MODULAR CMOS ALU CONTROL UNIT

by

PAVAN R. MULA, B.E.

A THESIS

IN

ELECTRICAL ENGINEERING

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

Approved

Accepted

August, 1999

ACKNOWLEDGEMENTS

I would like to express heartfelt gratitude towards, Dr. Micheal Parten, advisor and chairperson for this thesis, for his most valuable guidance and colossal support through the entire course of this work. The project done under his guidance has enriched me with priceless experience, which I will cherish forever. I am also greatly indebted to Dr. Michael G. Giesselmann and Dr. Donald J. Bagert Jr., thesis committee members, for their excellent cooperation.

A special thanks to the Department of Electrical Engineering for giving me this valuable opportunity to study at Texas Tech University. I wish to extend my deepest gratitude to my parents Mr. Ram Reddy and Mrs. Shakuntala, and my sister, Ms. Kavitha for their love, encouragement and blessings throughout my life. I am extremely thankful to all of my friends for their support, encouragement and well wishes.

TABLE OF CONTENTS

ABSTRACT

Modular based design is used for flexibility, simplicity and to lower the cost and labor in designing general purpose or application specific digital circuits. This research investigates the nature and advantages of digital building blocks by implementing a modular based design, 8-bit ALU control unit. The 8-bit ALU control unit is designed using Logic Works 3.0.3, laid out using L-Edit (Version 7) and simulated in PSpice. This paper contains the design method, layout considerations and simulation results. It also discusses history, advantages and problems with digital building blocks.

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 A brief history of CMOS circuits

Over the past decade, Complementary Metal Oxide Semiconductor (CMOS) technology has played an increasingly important role in the global integrated circuit industry. In 1925, J.Lilienfeld proposed the basic principle behind the MOS field-effect transistor. Experiments with early field-effect transistors led to invention of the bipolar transistor. The success of the later device led to a decline of interest in the MOS transistor. MOS devices remained an oddity until the invention of the silicon planar process around 1960. The use of both polarity devices on the same substrate was invented in the early 1960s. The initial circuits were developed using discrete MOS transistors and demonstrated what was for many years the hallmark of CMOS--low power dissipation. This low power attribute led CMOS to be initially used for very low power applications, such as watches. Since the processing technology required in the fabrication of CMOS circuits was more complex and the required silicon area was significantly larger than that for single polarity transistors, CMOS was applied sparingly to general system designs. As nMOS production process became more complicated, the additional complexity of the basic CMOS process decreased in importance. Through continued improvements, CMOS has become the technology of choice for a broad range of semiconductor products because of the advantages that CMOS provides: an exceptionally low power-delay product, the ability to accommodate millions of devices on a single chip, and flexible,

custom design methodologies which permit optimization, as required, for lowest cost, lowest power, or highest speed [2].

1.2 Evolution of integrated circuits

The field of semiconductor based design has grown over the last few decades as efficient design methods and devices have been developed. Each of the last five decades has seen a new technology come forward as the leading edge of that era. The first integrated circuit was developed in the 1950s by Jack Kilby of Texas Instruments and Robert Noyce of Fairchild Semiconductor.

In the 1950's, the building blocks were gate level discrete devices. A few transistors and other components combined to form AND, OR, or NOR gates. Integrated circuits are often classified by number of transistors or other electronic components they contain as shown in table 1.1 [3].

Table 1.1: Number of components per chip in different logics

Logic	**# of components per chip**
SSI (small-scale integration)	Up to 100
MSI (medium-scale integration)	From 100 to 3,000
LSI (large-scale integration)	From 3,000 to 100,000
VLSI(very large-scale integration)	From 100,000 to 1,000,000
ULSI(ultra large-scale integration)	More than 1 million

In the mid-1960's, SSI (Small-Scale Integration) logic was implemented. A few gates were lumped together as a means of improving the design. In the early 1970's MSI (Medium Scale Integration) logic was implemented where up to 3,000 transistors were packed together in a single chip for the same reason. MSI allowed for more modular designs, speeding up the process where the blocks would be applied. In the late 1970's,

LSI (Large-Scale Integration) logic was implemented, where several thousand transistors were placed on the same chip. Arithmetic Logic Units (ALUs) with on-board registers, microprogrammable sequencers and interrupt controllers in a bit-slice format became available. Memory chips (ROM, PROM, RAM) in increasing sizes became readily available. LSI culminated in the one-chip microprocessor. In the early 1980's, VLSI (Very Large-Scale Integration) and VHLSI (Very High-Speed Large-Scale Integration) logics were implemented. Up to 1,000,000 transistors were used to build large functional blocks and their related circuitry in low power, faster chips. CPUs (Central Processing Units) and complex functional blocks became available. Now more than one million transistors may be used on a single chip using ULSI (Ultra Large-Scale Integration) techniques [9].

LSI and VLSI further increase the modular block size, reducing design time, space, and power considerations and increasing reliability. Many LSI and VLSI blocks are designed by their manufacturers and referred to as FIS (Fixed Instruction Set) modules. In the late 1980's, up to 30,000 gates were used for ASIC (Application Specific Integrated Circuit) designs. In the early 1990's, up to 100,000 gates and with increasing speed up to 1.4GHz were used for ASIC designs [9].

1.3 Design Methods

Today designers are faced with the following basic choices of design methods[9]:

- Hard wired logic.
- Fixed Instruction Set microprocessors.
- Microprogrammable or Bit-Slice architecture.

- ASIC design.

For any given design, if the architecture of fixed LSI and VLSI blocks suits the application, then the design time is considerably shortened. When a one-chip microprocessor is not quite suitable, microprogrammable architecture can often provide sufficient customization. Microprogrammable architectures such as bit-slice architectures, allow closer control over the architecture, but not total control. Bit-slice architecture includes interruptible sequences and ALUs. The customization of bit-slice modules to an application is done through customer design module interconnection, the implemented commands and their sequences. The commands in the instruction set are called the microprogram for the design.

ASICs (Application Specific Integrated Circuits) are VLSI devices that are large enough to allow designers to implement architectures that are suited for solving the design problem rather than focusing on one architecture to solve every thing. This is a natural extension of bit-slice architecture, where some control of the architecture is possible through microprogramming but where the basic building blocks are of fixed design. These are designed specifically for mass application, if the market for that device is large. A chip for a toy bear that talks, a chip for satellite, a chip designed to handle the interface between memory and a microprocessor for a workstation CPU are examples of ICs that are ASICs [3].

Design tradeoffs are summarized in Table 1.2. A number of factors influence the decision as to which design method is best for the application. These factors can be categorized as architecture, size, word length, instruction set and speed. Basically where high speed, long word lengths or critical instruction sets occur, MOS FIS cannot be used.

If design time, part count or board space restrictions also exist, or if production volumes do not support the effort required to do an SSI/MSI design, (considered the most difficult to do correctly in a given time frame) bit-slice devices are the best choice.

