DESIGN AND SIMULATION OF A FINITE STATE MACHINE SUITABLE FOR MICRO-CODE EXECUTION

ELIJAH KITHUKU MITHANGA

I56/7706/2002

A thesis submitted in partial fulfillment of the requirements for the award of the degree of Master of Science in Physics (**Electronics and Instrumentation)** in the school of Pure and Applied sciences of Kenyatta University.

MAY, 2010

DECLARATION

This thesis is my original work and has not been presented for a degree in any other

University.

Elijah Kithuku Mithanga
Department of Physics
Kenyatta University.

Signature …….. Date ……..

This thesis has been submitted with our approval as university supervisors.

Prof. Elijah Mwangi
Department of Electrical and Electronic Engineering
University of Nairobi.

Signature …….. Date ……..

Prof. I. V. S. Rathore
Department of Physics
Kenyatta University.

Signature …….. Date ……..

DEDICATION

To my wife Maryanne and my two sons Cephas and Dan for their unconditional love and greater sacrifices they have made to me.

ACKNOWLEDGEMENT

I wish to sincerely express my gratitude to my supervisors Prof. Elijah Mwangi of the Department of Electrical and Electronic Engineering, University of Nairobi and Prof. I.V.S Rathore, Department of Physics, Kenyatta University, in guiding me through my research work. I am grateful to Denis Kombwa, an undergraduate Student University of Nairobi for the technical support offered especially on the Electronic Work bench software.

Acknowledgements are made to the entire technical staff of Department of Physics, Kenyatta University for the immense support over the entire period of study. I wish also to acknowledge the Kenya Civil Aviation Authority, the employer and many helpful discussions with my fellow colleagues at work and at Kenyatta University.

Finally I greatly appreciate the moral and material support received from a number of people. I sincerely thank my dear wife Maryann Kindiki for her understanding, support and encouragement through out the period of study. Above all, I would like to thank the Almighty God for giving me the strength and knowledge to achieve this goal.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS AND ACRONYMS

AC	Accumulator
ADD	Addition
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
CPU	Central Processing Unit
CAD	Computer Aided Design
C	The C programming language
DEC	Decrement
DRAM	Dynamic Random Access Memory
EWB	Electronic Work Bench
EPROM	Erasable Programmable Read Only Memory
FPGA	Field Programmable Gate Array
HDL	Hardware Descriptive Language
HLL	High Level Language
IC	Integrated Circuit
INC	Increment
IR	Instruction Register
LD	Load
LSI	Large Scale Integration
MAR	Memory Address Register
MOV	Move
MOS	Metal Oxide Semiconductor
MSI	Medium Scale Integration
Opcode	Operation Code
PC	Program Counter

P.D.	Propagation Delay
PIPO	Parallel in parallel out
PISO	Parallel in serial out
PLD	Programmable Logic Device
PROM	Programmable Read Only Memory
RAM	Random Access Memory
ROM	Read Only Memory
SRAM	Static Random Access Memory
SHR –	Shift Right
SIPO	Serial in parallel out
SISO	Serial in serial out
SSI	Small Scale Integration
STA	Store
TTL	Transistor Transistor Logic
VHDL	Very High Speed Integrated Circuit Hardware Descriptive Language
VLSI	Very Large Scale Integration
JNZ	Jump Non- Zero

ABSTRACT

Digital processors are built from very large scale integration circuits and the software to design them is only available to the manufacturers themselves. There is a need to design a central processing unit using large, medium and small scale integration circuits which can be used as a teaching aid in computer architecture and design. In this thesis a simulation model of a digital processor is designed using arithmetic logic unit, control unit and registers as building blocks with their interconnections. The presented digital processor is a 4-bit, 80 MHz incorporating sixteen instructions. The modules for the central processing unit were built from large, medium and small scale transistor transistor logic integrated circuits. The design and simulation is based on Electronic workbench software. The circuit is capable of modeling the functional and timing behavior of the digital processor to use instructions in the groups of data transfer, arithmetic, logical, bit manipulation, shift and program control. A state machine control unit was used for sequencing of the instructions. The clock pulses were supplied by a 555 timer instead of a quartz crystal in order to make it possible to vary the frequency to maximum by use of different values of capacitance and resistances. The results of frequency against different values of resistances were also plotted; this was carried out in order to calculate the optimum simulation time for different instructions. A set of four switches was used for selecting up of the opcodes and imputing data to the digital processor through the input port. Four probes were used to indicate the status of the registers, memory unit and the output port. In order to implement program control a conditional jump instruction is implemented when the state of the status register is logic 1. The mnemonic codes were also simulated and the results were found to be in agreement with theoretical values. With the sixteen opcodes, circuits for subtraction, addition and multiplication were simulated. The circuits yielded good data validity at the frequency of operation. Further work can be carried out in order to improve the processor to handle large amounts of data by using 8-bit and 16-bit registers. Five or six bit length opcodes could also increase the number of instructions to be executed from sixteen. Verilog hardware descriptive language and Very high speed hardware descriptive language can be used since they offer more tools for design.

CHAPTER ONE

INTRODUCTION

1.1 Background to the study

Microprocessors have been recognized as important tools by electronic and system designers. As more and more designers get familiarized with the capabilities of the microprocessor, the number of microprocessor-based applications will increase very rapidly. Microprocessors are the result of the semi-conductor industry's effort to accommodate an ever increasing number of transistors in a single integrated circuit so as to implement a given function with a smaller number of chips than any previous devices (Uffenbeck, 1985; Mano, 2005). The central processing unit is the major component of a microprocessor. Today's microprocessor technology has increased the speed of operation due to the very large scale integration of components in a single chip. The central processing unit (CPU) performs a variety of functions as dictated by the type of instructions that are incorporated in the computer in order to fetch and execute data and instructions. The functions of the CPU can be demonstrated by a simple processor built with medium scale integration and small scale integration TTL modules (Downton, 1994). The aim of this thesis is to simulate a simple CPU with the basic architecture of digital systems available in the modern data processors which can assist in the teaching of computer architecture course using available components and software resources.

1.2 Statement of the research problem/justification

Modern digital systems are large and complex since most integrated circuits contain many thousands or millions of transistors. Such systems are designed by decomposing them in to smaller manageable parts. Each sub-circuit representing the digital processor is simulated to test its output logic circuits for it helps a designer to build and simulate the devices and their interconnections to form a circuit. (Thomas and Philip, 1994). Digital processor's design

software is expensive and not available to users especially students and are only limited to institutions and circuit designers who can afford to buy them. There is also need for the development of a digital processor using medium and small scale integration circuits to be used as a teaching aid in the learning institutions. This project highlights the issues in design of a digital processor using medium scale integration and small scale integration to simulate mnemonic codes.

1.3 Objectives of the research project

1.3.1 Main objective

The main objective of this research project is to design and simulate a finite state machine suitable for micro-code execution to aid in teaching of computer architecture design.

1.3.2 Specific objectives

i) To design Central Processing Unit's internal registers, buses and determine the number of instructions to be performed by the CPU.
ii) To design a state machine and its associated instruction set.
iii) To design a control circuit and its associated signals to be used for the control in order to perform different functions.
iv) To write an assembly code to describe the finite state machine.
v) To simulate in order to synthesize the circuits from the code.

1.4 Rationale of the research project

For this study several softwares were considered but Electronic Workbench was selected because of the three primary reasons of documentation, simulation and synthesis. Although design is possible in C programming language, it currently does not work with all the tools needed in the design flow so it is unlikely to replace traditional design software

(Ravichandran, 2003). Examples of design software include Verilog hardware descriptive language (HDL), Very high speed HDL and Multisim. However Electronic work bench (EWB) has the following advantages:-

(i) Electronic Workbench provides design portability. Circuit specified in EWB can be implemented in different type of chips and with CAD(Computer Aided Design) tools provided by different companies, without having to change the EWB specification.

(ii) Both small and large logic circuit designs can effectively be represented in EWB

(iii) EWB indicates the real process duration of the circuit being simulated

(iv) It indicates the circuits operating temperature in real process.

(v) It has good convergence properties to errors.

Most of scientific experiments, electronic testing, fault finding and process control systems require stable, accurate and reliable controllers to generate their signals; this is offered via use of microprocessors. A processor offers greater flexibility since the bits in memories replace wired connections. This research demonstrates the design of a simple CPU using medium scale integration (MSI) and small scale integration (SSI) logic modules. It can be applied in enhancing the teaching of a computer architecture course.

1.5 Thesis organization

This research project is divided into six chapters, organized as follows: chapter 1 gives the general overview of the research, specific objectives of the research project and justification of the study. Chapter 2 is a literature review that describes the history, development, advantages, relevant research work carried out by different researchers, limitations of VLSI processors, and the improvement that would be made in terms of the applications. In chapter 3, the general description of various digital components used in the organization of digital processors is analyzed. This begins with a description of the different types of registers,

control unit and ALU's available. Different computer programming languages are also discussed. The design of the digital processor is presented in chapter 4. It gives a brief description of the research methodology adopted in carrying out the research. Hardware, software designs and construction of a digital processor are described in detail. For instruction set completeness instructions were included in all the groups namely: Data manipulation instructions, Data transfer instructions and program control instructions. The instructions implemented included six data transfer instructions: MOV (move), LD (load), STA (store), four arithmetic instructions: INC (increment), DC (decrement), ADD (addition), two shift instruction: SHR (shift right) and two program control instruction (HLT, JNZ). Each instruction is four bits in length and every mnemonic was assigned a code. State diagrams were used to give the sequence of operation for each instruction, with Karnaugh map and De-morgans theorem simplification techniques being used to optimize the instruction generation.

The control unit which consists of decoders, sequence counter and a number of control logic gates was designed to enable the sequence of operations that were required in the execution of instruction provided by the clock pulses. The results and discussions are presented in chapter 5. EWB was used to describe the digital processor for it provided a programming and simulation environment to test the digital designs at the gate level or behavioral level. It helped in exploring architectural alternatives through simulation and detection of design bottlenecks before detailed design began for it was used to describe the layout of the wires, resistors, integrated circuits like decoders, counters, memory and register. EWB software used two simulation techniques: Functional simulation to test the working of the circuitry and timing simulation in view of evaluating the propagation delays and reducing them to minimal. After verification of all the instructions, design circuits of addition, subtraction, multiplication and division were simulated. Subtraction was carried out in two's complement method which is used in digital processors. The simulation results yielded the expected theoretical results. Chapter 6 gives the summary of recommendations and suggestions for further work.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The evolution of microprocessors has been known to follow Moore's law when it comes to steadily increasing performance over the years. This law suggests that the complexity of an integrated circuit with respect to the maximum component cost doubles every eighteen months (Hodges and Jackson, 1993). This has been proven from evolution of computers in their first generation used as the drivers for calculator, which continued to increase in size, shape performance and power consumption over the years to form computers. Digital devices are widely used in many applications that include digital control systems, measuring and testing equipment. In the integrated circuit, they are used in applications requiring low level design integration (Ghausi, 1996; Mathur, 1994). Microprocessors came about due to the development of mathematics which led to the development of tools to help in computation. Blaise Pascal in seventeen century was credited with building the first calculating machine. This machine had a mechanical computation. It used punch cards to store the numbers and processing requirements (Downton, 1994; Levental, 1978).

Research work has been conducted on using microprocessors in control and processing by many researchers. Garreth AiResearch *(*1968*)* produced a microprocessor to compete with electrochemical systems then under development for the main flight control computer. The design used Metal Oxide Semiconductor-based chipset as the core CPU. The system contained twenty bit pipelined, parallel multi-processor (Garatt, 1995; Raghuram, 1989). The system was considered advanced and the U.S.A navy refused to allow publication of the design until 1997. Garreth AiResearch instead adopted a four bit microprocessor and stressed pre-programmed embedded applications in implementing a calculator on a chip. McDemott

(1983) developed two simple programs using flowchart diagrams to simulate digital logic circuits in Basic language and running on the TRS model 80 microcomputer. The first program introduced simulation of digital circuits as a computer aided design modeling and simulating higher level devices such as registers and counters.

Research work on digital processors has also been carried out by Edward Rogers in the Department of Electrical and Computer Engineering, University of Toronto (2000) in several design exercises. In one of their designs they implemented a simple 16-bit processor in the FLEX 10K FPGA. The processor was able to execute a limited set of instructions which was stored in the SRAM. It consisted of 256 instructions and 256 words of data. Four general purpose registers were utilized (R0, R1, R2 and R3), Arithmetic Logic Unit, Program counter and was able to perform the following operations Move data between registers, Load, Store, Add, Sub, and Halt (Breuer, 1997; Mano and Kime 1997; Malvino, 1980).

Qualey, (2006) implemented 4-bit CPU from TTL ICs. Used counters, latches, gates, EPROMs, and RAM. Fifteen Instructions were implemented including STORE, LOAD, INC, IN, OUT and JMP. Micro-programmed method was used to design the control unit. Prototypes of these circuits are first designed and tested before fabrication in printed circuit boards. The convectional methods for testing such digital circuits involve solving a Boolean equation of the circuit and use of the hardware breadboard. The latter involves implementing the design on a breadboard, applying test signals and observing the outputs (PC instruments datasheet, 1999 and Texas instruments, 1980). A Boolean equation of a digital circuit with many devices such as integrated circuit is difficult to solve and the hardware bread boarding technique is preferred. But, the experimental method of bread boarding digital designs is tedious and costly in terms of the hardware materials, time consuming and practically impossible for digital ICs.

It is also difficult in fault finding and trouble shooting diagnosis (Thomas and Philip, 1994; Zeigher, 1976). Thus there may be a propagation of errors in the circuit design. The use of a programming and simulation environment to test digital designs at the gate and behavioral level can detect design bottlenecks before a detailed design begins.

2.2 Microprocessor overview

A microprocessor is a Very Large Scale Integration (VLSI) component that implements most of the functions of the Central Processing Unit (CPU) on a single chip (Chand and Kuthe, 2004; Douglas, 1994; Eincheberger and Williams, 1992). Microprocessors has been recognized as important tools by electronic and system designers. Microprocessors are as a result of semi-conductor industry's effort to accommodate an ever increasing number of transistors in a single integrated circuit so as to implement a given function with a smaller number of chips than any previous devices. As markets get better defined, products initially developed with the microprocessor as the controller will be replaced by custom circuits implemented using logic circuits and reprogrammable logic arrays (Eccles, 1985; Hamacher *et al*., 2002). It is more advantageous to use FPGAs than microprocessor because they offer a good degree of flexibility and a wide range of functions for the user (Brown and Vranesic, 2003; Navtei, 2005; Taub and Schilling, 1997).

