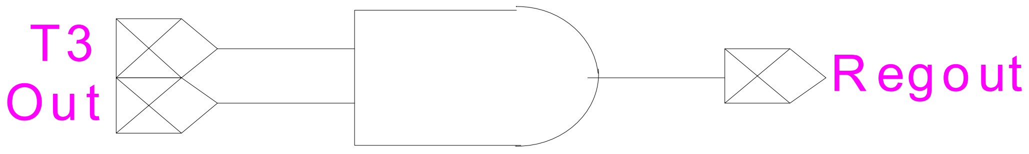


Figure 6.38 Regout signal circuit implementation

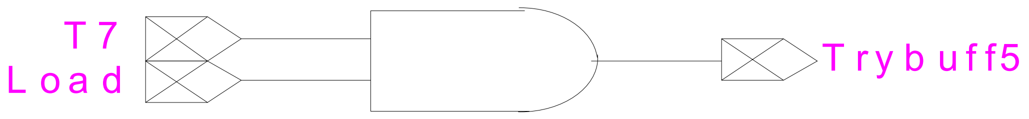


Figure 6.39 Tristate buffer 5-signal circuit implementation

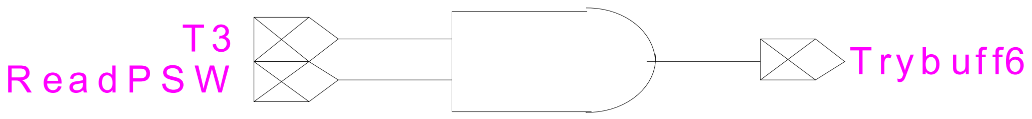


Figure 6.40 Tristate buffer 6-signal circuit implementation

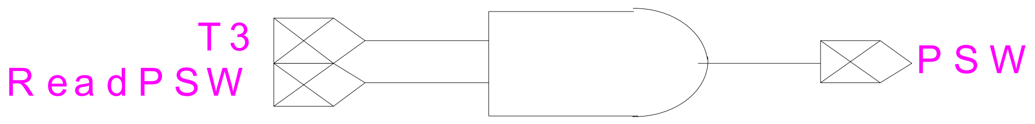


Figure 6.41 PSW register signal circuit implementation

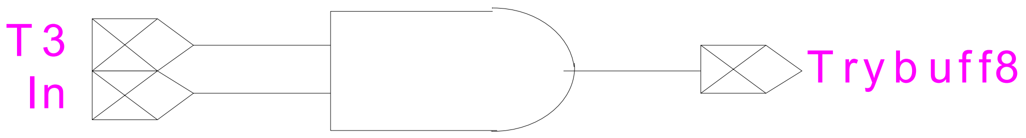


Figure 6.42 Tristate buffer 8-signal circuit implementation

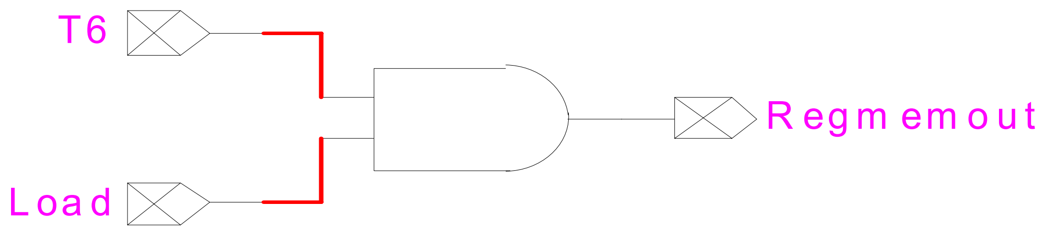


Figure 6.43 Regmemout signal circuit implementation

The Control Unit Encoder Implementation

After the 38 control signal logic circuits have been defined, the next step consists in connect all of them in just one unit called the control unit encoder. The logic circuits of this unit will receive input signals from the timer and the operational code (opcode) decoder and will activate the corresponding signals for the instruction execution. Figures 6.44 to 6.53 illustrate the circuit interconnection that forms the control unit encoder.

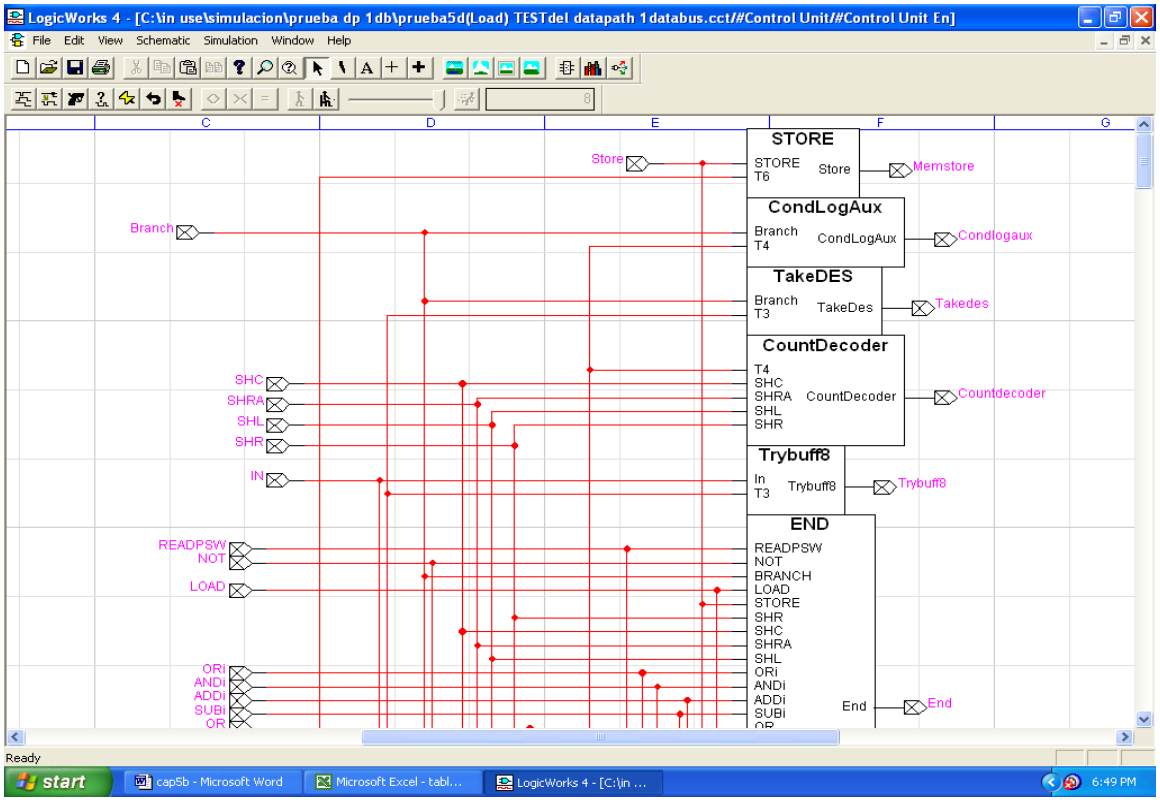


Figure 6.44 Control Unit Encoder (a)

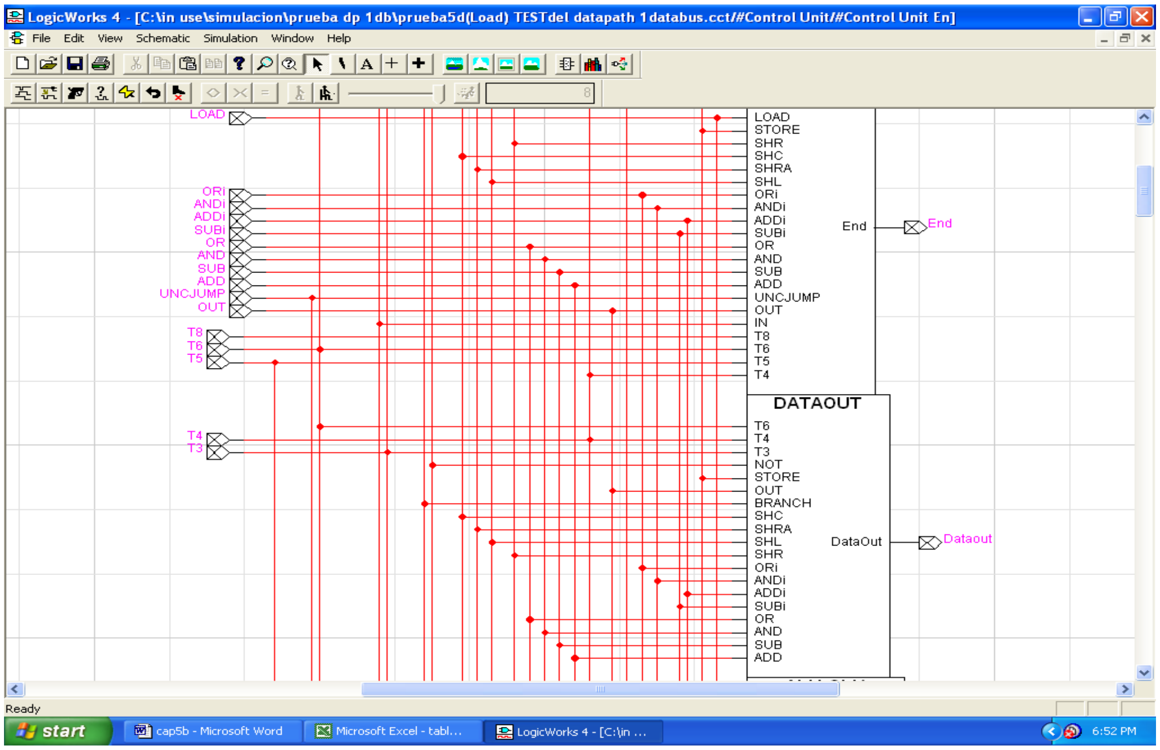

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit/#Control Unit En]
LOAD
ORI
ANDi
ADDi
SUBi
OR
AND
SUB
ADD
UNCJUMP
OUT
T8
T6
T5
T4
T3
End
DATAOUT
DataOut
Dataout
Ready

Figure 6.45 Control Unit Encoder (b)