Table 1.2: Design Tradeoffs [9]

	SSI/MSI	**Bit-Slice Devices**	**FIS MOS Microprocessor**
Architecture	Any desired	Pseudo-flexible	Pre-designed
Size(typical)	500 chips	50	3-6
Word length	Any desired	Multiples of 2,4	4,8,16,32,64
Instruction set	Any desired (May be hardwired)	Any desired (may be microprogrammed)	Constrained if speed is a problem
Operating speed	Relatively faster	Relatively faster	Relatively slower
Design time	Long, slow	Fast	Fast
Debug	Difficult	Development systems aid process	Development systems aid process
Documentation	tedious	Forced via microprogram	Software is a major portion
Upgrades	Up to full redesign may be required	Easily done, can be pre-planned upgrade	Easily done (software)
Cost	Highest	Medium	Lowest

1.4 Present trends in chip and microprocessor design

The late nineties have seen new trends in digital design, Designers began constructing chips with greater speed and high computational power for a wide variety of applications. Consequently SSI/MSI or bit-slice devices are seldom used, but microprocessors have become very fast with higher computational capabilities. As a result complexity, design time and cost have also increased. So designers have been concerned with several constraints before making design decisions. Based on the need, designers choose application specific or general purpose processors. Both of these devices have their own advantages and disadvantages. General processors are common processors like Intel's

80X86 microprocessor. They are designed for general computation applications. General purpose computers achieve flexibility and generality by sacrificing some areas of performance. In contrast, Application Specific Processors are optimized for their intended application, often achieving orders of magnitude improvement in performance. The main advantage with application specific processors is that they do not waste processing power on unnecessary capabilities. On the other hand the main disadvantage is its high cost because of its unique design made for a specific application. The main advantage of general purpose processors are the low cost, because the cost is amortized over a potentially long production run. In addition the risk of fundamental defects in the design is reduced. However, they generally offer inadequate throughput for many problems [7].

Application Specific Programmable Processor (ASPP) is a hybrid of application specific and general purpose processors. These are designed to overcome to some of the problems of application specific and general purpose processors. These are designed for some applications in specific domains such as digital filtering. ASPPs can implement different functions under different situations. ASPPs can be used as processor cores to speed up the design of complex chips and also open the doors for a new trend called design reuse. Design reuse is using previous designs as building blocks in new designs. This concept makes a strong case for modular design. Also low cost, high speed and low power are the other reasons for using ASPPs.

Most present and future video and multi-media applications are too large to fit on a single chip. Recently, due to advantages in packaging technologies, an efficient solution to this problem is offered by the Multi-Chip Module (MCM) technology. An MCM has several chips bounded to a single substrate. The inter-chip connections are provided at the

substrate level. The major advantages of MCM are electrical performance, reduced size and high reliability due to short interconnect distances and fewer parasitics. Designers can incorporate different technologies (like analog and digital devices) in the same module [7].

1.5 The building blocks for chip design

From the above discussion, it is clear that chip design has grown to a very high level of complexity in an attempt to execute more instructions per cycle. However, complexity often fails to yield the expected throughput and winds up costing a lot in terms of die size, clock cycles and scheduling. Even though many computer aided design tools are in use, they do not incorporate huge module building blocks which are necessary to reduce the time and effort to implement MCMs or ASPPs. Again there is no standard for modular systems on a chip. The Answer to all these problems can be designing fundamental building blocks for digital design. With this, engineers will be provided with some powerful, fast, expandable and flexible building blocks to be tied together to meet the requirements in an ASIC design or in general purpose application. Even though some of the present integrated circuit layout tools have libraries with building blocks, they are frequently primitive gates (AND, OR, XOR, INVERTER, etc.) only. It takes a long time and significant labor to design complicated circuits from these gates. To obtain the advantages of modular based design, tool libraries must have building blocks at the MSI or LSI level [7].

The motivations of large-scale digital design are lower cost, higher speeds, higher reliability, shorter design times and flexibility. Basic building blocks can meet all the

above objectives in most applications. Building blocks can be designed using existing computer aided design tools and are basically LSI blocks where several hundred gates may be used. As complexity is not a big issue in designing building blocks, required design time and cost can be kept low.

Building blocks solve any potential pin-out problem. There is an upper limit of number of pins on the package holding the chip to keep the die size small and the cost low. For example, if the upper limit of the pins on a chip is one hundred then it is not feasible to construct a 32-bit ALU control chip, since such a device will require over one hundred pins. As building blocks are made units of smaller bits and can be cascaded to make higher bit units that do not have any pin-out problem [7].

Basic building blocks provide flexibility in designing complicated systems by solving the "part proliferation" problem of VLSI design. In producing a VLSI device the design cost is extremely high, but the production cost per unit is extremely low. Hence, the economics of VLSI are very attractive when a large number of each type can be used. This leads to a problem. The large number of circuits on a VLSI device could easily imply that such devices are specialized toward a particular system and unusable in other design. For example, suppose that a computer company was developing a family of processors each with distinct cost and performance objectives. If one could design several chips for use in processor A, it is unlikely that they could be used in processor B, because the processors are likely to have dissimilar internal designs (different data path widths, different ALU functions, different degrees of internal parallelism). This puts a restriction on the volume in which a chip can be manufactured.

The road to a solution is the realization that, in VLSI design, one should not be pre-occupied with minimizing the number of elementary components or gates in a system, but rather with minimizing the number of chip types. So, some universal chip types are needed. Rather than containing static functions, the functions of these devices should be capable of being controlled externally (i.e., by logic external to the device). To serve as building blocks in a large number of systems these devices should be capable of performing a large number of functions, many of which might not be used in a particular design. At the same time these universal devices should place few, if any, restrictions on the design in which they might be used (e.g., restrictions on the data path size). With this in mind, the main objective of this research is to investigate the nature, advantages of this kind of digital building block and to implement this idea into an 8-bit ALU control unit.

1.6 Overview

This research aims at investigating the nature and advantages of building blocks (Modular design concept) and implements a modular ALU control unit with an emphasis on micro programming. In achieving this, an ALU control unit was designed (supplying inputs to ALU, setting flags after the result is available at the out put of ALU, and making these available in both straight and complemented form) in logic works, the physical design of the CMOS ALU control unit was laid out using L-Edit (a software package used for layout of physical design of any CMOS circuit) and the results verified in PSpice using extracted file from layout.

Chapter II discusses the microprocessor structure and the microprogramming approach. Chapter III discusses the design procedure of 8-bit ALU control unit. Chapter

IV discusses the layout procedure of that ALU control unit. Chapter V includes the simulation results from PSpice and conclusions are presented in Chapter VI.

CHAPTER II

CONTROL UNIT DESIGN

This chapter deals with the fundamental concepts that govern the design of a computer control unit.

2.1. Microprocessor Structure

The basic generic structure of a typical microprocessor can be considered to be the following [9].