Development of Microprocessors started with the 4-bit microprocessors, with the technology advancing to the 8-bit, 16-bit and 32-bit devices. Each type of microprocessor has its exclusive market of application (Ahson,1998; Osborne, 1987). For example 8-bit microprocessors are used for embedded control applications. Microprocessors are used as controllers in motorcars, washing machines, and other appliances. They will have application through out industry and in many commercial undertakings (Milne and Fraser, 1990). Most popular 8-bit CPUs include the Z-80 and Intel 8085. The processors use 8-bit bytes which are

stored in memory. These bytes contain both the program that the processor is executing and the data items that the program is working on. The processor uses 16-bit addresses to access these bytes. A microprocessor is used in conjunction with a random access memory (RAM), a read only memory (ROM), and one or more input/output (I/O) circuits. These circuits, assembled on one or more boards, form a microcomputer with a microprocessor being the control processing unit. Recent microprocessors are also implemented using the Field-Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC) that can be re-programmed after it is manufactured rather than having its programming fixed during the manufacturing. FPGA is configured or designed after manufacture (field programmable). ASIC is customized for a particular use rather than intended for general purpose use, e.g. a chip designed for a cell phone. FPGAs are generally slower than ASIC and draw more power although FPGAs have a shorter development cycle and lower development costs (Sternheim *el al.,* 1993; Baher, 1986). The CPU consists of the following:

i) Registers are flip-flop circuits arranged in groups. They store immediate data used during the execution of instructions.

ii) Arithmetic Logic Unit (ALU) performs arithmetic and logical operations like Add, Sub and logical functions like NOT, OR.

iii) Control circuit produces control and synchronization signals. It performs instruction fetch, decode and execute.

Buses are used for communication between the different units of the microprocessor system. It is carried out by address and data buses along with various control lines. Modern digital processors are designed using very large scale integration and large scale integration on a single chip. In this research the design of a simple processor is realized using Small Scale Integration and Medium Scale Integration (Green, 1985; RS data library, 1999; Tzafestas, 1973).

CHAPTER THREE

MICROPROCESSOR OPERATION THEORY

3.1 Central processing unit

A digital data processor comprises of a set of registers, the Arithmetic Logic Unit and the digital functions that implement the arithmetic, logic shift and transfer micro-operations as indicated by the control unit. Instructions are implemented either with a single micro-operation or with a sequence of operations (Mano, 2005; Mittle, 2004). The hardware of the digital data processor consists of the CPU, memory, input and output devices as shown in figure 3.1. The central processing unit contains an arithmetic and logic unit for manipulating data, a number of registers for storing data and control circuits for fetching and executing instructions. The memory contains storage for instructions and data. The input and output processor contains electronic circuits for communicating and controlling the transfer of information between the digital processor and the outside world. There are two types of computer architectures namely:

i) Harvard computer architecture; it uses separate memories (storage) for instructions and data hence requiring different signal pathways for instruction and data (dedicated bus for each of them). This makes it possible to read/write data from/to memory at the same time. Harvard architecture computer can thus be faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.

ii) Von Neumann architecture; it has single storage to hold both instructions and data. Instructions have to be fetched in a sequential order. Reading/writing data from/to memory cannot occur at the same time because it uses the same data bus.

In both computer architectures, pipelining and cache memories are employed as ways of enhancing of computational speed. Pipelining is a technique of decomposing a sequential

process into sub-operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments. Cache memory is a special very high speed memory used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.

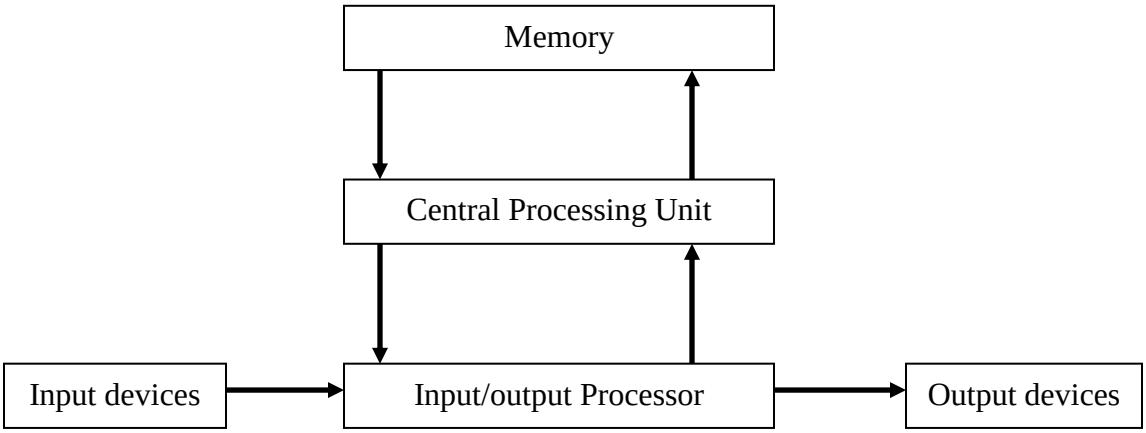


Figure 3.1: Major components of a CPU

3.2 Registers

A register is a group of flip-flops with each flip-flop capable of storing one bit information. An n-bit register has a group of n flip-flops and is capable of storing any binary information of n bits. In addition to flip-flops, a register may have combinational gates that perform certain data processing tasks. In digital processing there are general purpose registers and flags. classification of registers depends on how data is entered and retrieved. This leads to four possible modes of operation of either serial or parallel data entry or retrieval:

i) Serial in serial out (SISO).

ii) Serial in parallel out (SIPO).

iii) Parallel in parallel out (PIPO).

iv) Parallel in serial out (PISO).

Registers in which data is entered or /and taken out in serial form are referred to as shift registers. Bits are shifted in each flip-flop with the occurrence of clock pulses either right or left. Register configuration processing involves inter- register movements as dictated by the control unit. Register size define a processor in terms of processing power. Larger registers of 32 bits and 64 bits used in modern day processors allow faster processing by handling large bytes of data at any instant (Mathur, 1983; Mano and Kime, 1997). The next section discusses the different types of registers.

3.2.1 Program counter

The program counter holds the address of the next instruction to be read from memory and goes through a step by step counting sequence and causes the computer to read successive instructions previously stored in memory always incrementing by one from one instruction to the other (Douglas, 1994).

3.2.2 Accumulator register

The accumulator is an intermediate register between processing and input/output devices. The results of all mathematical operations performed by the Arithmetic Logic unit are left in this special register. Its name is derived from the arithmetic addition process encountered in digital computers.

3.2.3 Instruction register

Instruction register holds the operation code bits of the current instruction code fetched from memory.

3.2.4 Status register

Relative magnitudes of two numbers may be determined by logically subtracting one number from the other then checking certain bit conditions in the resultant difference. If the two numbers are unsigned, the bit conditions of interest are the output carry and the possible zero result. If the two numbers include a sign bit in the highest order position, the bit conditions of interest are the sign of the result, a zero indication and an overflow condition. It is sometimes convenient to supplement the arithmetic logic unit with a status register where the status bit conditions are stored for further analysis. Status bit conditions are also called condition-code bits or flags bits. The result of the analysis will be used mainly for flagging purposes in instruction execution (Osborne, 1987; Milne and Fraser, 1990).

3.2.5 Sequence register

It is a counter that produces the timing signals for the computer. It is decoded to supply timing variables for the control unit. The timing variables together with other variables produce the control functions that initiate all the micro operations for the computer.

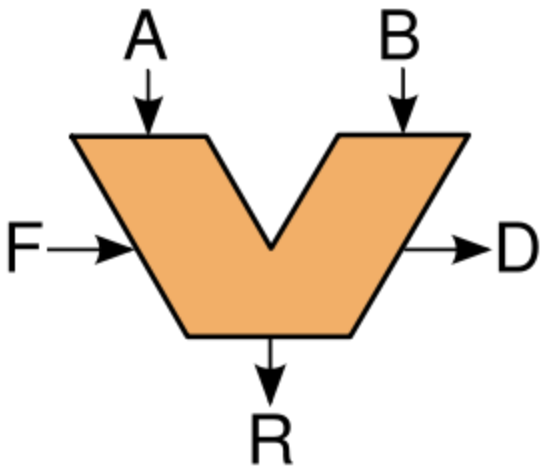
3.2.6 Input and output registers.

Input/output register receive data from an input/output device e.g. keyboard/monitor and is a means by which users can access utilities of the microprocessor.

3.3 Arithmetic logic unit

Arithmetic logic unit is that functional part of a digital computer that performs arithmetic and logical micro-operations for executing instructions i.e., carry out the arithmetic and logic operations. The ALU is a fundamental building block of the central processing unit of a computer, and even the simplest microprocessors contain one ALU for purposes such as maintaining timers. The processors found inside modern CPUs' have inside them very

powerful and very complex ALU. Figure 3.2 shows the schematic symbol of ALU with select lines for operands A and B. The function select input F from the control unit is used to distinguish between arithmetic and logic operations. R denotes the output signals. The input information received by the ALU consists of operands, operation code and format code. The operands are machine words that represent numeric and alphanumeric information (Hodges and Jackson, 1993; Peatman, 1997).



Keys:

A & B are operands

R is the output

F is the input from the Control Unit

D is an output status

Figure 3.2: A typical schematic symbol for an Arithmetic Logic Unit

A truth table of a typical ALU 74181 with four select inputs S_3, S_2, S_1, S_0 and its logic functions are summarized in table A.3. The basic component of the arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations. Arithmetic operations include addition, subtraction, multiplication, subtraction, increment, decrement and shift. Logical operations include AND, NAND, EX-OR. Table 3.1 shows the symbolic designation and description of arithmetic micro-operations.

Table 3.1: Arithmetic Micro-operations

Symbolic designation	Description
$R_3 \leftarrow R_1 + R_2$	Contents of R_1 plus R_2, output transferred to R_3
$R_3 \leftarrow R_1 - R_2$	Contents of R_1 minus R_2 output transferred to R_3
$R_2 \leftarrow \overline{R}_2$	Complement the contents of R_2 (1's complement)
$R_2 \leftarrow \overline{R}_2 + 1$	Two's complement the contents of R_2 (negate)
$R_3 \leftarrow R1 + \overline{R}_2 + 1$	R_1 plus 2's complement of R_2 (subtraction)
$R_1 \leftarrow R_1 + 1$	Increment the contents of R_1 by one
$R_1 \leftarrow R_1 - 1$	Decrement the contents of R1 by one

3.4 Control unit

This is the unit that controls the flow of information through the processor, and coordinates the activities of the other units within it. In a way, it is the "brain within the brain", as it controls what happens inside the processor, which in turn controls the rest of the computer i.e., component of the central processing unit that decodes, synchronizes, and executes program instructions. It does this by providing command signals for specifying the sequence of micro-operations in conjunction with the timing circuit. The functions performed by the control unit vary greatly by the internal architecture of the CPU, since the control unit really implements this architecture (Tzafestas, 1973; Malvino, 1980). The control unit performs the tasks of fetching, decoding, managing execution and then storing results. The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform. The whole CPU is synchronized by a clock signal as it goes through various states, each state doing something different. The control unit consists of sequence register, decoders, counters and control logic gates are as shown in figure 3.3. An instruction read from memory is placed in the Instruction Register. The operation code bits are decoded

with a 3 × 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. The 4-bit sequence counter, counts in binary from 0 through 15. The counter is incremented to provide the sequence of timing signals out of the 4 × 16 decoder. Once in a while the counter is cleared to 0, causing the next active timing signal to be T_0.

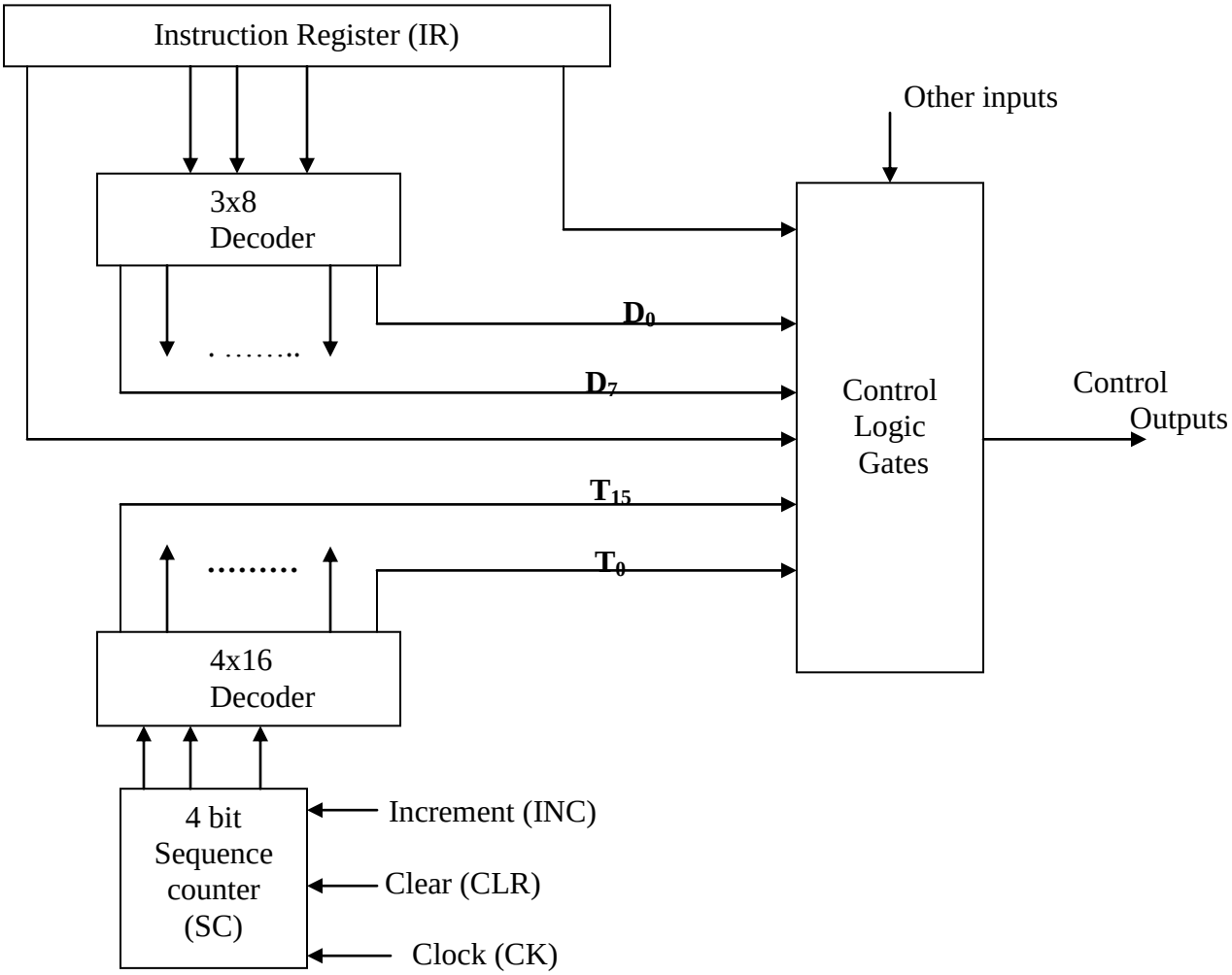


Figure 3.3: Control unit of basic computer

Control units are simple state machines that can be implemented either by Hardwired control or Micro-programmed control. In the hardwired control, the control logic is implemented with gates, flip-flops, decoders and other digital circuits. It has the advantage that it can be optimized to produce fast mode of operation, but, once implemented it becomes expensive to change. In the microprogrammed control, the control information is stored in the control

memory. The control memory is programmed to initiate the required sequence of micro-operations (Thomas and Philip, 1994; Sternheim *et al*, 1993). Hardwired control can be divided into four methods as discussed in the next sections.