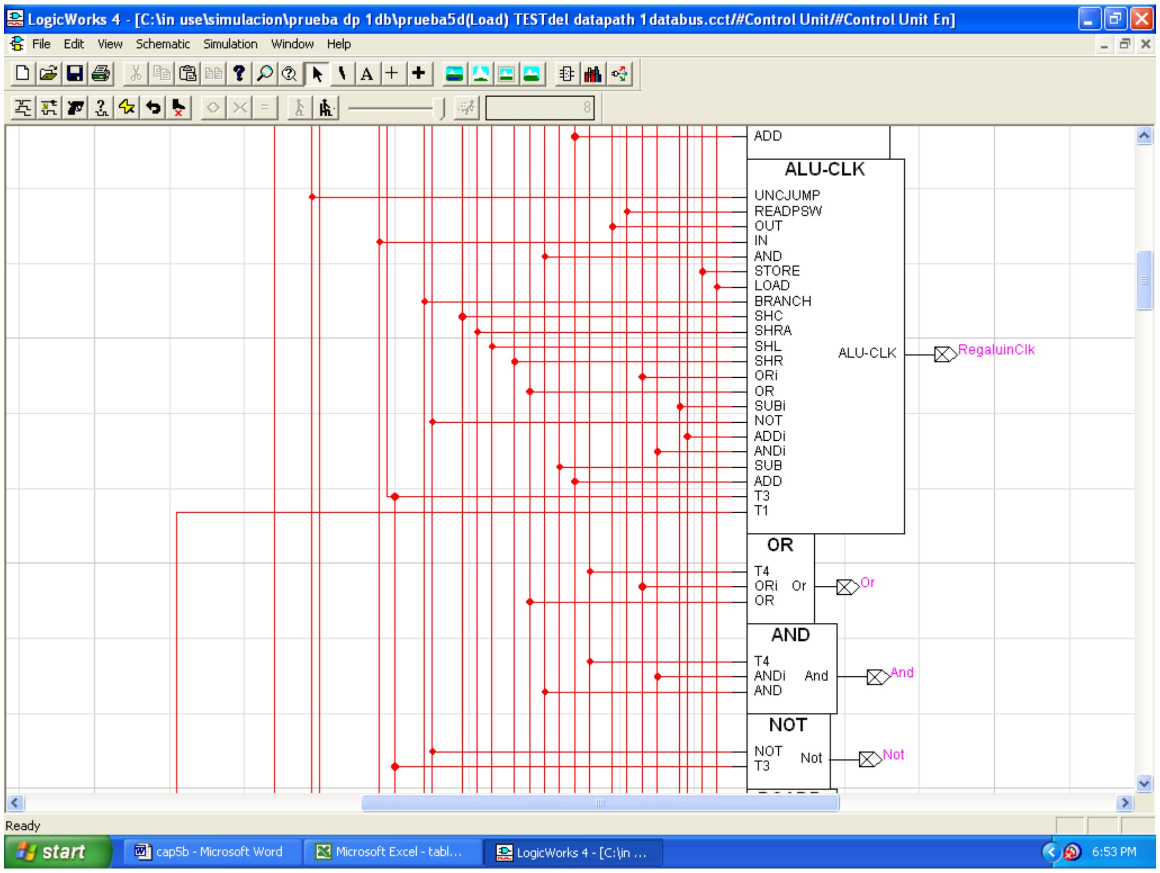


Figure 6.46 Control Unit Encoder (c)

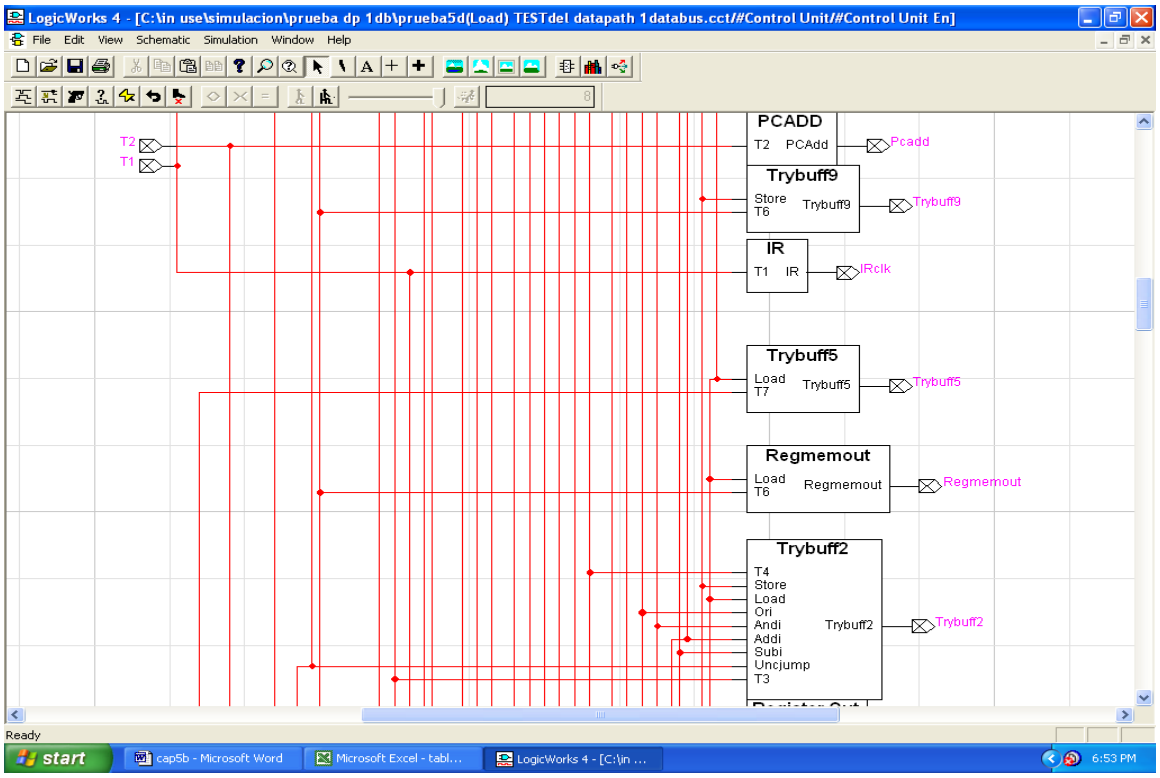


Figure 6.47 Control Unit Encoder (d)

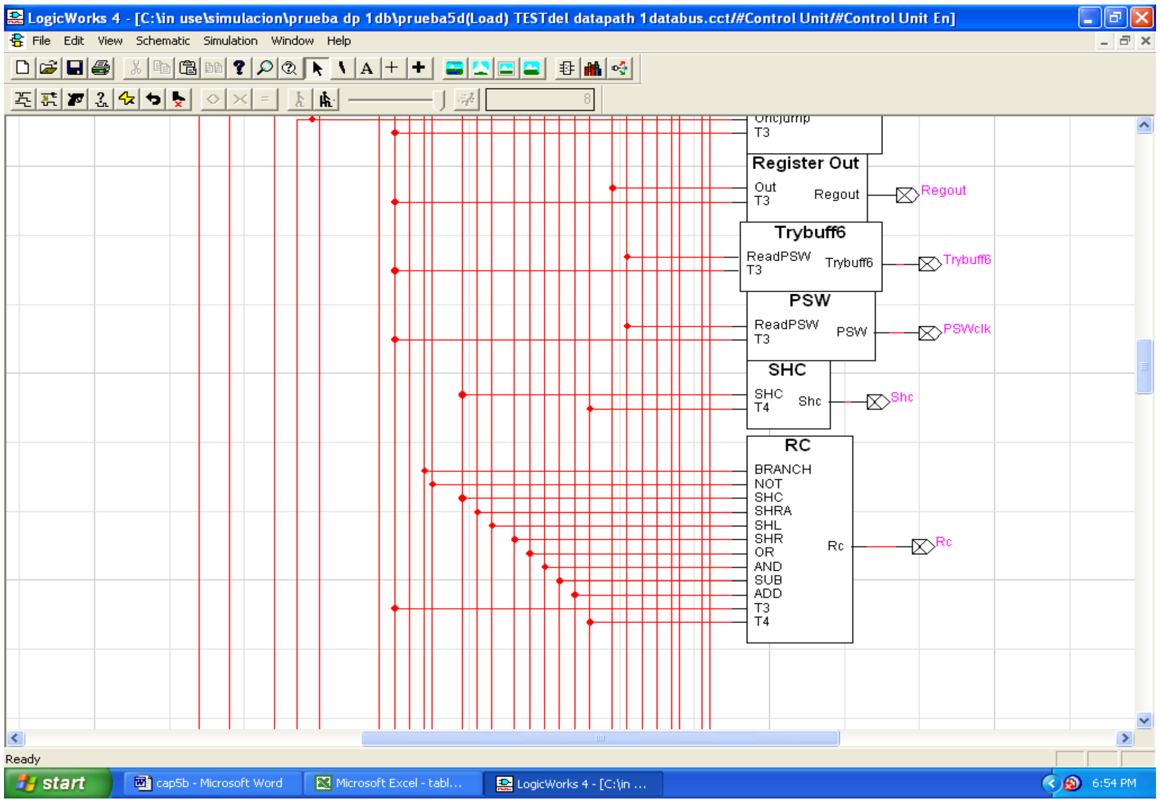

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit/#Control Unit En]
File Edit View Schematic Simulation Window Help
Uncjump
T3
Register Out
Out
T3
Regout
Regout
Trybuff6
ReadPSW
T3
Trybuff6
Trybuff6
PSW
ReadPSW
T3
PSW
PSWclk
SHC
SHC
T4
Shc
Shc
RC
BRANCH
NOT
SHC
SHRA
SHL
SHR
OR
AND
SUB
ADD
T3
T4
Rc
Rc
Ready
start
cap5b - Microsoft Word
Microsoft Excel - tabl...
LogicWorks 4 - [C:\in ...
6:54 PM

Figure 6.48 Control Unit Encoder (e)

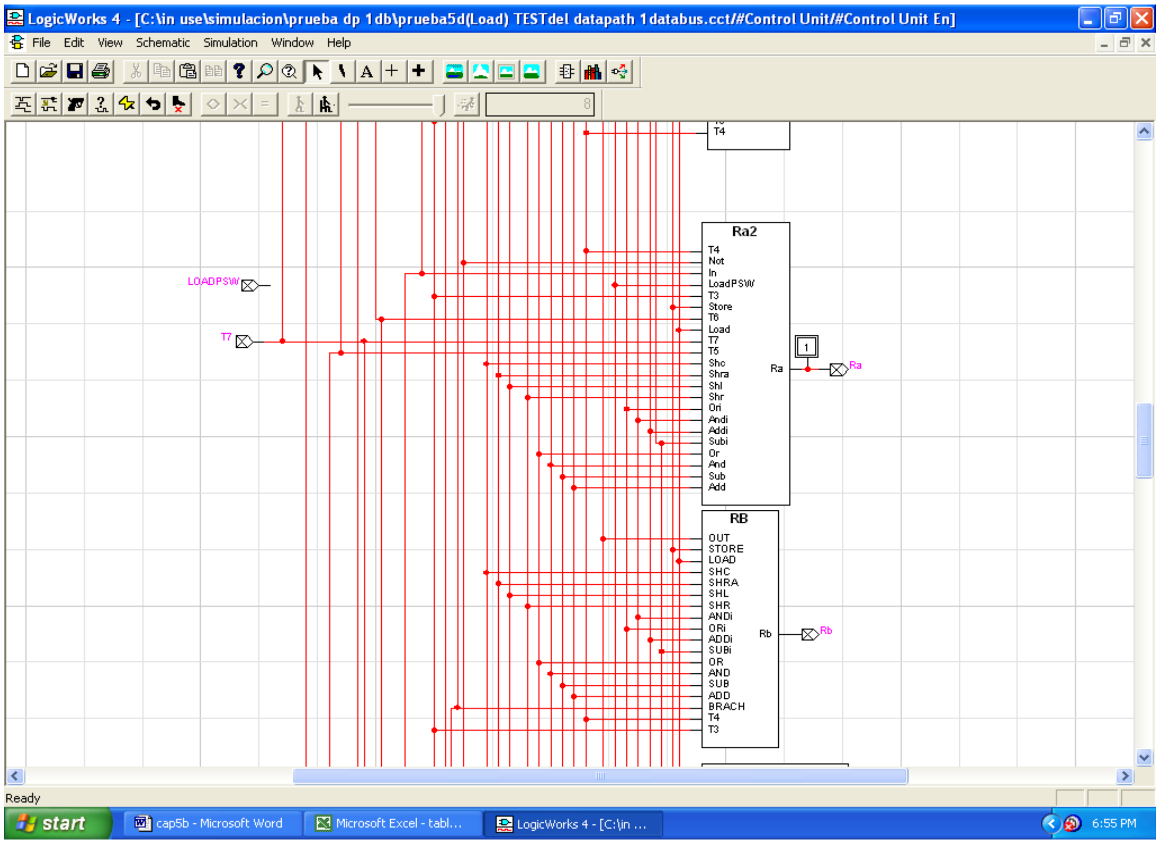

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit/#Control Unit En]
File Edit View Schematic Simulation Window Help
T4
Ra2
T4
Not
In
LoadPSW
T3
Store
T6
Load
T7
T5
Shc
Shra
Shl
Shr
Ori
Andi
Addi
Subi
Or
And
Sub
Add
Ra
1
Ra
LOADPSW
T7
RB
OUT
STORE
LOAD
SHC
SHRA
SHL
SHR
ANDi
ORi
ADDi
SUBi
OR
AND
SUB
ADD
BRACH
T4
T3
Rb
Rb
Ready
start
cap5b - Microsoft Word
Microsoft Excel - tabl...
LogicWorks 4 - [C:\in ...
6:55 PM