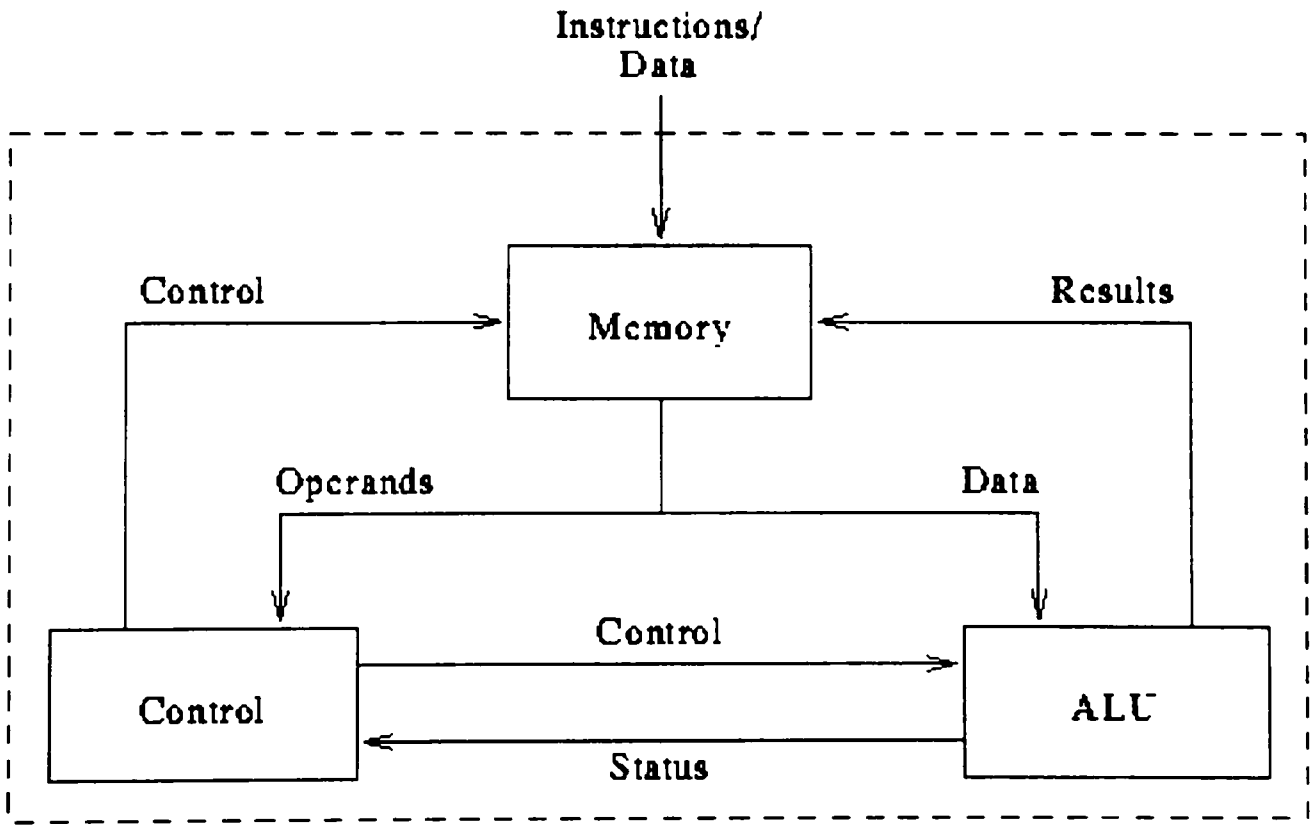


Figure 2.1. Basic microprocessor organization

There are three main areas. The memory area is used for internal storage of data within the microprocessor. The second aspect of interest is the control. Microprocessors work by the use of instructions. The purpose of the control circuitry within a microprocessor is

to take a given instruction, and decode it such that the desired operation is performed on the desired data. There are two basic approaches, hard-wired control, and microprogrammed control.

2.1.1. Hard-Wired control

In a hard-wired control system, the control circuitry has a set of output lines connected directly to each part of the ALU which has some control aspect associated with it. The decoding of the instruction involves generating the appropriate set of outputs on the lines to perform the desired function. Since the number of output lines is fixed, this process can simply be implemented as a combinational circuit, or as a ROM device [9].

2.1.2 Microprogrammed Control

Microprogramming is an approach that implements complex instructions from a basic set of functions. This reduces significantly the amount of circuitry required in the ALU at the expense of speed. Microprogramming works by breaking down each processor instruction in to a sequence of microinstructions which, when executed in sequence, implement the desired instruction. Consider as an example, binary subtraction. It is necessary to break down the operation in to a series of simpler steps. Such steps might be

- Load B in to accumulator (data transfer).
- Negate B (and store result back in accumulator). (Bitwise logic).
- Increment accumulator (addition).
- Add A to accumulator (addition again).

Each of these operations is simple, but the original single instruction has now produced four microinstructions. The control circuitry therefore involves taking in the original instruction and issuing the appropriate sequence of microinstructions. The microinstructions may be stored in some form of ROM device. An alternative form may be to construct a Finite State Machine that issues the appropriate sequence of microinstructions as states, based on the input instruction. In order for a processor to function, it requires a series of instructions to tell it what to do. The next block in consideration is the Arithmetic and Logic Unit. This is the part of the processor, which undertakes the operations on the data. This is the area of a microprocessor where the majority of the work is done. It is a component capable of performing a range of operations on data and producing the desired output [9].

There are number of factors which influences the decision as to which implementation is best for the application. These factors are explained in detail in section 1.3. SSI/MSI hardwired implementation allows the designer to specify the exact detail of the architecture desired. With bit-slice devices, some constraints are placed on the designer, but most of the system architecture is left to user definition via the selected interconnections and microprogram [9].

2.2. Microprogramming Architecture

Microprogramming is to hardware design what structured programming is to software design. If a bipolar Schottky TTL machine is to be built in bit-slice or in SSI/MSI or ASIC, its control should be done in structured microprogramming.

First random sequential logic circuits are replaced by memory (writable control store or ROM [read only memory] or PROM [programmable ROM] or related devices). This results in a more structured organization of the design. Second, changes in the microprogram can facilitate sufficient upgrading. As this is an augmentation in the microprogram code no hardware is disturbed. Third, an initial design can be done such that several variations exist simply by changing the microprogram, and enhanced versions can be preplanned such that version B is constructed simply by adding a new microprogram code or two to version A, simplifying production.

Version B = Version A + more microprogram code.

Machine level instructions are what the computer control unit (CCU) receives. In a microprogrammed machine, each machine level instruction (referred to as macroinstruction) is decoded and a microroutine is addressed which, as it executed, sends the required physical control signals in their proper sequence to the rest of the system. This is where software instruction via a firmware microprogram is converted into hardware activity.

2.3. Microprogram Storage

The various software programs (high-level user programs) will vary from hour to hour, or more often, and the data too will vary; therefore read-write or RAM memory is required as their storage area. The microprogram, however, will usually remain the same. There are a few machines that loaded a different microprogram for each of several specific application languages.

Where one microprogram is to be used by the system, ROMs (high-production) or PROMs (lower production, prototype) are used for the microprogram memory. Such systems are called microprogrammable systems. When a microprogram may be replaced by another, for example, to emulate another machine or to do a diagnostic run, then either a separate read-write memory, called a writable control store (WCS), or part of the system main memory is used as the microprogram memory with area of main memory as the auxiliary storage for the microprogram. Throughout the remainder of this thesis, the microprogram control memory will be assumed to be a PROM memory for the sake of simplicity.

2.4. Instruction Format

Each machine-level instruction has an op-code and operand field. There may be several different formats for the instructions in any one machine. The control unit decodes these instructions, and the decoding produces an address, which is used to access the microprogram memory. The microroutine for the individual machine or macroinstruction is called into execution and may be one or more microinstructions in length. (A microinstruction will be assumed to execute in one microcycle.) Each microinstruction field directs or controls one or more specific hardware elements in the system. Every time that a particular machine instruction occurs, the same microroutine is executed. The particular sequencing of the available microroutines constitutes the execution of a specific program.