3.4.1 One flip-flop per state method

Only one flip-flop is set at any particular time and all the others are cleared. A single bit is made to propagate from one flip-flop to the other under the control of design logic. In such an array, each flip-flop represents a state and is activated only when the control is transferred to it. The advantage of this method is the simplicity with which it can be designed as shown in figure 3.4. This type of controller can be designed by inspection from the state diagram that describes the control sequence. The outputs T_0, T_1, T_2, and T3 are the timing signals.

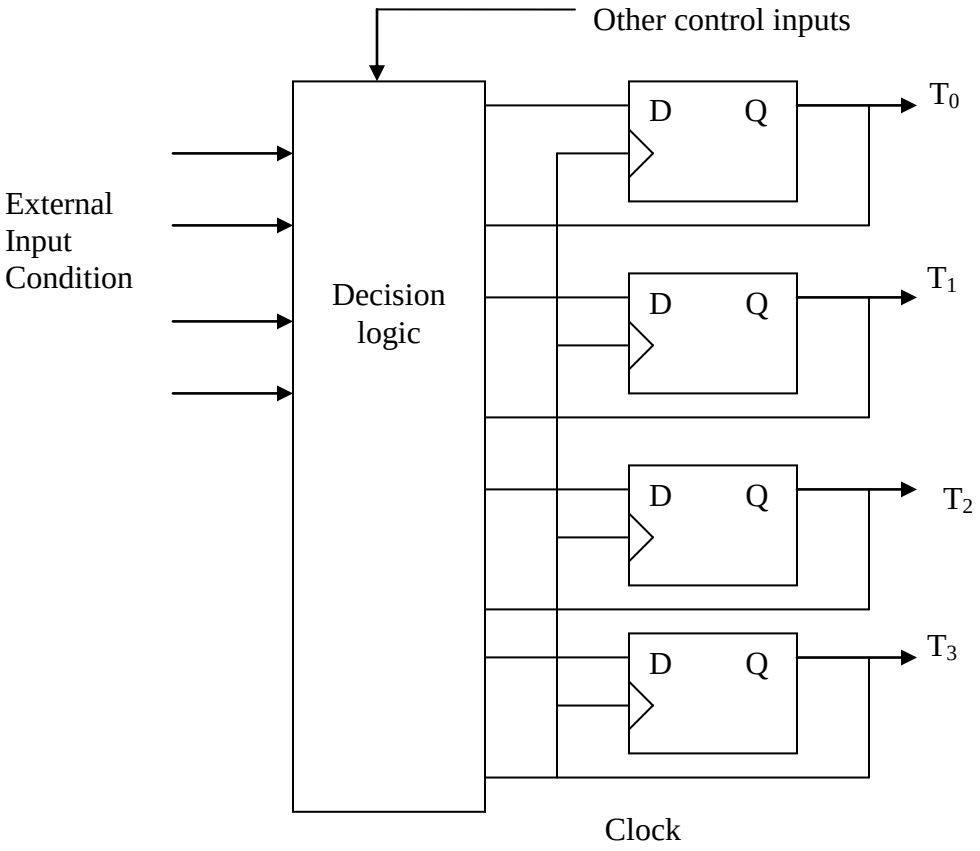


Figure 3.4: Control logic with flip-flop per state

3.4.2 Sequence register and decoder method

Uses a register to sequence the timing signals for the computer as shown in figure 3.5. The register output is decoded to provide one output for each state. For *n* flip-flops in the sequence register, the circuit will have 2^n states and the decoder will have 2^n outputs. The timing variables together with other variables produce the control functions that initiate all the micro-operations for the computer. It determines the address sequence that is read from control memory. The address of the next micro-instruction can be specified in several ways depending on the sequencer inputs. Typical functions of the micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations.

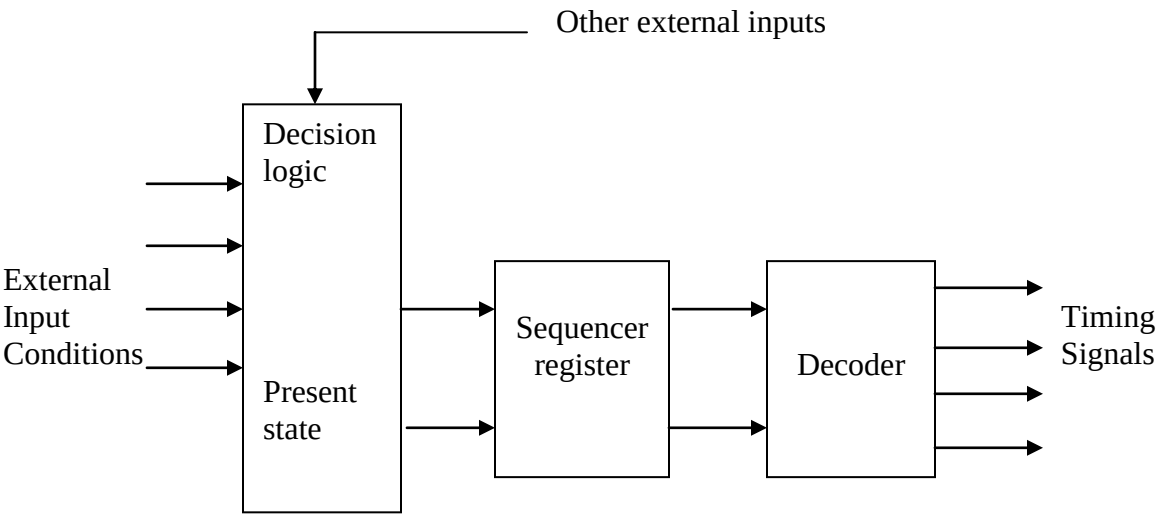


Figure 3.5: Control logic with sequence register and decoder.

3.4.3 Control signal generator

Examines the instruction code in the Instruction Register (IR) and sends out control and synchronization signals for the instruction to be implemented. The conditions under which each micro-operation occurs dictate the design of the control unit. The control unit generates the control signals that load, increment, and clear the registers in the register section. The

control unit also enables the buffers used to control the CPU's internal buses. The function to be performed by the ALU is specified by control unit. By outputting the control signals in the proper sequence, the control unit causes the CPU to properly fetch, decode and execute every instruction in the instruction set. Control unit can be hardwired or micro-sequenced. Hardwired control uses sequential and combinational logic to generate control signals. Micro-sequenced control uses a lookup memory to output the control signals. The simplest control unit has three components namely: The counter which contains the current state, decoder which takes the current state and generates individual signals for each state and some combinational logic to take the individual state signals and generate the control signals for each component as well as the signals to control the counter (Raghuram, 1989; Malvino, 1980). The control signal generator is designed in two ways. One using Karnaugh map simplification resulting in a large number of gates or using decoders and de-Morgan's theorem to result to a fewer number of gates.

3.4.4 Micro-program control

The control unit initiates a series of sequential steps of micro-operations. During any given time, certain operations are to be initiated while all others remain idle. Thus the control variables at any given time can be represented by a string of ones and zeros called the control word. Control words can be programmed to initiate the various components in the system in an organized manner. In a control unit consisting a micro-programmed control unit, each control word of memory is called a micro-instruction and a sequence of micro-instructions, is called a microprogram. The control memory can be a Read Only Memory (ROM), because alteration of the microprogram is seldom needed. The use of microprogram involves placing all control variables in words of the ROM for use by the control unit through successive read operations. The content of the word in ROM at a given address specifies the micro-operation

for the system. Once these operations are executed the control unit must determine its next address. The location of the next micro-instruction may be the next one in sequence, or it may be located somewhere else in the control memory (Mano, 2005; Malvino, 1980). For this reason some bits of the microinstruction are used to control the generation of the address for the next micro-instruction. Figure 3.6 shows the microprogram control logic.

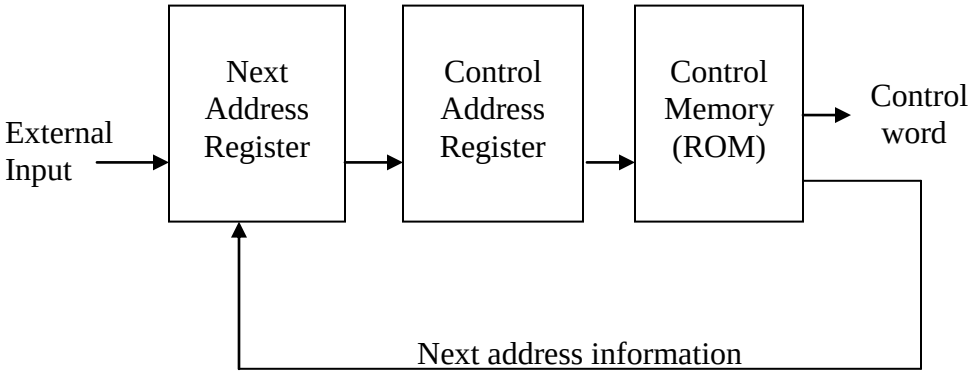


Figure 3.6 Microprogram control logic

3.5 System bus

A bus as used in digital systems refers to a group of wires through which binary information is transferred. It is used to facilitate the transfer of data and instructions between the microprocessor, memory and the input/output devices. A common system bus can be constructed with multiplexers (Zeigher, 1976). The following major buses are used:-

3.5.1 Data bus

This is a group of wires (pins) used to send and receive data. It is bi-directional facilitating shared path and data movement in both directions. The data bus width is the same as the word size of a microprocessor which is the number of bits that it can process at a time. The more signals that can be sent at the same time, the more data can be transmitted in a specific interval and therefore the faster is the bus.

3.5.2 Address bus

This is a set of wires that carry the addressing information used to describe the memory location to which the data is being sent or from which the data is being retrieved. The size of memory which can be addressed with p lines in the address bus is 2^p, referred to as address space or memory space. This number of locations which can be accessed by the processor is known as its physical address space and is often expressed in terms of kilobytes, megabytes and gigabytes. In many real situations, latches are present at the address input lines of the memory which is used to latch the address as soon as it arrives. Once the address is latched, the address bus is no longer required for the rest of the operation of the memory read or memory write cycle and can be used for other purposes.

3.5.3 Control bus

A set of wires used to provide timing signals to synchronize the operation of the processor with the operation of the external circuitry. This is between the microprocessor, memory or the input and output devices. It has also the facility of interrupting the execution of the normal sequence of the program.

3.6 Clock system

A crystal oscillator controls clock speed using silver quartz in a small tin container. When voltage is applied to the quartz, it begins to oscillate at frequency rates dictated by the crystal in the form of current that alternates at the frequency of the crystal. The alternating current is the clock signal. A typical speed is usually measured in megahertz (MHz). The microprocessor is normally tested at different speeds, temperatures and pressures for a safe maximum speed at which it will operate under variations of temperatures and pressure usually encountered during operation. The size of data bus, address bus and the type of control bus as

well as the power supply and clock requirements vary from one microprocessor to the other.

3.7 Memory

Memories are arrays of registers arranged in a sequence. Microprocessor memories usually have a space that is fixed for a number of binary numbers in each line. They are mainly of the semiconductor type and are used for storage of data and instructions to be used by microprocessor system. They are popular because of small size, low cost, high speed, high reliability and ease of expansion of memory size. The basic element is the flip-flop and memory is specified by the number of locations available and number of bits at each location. Flip-flops are grouped together to form registers. Commonly used memories are 64K, 128K, 256K and 512K, whereas commonly used values for word size are 1, 4,8,16 and 32 bit. The data bus is time multiplexed and it is used as input bus for some specific time and as an output bus for some other time depending upon a read/write control input. Memory locations are accessed internally by means of decoders and multiplexers. The write operation requires selection of a memory location and application of logic one voltage to CS (chip select) and a logic zero to R/W (Ufenbeck, 1995).

3.7.1 Write cycle

Important timing characteristics of the write cycle are:-

i) Write cycle time; minimum time required between successive write operations

ii) Write pulse time; minimum length of a write pulse

iii) Write release time; minimum amount of time for which the address must be valid after the write pulse ends.

iv) Data set up time; minimum amount of time for which the data must be valid before the write pulse ends.

v) Data hold time; minimum amount of time for which the data must be valid after the write pulse ends.