Figure 6.49 Control Unit Encoder (f)

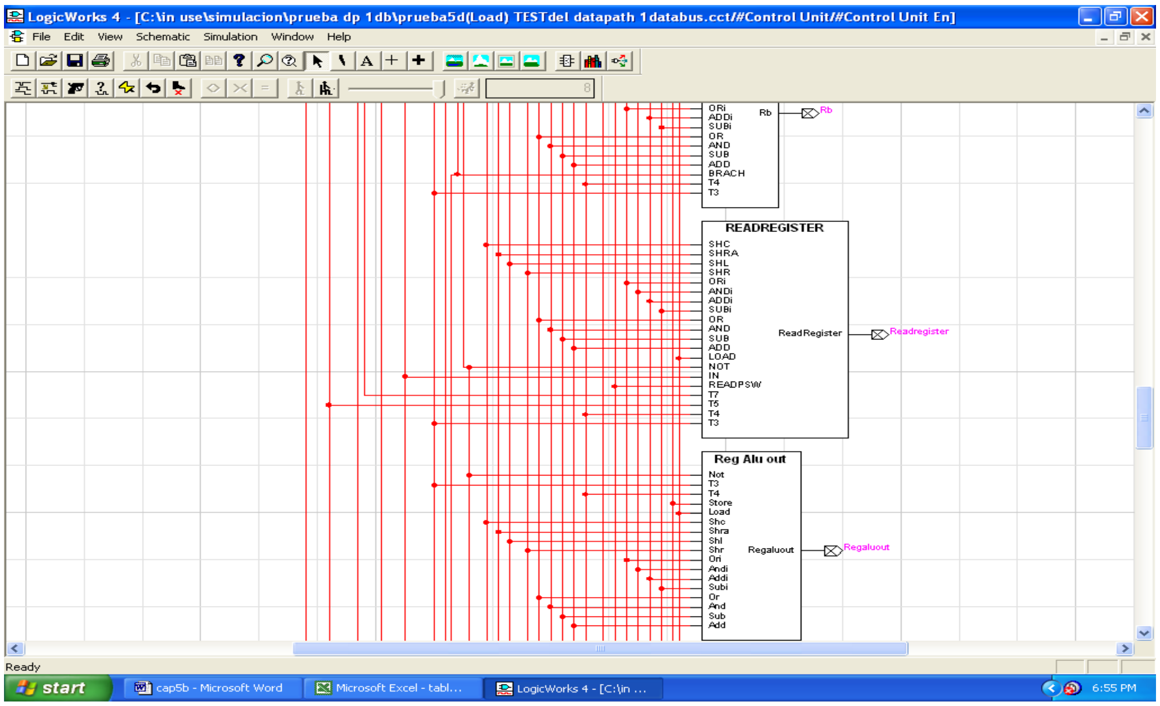


Figure 6.50 Control Unit Encoder (g)

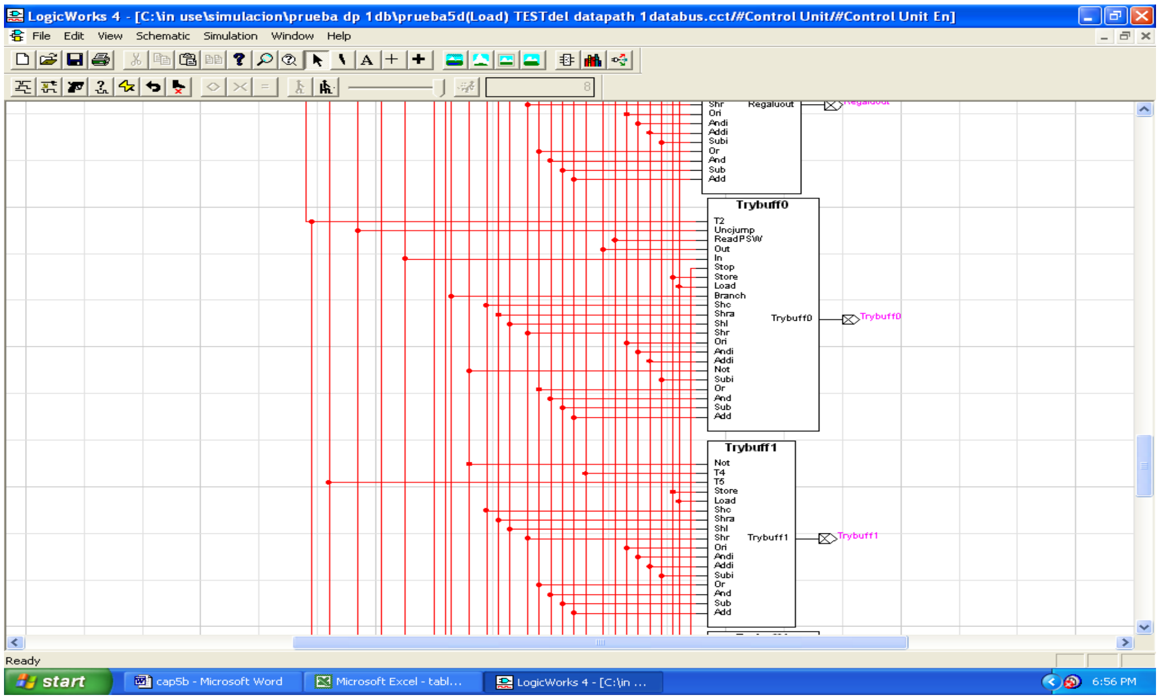

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit/#Control Unit En]
File Edit View Schematic Simulation Window Help
Regaluout
Tryhuff0
Trybuff1
Ready
start
cap5b - Microsoft Word
Microsoft Excel - tabl...
LogicWorks 4 - [C:\in ...
6:56 PM

Figure 6.51 Control Unit Encoder (h)

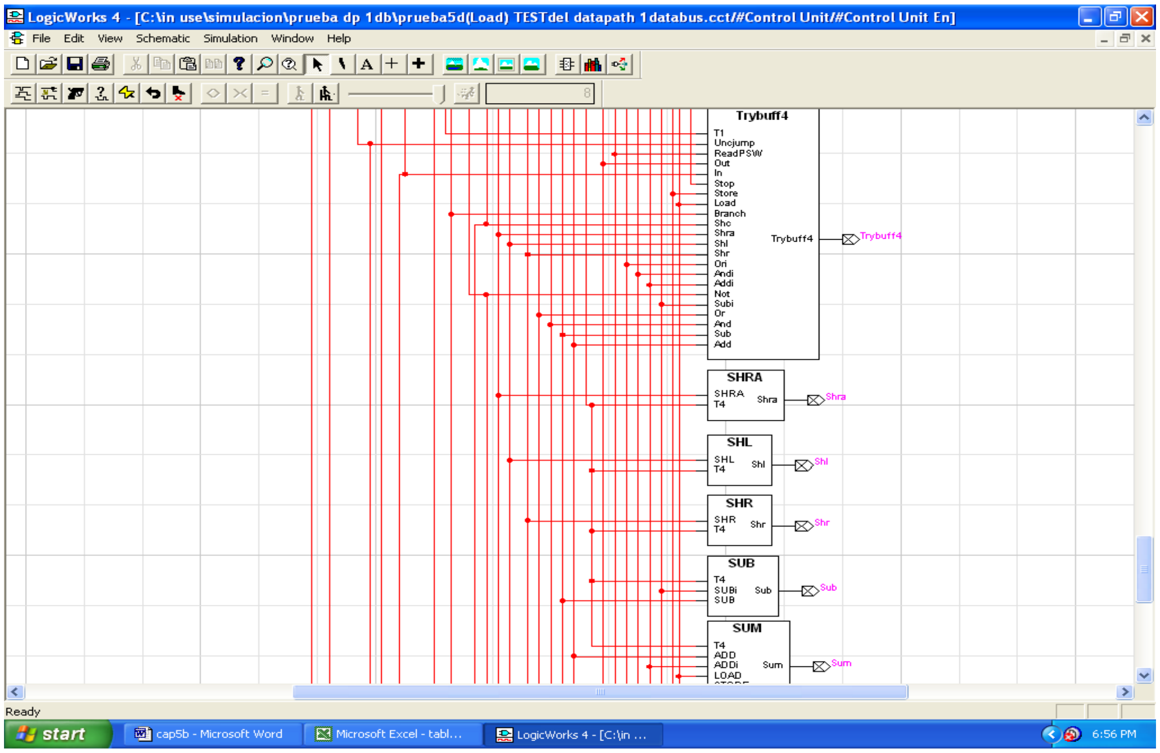

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit/#Control Unit En]
File Edit View Schematic Simulation Window Help
Trybuff4
SHRA
SHL
SHR
SUB
SUM
Shra
Shl
Shr
Sub
Sum
Ready
start
cap5b - Microsoft Word
Microsoft Excel - tabl...
LogicWorks 4 - [C:\in ...
6:56 PM

Figure 6.52 Control Unit Encoder (i)

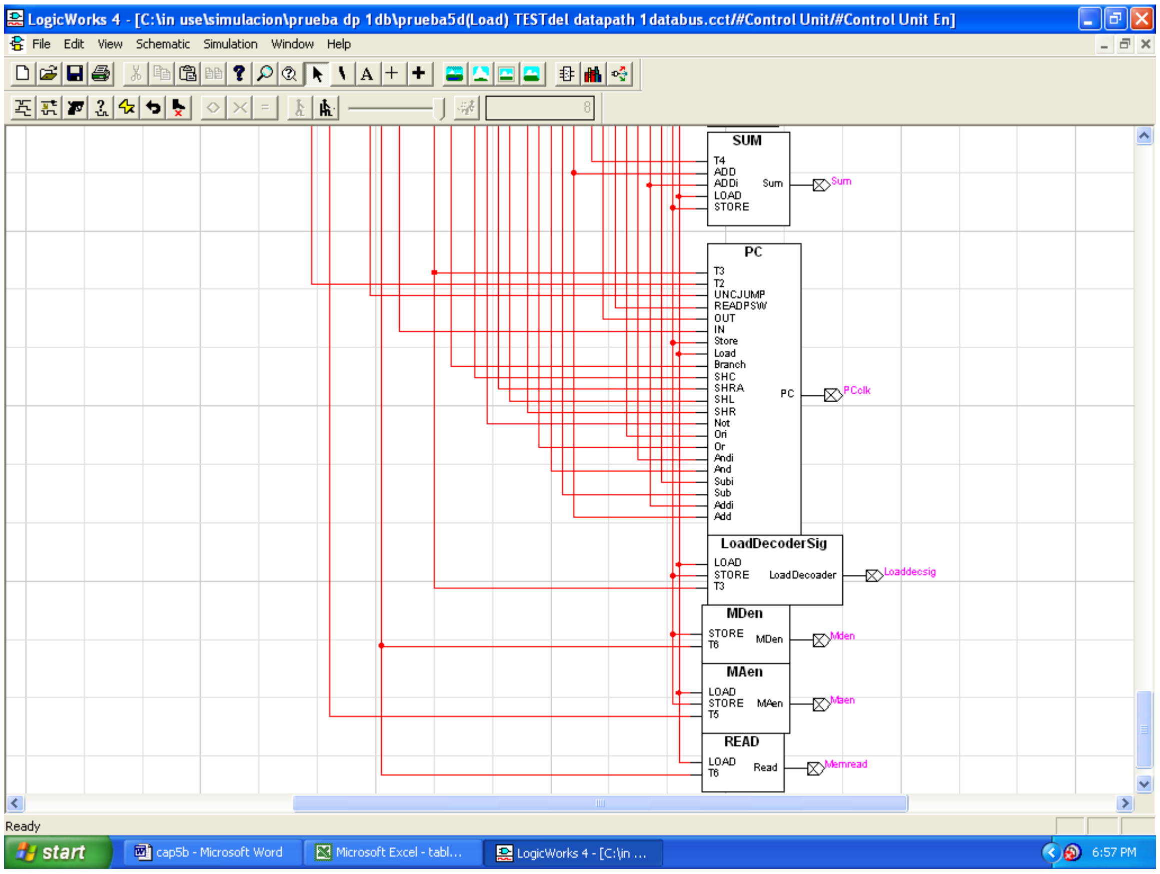


Figure 6.53 Control Unit Encoder (j)