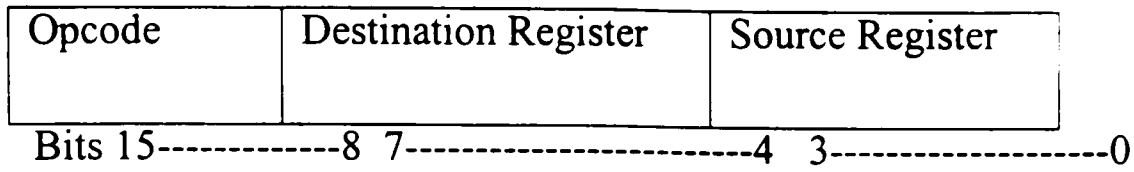


Figure 2.2. Machine Level Instruction

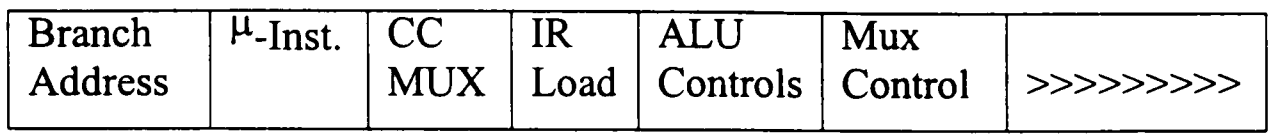


Figure 2.3. Microprogram Instruction

Microprogramming is done in the format or formats designed by the programmer. Once chosen, the format becomes fixed. Each field controls a specific hardware unit or units and the possible bit patterns for each field are determined by the signals required by the hardware units being controlled. Simple, short microprograms can be recorded in bit string fashion and prototype PROMs created using manually operated PROM burners. A sample microinstruction format is shown in Figure 2.3. More than one microinstruction can exist for a given microprogram. When two or more exist, extra bits must be added to the microword and these are assigned the task of signaling the controller as to which format is being decoded. The simplest microprogrammed computer control unit would require an instruction register, decode logic, a clock source and a ROM-PROM based control memory. Output from the control memory would include the control signals for the rest of the system [9].

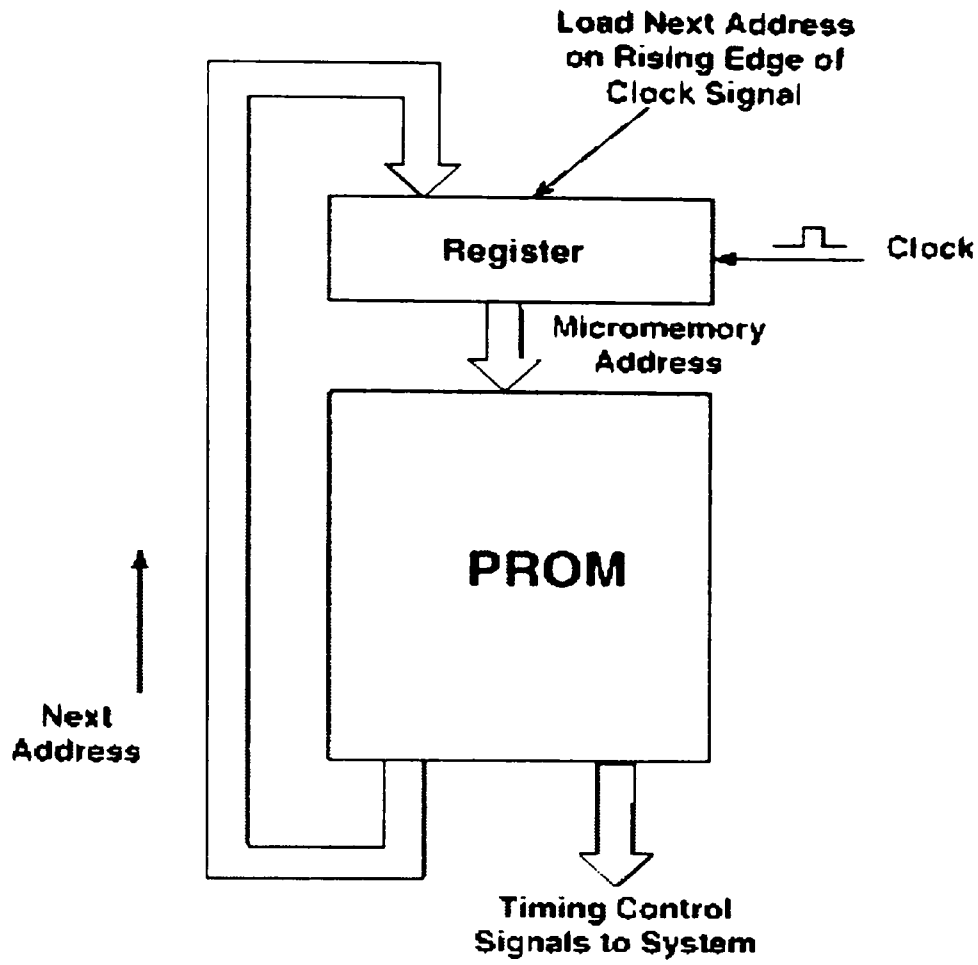


Figure 2.4. Simplest Control Bit implementation [9].

2.5. Implementation of a Sequential Machine

The very simple microprogram controller can be constructed from a PROM and a register as shown in Figure 2.5. The load enable on the register is connected to the clock signal. The register outputs an address to the PROM memory, and this address is used to fetch the next microinstruction that is to be executed. No next address logic is included or required. After a time delay equal to that needed for stabilizing the register outputs plus the read access time of the memory, the memory outputs to the control signals to the rest of the system and the next address to be loaded into the register. The PROM memory is also referred to as the control memory. The output form the memory must be stable before the next clock pulse (Cp).

$$Cp = t_{\text{read access of memory}} + t_{\text{register Cp to output}} + t_{\text{setup time for register}}.$$