3.7.2 Read cycle

The read cycle can be described with the following attributes:

i) Read cycle (t_A); this is the minimum amount of time the valid address must be present for reading a word from memory

ii) Access time (t_{RC}); maximum amount of time from the start of the valid address of the read cycle to the time when the valid data is available at the data outputs. The access time is at most equal to the read cycle time where t_A is less or equal to t_{RC}.

iii) Read to output active time; minimum delay time between the beginning of the read pulse and the output buffers coming to active state from the high impedance state.

iv) Chip select to output valid time; maximum delay time between the beginning of the chip select pulse and the availability of valid data at the outputs

v) Chip select to output active time; this is the minimum delay time between the beginning of the chip select pulse and the output buffers coming to active state.

vi) Output tri-state from read; this is the maximum time delay between the end of the read pulse and the output buffers to high impedance state.

vii) Data hold time; the minimum time for which the valid data is available at the data outputs after address ends.

The timing waveforms for the read and write are shown in figure A.1

To execute programs the microprocessor needs instructions and data from memory and performs the computing operations in its Arithmetic Logic Unit. Results are transferred to the output section for display or stored in memory. Two types of memories used in microprocessor system;

3.7.3 Random access memory

It is a special type of semiconductor chip with a cell as the basic element. The information/data stored in RAM is lost when the power is removed from it. This RAM is used for temporary storage of user programs and data. The information stored in it can easily be read or altered. RAMs are of two types; Static RAM (SRAM) and Dynamic RAM (DRAM). DRAM requires periodic refreshing for maintaining the stored information while SRAM does not. SRAMs are preferred when the memory size is not too large, for systems requiring large memories, DRAMs are used since they have high density, low power consumption and are inexpensive.

3.7.4 Read only memory

It is a programmable chip which can only be read by a microprocessor. It is non-volatile hence necessary for long term storage of instructions. They are of two basic types namely; user programmable ROM and masked programmable ROM. User programmable ROMs are of two types: Programmable ROM (PROM) which can be programmed by the user only once and Erasable PROM (EPROM) which can be programmed, erased and reprogrammed by the user a number of times. A Masked programmable ROM has the information written into it at the time of manufacture hence the user uses the pre- stored information in this chip as it is.

3.8 Computer programming languages

Computers may be programmed using a variety of programming languages**.** At one end of scale, the binary coded machine instructions are the programming language which the computer can understand and execute directly. Human beings find it very difficult to program directly in machine code however they prefer to write programs in the languages which are closer to English language (Green, 1985). There are three levels that can be used to write a

program for microprocessors: machine language, Assembly language and High level languages.

3.8.1 Machine language

Instructions are patterns of bits with different patterns corresponding to different commands to the machine. Machine code or machine language is a system of instructions and data directly executed by a computer's central processing unit. Machine code is the lowest-level of abstraction for representing a computer program. While easily understood by computers, machine languages are almost impossible for human beings to use because they consist entirely of zeros and ones. Programmers therefore, use either a high-level programming language or an assembly language.

Every CPU model has its own machine code, or instruction set defined by the designer of the computer and constrained by the architecture chosen. Some nearly completely compatible processor designs may have slightly different effects after similar instructions system. A machine code instruction set may have all instructions of the same length, or may have variable-length instructions. How the patterns are organized depends largely on the specification of the machine code. Common to most is the definition of the opcode field which specifies the exact operation to be executed, for example "add". Other fields may give the type of the operands, their location, or their value directly, for example operands contained in an instruction are called immediate. Every CPU has its own unique machine language. Programs must be rewritten or recompiled to run on different types of computers. The disadvantage of this language is that programs are hand coded, which is slow and prone to translation errors. The machine code programs are not portable. However they are fast to execute since no translation is required and occupy less memory space.

3.8.2 Assembly language

It is a much more readable edition of machine language, which uses mnemonic codes to refer to machine code instructions, rather than simply using the instructions' numeric values. An assembly language contains the same instructions as a machine language, but the instructions and variables have names instead of being just numbers. The manufacturer usually supplies the complete description of each instruction and therefore the assembly language and mnemonics is specific to each microprocessor. There is one to one correspondence between the mnemonic codes and the actual machine codes. The programmer does not have to calculate address to allocate memory by hand but labels are used to represent address in assembly language. Mnemonics are used because;

i) They are more meaningful than hex or binary values.

ii) Reduces the chances of making an error.

iii) They are easier to remember than bit values.

Assembly language statements are usually written in standard form that has four fields: label field, opcode field, operand field and comment field. For example:

Label field	Opcode field	Operand field	Comment field
START:	LD	A, 00H	; Clears A register

A label is a group of symbols used to represent an address, which is not specifically known at the time the statement is written. A colon usually follows labels. Here the label START is equal to the address of the instruction LD A, 00H

The operation code of the instruction contains the mnemonic for the instruction performed. The operation field of the statement contains the data, the memory address, the port address or the name of the register on which the instruction is performed. The comment field starts with a semi-colon or an asterisk and is not part of the machine language program. The comments in

a program remind one of the functions that an instruction performs in the program. Assembly language programs are translated into machine language by a program called an assembler.

The advantages of assembly language program are:-

i) Faster translation times hence more efficient.

ii) Reduced translation errors.

iii) It requires less memory space since they are compact compared to high level languages.

iv) Changes can be made easier and faster.

However the disadvantages are that the programmer requires knowledge of the processor architecture and instruction set and thus any change in the hardware requires complete rewriting of the program.

3.8.3 High level language

It uses program statements that are more English like than those of assembly language. These statements may represent many machine code instructions. High level languages (HLL) is problem oriented thus a programmer can master this language without knowing the detailed structure of the computers processor architecture. It has some disadvantages in that programs written in HLL require more memory space and are executed more slowly compared to the same program written in assembly language. Its main application is in complex data processing programs that manipulate large amounts of data such as inventory records, however programs that involve a lot of hardware control such as robots and factory control systems, that must run as quickly as possible are usually written in assembly language. HLL include languages such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from

machine languages. In contrast, assembly languages are considered low-level because they are very close to machine languages. The main advantage of high-level languages over low-level languages is that they are easier to read, write, and maintain. Ultimately, programs written in a high-level language must be translated into machine language by a compiler or interpreter. Interpreter reads the source code one instruction or line at a time, converts this line into machine code and executes it. The machine code is then discarded and the next line is read. It has the advantage of simplicity and can be interrupted when running to change a program and either continue or start again. Its only limitation is that every line must be translated even if it appears more times than one. A compiler reads the whole source code and translates it into a computer machine program to perform the required tasks which is output as a new file. It is fast since translation is done only once although one cannot change the program without going back to the original source code, editing and recompiling.

CHAPTER FOUR

RESEARCH METHODOLOGY

4.1 Introduction

The design technique used in this study took in to consideration the fact that the circuit was a demonstration module that was to execute not less than twelve instructions. The module used Medium Scale Integration (MSI) and Small Scale Integration (SSI) ICs (Mano, 2005). Only Transistor Transistor logic devices were used as one of project objectives. The initial stages of the design involved the design of an instruction set and based on this set, the formulation of a timing and control network was made. The listing of the register transfer operations needed to execute all the instructions was done. After designing the circuit based on these considerations, simulations were carried out on the circuit to determine the functionality of the circuit in implementing these instructions. The instructions were then used to design a program written using the mnemonics and the circuit used to run the program using a state machine control method.

4.2 Design procedure

Methodology adopted in this research work is illustrated by means of a flow chart given in figure 4.1 The appropriate hardware components for the system design were selected on the basis of the following factors:-

i) Voltage/Current ratings.

ii) Convergence errors generated and frequency response.

iii) Operating temperature.

iv) Reliability.

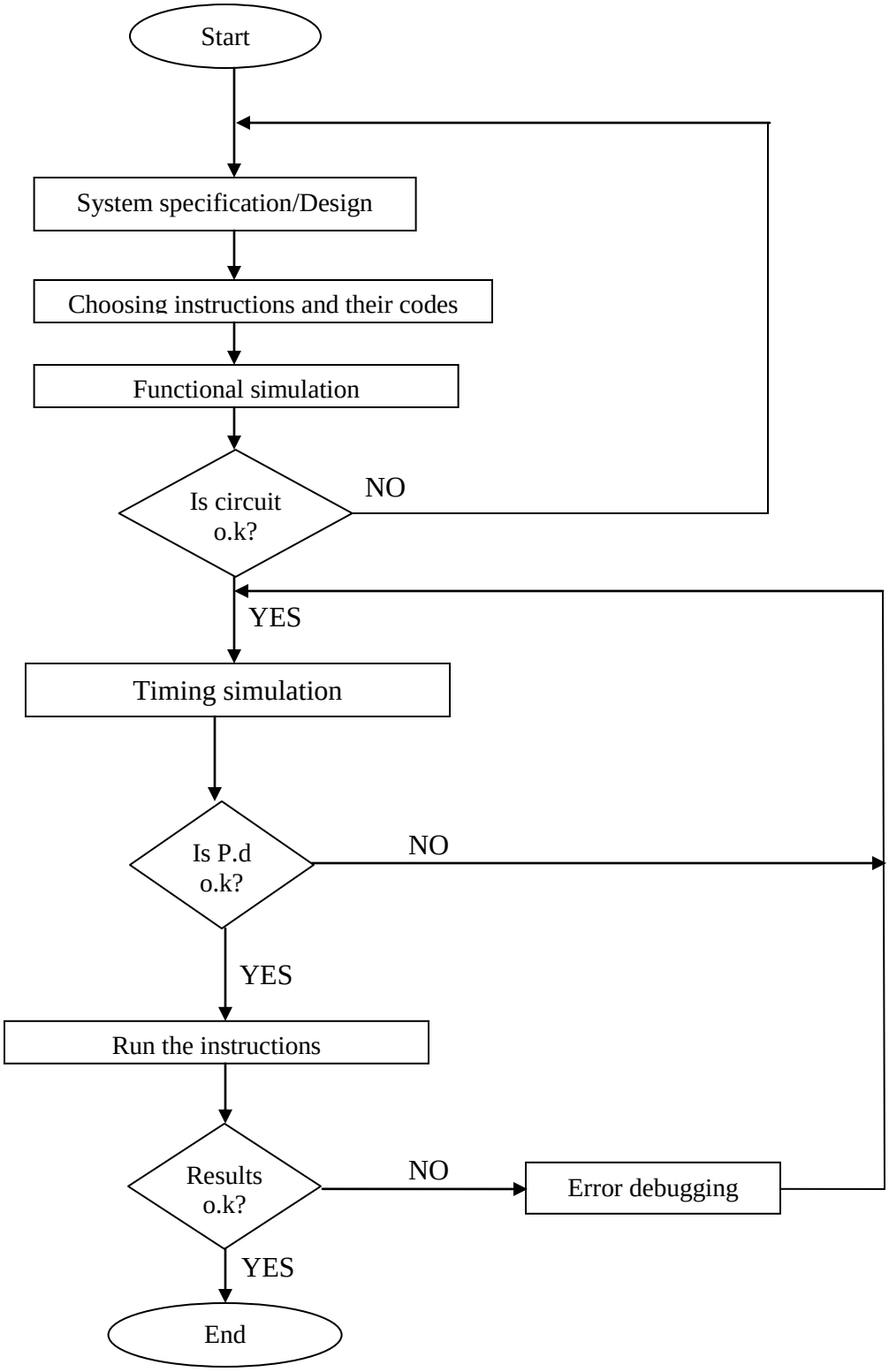


Figure 4.1: Design procedure flowchart

P.d. is the propagation delay

4.3 Instruction specification.

The instructions were chosen in order to avoid any spare opcodes. Four bit opcode word length was chosen to give rise to 2^4 = sixteen possible instructions. Since the circuit was used to demonstrate digital processing, instructions in all types of the basic processor instructions were included in the instruction set.

4.3.1 Data transfer instructions

These instructions move data between memory and processor registers, processor registers and input or output, and between the processor registers themselves. These instructions include:

i) **Load** instruction is used to designate a transfer from memory to a processor register, usually an accumulator.

ii) **Move** instruction is used to designate a transfer from one register to another. Other instructions include Input, Output (Uffenbeck, 1985).

Table 4.1 gives a summary of data transfer instructions implemented.

Table 4.1: Data Transfer Instructions

MNEMONIC	DESCRIPTION
MOV A,r1	Transfer of data at input port r1 to input register A.
MOV B, r2	Transfer of data at input port r2 to input register B.
MOV D, A	Transfer of data at register A to output register D.
STA	Storage of data in register A in temporary storage register.
STB	Storage of data in register B in temporary storage register.
LDA	Loads register A with data in storage register, usually the memory

4.3.2 Data manipulation instructions

Instructions in this group perform operations on data and provide the computational capabilities for the computer (Mano, 2005). They are divided into three:

i) Arithmetic instructions

They include the four arithmetic operations: Subtraction, Addition, Multiplication and division. Also they include Increment and Decrement instructions. Instructions implemented are illustrated in table 4.2.

Table 4.2: Arithmetic instructions

MNEMONIC	DESCRIPTION
INC A	Increments the contents in register A by 1.
DCA	Decrements the value stored in register A by 1.
ADD A	Adds contents of register A to contents of storage register and places result in register A.
ADD A, B	Adds contents of register A to contents of register B and stores the result in register A.

ii) Logical and bit manipulation instructions

Perform binary operations on strings of bits stored in registers. They are useful for manipulating individual bits or a group of bits that represent binary coded information. This group includes the logical operators AND, OR and EXCLUSIVE-OR.

Table 4.3 gives instructions implemented in the logical and bit manipulation group.

Table 4.3: Logical and Bit Manipulation Instructions

MNEMONIC	DESCRIPTION
CMA	One's complement of contents in register A.
OR A, B	Performs bit by bit ORed operation on contents of registers A and B. and stores the result in register A.

iii) Shift instructions

Instructions to shift the content of an operand in which the bits of a word are moved to the left or right described in table 4.4. The bit shifted in at the end of the word determines the type of shift used. Shift instructions may specify logical shifts, arithmetic shifts, or rotate-type operations.

Table 4.4: Shift Instructions

MNEMONIC	DESCRIPTION
SHR B	shifts to the right data available in register B.
SHR A	shifts to the right data available in register A.