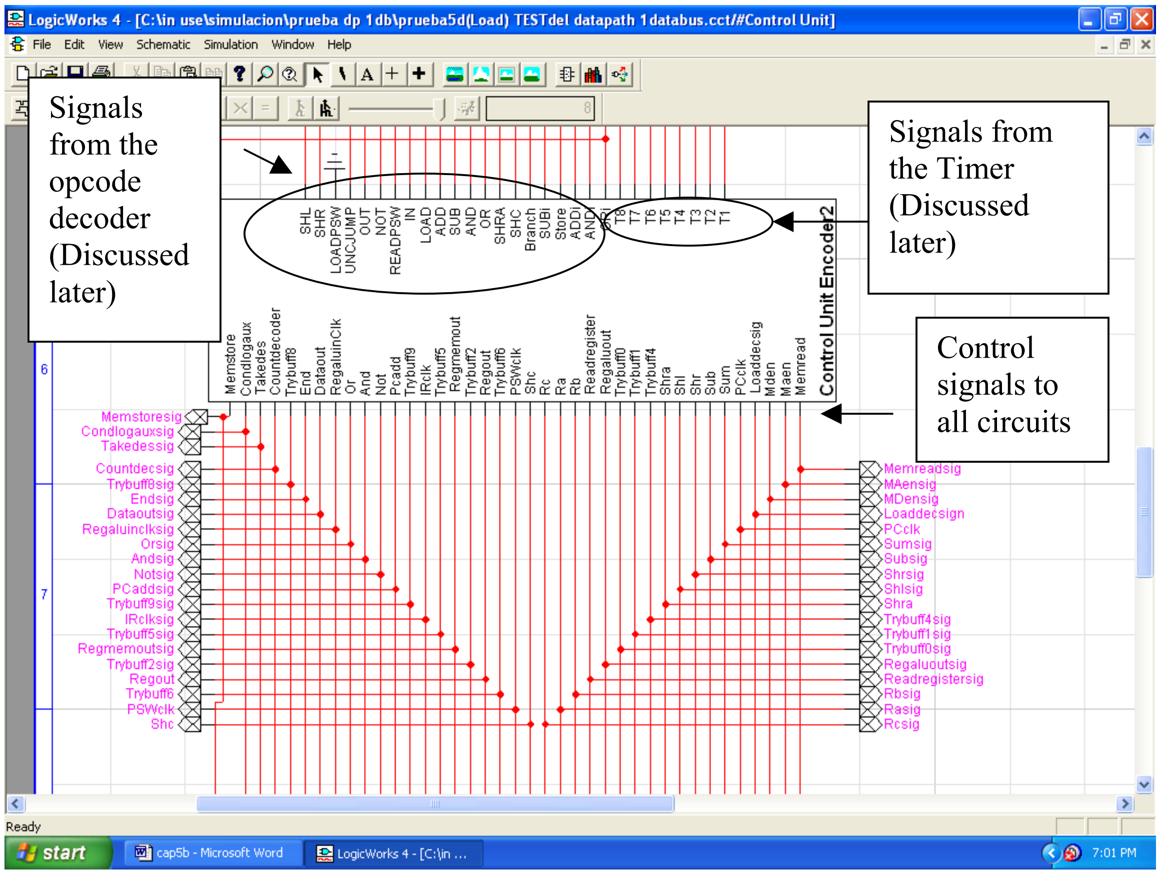

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit]
File Edit View Schematic Simulation Window Help
Signals from the opcode decoder (Discussed later)
Signals from the Timer (Discussed later)
Control signals to all circuits
Control Unit Encoder2
SHL SHR LOADPSW UNCJUMP OUT NOT READPSW IN LOAD ADD SUB AND OR SHRA SHC Branch SUBi Store ADDi ANDi
T8 T7 T6 T5 T4 T3 T2 T1
Memstore Condlogaux Takedes Countdecoder Trybuff8 End Dataout RegaluinClk Or And Not Pcadd Trybuff9 IRclk Trybuff5 Regmemout Trybuff2 Regout Trybuff6 PSWclk Shc Rc Ra Rb Readregister Regaluout Trybuff0 Trybuff1 Trybuff4 Shra Shl Shr Sub Sum PCclk Loaddecsig Mden Maen Memread
Memstoresig Condlogauxsig Takedessig Countdecsig Trybuff8sig Endsig Dataoutsig Regaluinclksig Orsig Andsig Notsig PCaddsig Trybuff9sig IRclksig Trybuff5sig Regmemoutsig Trybuff2sig Regout Trybuff6 PSWclk Shc
Memreadsig MAensig MDensig Loaddecsign PCclk Sumsig Subsig Shrsig Shlsig Shra Trybuff4sig Trybuff1sig Trybuff0sig Regaluoutsig Readregistersig Rbsig Rasig Rcsig
Ready
start
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
7:01 PM

Figure 6.54 Control Unit Encoder implementation

6.3 The control unit operational code decoder

The operational code decoder receives the first five bits of the Instruction Register. This unit decodes those five bits and generates one signal that corresponds to the instruction that will be executed. This signal goes to the Control Unit Encoder and together with the timer decide which signals will be activated. Table 6.34 shows the Opcode Decoder truth table. Figure 6.55 shows the Opcode Decoder implementation.

OPCODE DECODER						
	CODE					
POSITION	A	B	C	D	E	NAME
0	**0**	**0**	0	0	0	Ori
1	0	0	0	0	1	ANDi
2	0	0	0	1	0	ADDi
3	0	0	0	1	1	STORE
4	0	0	1	0	0	SUBi
5	0	0	1	0	1	BRANCH
6	0	0	1	1	0	SHC
7	0	0	1	1	1	SHRA
8	**0**	**1**	0	0	0	Or
9	0	1	0	0	1	AND
10	0	1	0	1	0	SUB
11	0	1	0	1	1	ADD
12	0	1	1	0	0	LOAD
13	0	1	1	0	1	IN
14	0	1	1	1	0	READ PSW
15	0	1	1	1	1	NOT
16	1	0	0	0	0	OUT
17	1	0	0	0	1	UNCJUMP
18	1	0	0	1	0	LOAD PSW
19	1	0	0	1	1	SHR
20	1	0	1	0	0	SHL
21	1	0	1	0	1	UNUSED
22	1	0	1	1	0	UNUSED
23	1	0	1	1	1	UNUSED
24	1	1	0	0	0	UNUSED
25	1	1	0	0	1	UNUSED
26	1	1	0	1	0	UNUSED