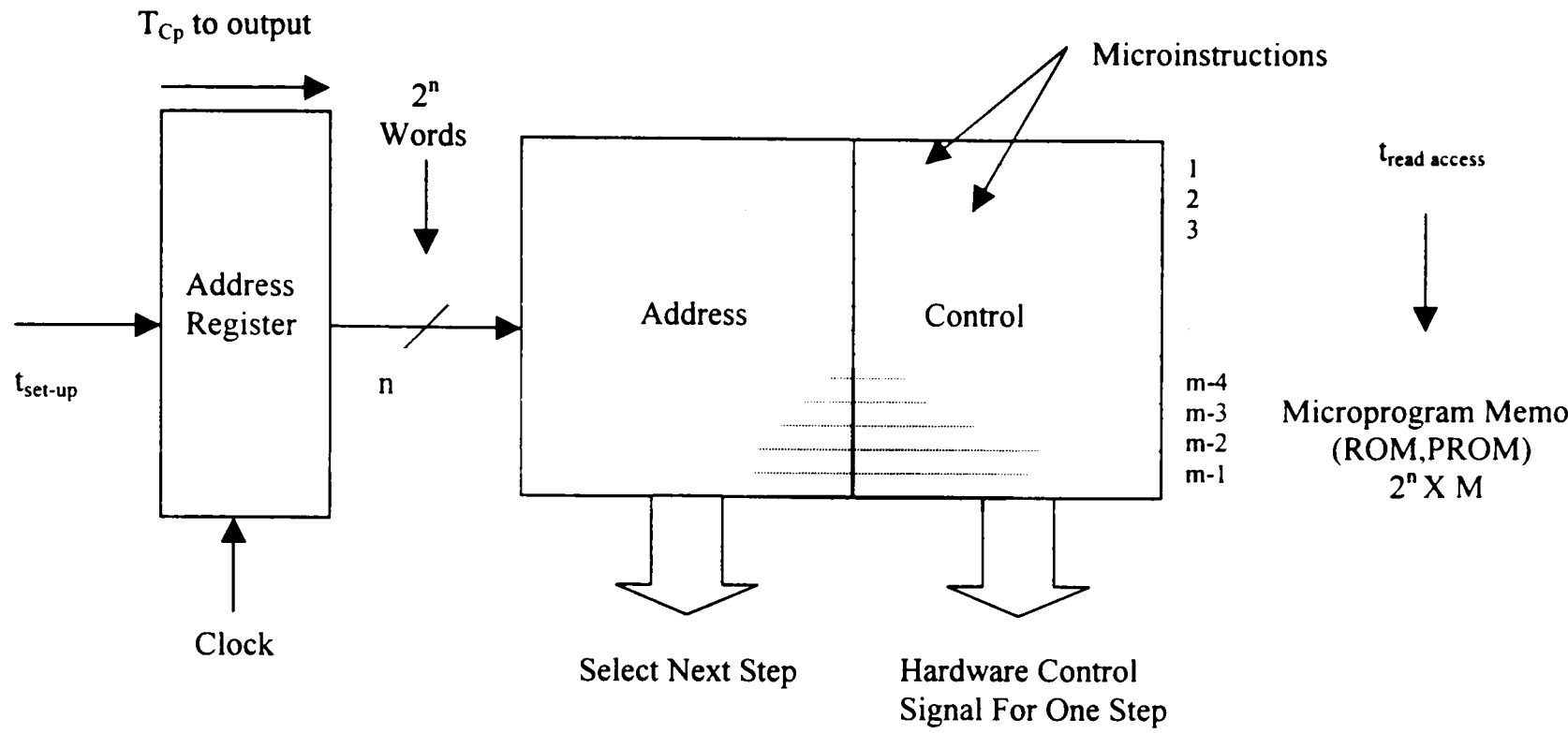


Figure 2.5. The simplest implementation of a sequential machine

The size of the memory is 2^n words, with each word M bits long. The M bits are formed from the C control bit plus the n address bits required specifying the next instruction. M = C + n [9].

Word width equals the number of control bits plus the number of next-address bits. The programmer is free to place a microinstruction in any order as long as each one references the next executable address. This system will run from clock power-up until clock power-down. Assume on power-up that the register is cleared and the first address executed is address O. Since no address logic is provided, only one sequence is possible.

2.6. Sequential Execution

A reduction in required PROM memory is possible by removing the requirements of the next address field. This is reasonable because the microprogram can be loaded into PROM in its executed order as one long sequential routine, as diagramed in Figure 2.6. In this case, the next microinstruction address is always equal to zero on start up [9].

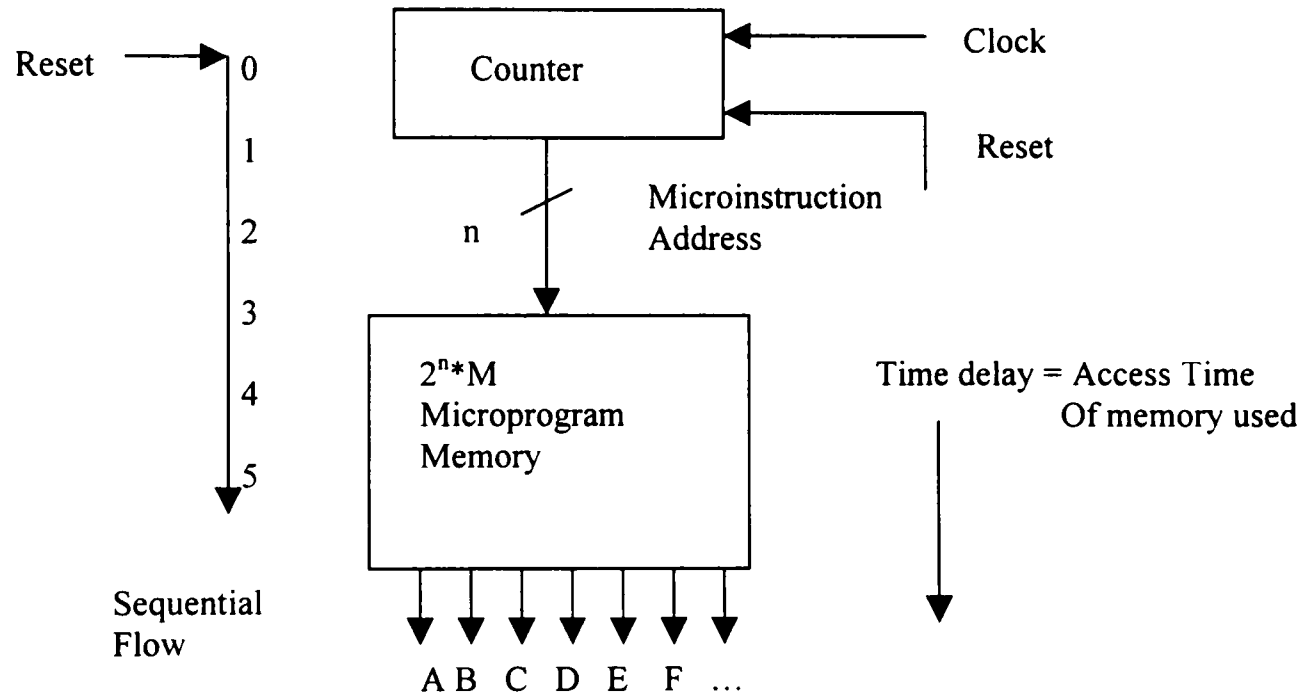


Figure 2.6. Sequential Control

The clock pulse width is determined by:

$Cp = t_{\text{read access}} + t_{\text{counter Cp to output}}$ which is approximately the same as before, since:

$t_{\text{read access}} >> T_{\text{Cp to output}}$

To derive the control portion of the microcode of the two control units described, assume a timing diagram. By digitizing the timing signal using the clock step and assuming all changes correspond to the rising edge of the clock, the microprogram

control field is simply the binary word at each time slice. The procedure is shown in Figure 2.7.