4.3.3 Program control instructions

After execution of a data transfer or data manipulation instruction, control returns to the fetch cycle. The program counter which was incremented earlier now contains the address of the next instruction in the sequence. Other instructions may cause the flow of control to be altered. These instructions include Branch, Jump, Skip, Call, Return, and Halt. In this group the instruction included is Halt and JNZ in which by default when the microprocessor is off the op-codes are 0000 hence no operations are performed and 1110 respectively

.

4.4 System design

The system consisted of two parts namely: the System hardware and System software. Hardwire technique was used in the system design as opposed to microprogrammed technique and Electronic Workbench as simulation software. Hardware design control can be optimized to produce a fast mode of operation. Qualey (2006) used a microprogrammed control in EPROM chip.

4.4.1 System hardware

Four - four bit registers were chosen to provide a data path for the transfers. An ALU was chosen for the logical and arithmetic operations. Two registers were used as process registers. One register was selected as an accumulator register and one as an output register. No input register was included because it was considered appropriate to use one of the process registers as an input register during the data input instruction rather than to have one more register to be used as an input register. Data processing was done in parallel and therefore the registers were used in the Parallel in parallel out (PIPO). The application of the processor did not involve large amounts of programs and data storage and flip-flop elements were used. Two tri-state buffers were used; one to enable the memory read and the other for enabling the memory write process. The processor had to perform operations on four bit data and hence a four bit bus lines were used. This gave a word size of four bits and classified the processor as four bit processor. Four switches were used for purposes of transferring data into the processor via the input port. The contents of the output port, registers and the memory unit were shown using probes to represent indicators. The configuration block diagram is shown in figure 4.2.

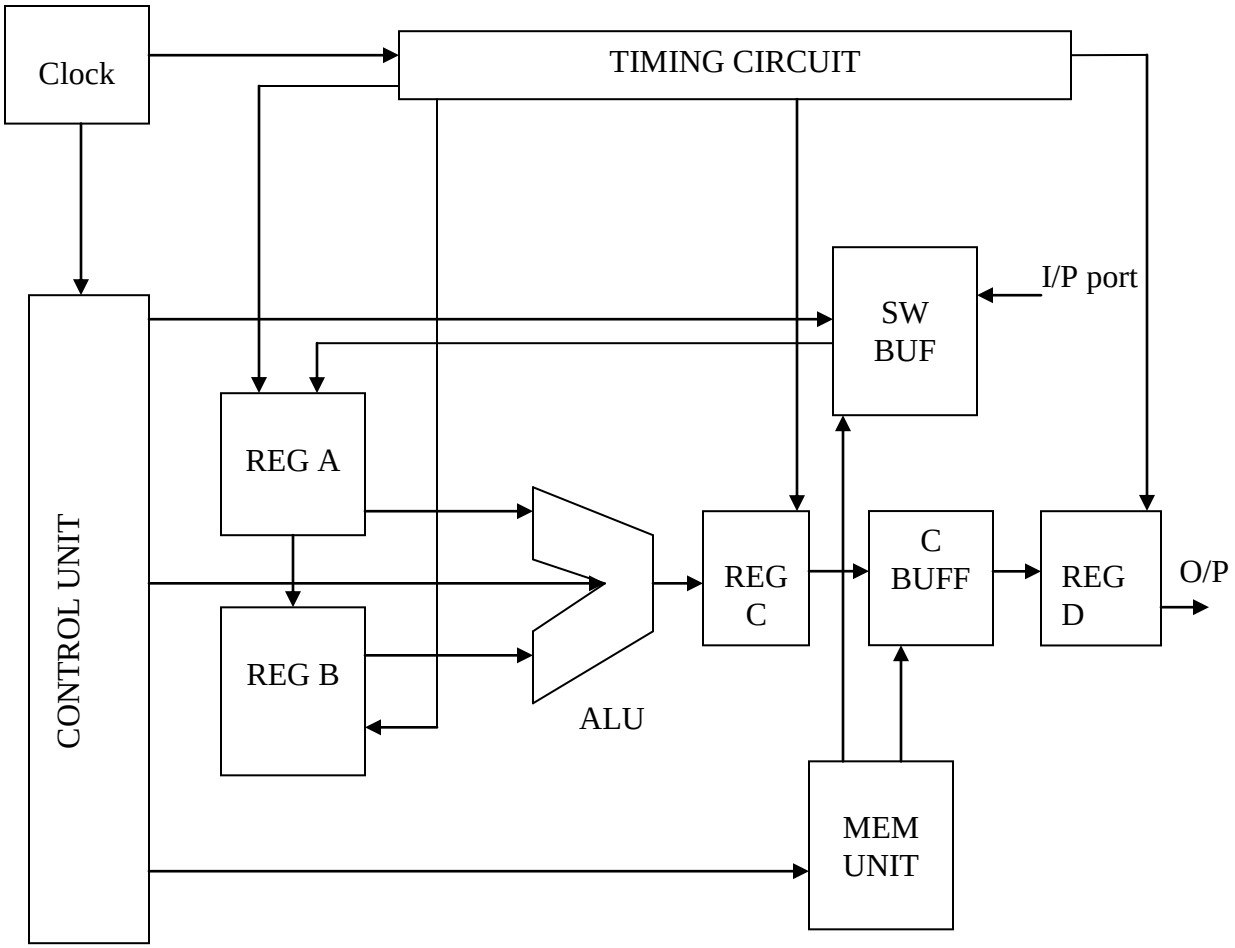


Figure 4.2: System configuration block diagram.

Register C is an accumulator register and stores immediate output of the ALU. Register D is an output register, which stores the results of the accumulator. Registers A and B are process registers and are used to input data to the ALU. SW BUF is the switch input buffer, MEM UNIT is the memory unit and C BUFF is the C register output buffer. For the implementation of these instructions using digital circuitry the code assignment was carried as shown in table 4.5.

Table 4.5: Mnemonics and their code assignment.

MNEMONIC	**4-BIT OP-CODE**
HLT	0000
MOV A,r1	0001
MOV B,r2	0010
MOV D,A	0011
SHR A	0100
SHR B	0101
STA	0110
CMA	0111
OR A,B	1000
INC A	1001
ADD A	1010
ADD A,B	1011
LDA	1100
STB	1101
JNZ	1110
DCA	1111

These op-codes were then used for control purposes to determine the different inter-register movement and functions to be performed by the ALU. They also determined the conditions of the different buffers used in the arrangement, disabling or enabling them as required. Three registers were used for the processing. Registers A and B which are also input registers and register C. Each of the instruction was then executed in three stages, each stage including the clocking of one of the registers for loading or shifting purposes.

4.5 Operating sequence

Sixteen instructions were assigned opcodes from 0000 to 1111. They were simulated and respective registers clocked during execution as discussed in the next section.

HLT

During this instruction, there are no changes in register status although register C is clocked and the buffer enabled. No clocking of register A and register B. The inclusion of clocking of register C and enabling the buffer is for simplification of design.

MOV A, r1

This instruction placed the contents of the input port r1 into register A. No clocking of register B. Set ALU to F=A and clock C. Enable buffer connected to switches and clock register A.

MOV B, r2

This instruction placed the contents of the input port r2 into register B. Enable register B input buffer and clock register B. Set ALU to F=A and clock C. No clocking of register A.

MOV D, A

This instruction was used to place the contents of register A to the output port via register D. No clocking of register B. Enable the C output buffer and clock A and D instantaneously. Set ALU to F=A and clock C.

ADD A

Add the contents of the memory unit to contents of register A and store the result in register A by first copying the contents of storage register to register B. Set ALU function to ADD and clock register C. Enable register C output buffer and clock register A.

LDA

Copy contents of memory unit into register A. Set ALU to F=B and clock register C. Enable register C output buffer and clock register A.

STA

Copy contents of register A in to the memory unit. No clocking of register B. Enable the C output buffer and clock register A at the same time when writing into Memory.

DCA

Decrement contents of register A by one. No clocking of register B while Setting the ALU function to 'F=A minus one' and clock register C. Enable the C output buffer and clock register A.

JNZ

It examines the contents of the status register and jumps if the contents are non-zero

CMA

Complement contents of register A. No clocking of register B. Set the ALU function to 'F= A and clock register C. Enable the C output buffer and clock register A.

ADD A, B

Add the contents of register B to register A and result left in register A. No clocking of register B. Set the ALU function to 'F=ADD' and clock register C. Enable the C output buffer and clock register A.

STB

Copy contents of register B in to the memory. No clocking of register B. Set the ALU function to 'F=B' and clock register C. Enable the C output buffer and clock register A at same time writing to memory.

SHR A

Arithmetic shift right of the contents of register A. No clocking of register B. Set the ALU function to ‘F=A ’ and clock register C. Clock register A while enabling the A shift operation.

SHR B

Arithmetic shift right of the contents of register B. Clock register B while enabling the B shift enable. Set the ALU function to ‘F=A’ and clock register C. Enable the C output buffer and clock register A.

INC A:

Increment the contents in register A by one. Increment the contents in register A by one. Clock register B. Set the ALU function to ‘F=ADD’ and clock register C. Enable the C output buffer and clock register A.

OR A, B:

The contents of the Accumulator is ORed to contents of register B. No clocking of register B. Set the ALU function to ‘F=ADD’ and clock register C. Enable the C output buffer and clock register A. The truth table for enabling of the ALU functions is shown in the appendix A.4.

4.6 Control signals

In order to implement the instructions, control signals were required. A table showing the relationship between the opcode value and effect on control signals was constructed as shown in table 4.6. There were seventeen control signals to be generated namely:

i) Four function select input for the ALU namely S_o, S_1, S_2 and S_3.

ii) Mode control input for the ALU, M.

iii) C_{in}- Carry input for the ALU.

iv) A register input buffer.

v) B register input buffer.

vi) C register input buffer.

vii) Storage register (memory read).

viii) Storage register (memory write).

ix) Clock for register A.

x) Clock for register B.

xi) Clock for register C.

xii) Clock for register D.

xiii) SH/$\overline{\text{LOAD}}$ control for register A.

xiv) SH/$\overline{\text{LOAD}}$ control for register B.

xv) SH/$\overline{\text{LOAD}}$ control for register D.

Generation of the control signals could be done by use of basic logic gates. The objective was to generate these signals using an optimal number of gates. The Karnaugh map technique was chosen because it could simplify the maxterms and minterms to their minimal canonical level.

Table 4.6: Effect of instruction code on the control signals

	0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**111**
S_3	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
S_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S_1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
S_0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1
M	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1
C_{in}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A_{Buff}	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
B_{Buff}	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
MEM Read	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1
MEM Write	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
Clock A	0	1	0	1	0	0	1	1	1	0	1	1	1	1	1
Clock B	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0
Clock C	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Clock D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
SH/LD A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SH/LD B	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
SH/LD D	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
C_{Buff}	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0

4.6.1 Sequence register

This is a multi-cycle execution of instructions with three clock cycles being used. We therefore have sequence register for generation of the three clock cycles in sequence. The register is decoded to provide one output for each state. The choice of 74LS160 counter and 74LS138 was used to generate the sequence. The counter produced the timing signals which are decoded to produce the sequence required. For n flip-flops in the sequence generator, the circuit will have 2^n states and the decoder will have 2^n outputs.

4.6.2 Control signal generator

The control signal generator is designed in two ways. One using Karnaugh map simplification resulting to a large number of gates or using 4:16 decoder and de-Morgan's theorem to result to a fewer number of gates (Brown and Vranesic, 2003). Since the number of instructions are sixteen, a 4:16 decoder is used to generate all the possible minterms of a four bit binary value. This results to a logic output zero for each output that is enabled. The signals S_0, S_1, S_2 and S_3 which are the four function select input for the ALU as given in table 4.6 and by use of De-morgan's theorem are generated as follows:

$S_3 = 1001 + 1010 + 1011 + 1100 + 1101 + 1111$

$= \quad m_9 + \quad m_{10} + \quad m_{11} + \quad m_{12} + \quad m_{13} + \quad m_{16}$

The minterm generated from the decoder is logic 0 for this function

$$S_3 = \overline{m}_9 + \overline{m}_{10} + \overline{m}_{11} + \overline{m}_{12} + \overline{m}_{13} + \overline{m}_{15}$$

Using

$$\overline{A+B+C} = \overline{A} \cdot \overline{B} \cdot \overline{C} \quad \text{then}$$

$$S_3 = \overline{\overline{m}_9 \cdot \overline{m}_{10} \cdot \overline{m}_{11} \cdot \overline{m}_{12} \cdot \overline{m}_{13} \cdot \overline{m}_{15}}$$

$$S_0 = \overline{\overline{m_8} \cdot \overline{m_9} \cdot \overline{m_{10}} \cdot \overline{m_{11}} \cdot \overline{m_{14}} \cdot \overline{m_{15}}}$$

$$S_1 = \overline{\overline{m_{13}} \cdot \overline{m_{14}} \cdot \overline{m_{15}} \cdot \overline{m_{16}}}$$

$$S_2 = \overline{\overline{m_{16}}}$$

$$M = \overline{\overline{m_7}} \cdot m_{12} \cdot \overline{\overline{m_{13}}} \cdot \overline{\overline{m_{14}}}$$

$$\left.\begin{aligned} \overline{A_{Buff}} &= \overline{m_1} \\ \overline{B_{Buff}} &= \overline{m_2} \end{aligned}\right\} \text{From the decoder}$$

For the generation of signals for enabling of the C buffer, then consideration is made of the fact that the output buffer is enabled when register A is clocked. Measures were taken so that when register B is clocked the buffer is disabled to avoid erroneous transfer of data in register A to register B. This is done by inclusion of an OR gate so that the buffer can also be disabled by the clocking of register B.

$$C_{Buff} = \overline{\overline{m_5} \cdot \overline{m_4} \cdot \overline{m_2} \cdot \overline{\overline{m_1}} + G_1}$$

The signal read from memory is used for clocking the storage register D

$\overline{\text{MEM Read}} = (\overline{m_{10} + m_{12}})$ used for enabling MEM output buffer

MEM Write = STA + STB

$= (\overline{m_6} + \overline{m_{13}})\overline{G_3}$ for clocking storage register

$$\text{CLK A} = (\overline{m_0} + \overline{m_2} + \overline{m_5} + \overline{m_{13}}).G_3 \; \overline{(\overline{m_0}\, m_2\, \overline{m_5}\, \overline{m_{13}})}.G_3$$

The storage register (temporary memory) was to be enabled only when reading from memory.