27	1	1	0	1	1	UNUSED
28	1	1	1	0	0	UNUSED
29	1	1	1	0	1	UNUSED
30	1	1	1	1	0	UNUSED
31	1	1	1	1	1	UNUSED

Table 6.34 The Opcode Decoder truth table

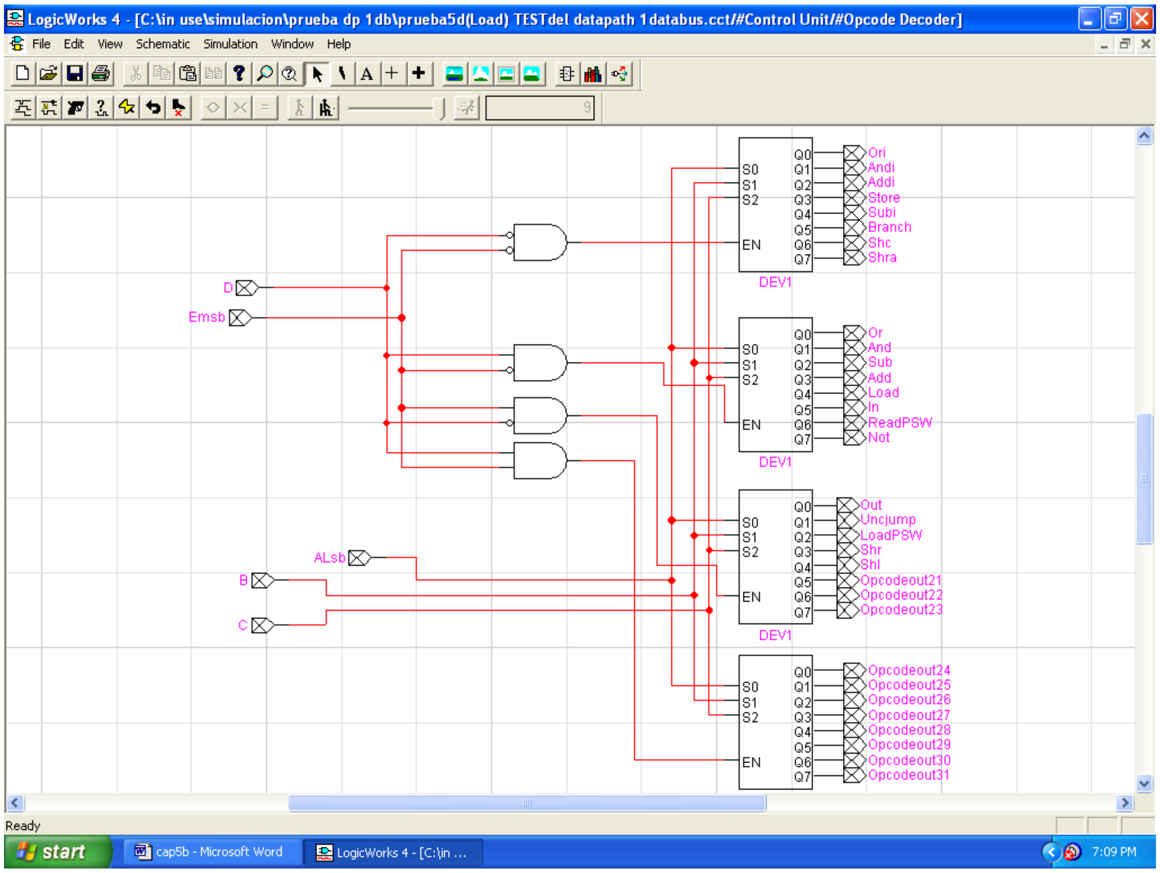


Figure 6.55 Operational Code Decoder circuit

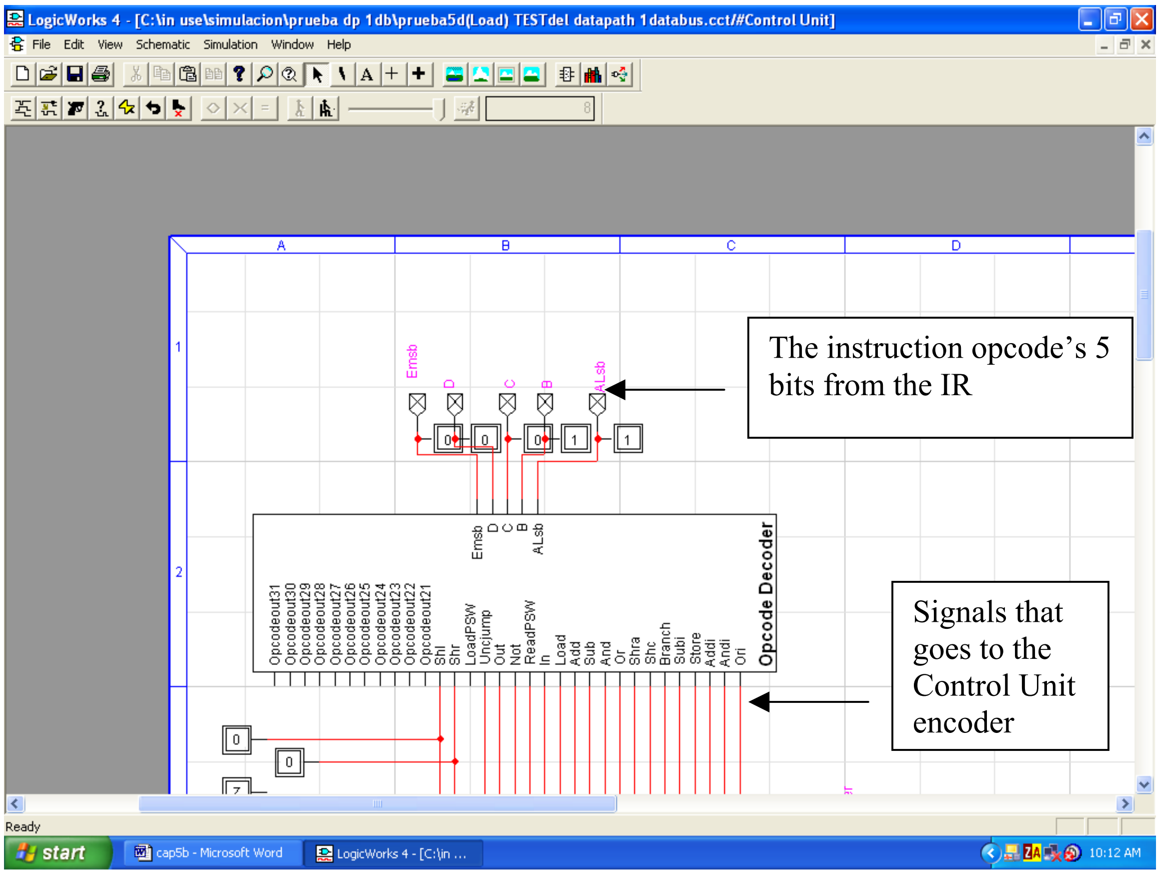

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit]
File Edit View Schematic Simulation Window Help
The instruction opcode's 5 bits from the IR
Signals that goes to the Control Unit encoder
Emsb
D
C
B
ALsb
Opcode Decoder
Opcodeout31
Opcodeout30
Opcodeout29
Opcodeout28
Opcodeout27
Opcodeout26
Opcodeout25
Opcodeout24
Opcodeout23
Opcodeout22
Opcodeout21
Shl
Shr
LoadPSW
Uncjump
Out
Not
ReadPSW
In
Load
Add
Sub
And
Or
Shra
Shc
Branch
Subi
Store
Addi
Andi
Ori
Ready
start
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
10:12 AM

Figure 6.56 Operational Code Decoder Implementation

6.4 THE CONTROL UNIT TIMER

The Control Unit timer is really one zero to seven counter. It was selected to seven because the largest number of clock cycles in the instruction set is 8. The timer specifies each instruction clock cycle. It works with the opcode decoder and sends its signal to the Control Unit Encoder as shown in figure 5.114.

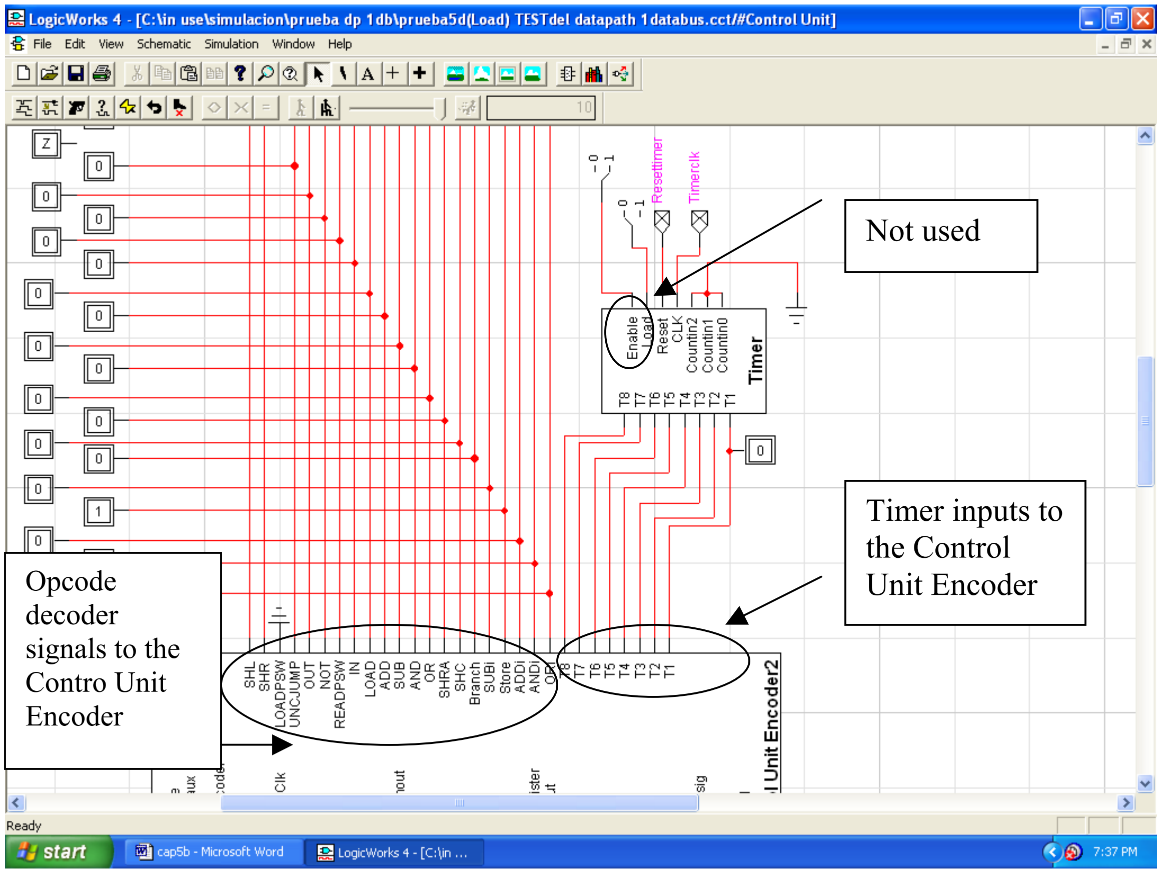

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit]
File Edit View Schematic Simulation Window Help
Resettimer
Timerclk
Not used
Enable
Load
Reset
CLK
Countin2
Countin1
Countin0
Timer
T8 T7 T6 T5 T4 T3 T2 T1
Timer inputs to the Control Unit Encoder
Opcode decoder signals to the Contro Unit Encoder
SHL SHR LOADPSW UNCJUMP OUT NOT READPSW IN LOAD ADD SUB AND OR SHRA SHC Branch SUBi Store ADDi ANDi ORi
Unit Encoder2
Clk
Ready
start
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
7:37 PM

Figure 6.57 The Control Unit Timer

The enable and Load ports will not be used in this work. Reset makes the timer to start over again and count from zero. The clock will be used, as the main clock, and it will control the movement from one microcontroller state to the other. Count bits 0,1 and 2 are the bits that specify where the timer starts its count.

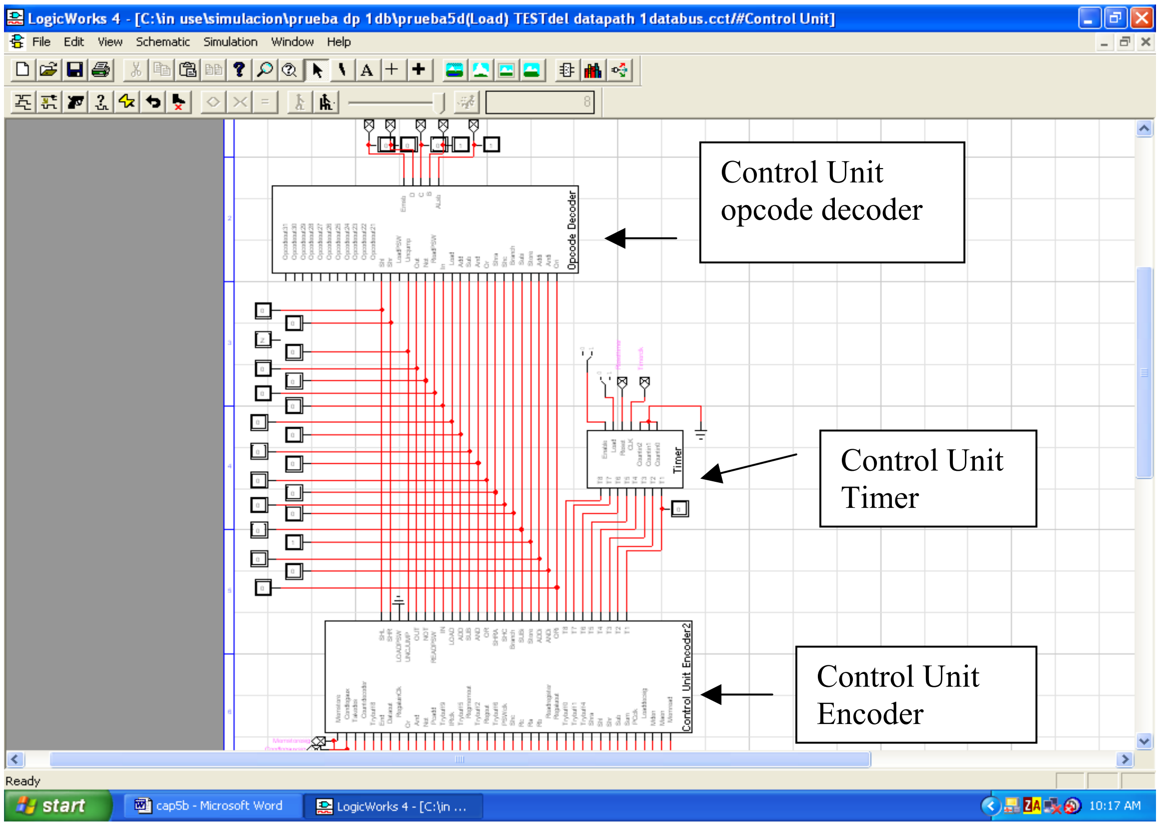

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct/#Control Unit]
File Edit View Schematic Simulation Window Help
Control Unit opcode decoder
Opcode Decoder
Control Unit Timer
Timer
Control Unit Encoder
Control Unit Encoder2
Ready
start
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
10:17 AM