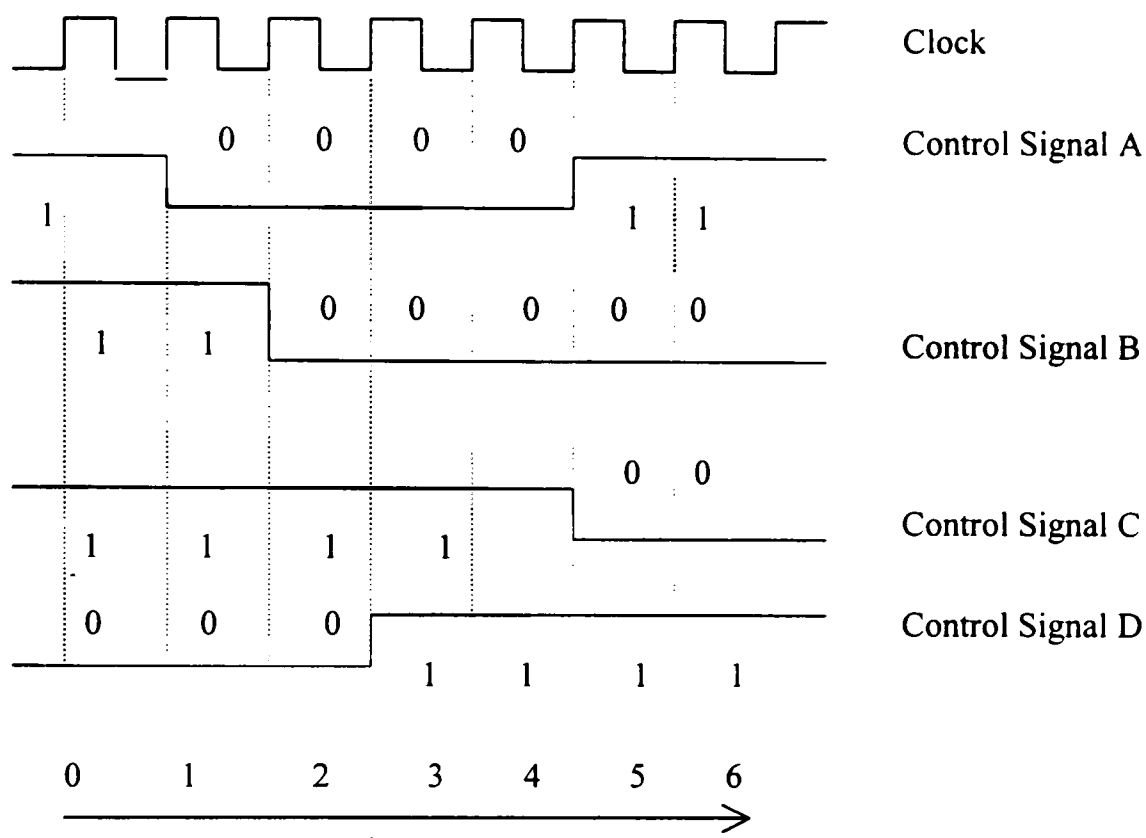
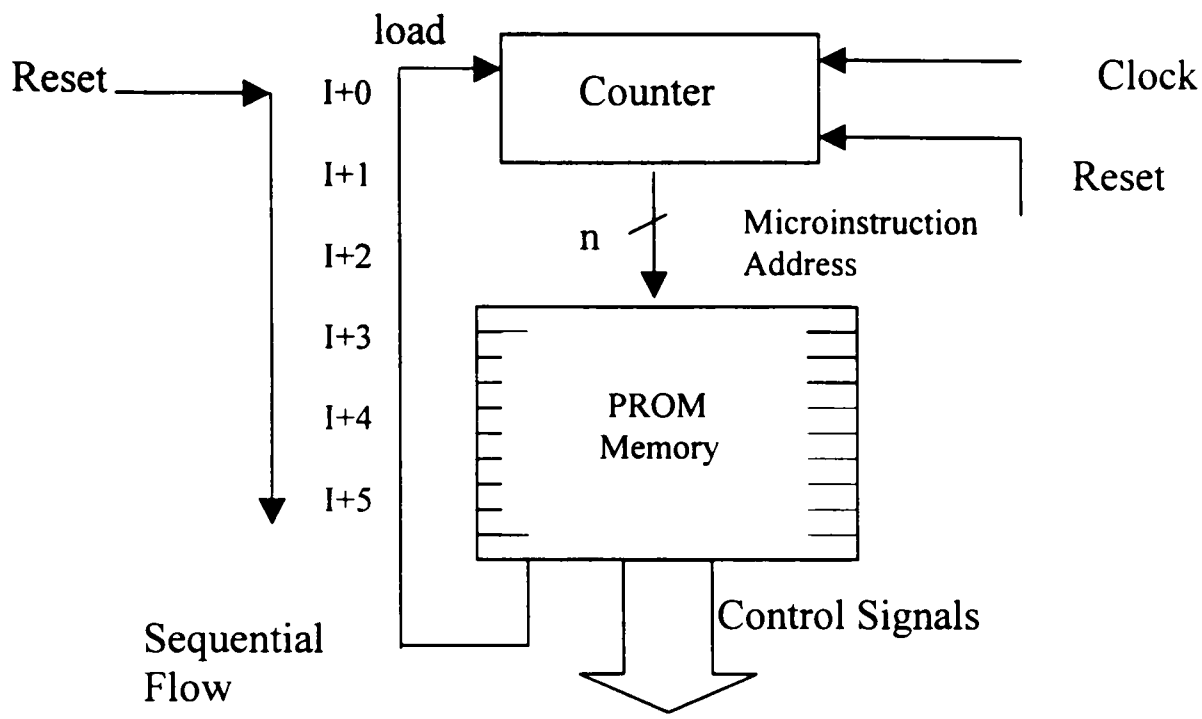


Figure 2.7. Sample sequential microcode

2.7. Multiple Sequences

The controller may be made to execute multiple sequences by adding one control bit to the word width, the load control bit. This bit connects to the load control line of the loadable counter, as shown in Figure 2.8.



(a) block diagram

Load Control Bit	ALU Fn Carry C_{in}	ACC Load Enable	Op code Load	A Address	B Address	>>>>>>>>>>>
1 Bit	4-6 Bits	1 Bit	1Bit	1-4 Bits	1-4 Bits	

(b) Microword Format

Figure 2.8. Multiple Sequence Controller

The data input to the counter receives the start address. The new start address is gated into the counter when the load control bit equals "1." The counter operates as a counter as long as the load control field is zero. A "1" in the load control field in the last microinstruction of the sequence will load the counter.

If the controller is a computer control unit (CCU), the start address of each microroutine is derived from the current machine instruction. At the minimum, an instruction register must be added between the data bus and the counter data inputs to store this instruction. A load control bit must be added to the microword for the

instruction register, which must load prior to the counter load as shown in Figure 2.9. This scheme requires that the opcode equal the high-order bits of the start address, with low-order bits tied to logical "0". This is necessary to allow the starting addresses to be specialized by the minimum number of addresses required by the longest microroutine.