MEM O/P BUFFER = MEM READ + G_3 + G_2

G_3 and G_2 prevent erroneous read from memory when registers A and C are clocked

$C_{in} = 1$, Clock C = 1

Clock D = M_3

$\overline{SH/LOAD}$ A = $\overline{\overline{m_4}}$

$\overline{SH/LOAD}$ B = $\overline{\overline{m_{15}}}$

$\overline{SH/LOAD}$ D = $\overline{\overline{m_3}}$

CK B = $(\overline{\overline{m_9}} + \overline{\overline{m_5}} + \overline{\overline{m_2}} + \overline{\overline{m_{12}}})$ G1 + (MEM READ) G_1

$= (\overline{\overline{m_9}}.\overline{\overline{m_5}}.\overline{\overline{m_2}}.\overline{\overline{m_{12}}})$ G1 + (MEM READ) G_1

4.6.3 Instruction register design

The instruction register was designed as a state machine to perform the addition and subtraction operations. This is done by use of a counter that would go through states representing the codes for the various instructions. The mnemonics are used to design an assembly language programs for addition or subtraction. The program consists of sequences of instructions that are executed by the processor in the order in which they occur. The sequences were viewed as micro programs to implement the program. The state machine was viewed as a micro-program sequence. The sequences are represented in state diagram form. The state machine was viewed as a counter passing through the given states as per the program. The Instruction Register (IR) is a non-binary synchronous counter. The external input for enabling the IR is the simulation switch.

4.6.4 Addition

In order to achieve addition five states were used. By default the initial state was HLT.

The assembly program is shown in table 4.7

Table 4.7: Assembly program for addition

MNEMONIC	CODE
HLT	0000
MOV A,r1	0001
MOV B,r2	0010
ADD A,B	1011
MOV D,A	0011

The counter would then count through these states in the sequence and stop at state 0011.

The state diagram would appear as shown in figure 4.3

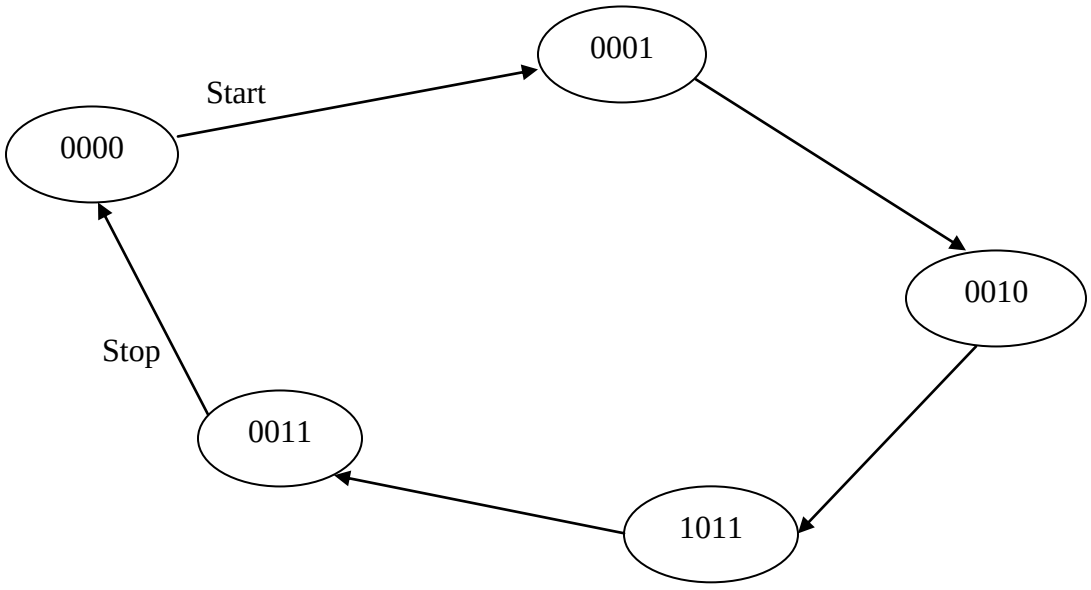


Figure 4.3: Addition state diagram

The sequence in figure 4.3 represented in table 4.8, which shows the transition states for the addition circuit.

Table 4.8: Addition sequence

Present state	Next state	Register input (instruction register)
$Q_DQ_CQ_BQ_A$	$Q_DQ_CQ_BQ_A$	$Q_DQ_CQ_BQ_A$
0000	0001	0001
0001	0010	0010
0010	1011	1011
0011	0011	0011
0100	0001	0001
0101	0001	0001
0110	0001	0001
0111	0001	0001
1000	0001	0001
1001	0001	0001
1010	0001	0001
1011	0011	0011
1100	0001	0001
1101	0001	0001
1110	0001	0001
1111	0001	0001

The information in the table 4.4 and Karnaugh map simplification were used to derive the optimum logic circuit for generation of inputs to the generator that would enable this count sequence (Brown and Vranesic, 2003). From the table 4.8 it can be deduced that,

$$D_D = \overline{Q}_D \overline{Q}_C Q_B \overline{Q}_A$$

$$D_C = 0$$

$$D_A = \overline{\overline{Q}_D \overline{Q}_C \overline{Q}_B Q_A}$$

The outputs of this instruction register state machine were used as the inputs for the instruction decoder and simulations run for typical values of data at input ports A and B being added.

4.6.5 Subtraction

In order to perform digital subtraction using the processor, then it was necessary to define the order of binary subtraction. For two values to be subtracted from each other we have M as the Minuend and N the subtrahend. For M – N, 2's complement was used where no carry

indicates that the subtrahend is a negative number and a carry indicates that the subtrahend is a positive number (Mano, 2005). The assembly language for this instruction was thus;

MOV B, r2 contains minuend.

MOV A,r1 moves subtrahend to register.

CMA complements the subtrahend in register A.

ADD A, B if a carry occurs, its stored in the storage register.

ADD A adds contents of storage register to contents of register leaving the results in register A.

MOV D, A places the contents of the register A to the output port which is the result.

The state diagram for the subtraction is as shown in figure 4.4

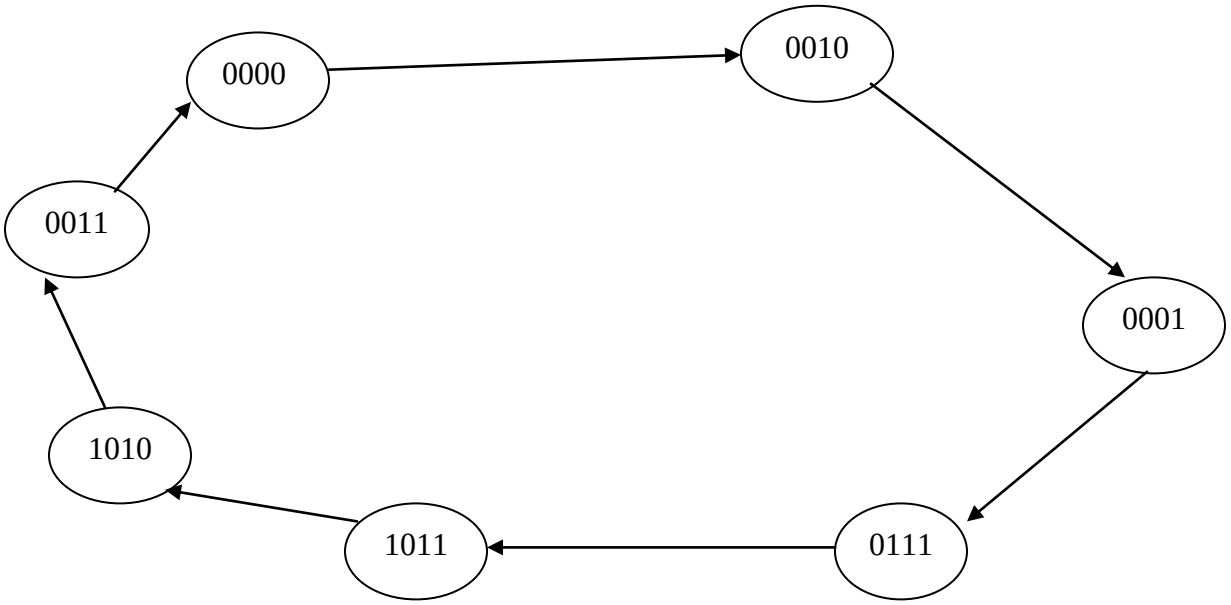


Figure 4.4: Subtraction state diagram

The storage was now used as a special function generator to store the carry generated by the ALU. State machine design of the IR is represented in table 4.9

From table 4.9 it can be deduced that,

$D_D = \overline{Q}_D Q_C Q_B Q_A + Q_D \overline{Q}_C Q_B Q_A$

$D_C = \overline{Q}_D \overline{Q}_C \overline{Q}_B Q_A$

$$\overline{D_A} = \overline{\overline{Q_D}\,\overline{Q_C}\,\overline{Q_B}\,\overline{Q_A} + Q_D\overline{Q_C}Q_B\,Q_A}$$

$$D_B = \overline{Q_D}\,\overline{Q_C}\,\overline{Q_B} + \overline{Q_D}\,Q_B\,Q_A + Q_D\,\overline{Q_C}\,Q_B$$

The outputs of this instruction register were used as the inputs for the instruction decoder and simulations run for typical values of data of input ports A and B being subtracted.

Table 4.9: State machine design for instruction register.

Present state	Next state	Register Inputs
$Q_DQ_CQ_BQ_A$	$Q_DQ_CQ_BQ_A$	$D_DD_CD_BD_A$
0000	0010	0010
0001	0111	0111
0010	0001	0001
0011	0011	0011
0100	0001	0001
0101	0001	0001
0110	0001	0001
0111	1011	1011
1000	0001	0001
1001	0001	0001
1010	0011	0011
1011	1010	1010
1100	0001	0001
1101	0001	0001
1110	0001	0001
1111	0001	0001

4.6.6 Program control

The conditional jump instruction is implemented when the state of the status register is logic 1. When the contents of registers A and B are not equal (A>B or A<B), the contents of the status register is logic 0 and JNZ instruction is not executed. However if the contents of the register's A and B is equal (A=B), the JNZ is active and the program is transferred to a new control to execute a multiplication subroutine. The assembly language for the instruction is:

MOV B, r2 contains minuend.

MOV A,r1 moves subtrahend to register.

JNZ complements the subtrahend in register A.

ADD A, B if a carry occurs, its stored in the storage register.

ADD A doubles the contents of register A

MOV D, A places the contents of the register A to the output port which is the result.

Table 4.10: Assembly program for jump instruction

MNEMONIC	CODE
HLT	0000
MOV A,r1	0001
MOV B,r2	0010
JNZ	1110
ADD A,B	1011
ADD A	1010
MOV D,A	0011

The counter would then count through these states in the sequence and stop at state 0011.

The state diagram would appear as shown in figure 4.3

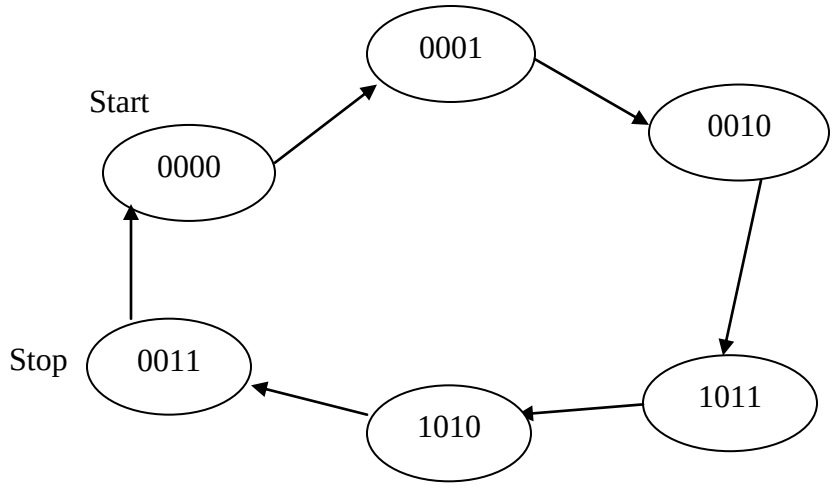


Figure 4.5: Jump state diagram

4.6.7 Choice of logic family

Factors considered in choosing a type of a digital family to be used in the circuit design were the cost, availability, switching speed, noise margin, fan out, fan in and the project objective. Switching speed of a digital data processor is fundamental figure of merit since an efficient processor should be able to execute a long program in the most minimal time as possible (Rs data library, 1999). After studying the tables on the appendix A, it was apparent that low schottky subfamily had the most desired characteristics. Market surveys revealed that most low power schottky I.Cs were available. For simulation purposes, then the I.C components used are as given in table 4.10

Table 4.11: Components and simulation purpose

Component	Simulation purpose
74LS367	Tri-state buffers
74LS195	Four bit parallel access shift registers
74LS138	3- to-8 decoder
74LS181	ALU/Function generator
74LS160	synchronous four bit decade counter
74LS159	4-to-16 bit decoder/demultiplexer
74LS08	Quad two input AND gate
74LS04	Hex inverter
74LS00	Quad two input NAND gate
74LS32	Quad two input OR gate

4.7 System Software

4.7.1 Circuit for the control generator.

The circuit was designed on the Electronic Workbench by using the design tools available on the menu. This involved the interconnection of the various parts to form the complete module. A set of four switches were then used for selecting up of the op-codes and inputting data to the system through the input ports. Four probes were then used as indicators of the status of the various registers, the memory unit and output port. The circuit was then tested by running the simulation switch available in the tool bar. All simulation yielded the expected results as indicated by the simulation diagrams available in the appendix B.

4.7.2 Clocking

The clock generator for the processor can be generated using either a 555 timer or a crystal oscillator. In the Electronic Workbench menu bar, one can select the crystal oscillator item among the miscellaneous items. However there are a limited number of models available with no utility for adjustment in order to maximize the frequency properties. This meant that the user is only limited to the highest frequency item available, 80 MHz, and thus for higher frequencies which are more practical of data processors crystals are not suitable.