Figure 6.58 Control Unit Circuit

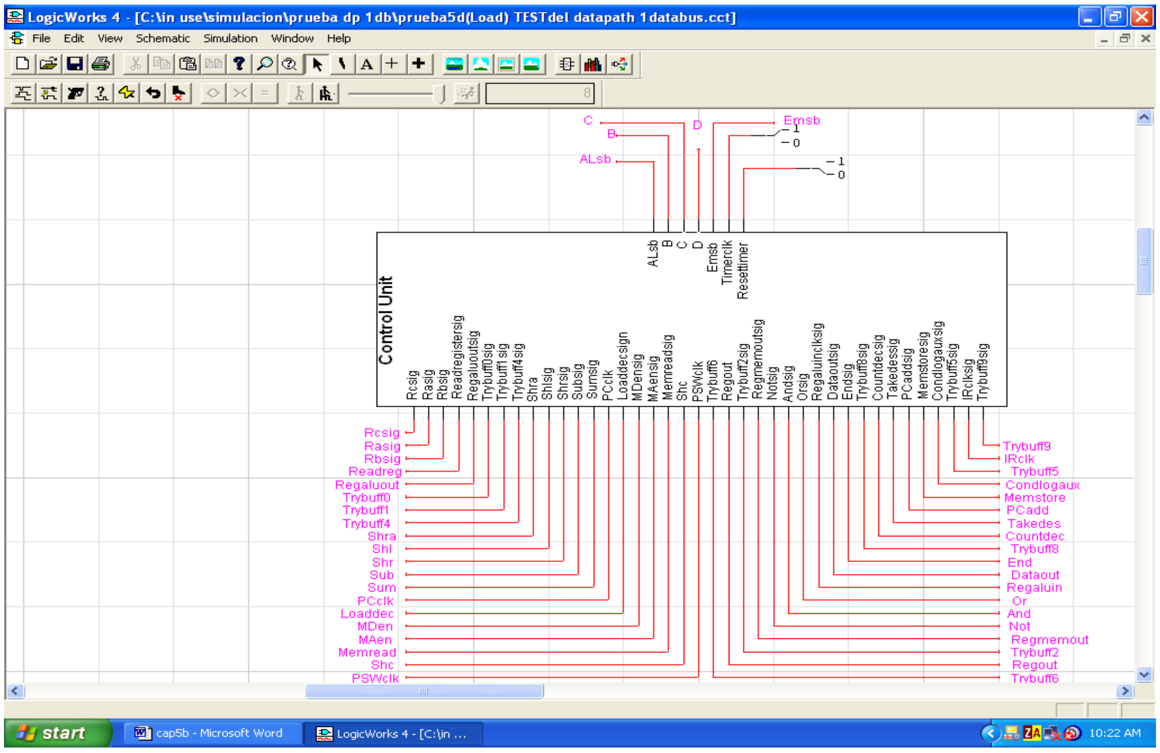

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct]
File Edit View Schematic Simulation Window Help
C
B
D
Emsb
ALsb
1
0
1
0
Control Unit
ALsb B C D Emsb Timerclk Resettimer
Rcsig Rasig Rbsig Readregistersig Regaluoutsig Trybuff0sig Trybuff1sig Trybuff4sig Shra Shlsig Shrsig Subsig Sumsig PCclk Loaddecsign MDensig MAensig Memreadsig Shc PSWclk Trybuff6 Regout Trybuff2sig Regmemoutsig Notsig Andsig Orsig Regaluinclksig Dataoutsig Endsig Trybuff8sig Countdecsig Takedessig PCaddsig Memstoresig Condlogauxsig Trybuff5sig IRclksig Trybuff9sig
Rcsig
Rasig
Rbsig
Readreg
Regaluout
Trybuff0
Trybuff1
Trybuff4
Shra
Shl
Shr
Sub
Sum
PCclk
Loaddec
MDen
MAen
Memread
Shc
PSWclk
Trybuff9
IRclk
Trybuff5
Condlogaux
Memstore
PCadd
Takedes
Countdec
Trybuff8
End
Dataout
Regaluin
Or
And
Not
Regmemout
Trybuff2
Regout
Trybuff6
start
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
10:22 AM

Figure 6.59 Control Unit Implementation

6.5 Implementation Problems

Once the Control Unit is finished and ready for implementation, designers should realize that during its implementation some problems arise. The following is a small list of problems and important points to keep in mind at the Control Unit implementation stage.

1) Due to the many existing control lines, designers must ensure that every signal that goes from the control unit is properly connected to its corresponding circuit. In this report, Logic Works offers one feature that allows connections just giving the signal source and its destination the same name. If designers use this feature, they must ensure that both signal ends have exactly the same name. If not the software does not recognize the signal and the hardware will not work properly, as a consequence, circuits that depend on the circuit data and an entire operation can be affected.
2) Care should be taken at the interconnection stage because involuntary disconnections may happen.
3) More than one signal is activated per clock cycle, this means that some circuits have to wait for data because probably it is not ready for processing at the circuit signal activation moment. To solve this problem, once the control unit is connected to all circuits, designers have to run manually with the control unit clock, each and every one of the microcontroller instructions to see per clock its performance.
4) Once a time delay problem has found (you will know that this problem happen because in its respective instruction clock cycle, when you run it manually, there is not data in some circuits that is supposed to be. This means that a time delay must be added to the circuit element that does not receive the data. Figure 6.60 illustrates two inverters with added time delay (in nanoseconds) necessary at the ALU port A and out put ports to function properly.

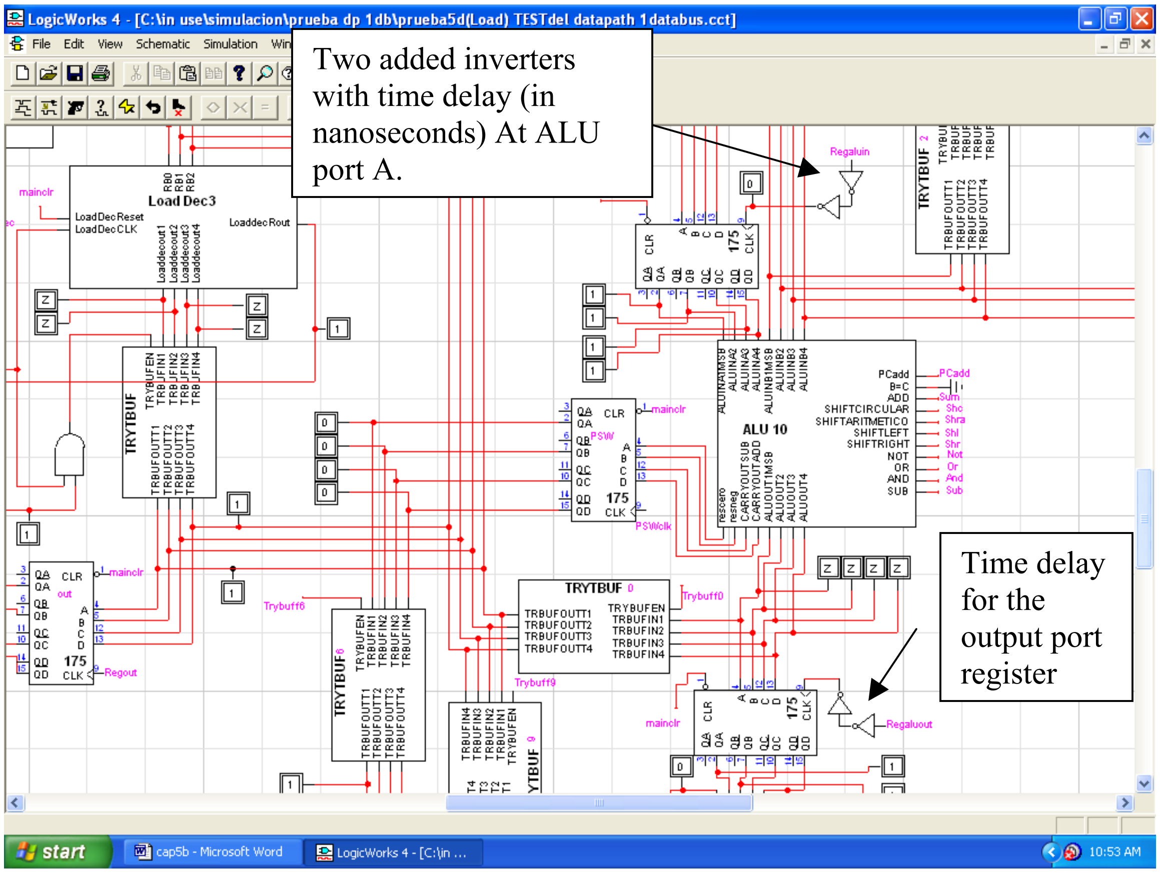

LogicWorks 4 - [C:\in use\simulacion\prueba dp 1db\prueba5d(Load) TESTdel datapath 1databus.cct]
File
Edit
View
Schematic
Simulation
Two added inverters with time delay (in nanoseconds) At ALU port A.
Time delay for the output port register
Load Dec3
ALU 10
TRYTBUF
Regaluin
Regaluout
Regout
mainclr
PSWclk
Trybuff0
Trybuff6
Trybuff9
start
cap5b - Microsoft Word
LogicWorks 4 - [C:\in ...
10:53 AM

Figure 6.60 Delay for signals

Chapter 7. Detailed Description of the Instruction Set

Next is a detailed description of each instruction that can be executed with the microcontroller simplest data path. Details like the instruction format; clock cycle number and task by clock cycle are discussed. The fetch process is discussed later at the Control Unit design stage, now it is just only explained as part of the instruction execution process.

7.1 ADDITION

Importance and justification

All microcontrollers and microprocessors must be able to perform mathematical computations in order to execute its own instructions and be useful. The operation of addition is one of the most important and basic mathematical computations.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01011					Ra			Rb			Rc			XXXX			

Operation: A ←(B +C)

Flags Affected: PSW[4]

ADDITION		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File signal activates the address **bits of Rb** to locate the register specified by Rb. Also the Register File **data out** signal is activated to release the data specified by Rb. The **ALU port A register read signal is activated** to store this data.
4	3	The Register File signal **Rc** is activated to locate the register specified by Rc. Also the Register File **data out signal** is activated to release the data specified by Rc to the ALU port B. **The ALU addition signal is activated** to perform the operation. **The read signal of the ALU output port register is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**.** The Register File signal for Ra is activated to locate its specified register. **The Register File read signal is activated to store** the processed result in the data bus.

Table 7.1 Add instruction signal activation verbal descriptions by cycle

7.2 BIT WISE AND

Importance and Justification

Compares two 4 bits numbers (first bit of first number with first bit of second number and so on) and send a high signal when a compared pare of bits have both bits in high (1), and send a low (0) when at least one or both compared bits are low. Performs a useful logic task to compare two binary numbers and to take decisions.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01001					Ra			Rb			Rc			XXXX			

Operation: A ←(B • C)

AND		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File signal that activates the address **bits Rb** is activated to locate the register specified by Rb. Also the Register File **data out** signal is activated to release the data specified by Rb. The **read signal** of the register at the **ALU port A is activated** to store this data.
4	3	The Register File signal that activates the **bits Rc** is activated to locate the register specified by Rc. Also the Register File **data out signal** is activated to release the data specified by Rc to the ALU port B. **The ALU AND signal is activated** to perform the operation. **The read signal of the ALU output port register is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release the processed result to the data bus**.** The Register File signal for **Ra is activated** to locate its specified register. **The Register File read signal is activated to** read the processed result in the data bus and store it in the specified register.