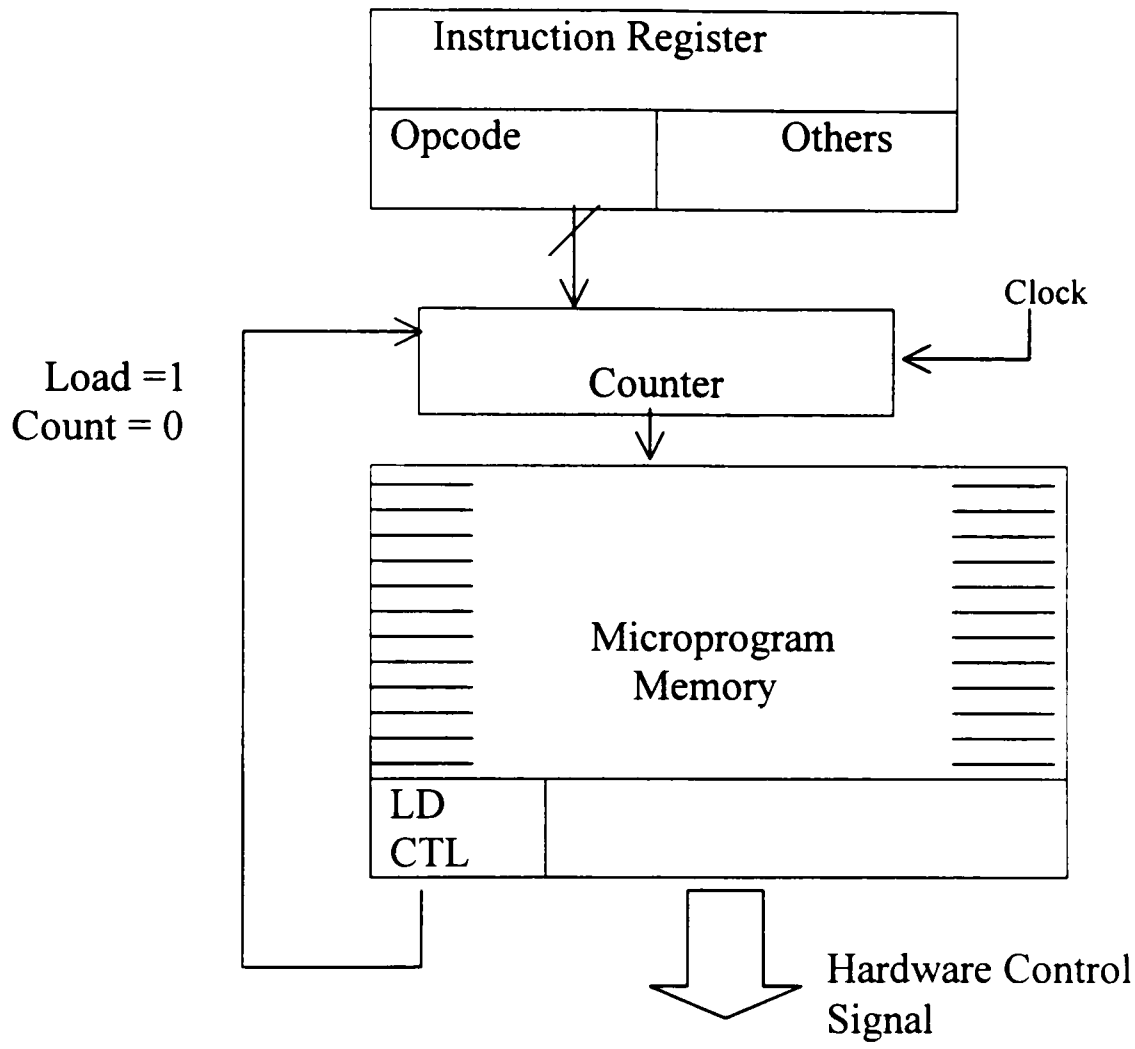


Figure 2.9. Basic CCU

It is to be noted that the clock pulse required by the controller has not changed. and also that the width of the memory is not a function of its depth. This scheme is adequate if

- There is sufficient room in the PROM memory
- Space locations are acceptable
- No microroutine exceeds 16 steps.

If a microword exceeds 16 steps (here it is assumed that each macroinstruction is represented by fixed 16 microinstruction), it would out run the start address, reducing the

number of opcodes possible. Short routines leave discontinuous unused areas scattered throughout the PROM memory (fragmentation), which in this type of design may be acceptable [9].

2.8. Mapping PROM

If fragmented space and reduced numbers of available opcodes are not acceptable, one solution is to add a mapping PROM between the instruction register and the counter. The opcode is the address into the map, which in turn outputs the full start address of the microroutine to the counter, as shown in Figure 2.10 [9].

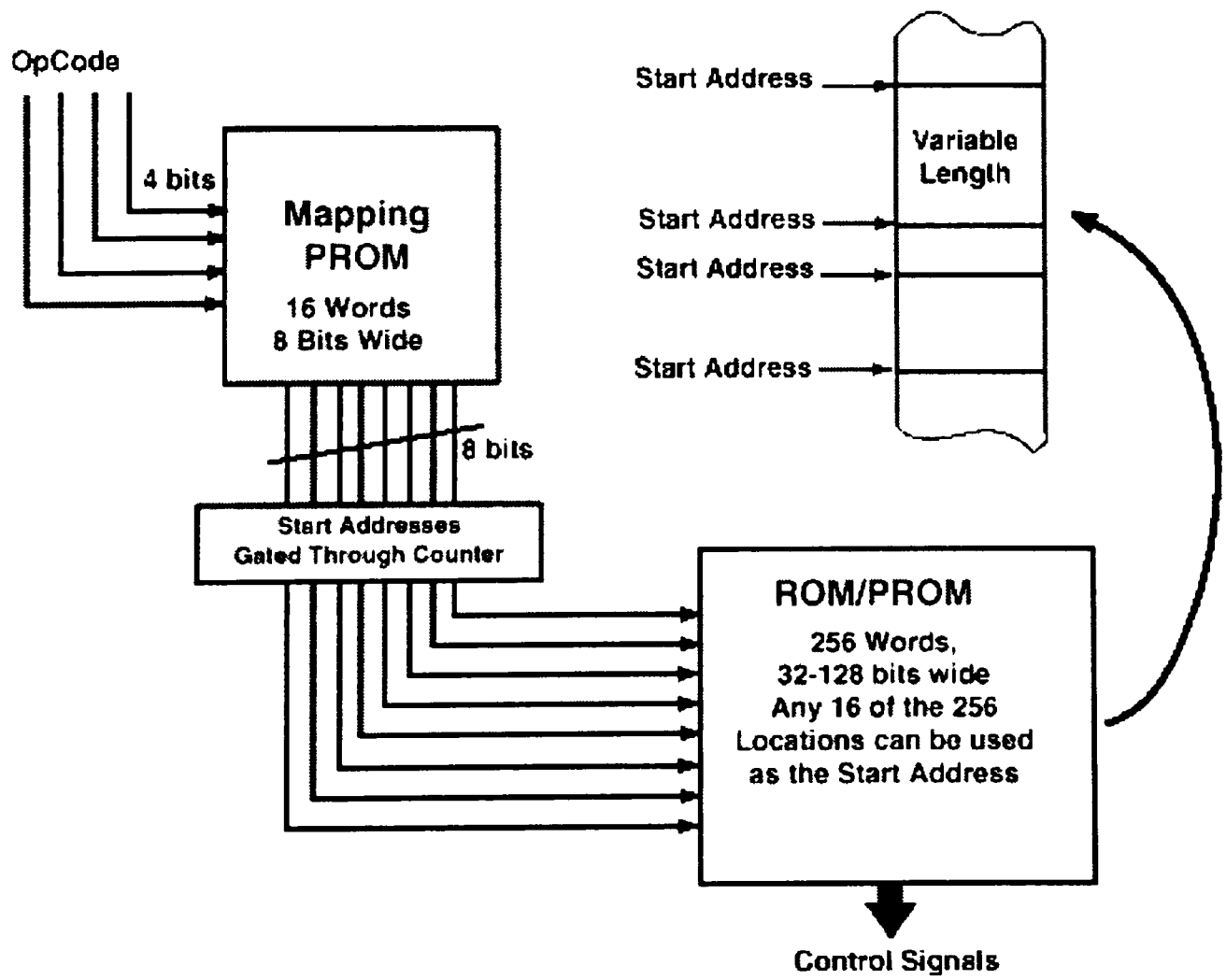


Figure 2.10. Mapping PROM

Start addresses may now be assigned at any location in the PROM memory rather than requiring them to be equidistant from each other, and microroutines may be compacted in this way to delete excessive fragmented space. It is, however, a good idea to allow some unused areas within the PROM to allow for enhancement changes to the system. The final placement of the microroutines in the production PROMs should be done after the debug cycle to minimize mapping PROM changes.

Another feature may be added once the mapping PROM approach is chosen. The mapping PROM may be made larger than required for normal running and contain address lines driven by switches to allow a privileged state, where all opcodes are valid, and a normal state, where certain opcodes are invalid, "trapping out" to an error trap address in the control memory.