The 555 timer is a stable and user friendly integrated circuit for both monostable and astable applications. The timer uses transistors, diodes, and resistors. By conveniently adjusting the values of resistances R1 and R2 and capacitor C1 for the timer according to equations 4.1, 4.2 and 4.3, the frequency of operation can be varied (Ghausi, 1996).

$$F = \frac{1}{0.693\ C\ (R1 + 2R2)} \qquad \text{.....} (4.1)$$

$$R2 = \frac{1 - D}{0.693 C F} \quad \text{.....} (4.2)$$

$$and, R1 = 0.5R2$$

where,

$$D = \frac{R1 + R2}{R1 + 2R2} \quad is\ the\ duty\ cycle \quad \text{.....} (4.3)$$

The 555 timer can easily be adjusted to a given frequency by adjusting the values of the resistors R1, R2 and the capacitor C1 i.e. the time constant RC. The circuit diagram of the 555 timer is given in figure 4.6 shown.

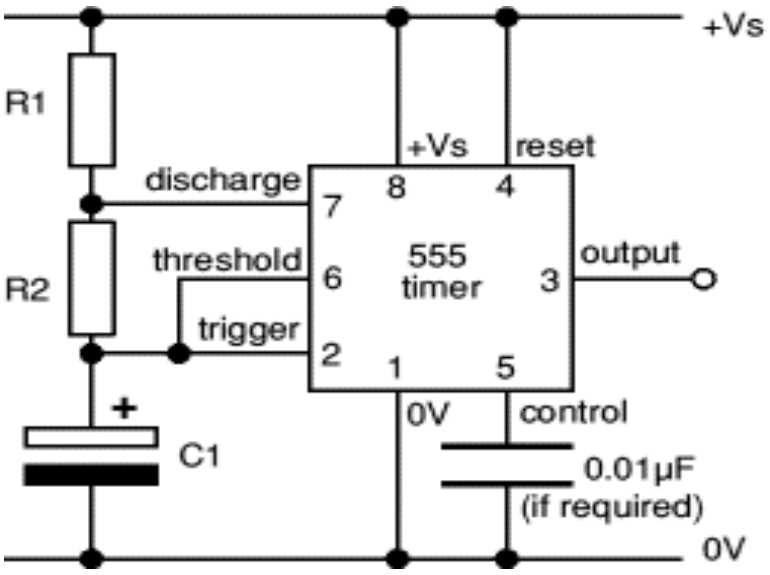


Figure 4.6: Circuit diagram of a 555 timer

Pin 1 This pin is the ground: pin is the most negative supply potential of the device, which is connected to circuit ground

Pin 2 (Trigger): this pin is the input to the lower comparator and is used to set the latch, which

in turn causes the output to go high. Triggering is accomplished by taking the pin from above to below a voltage level of 1/3 V+. The trigger pulse must be of shorter duration than the time interval determined by the external R and C.

Pin 4 (Reset): this pin is used to reset the latch and return the output to a low state. The reset voltage threshold level is 0.7V, and sink current of 0.1mA from this pin is required to reset the device.

Pin 5 (Control Voltage): this pin allows for the direct access of the 2/3 V through the voltage divider point, the reference level for the upper comparator. It also allows indirect access to the low comparator, as there is a 2:1 divider

Pin 6 (threshold): one input to the upper comparator and is used to reset the latch, which causes the output to go low. Resetting via a terminal is accomplished by taking the terminal from below to above a voltage level of 2/3 V. The action of the threshold pin is level sensitive, allowing slow rate- of change waveforms.

Pin 7 (Discharge): the pin is connected to the open collector of an NPN transistor; the emitter of which goes to ground, so that when the transistor is turned “on”, pin 7 is effectively shorted to ground. Usually the timing capacitor is connected between pin 7 and ground and is discharged when the transistor turns “on”.

Pin 8 (V+): also referred to as VCC is the positive supply voltage terminal of the 555 timer IC. Supply voltage operating range for 555 timer is +4.5Vminimum to +16V maximum, and it is specified for operation between +5 and +15V.

4.7.3 Frequency characteristics

The frequency characteristics of a data processor is mainly affected by the paths used by the signals to propagate i.e. propagation delay caused by the component's switching speed and the transmission line effects. Performance is measured in terms of the time it takes to complete a given task. As shown in equation 4.4, the performance depends on many parameters such as the processor itself i.e. the logic family used and system configuration (Hodges and Jackson, 1993; Parzon, 1983).

$$Performance = \frac{1}{Execution\ time}$$

$$= \frac{Ipc \times frequency}{Instruction\ count} \qquad \ldots\ldots\ldots\ldots\ldots (4.4)$$

Where, Ipc is the average number of instruction completed per cycle, frequency is the number of clock cycles per second, and instruction count is the total number of instructions executed. Performance can be improved by increasing IPC and or frequency or by decreasing instruction count. There exists a direct method of improving the performance by simply adjusting the elements of the 555 timer so as to reach a targeted frequency. However, as performance of the processor improves due to this, there reaches a certain optimum frequency beyond which the data output is invalid irrespective of speed enhancements. This was chosen as the frequency of the data processor after carefully analyzing the frequency characteristics for various elements.

CHAPTER FIVE

RESULTS AND DISCUSSION

5.1 Simulation of instruction codes

Electronic Workbench software version 5.12 was used to perform the simulations. This software was because it has good convergence to errors, the operating temperature of the circuit is displayed during simulation to give the optimum temperature during real processing. It also works well in the digital mode and the components chosen for the research project were available. The simulation began by performing functional simulation to test whether the circuit is operational in implementing the instructions. The opcodes were set by means of switches with the ON state representing logic high and an OFF state representing a logic low. Since the opcode was a four bit word length, then four switches were used to select the opcodes. A set of four indicator probes were used to indicate the contents of registers A, B, D and the memory unit. After functional simulation, timing simulation was also carried out in order to minimize the propagation delays and increase the speed of operation of the circuit. The speed of operation was adjusted in the 555 timer using the resistors and the capacitor (Parson, 1983). Different frequencies were compared and listed in tables 5.1 5.2, 5.3, and 5.4. The instructions were simulated as follows.

MOV A, r1: This instruction placed the contents of the input port r1 into register A.

This was the first instruction to be tested. The opcode switches were set to 0001 in order to select the four bit opcode. The four bit input port A was also set to a given value. For example 0100. Then the start switch was put ON and the simulation started using the simulation switch. The probe indicators were connected at the output of register A and it displayed the required results.

DEC A: Decrement contents of register A by one.

The four bit opcode switches were set for this function by selecting code 1111. Simulation results indicated that the contents of register A which was input from the

input port was decremented by one. Register A contents were 0100 and decremented to 0011.

STA: Copy contents of register A in to the memory unit.

This instruction was used to copy the contents of register A into the memory, the opcode for the selection of this instruction is 0110. The probe indicators were now connected at the output of the memory unit flip-flop elements. The output was observed as 0011 which corresponded to the output of register A.

SHR A: Arithmetic shift right of the contents of register A.

This instruction was used to shift right the contents of register A. The four bit opcode is 0100. Originally register A contained 0011. After enabling the instruction the contents of register A were shifted one position to the right and indicator probe indicated 0110 which was the desired result.

CMA: Complement contents of register A.

The opcode switches were set to 0111. This performed 1's complement to the contents of register A. The desired results were obtained by observation of the indicator probes which resulted to 1001; this is one's complement of 0110

This was the required result since the contents of register A were 0110. The indicator probes were in the output of register A.

JNZ: Jump None Zero.

This instruction was used to examine the contents of the status register. If the contents were logic 0 the flow the program was not interrupted. If the contents of the status

register were logic 1, the flow of the program would be transferred to a new location. The opcode was 1110.

LDA: Copy contents of memory unit into register A.

This instruction was used to copy the contents of memory into register A. since the memory had previously been loaded with 0011 with the instruction STA and register A was cleared with instruction CLR A. By setting the opcode switches to 1100, and using the restart switch and running the simulation, the memory contents were successfully copied to register A.

INC A: Increment the contents in register A by one.

The four bit opcode switches were set for this function by selecting code 1001. simulation results indicated that the contents of register A which was input from the memory was incremented by one. Register A contents were 0011 and incremented to 0100.

ADD A: Add the contents of the memory unit to contents of register A and store the result in register A by first copying the contents of storage register to register B.

Since data was available in memory unit and in register A. By selecting the four bit opcode to 1010. The results were observed in the indicator probes in register A.

MOV D, A: This instruction was used to move the contents of register A to the output port via register D. The opcode switches were set to 0011. Before simulation there was no data at the output port by observing the indicator probes. On simulation the contents of register A were observed at the output port.

MOV B,r2: This instruction placed the contents of the input port r2 into register B.

The opcode switches were set 0010 in order to select the four bit opcode. The four bit input B port was also set to a given value. For example 0100. Then the start switch was put ON and the simulation started using the simulation switch. The probe indicators were connected at the output of register B and it yielded the required results.

SHR B: Arithmetic shift right of the contents of register B.

This instruction was used to shift right the contents of register B. The four bit opcode is 0101. Originally register B contained 0100 from instruction MOV B,DATA. After shifting right one position the indicator probe indicated 0110 which was the desired result.

STB: Copy contents of register B in to the memory.

This instruction was used to copy the contents of register B into the memory. The opcode for the selection of this instruction is 1101. The probe indicators were now connected at the output of the memory unit flip-flop elements. The output was observed as 0110 which corresponded to the output of register B.

OR A,B: The contents of the Accumulator is ORed to contents of register B.

It performs bit by bit OR operation on contents of registers A and B. The four bit opcode is 1000. This instruction is performed after STB. The indicator probes were connected at the output of register A.

ADD A,B: Adds contents of register A to the contents of register B and stores the result

in register A. This is accomplished by four bit opcode 1011 and connecting the indicator probes in register A.

HLT: This instruction was used to indicate that the processor is not performing any operation. Accomplished by setting the opcode switches to 0000.

The same procedure was repeated with different data combination at the input port and each yielded the required results. Simulation for the circuit of addition of four bit numbers were also simulated.

5.2 Subtraction.

Subtraction was carried out as follows: take the two's complement of the subtrahend (Mano and Kime, 1997). Add the 2's complement to the minuend. An overflow indicates that the answer is positive and ignore the overflow. No overflow indicates that the answer is negative; take the 2's complement of the original addition to obtain the true magnitude answer.

For example; $1100_2 - 0100_2 = 1000_2$

1's complement of $0100_2 = 1011_2$

2's complement of $0100_2 = 1100_2$

$1100_2 + 1100_2 = 11000_2$, ignore the end around carry

So, $1100_2 - 0100_2 = 1000_2$

0101 – 1001 since the subtrahend is bigger than the minuend

2's complement of 1001 = 0111

0101 + 0111 = 1100 which is the 2's complement of negative four. The circuit yielded the required results. As shown in table 5.3 and table 5.4, the simulation of the subtraction

instructions took more time compared to the addition instructions because of using 2's complement to convert them to addition problem.

Table 5.1: Addition simulation results

$R_1(\Omega)$	$R_2(\Omega)$	Frequency (**MHz**)	Data validity	Simulation Time (**µS**)
15	30	192	valid	0.315
20	43	140	valid	0.380
24	48	120	valid	0.410
27	57	101	valid	0.466
36	72	80	valid	0.540
39	78	77	valid	0.544

Table 5.2: Comparison for addition simulation results

R_1(**kΩ**)	R_2(**kΩ**)	Frequency (**kHz**)	Data validity	Simulation Time (**µS**)
15	30	192	valid	74
20	43	140	valid	112
24	48	120	valid	119
27	57	101	valid	162
36	72	80	valid	227
39	78	77	valid	230

The frequency of operation of the circuit was set at 80 MHz because the data validity for the subtraction circuit was good at that frequency, although the addition circuit could operate at higher frequencies. Frequency of operation was determined by the values of resistors R_1 and R_2. Comparison of the simulation times was obtained in table 5.2 where the values of the

resistors were lower compared to table 5.1. Tables 5.3 and 5.4, the frequency is maintained constant, the resistance is varied.

Table 5.3: Subtraction simulation results

$R_1(\Omega)$	$R_2(\Omega)$	Frequency (**MHz**)	Data validity	Simulation Time (**µS**)
15	30	192	invalid	0.440
20	43	140	invalid	0.550
24	48	120	valid	0.578
27	57	101	valid	0.650
36	72	80	valid	0.760
39	78	77	valid	0.778

Table 5.4: Comparison for subtraction simulation results

R_1(**kΩ**)	R_2(**kΩ**)	Frequency (**kHz**)	Data validity	Simulation Time (**µS**)
15	30	192	valid	148
20	43	140	valid	156
24	48	120	valid	165
27	57	101	valid	238
36	72	80	valid	311
39	78	77	valid	336

The simulation time needed for each instruction depended on the time required to execute the instructions as given in table 5.5.

Table 5.5: Mnemonics execution and timing

		Clock Frequency and Execution Time

MNEMONIC	4-BIT OP-CODE	80MHz	80kHz
		Execution time (µs)	Execution time (µs)
HLT	0000	Initialize	initialize
MOV A,r1	0001	0.074	4.0
MOV B,r2	0010	0.090	4.5
MOV D,A	0011	0.090	18.3
SHR A	0100	0.060	6.0
SHR B	0101	0.090	6.2
STA	0110	0.076	8.0
CMA	0111	0.097	9.9
OR A,B	1000	0.010	7.3
INC A	1001	0.079	6.9
ADD A	1010	0.075	8.9
ADD A,B	1011	0.095	11.0
LDA	1100	0.090	10.4
STB	1101	0.080	8.2
JNZ	1110	0.050	6.0
DCA	1111	0.090	9.0

Other than subtraction and addition, the mnemonics in table 5.5 can also be used to implement multiplication and division hardware algorithm.

5.3 Multiplication algorithm

Multiplication is an operation that can be implemented by three methods namely: Booth's algebra, Add and Shift method and successive addition (Peatman, 1997). Add and Shift method is easily implemented in both hardware and computer software program and it is faster in computation. Add and Shift method is implemented by first clearing the accumulator and loading the multiplier. If the lower significant bit is equal to logic 0, right shift the accumulator, but if the LSB is equal to logic 1, add the multiplicand to the accumulator and right shift the accumulator. The operation is repeated until all the multiplier bits are shifted. In a four bit processor, the highest number that can be accommodated by the registers is 1111_2 which the product of 0101_2 and 0011_2. Based on this the multiplication circuit was designed to be able to accommodate multiplication by 0000_2, 0001_2, 0010_2 and 0011_2. A multiplier of two limited the highest multiplicand to 0111_2 and a multiplier of 0011_2 limited the highest multiplicand to 0101_2.