Table 7.2 AND instruction signal activation verbal description

The fetch activation signals are not shown because those signals will be defined at the Control Unit stage.

7.3 ARITHMETIC SHIFT RIGTH

Importance and justification

Sometimes programmers must accomplish certain tasks and manipulate data in certain ways to accomplish specific tasks. Arithmetic Shift Right instruction is very useful because it allows the programmer to take one binary number and shift its leftmost bit one or several places to the right. The vacant places are filled with bits equal to the binary number leftmost bit. It can be used and combined with other instructions to make the microcontroller programming easier.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00111					Ra			Rb			Rc			count			

Operation: (Bi)(Bi+n)(B j+n)(B k+n) ← BiBjBkBl

Where n is the number of shift places and i, j ,k ,and l are the respective bits position 1 , 2, 3 and 4.

THE ARITHMETIC SHIFT RIGHT		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File address bits **Rb** are activated to locate the register specified by Rb. Also the Register File **data out signal** is activated to release the data in the address location specified by Rb. The read signal of **the register at the ALU port A is activated** to store this data.
4	3	The Register File address bits **Rc** are activated to locate the register specified by Rc and **the Count Decoder signal is activated**. The count decoder makes its logical decision and **the ALU Arithmetic Shift Right signal is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**. The Register File address bits Ra are activated** to locate the register specified by Ra. **The Register File read signal is activated** to read the processed result in its input port and store it in the specified register.

Table 7.3 Arithmetic Shift Right instruction signal activation verbal descriptions

7.4 BRANCH – JUMP IF CONDITION

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00101					Ra (unused)			Rb			Rc			condition			

Operation: IF CONDITION IS TRUE: PC ← (ADDRESS)

BRANCH		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	Register File **Rc** bits are activated to locate the register specified by them. Also the Register File **data out signal** is activated to release the data specified by Rc to the data bus. **The take decision signal of the conditional logic** is a**ctivated** to take the logic decision to jump or not.
4	3	The Register File **bits of Rb** are activated to locate the register specified by them. Also the Register File **data out signal** is activated to release the data specified by Rb to the data bus. **The signal port of AUX circuit is activated** to release its logic decision to the PC.
5	4	The next fetch process begins.

Table 7.4 Branch signal activation verbal descriptions

7.5 BRANCH – UNCONDITIONAL JUMP

UNCONDITIONAL JUMP

In this set it is included one additional kind of jump, the unconditional jump.

Importance and justification

Unconditional jump allows programmers to execute non-continuous programming code in memory. The main difference between branch and the unconditional jump is that the latter does not have to be tested or has to take any decision, just jump to other memory location and execute its code.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10001					Ra (unused)			Rb (unused)			Rc (unused)			Immediate value			

Operation: PC ← (LAST 4 BITS)

Signal activation table for the instruction by cycle

UNCONDITIONAL JUMP		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The instruction **second task** consists on **activating the tristate buffer 2 and the PC clock** to load it with the new value for jump.

Table 7.5 Unconditional Jump signal activation verbal descriptions

7.6 CIRCULAR SHIFT

Importance and justification

Sometimes programmers must accomplish certain tasks and manipulate data in certain ways to accomplish specific tasks. Circular Shift instruction is very useful because it allows the programmer to take one binary number and shift its leftmost bit one or several places to the right. The vacant places are filled with bits equal to the rightmost bit of the binary number. It can be used and combined with other instructions to make the microcontroller programming easier.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00110					Ra			Rb			Rc			count			

Operation: (Bl)(Bi+n)(B j+n)(B k+n) $\leftarrow$ BiBjBkBl

Where n is the number of shift places and i, j ,k ,and l are the respective bits position 1 , 2, 3 and 4.

THE CIRCULAR SHIFT		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File address bits **Rb** are activated to locate the register specified by Rb. Also the **data out signal** of the Register File is activated to release the data in the address location specified by Rb. The read signal of **the register at the ALU port A is activated** to store this data.
4	3	The Register File address bits **Rc** are activated to locate the register specified by Rc and **The Count Decoder signal is activated**. The count decoder makes its logical decision and **the ALU Circular Shift signal is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**. The Register File address bits Ra are activated** to locate the register specified by Ra. **The reading signal of the Register File is activated to** read the processed result in its input port and store it in the specified register.

Table 7.6 Circular Shift instruction signal activation verbal descriptions

7.7 IN

Importance and justification

This instruction is used to obtain data from the outside. The data arrives into the microcontroller data path and is stored in a Register File location.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01101					Ra			Rb (unused)			Rc (unused)			XXXX			

Operation: ADDRESS ← DATA
Where address means one Register File location

Signal activation table for the instruction by cycle

IN		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	**The second** instruction task consists on **activating the tristate buffer 8** (see figure 5.55) **and the Register File Ra and read signals.**

Table 7.7 IN instruction signal activation verbal descriptions

7.8 IMMEDIATE ADDITION

Importance and justification

All microcontrollers and microprocessors must be able to perform mathematical computations. The operation of addition is one of the most important and basic mathematical computations .The immediate addition allows the programmer to specify in the instruction the second value that will be processed.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00010					Ra			Rb			Rc (unused)			Immediate operand			

Operation: A ←(B + last 4 bits)

Flags Affected: PSW [4]

IMMEDIATE ADDITION		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File address **bits Rb** are activated to locate the register specified by Rb. Also the Register File **data out signal** is activated to release the data specified by Rb to the data bus. The read signal of **the register at the ALU port A is activated** to store this data.
4	3	**The tristate buffer 2 holding the last four bits of the IR is activated** to allow those bits to pass to the data bus. **The ALU addition signal is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**. The Register File Ra bits are activated** to locate the register specified by Ra. **The read signal of the Register File is activated to** read the processed result in the data bus and store it in the specified register.

Table 7.8 Immediate Addition instruction signal activation verbal descriptions

7.9 IMMEDIATE AND

Importance and justification

Compares two 4 bits numbers (first bit of first number with first bit of second number and so on) and send a high signal when a compared pare of bits have both bits in high (1), and send a low (0) when at least one or both compared bits are low. The immediate and operation allows the programmer to specify in the instruction the second value that will be processed.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00001					Ra			Rb			Rc (unused)			Immediate operand			

Operation: A ←(B • last 4 bits)

INMEDIATE AND		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File address **bits Rb** are activated to locate the register specified by Rb. Also the Register File **data out signal** is activated to release the data specified by Rb to the data bus. The read signal of **the register at the ALU port A is activated** to store this data.
4	3	**The tristate buffer 2 holding the last four bits of the IR is activated** to allow those bits to pass to the data bus. **The ALU AND signal is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**. The Register File Ra bits are activated** to locate the register specified by Ra. **The read signal of the Register File is activated to** read the processed result in the data bus and store it in the specified register.

Table 7.9 Immediate AND instruction signal activation verbal descriptions

7.10 IMMEDIATE OR

Importance and justification

Compares two 4 bits numbers (first bit of first number with first bit of second number and so on) and send a high signal when a compared pare of bits have at least one high (1) bit is present, and send a low when both compared bits are low (0).
The immediate OR operation allows the programmer to specify in the instruction the second value that will be processed.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00000					Ra			Rb			Rc (unused)			Immediate operand			

Operation: A ←(B (+) with last 4 bits)

INMEDIATE OR		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File address **bits Rb** are activated to locate the register specified by Rb. Also the Register File **data out signal** is activated to release the data specified by Rb to the data bus. The read signal of **the register at the ALU port A is activated** to store this data.
4	3	**The tristate buffer 2 holding the last four bits of the IR is activated** to allow those bits to pass to the data bus. **The ALU OR signal is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**. The Register File Ra bits are activated** to locate the register specified by Ra. **The read signal of the Register File is activated to** read the processed result in the data bus and store it in the specified register.

Table 7.10 Immediate OR instruction signal activation verbal descriptions

7.11 IMMEDIATE SUBTRACTION

Importance and justification

All microcontrollers and microprocessors must be able to perform mathematical computations. The operation of subtraction is one of the most important and basic mathematical computations .The immediate subtraction allows the programmer to specify in the instruction the second value that will be processed. The immediate availability of this value is one of the reasons to include it in the instruction set.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00100					Ra			Rb			Rc (unused)			Immediate operand			

Operation: A ←(B - last 4 bits)

Flags Affected: PSW [3]

INMEDIATE SUBTRACTION		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File address **bits Rb** are activated to locate the register specified by Rb. Also the Register File **data out signal** is activated to release the data specified by Rb to the data bus. The read signal of **the register at the ALU port A is activated** to store this data.
4	3	**The tristate buffer 2 holding the last four bits of the IR is activated** to allow those bits to pass to the data bus. **The ALU subtraction signal is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**. The Register File Ra bits are activated** to locate the register specified by Ra. **The read signal of the Register File is activated to read** the processed result in the data bus and store it in the specified register.

Table 7.11 Immediate Subtraction instruction signal activation verbal descriptions

7.12 LOAD

Importance and justification

Sometimes the programmer needs to load values from memory and then transfer the information to the Register File to store them for further processing. Once the Register File has information in it, the programmer can perform operations with those values. The load instruction is essential because without it will be impossible to load data from memory to process it.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01100					Ra			Rb			Rc			Mc			

Operation: $A \leftarrow M$

Where M is data in memory and A represent a Register File location.

Signal activation table for the instruction by cycle

LOAD		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	Register File **bits of Rb** are activated to locate the register specified by Rb. Also the **Load Decoder data out signal** is activated to release its decision. The read signal of **the register at the ALU port A is activated** to store the data.
4	3	The tristate buffer 2 signal (holding the Mc four bits) is activated in order to allow those bits to pass to the data bus. **The addition signal of ALU is activated** to calculate the memory location. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	The data in the register at the ALU output port is stored in MA, **activating the tristate buffer 1 signal and the Memory Address register read signal.**
6	5	**The memory chip read signal is activated** to read the address specified by MA. **The read signal in the register at the memory chip output port is activated to store the data.**
7	6	**The tristate buffer 3 signal is activated to release data from memory to the data bus. Bits of Ra** in the Register File are activated to locate the register specified by Ra. **The Register File read signal** is activated to read and store the data from memory.

Table 7.12 Load instruction signal activation verbal descriptions

7.13 NOT

Importance and justification

Sometimes the programmer needs to change the sign of the bits in use in order to make calculations or to perform operations to address some registers, etc. In those cases is very useful to have an instruction that makes that happen and that is the reason to include this operation in the instruction set.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01111					Ra			Rb			Rc (unused)			XXXX			

Operation: -(B) $\leftarrow$ B

NOT		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File **Bits of Rb** are activated to locate the register specified by Rb. Also the **data out signal** of the Register File is activated to release the data specified by Rb. **The not signal of ALU is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
4	3	**The tristate buffer 1 is activated** to release its data to the data bus**. Register File Ra signal is activated** to locate the register specified by Ra. **The read signal of the Register File is activated** to read the processed result in the data bus and store it in the specified register.

Table 7.13 NOT instruction signal activation verbal description

7.14 OR

Importance and justification

Compares two 4 bits numbers (first bit of first number with first bit of second number and so on) and send a high signal when a compared pare of bits have at least one high (1) bit is present, and send a low (0) when both compared bits are low. Perform a useful logic task to compare two binary numberss and to take decisions.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01000					Ra			Rb			Rc			XXXX			

Operation: A $\leftarrow$(B (+) C)

OR		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File signal that activates the address **bits of Rb** is activated to locate the register specified by Rb. Also the Register File **data out** signal is activated to release the data specified by Rb. The **read signal** of the register at the **ALU port A is activated** to store this data.
4	3	The Register File signal that activates the **bits of Rc** is activated so that the Register File locates the register specified by Rc. Also the Register File **data out signal** is activated to release the data specified by Rc to the ALU port B. **The ALU OR signal is activated** to perform the operation. **The read signal of the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**.** The Register File signal for **Ra** is activated to locate the register specified by Ra. **The Register File read signal is activated to** read the processed result in the data bus and store it in the specified register.

Table7.14 OR instruction signal activation verbal description

The fetch activation signals are not shown because those signals will be defined at the Control Unit stage.

7.15 OUT

Importance and justification

This instruction is used to release data from the microcontroller to the outside world. The processed data could be used for device control or just to deliver information.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10000					Ra			Rb (unused)			Rc (unused)			XXXX			

Operation: PORT ← ADDRESS

Where Port is the register where the data from Register File will be transferred. Address is the Register File address location where the data is.

Signal activation table for the instruction by cycle

OUT		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	**The instruction second task** consists on **activating the Register File Rb and data out signals to release the data.** (see figure 5.56)

Table 7.15 Out instruction signal activation verbal descriptions

7.16 READ PSW

Importance and justification

The PSW is the register that holds the microcontroller flags. This instruction is used to read the PSW and obtain valuable information of computational flags. Inside, the ALU flags are activated if the computational result is zero, negative and overflow for add and subtract. Those flags are very important because programmers can take important decisions with them.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01110					Ra			Rb (unused)			Rc (unused)			XXXX			

Operation: ADDRESS ← PSW
Where address is the Register File address location to store the PSW.

Signal activation table for the instruction by cycle

READ PSW		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	**The second instruction task** consists on **activating the PSW clock, the tristate buffer 6, the Register File Ra and read signals** to store the PSW. See figure 5.57.

Table 7.16 Read PSW signal activation verbal descriptions

7.17 SHIFT LEFT

Importance and justification

Sometimes programmers must accomplish certain tasks and manipulate data in certain ways to accomplish specific tasks. Shift Left instruction is very useful because it allows the programmer to take one binary number and shift its rightmost bit one or several places to the left. The vacant places to the right are filled with zeroes. It can be used and combined with other instructions to make the microcontroller programming easier.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10100					Ra			Rb			Rc			count			

Operation: (Bj-n)(Bk-n) (B1-n) (0) ← BiBjBkBl

Where n is the number of shift places and i, j ,k ,and l are the respective bits position 1 , 2, 3 and 4.

THE SHIFT LEFT		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File address bits **Rb** are activated to locate the register specified by Rb. Also the Register File **data out signal** is activated to release the data in the address location specified by Rb. The read signal of **the register at the ALU port A is activated** to store this data.
4	3	The Register File address bits **Rc** are activated to locate the register specified by Rc and **The count decoder signal is activated**. The count decoder makes its logical decision and **the ALU Shift Left signal is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release the data to the data bus**. The Register File address bits Ra are activated** to locate the register specified by Ra. **The reading signal of the Register File is activated to** read the processed result in its input port and store it in the specified register.

Table 7.17 Shift Left instruction signal activation verbal descriptions

Importance and justification

Sometimes programmers must accomplish certain tasks and manipulate data in certain ways to accomplish specific tasks. Shift Right instruction is very useful because it allows the programmer to take one binary number and shift its leftmost bit one or several places to the right. The vacant places are filled with zeroes. It can be used and combined with other instructions to make the microcontroller programming easier.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10011					Ra			Rb			Rc			count			

Operation: (0)(Bi+n)(B j+n)(B k+n) ← BiBjBkBl

Where n is the number of shift places and i, j ,k ,and l are the respective bits position 1 , 2, 3 and 4.

SHIFT RIGHT		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File address bits **Rb** are activated to locate the register specified by Rb. Also the **data out signal** of the Register File is activated to release the data in the address location specified by Rb. The read signal of **the register at the ALU port A is activated** to store this data.
4	3	The Register File address bits **Rc** are activated to locate the register specified by Rc and **The Count Decoder signal is activated**. The Count Decoder makes its logical decision and **the ALU Shift Right signal is activated** to perform the operation. The read signal of **the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**. The Register File address bits Ra are activated** to locate the register specified by Ra. **The Register File read signal is activated to** read the processed result and store it in the specified register.

Table 7.18 Shift Right instruction signal activation verbal descriptions

7.19 STORE

Instruction Name

STORE

Importance and justification

Sometimes the programmer needs to process data and store it in memory. The store instruction is essential because without it will be impossible to store data in memory after the data is processed.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00011					Ra			Rb			Rc			Mc			

Operation: $M \leftarrow A$

Where M is data in memory and A represent a Register File location.

STORE		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File **bits Rb** are activated to locate the register specified by Rb. Also the **Load Decoder data out signal** is activated to release its decision and also is activated its tristate buffer that holds the 0000. The **read signal of the register at the ALU port A is activated** to hold 000 from the load decoder or the data of Rb.
4	3	The signal of the tristate that holds the Mc four bits is activated in order to pass those bits to the data bus. **The addition signal of ALU is activated** to perform the operation. The read signal of **the register at the ALU output port is activated to store the result.**
5	4	The tristate buffer 1 is activated to release the data in the ALU output port and is stored in **the Memory Address register activating its read signal.**
6	5	The Register File **bits Ra** are activated to locate the register specified by Ra. **The Register File data out signal** is activated to release its data to the data bus. The register at the ALU port A is activated to store the value from the Register File. The tristate buffer 9 is activated to deliver the data from the Register File output port to MA. The MD read signal **is activated** to store the data specified by Ra. **The memory chip read signal is activated** to read the address specified by MA and store the data in MD.

Table 7.19 Store instruction signal activation verbal descriptions

7.20 SUBTRACT

Importance and justification

All microcontrollers and microprocessors must be able to perform mathematical computations in order to execute its own instructions and be useful to the user. The operation of subtraction is one of the most important and basic mathematical computations.

Instruction Format

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01010					Ra			Rb			Rc			XXXX			

Operation: A ←(B - C)

Flags Affected: PSW [3]

SUBTRACTION		
Cycle	Task	SIGNAL ACTIVATION
1	Fetch	
2	Fetch	
3	2	The Register File signal that activates the address **bits of Rb** is activated to locate the register specified by Rb. Also the Register File **data out** signal is activated to release the data specified by Rb. The **read signal** of the register at the **ALU port A is activated** to store this data.
4	3	The Register File signal that activates the **bits of Rc** is activated to locate the register specified by Rc. Also the Register File **data out signal** is activated to release the data specified by Rc to the ALU port B. **The ALU subtraction signal is activated** to perform the operation. **The read signal of the register at the ALU output port is activated** to store the result.
5	4	**The tristate buffer 1 at the ALU output port is activated** to release its data to the data bus**.** The Register File signal that activates **Ra** is activated to locate its specified register. **The Register File read signal is activated** to read the processed result in the data bus and store it in the specified register.

Table 7.20 Subtraction instruction signal activation verbal descriptions

The fetch activation signals are not shown because those signals will be defined at the Control Unit stage.

Chapter 8. Conclusions

One of this project's goals was to provide the reader the opportunity to see how all the basic circuit, digital logic, basic electronic and advanced digital design concepts are applied in order to produce one functional system: a microcontroller.

This work also provides the student the opportunity to develop and practice some of the fundamental microcontroller design skills like planning, organization and testing the microcontroller hardware. The user is encouraged to use the techniques in chapter four and five to develop a microcontroller data path. Decisions like the number of data path circuit elements, their interconnection to save clock cycles and each element design are some of the skills worked in those chapters.

This methodology will guide users' actions and design tasks, to think about the available resources, reliability, time and design cost to achieve the final product. Considering that sometimes students become confused when trying to develop new skills and that the microcontroller world is not easy to understand at first, the design was done using Logic Works. Students will not face the situation of dealing with complex algorithms and symbols when they are introduced to microcontrollers. This work thus tried to be graphically understandable showing the design, implementation and testing of each microcontroller part and operation.

This work also provides the reader a ten-step mechanism that will guide the microcontroller design. One of the most important characteristics of this method is that it is modular. All circuits design were done as independently as possible. The advantage is that new designs can be tested with small changes to the original one. For example the Arithmetic Logic Unit like other blocks was developed with parallel circuits. This allows users to "plug and play" their new circuits without making significant design changes.

Also, modules can be designed and stored for fast implementation in future designs, and this can accelerate new designs or projects using the already existing circuits. The method developed in this work was used for a four-bit microcontroller, but it can be used for bigger ones. Although the circuit will be more complex, all design steps still apply together with all its recommendations. This work should be useful for beginners in the microcontroller design and operation field or as a microcontroller class complement or laboratory.

It is expected that with this approach students will feel more confident with different microcontroller designs. All simulations in this work were done with Logic Works 4.0. But the economy of this method is paid by designer's ability to select the proper interconnection and hardware to execute the instructions in the fastest way using the minimum amount of clock cycles per instruction. Designers must ensure to orchestrate all microcontroller signals activation in such a way that no conflict between signal activation exist during each instruction execution clock cycle.

the method developed in this work was used for a four-bit microcontroller, but it can be used for bigger ones. Although the circuit will be more complex, all design steps still apply together with all its recommendations. This work should be useful for beginners in the microcontroller design and operation field or as a microcontroller class complement or laboratory.

A physical implementation of the microcontroller can be done using FPGAS's. This requires a VHDL code, which can be partially generated by modern CAD software. This is left for future work.

References

[1] Gene, H., 1999, Microcomputer Engineering, Prentice Hall, New Jersey, NJ.

[2] Douglas, J., 1996,HDL Chip Design, Doone Publications, Madison, AL,USA, pp.3-4.

[3] Escuela Politéctinica Superior de Alcoy

http://server-die.alc.upv.es/asignaturas/LSED/2002-03/micros/downloads/trabajo.pdf

[4] http://www.pbs.org/nerds/timeline

[5] http://www.fms.komkon.org/comp/misc/ancient.txt

[6] http://www.geocites.com/micros_van/cap54.html

[7] Payne, Sandra, Dictionary of Computing, Oxford University Press 1996.

[8] IBM RESEARCH

http://www.research.ibm.com.

[9] INTEL

http://www.intel.com.

[10] Fletcher, William, 1980, An Engineer Approach to digital Design, Prentice Hall.

[11] Floyd, Thomas, 1997, Digital Fundamentals 6th edition, Prentice Hall.

[12] Preparata Franco P., 1985, Introduction To Computer Engineering, Harper & Row, Publisher Inc.

[13] Palmer, James,1993, Perlman David, Introduction To Digital Systems, Mc Graw Hill.

[14] Wakerly John F, 2000, Digital Design Principles & Practices 3rd edition, Prentice Hall.

[15] Tokheim Roger L.,1994, Digital Principles, 3rd edition, Mc Graw Hill.

[16] Bartee Thomas C., 1984, Fundamentos De Computadores digitales, Quinta edicion, Mc Graw Hill.

[17] Carter Nicholas, 2002, Computer Architecture, Mc Graw Hill.

[18] Tokheim Roger L.,1991, Fundamentos De Los Microprocesadores, Mc Graw Hill.

[19] Wilson Graham, 2002, Embedded Systems & Computer Architecture, Newnes Publishing Company.

[20] Stallings William, 1993, Computer Organization and Architecture, 3rd edition, Macmillan Publishing Company.

[21] Patterson David A, Hennessy John, 1994, Computer Organization & Design, Morgan Kaufmann Publishers, Inc.

[22] Brey Barry B., 1997, The Intel Microprocessors 808X,80286,80386,80486,Pentium & Pentium Pro, Prentice Hall.

[23] VHDL Tutorial: Learn by Example

http://www.cs.ucr.edu/content/esd/labs/tutorial/