5.3.1 Design procedure for multiplication

It was necessary to implement the multiplication by use of three instruction registers. In Table 5.6, port B input switches were used for inputting of the multiplier values and port A for the multiplicand. During the multiplication by three, instruction register C is enabled, instruction register B is enabled during the multiplication by two. When the multiplication by zero is performed all the instruction registers are disabled. Three input OR gate prior to each input of instruction register is required since the instruction decoder was to receive its inputs from either instruction register A, B or C. The following instructions were implemented as follows,

Table 5.6: Multiplication code

MNEMONIC	CODE

HLT	0000
MOV A, r1	0001
STA	0110
ADD A	1011
MOV B, r2	0010
ADD A, B	1011
MOV D, A	0011

5.4 Division algorithm

Division is performed by making trial subtraction of the divisor from the partial dividend, If the divisor can be subtracted from the partial dividend without a negative difference, perform the subtraction and enter logic '1' in the quotient and continue with the division process if possible. If the subtraction gives a negative difference enter logic '0' in the quotient and do not alter the partial dividend. Shift the divisor one bit to the right and start again the division process by making trial subtraction. If the sign of the divisor and dividend is the same, the quotient is positive, if the sign of the divisor and the dividend are different, the quotient is negative.

CHAPTER SIX

CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

The research project aimed at designing, simulating and carrying out functional and timing evaluations of a finite state machine suitable for micro-code execution. The main contribution to this work is the introduction of a different approach to design a CPU using readily available components. This approach involves using small and large scale integration components. A simulation of a simple 4-bit processor operating at 80MHz was achieved and using sixteen instructions was achieved, although higher frequencies could be attained for the addition circuitry. The Arithmetic logic unit, control unit and registers which are the major components of a digital processor were designed. It was able to implement sixteen instructions using medium scale and small scale transistor transistor logic integrated circuits. A state machine control unit was used for sequencing of the instructions. The clock pulses were supplied by a 555 timer instead of a quartz crystal in order to make it possible to vary the frequency to a maximum as shown in tables 5.1, 5.2, 5.3, 5.4 and 5.5 of the variation of frequency of a 555 timer against the resistance. The circuit provided an insight of how registers, memory units timing modules and other logic components would be used to realize a demonstration model of a processor unit.

The need for microprocessor systems in conducting laboratory experiments is elaborately exemplified and use of software design tools. Microprocessor technique adopted conforms to the current global trends where it has turned out that almost all disciplines are shifting towards automation. Instructions in all instruction types were implemented including the groups of data transfer, arithmetic, logical and rotate and bit manipulation.

The digital processor main application is in university and college laboratories for experimental work to aid in teaching computer architecture and design. The objectives of the research project were realized.

6.2 Recommendations.

This study has made some meaningful contributions by giving a different approach to the design and simulation of a CPU using medium and small scale integration circuits, however there is still some considerations that could potentially be improved. Further work can be carried out in order to improve the processor to handle large amounts of data by using 8-bit and 16-bit registers and also include status register. Memory modules would have to be included in the circuit i.e. Random Access Memory (RAM). Five or six bit length opcodes will increase the number of instructions to be executed from sixteen, especially instructions to execute interrupts and error checking and correction. Design software to include LSI and VLSI in order to design most of the instructions in the modern computers. These shortcomings are overcome by using Hardware descriptive languages: Verilog Hardware Descriptive Language and VHDL offer more design to model electronic systems.

Simulation of other instructions in the same groups or in other instruction groups to include branch instructions and interrupt service routines.

REFERENCES

Ahson, S.I. (1998). *Microprocessor in process control, computer and Electrical Engineering*. D.P. Publications ltd, London, Great Britain. Pp1-23.

Baher, H. (1986). *Analogue and digital processing*. John Wiley and sons, New

York, USA. Pp15-53.

Breuer, M.A. (1997). *Diagnosis and reliable design of digital systems (2nd Ed).* Nelson, Great Britain. Pp100-169.

Brown,S. and Vranesic, Z. (2003). F*undamentals of Digital Logic with Verilog Design.* McGraw-Hill, New York, USA. Pp54-836.

Chand, S. and Kuthe, A. (2004). *Computer graphics including CAD, AutoCAD and C* (3rd Ed). Rajedra Ravindra publishers, New Delhi, India. Pp12-189.

Douglas, H. (1994). *Microprocessor and Digital Systems*. Chapman and Hall, Great Britain. Pp20-90.

Downton, A.C. (1994). *Computers and microprocessors components and systems*. Van Nostrand Reinhold, Berkshire, England. Pp102-190.

Eincheberger M. and Williams G. (1992). *A Logic design structure for LSI testability*. Van Norstrand Reinhold, New York. USA. Pp5-11.

Eccles W.J. (1985). *Microprocessor systems A 16-bit approach*. Addison– Wesley publishing company inc. California, USA. Pp211-312.

Garratt, J. (1995). *Design and technology*. Cambridge university press, Edinburg building, Cambridge, UK. Pp156-220.

Ghausi, M.S. (1996). *Electronics devices and circuits, discrete and integrated*. Holt, New York, USA. Pp222-287.

Green, D.C. (1985). *Digital techniques and systems.* PITMAN publishing ltd. London, U.K. Pp20-112.

Hamacher, V.C. Vranesic Z.G. and Zaki S.G. (2002). *Computer Organization (5th Ed).* McGraw-Hill, New York. Pp200-312.

Hodges, D.A. and Jackson, H.G. (1993). *Analysis and design of digital integrated circuits.* McGraw-Hill book company, New York, USA. Pp6-40.

Leventhal, L.A. (1978). *Introduction to microprocessors software and hardware programming*. Prentice-Hall international inc New York, USA. Pp61-170.

Mano, M.M. (2005). *Computer System Architecture* (3rd Ed). Prentice-Hall of India Private Limited. Pp100-240.

Mano, M.M. and Kime, C.R., (1997). *Logic and Computer Design Fundamentals*. Prentice-Hall, New Jersey, USA. Pp50-90.

Malvino, P.A. (1980). *Digital computer electronics, Introduction to Microcomputers.* White Hall Books Publishers, Arkansas, USA. Pp34-45.

Mathur, A.P. (1994). *Introduction to microprocessors 3rd edition*. McGraw-Hill, USA. Pp81-210.

McDermott, R. (1983). *The design of an advanced logic simulation byte.* The small systems journal, Vol.8, No. 4. Pp 398-438

Milne, J.S. and Fraser C.J. (1990). *Microcomputer applications in measurement systems*. Macmillan Education ltd. Oxford, UK. Pp100-130.

Mittle, U.N. (2004). *Basic Electrical Engineering.* Tata McGraw-Hill publishers, New Delhi. Pp140-245.

Navtei, N. (2005). Cadence design systems.
Website: http://*www.cadence.com/*

Osborne, A. (1987). *An introduction to microcomputers, Vol 1*. McGraw-Hill, London, Great Britain. Pp140-200.

Parzon, B. (1983). *Design of crystal and other harmonic oscillators*. Wiley, New York, USA. Pp70-190.

Peatman, J.B. (1997). *Microcomputer based design*. McGraw-Hill, New York, USA. Pp243-289.

PC instruments Datasheet (1999). *PC instruments limited.* Texas instrumenents inc., Dallas, Texas, USA. Pp139-300.

Qualey, J. (December, 15th,2006). Galactic electronics.
Website: http://*www.galacticelectronics.com/*

Ravichandran, D. (2003). *Programming with C++ (2nd Ed).* Tata McGraw- HIll Publishing Company Limited, New Delhi, India. Pp40-400.

Raghuram, R. (1989). *Computer simulation of electronic circuits*. Wiley Eastern ltd, New Delhi, India. Pp140-248.

Rs data library. (1999). *Rs components ltd*. Corby, Northamptonshire, UK Pp32-78

Sternheim, E. Singh, R. Trivedi, T. Madhaven R. and Stapleton, W., (1993). *Digital Design and Synthesis with Verilog HDL*. Automata publishing
Co.,Cupertino, New Delhi, India. Pp32-78.

Taub, H. and Schilling, D. (1997). *Digital integrated electronics*. McGraw-Hill book company, New York, USA. Pp82-166.

Texas instruments. (1980). *The TTL data book for design engineers, 2nd edition*.New York, USA. Pp120-331

Thomas, D.E. and Philip, R.M. (1994). *The Verilog Hardware Description Language (2nd Ed).* Kluwer Academic Publishers, Norwell, USA. Pp20-31.

Tzafestas S.G. (1973). *Microprocessor in signal processing.* Prentice Hall Incl., New Jersey, USA. Pp32-67.

Ufenbeck J. (1985). *Microcomputers and Microprocessors*. Prentice Hall Inc., New Jersey, USA. Pp45-101.

Zeigher, B.P. (1976). *Theory of modeling and simulation*. Prentice Hall Inc., upper saddle river, New Jersey, USA. Pp145-201.

APPENDIX A

74138 (3-to-8 Dec)

Table A.1 3-to-8 decoder/demultiplexer truth table

$\overline{G_L}$	G1	$\overline{G_2}$	Select C	Select B	Select A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	X	1	X	X	X	1	1	1	1	1	1	1	1
X	0	X	X	X	X	1	1	1	1	1	1	1	1
0	1	0	0	0	0	0	1	1	1	1	1	1	1
0	1	0	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	1	0	1	1	0	1	1	1	1	1
0	1	0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	1	0	0	1	1	1	1	0	1	1	1
0	1	0	1	0	1	1	1	1	1	1	0	1	1
0	1	0	1	1	0	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	1	1	1	1	1	0
1	1	0	X	X	X	Output corresponding to stored address 0; all others 1							

74159 (4-to-16 Dec/DEMUX (LS-OC))

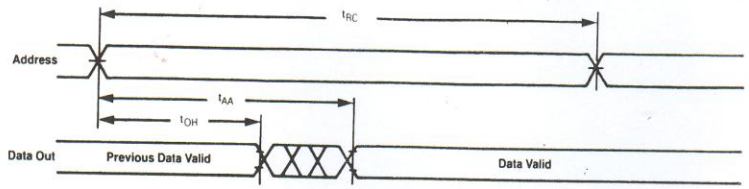
For this IC, we recommend the LS-OC model for realistic performance.

Table A.2 Decoder/Demultiplexer truth table

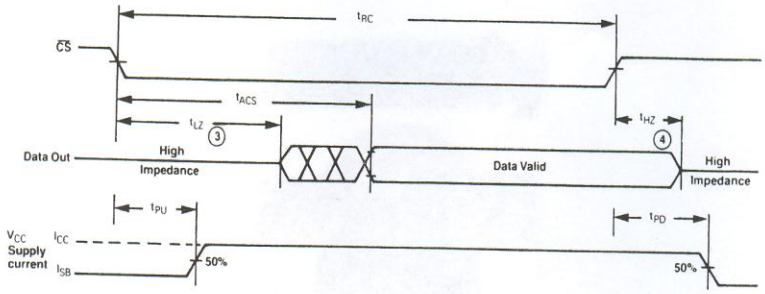
_	_	Inputs				Outputs													
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Timing Waveforms

Read Cycle 1 ⑤⑥



Read Cycle 2 ⑤⑦

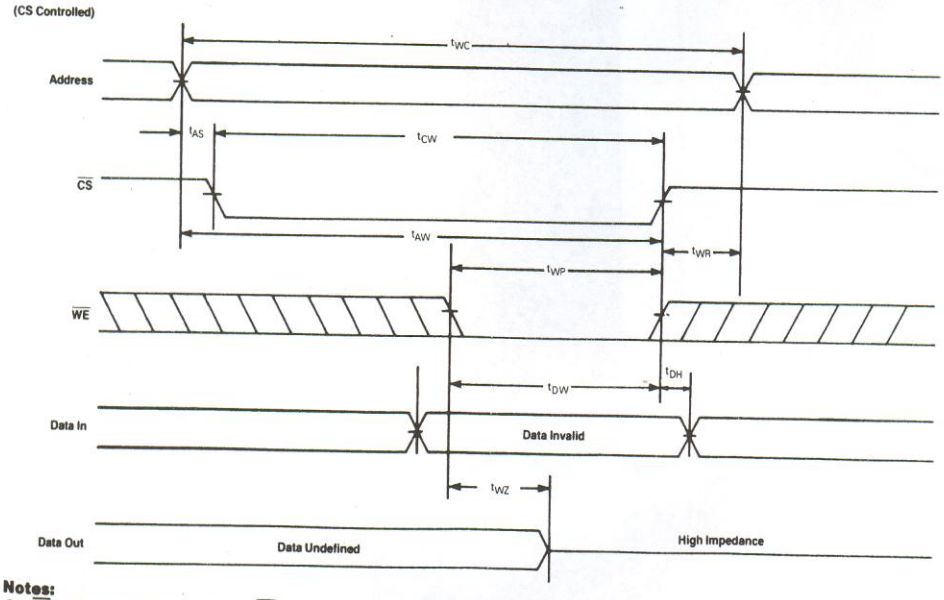


Notes:

① All Read Cycle timings are referenced from the last valid address to the first address in transition.
② At any given temperature and voltage condition, t_{HZ} max is less than t_{LZ} min both for a given device and from device to device.
③ Transition is measured +200mV from steady state voltage with specified loading in Figure 2.
④ Transition is measured at V_{OL} +200mV and V_{OH} −200mV with specified loading in Figure 2.
⑤ $\overline{WE}$ is high for Read Cycles.
⑥ Device is continuously selected, $\overline{CS} = V_{IL}$.
⑦ Addresses valid prior to or coincident with $\overline{CS}$ transition low.

Write Cycle

(CS Controlled)



Notes:

① If $\overline{CS}$ goes high simultaneously with $\overline{WE}$ high, the output remains in a high impedance state.
② All Write Cycle timings are referenced from the last valid address to the first transitioning address.
③ Transition is measured at V_{OL} +200 mV and V_{OH} −200 mV with specified loading in Figure 2.
④ Transition is measured ±200 mV from steady state voltage with specified loading in Figure 2.

Figure A.1 Read/Write timing waveforms