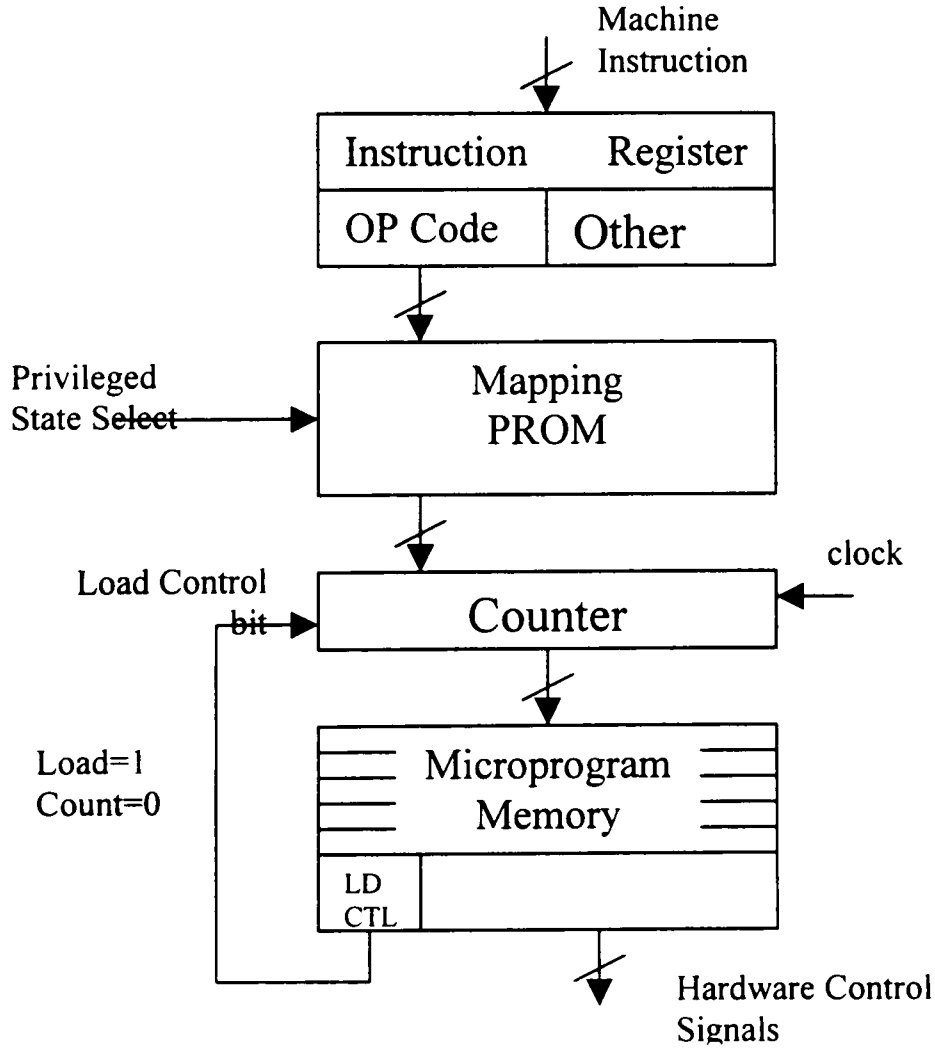


Figure: 2.11: CCU with mapping PROM

The control memory would not necessarily be larger than before. The CCU is shown in Figure 2.11. A good amount of control capability has been incorporated and all the microroutines are simple sequences.

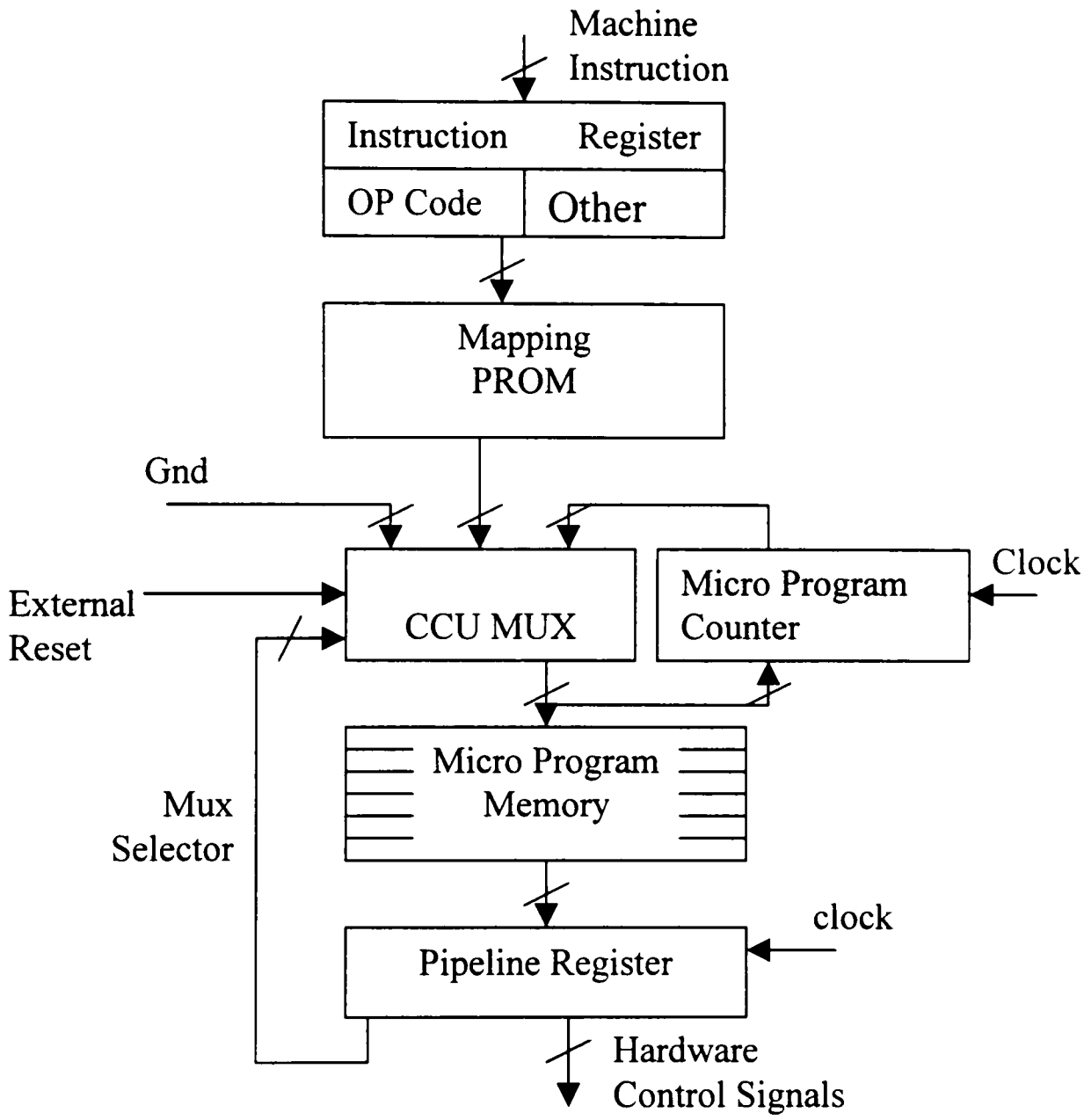


Figure 2.12 CCU

Figure 2.12 shows another approach towards implementing a Computer control unit. It contains computer control unit multiplexer (CCU MUX). CCU MUX selects one of the three buses viz. Mapping PROM, micro program counter and logical zero. Micro program counter always contains the value always 1 greater than the current instruction being executed. The pipeline register holds a micro program word that defines a stable

state of the microprocessor. Its bits define the logic states of the control lines within it. The states in which microprocessor rests are always followed by another stable state generated by a change in the pipe line register. Each state is generated by a single microprogram word stored in the microprogram memory. The processor is advanced from state to state by the advancing of addresses supplied to the micro program memory. This is where the CCU MUX performs its function. The two most significant bits of pipeline register determine which of the three address sources are sent on to the microprogram memory.

As the mapping PROM contains the list of starting address of the micro programs stored in the microprogram memory, the CCU MUX will switch to the mapping PROM input at the end of the fetch cycle. When CCU MUX is switched to logical 0, the microinstruction stored in 0^{th} address of the microprogram memory. If the fetch instruction's micro program word is there in 0^{th} address of the microprogram memory, the control lines of CCU MUX (2 MSBs) of last microinstruction of any microprogram should be in such a way to select logical zero.

2.9 A Simple Processor Architecture

Figure 2.13 shows a simple processor architecture. This can be organized into three major components as Computer Control Unit, Program Control Unit, Arithmetic Logic unit. The Computer Control Unit is explained in detail in the previous sections. The CCU consists of the instruction register , mapping ROM, CCU multiplexer, combinational Incrementer, microprogram counter, microprogram ROM, and pipeline register.

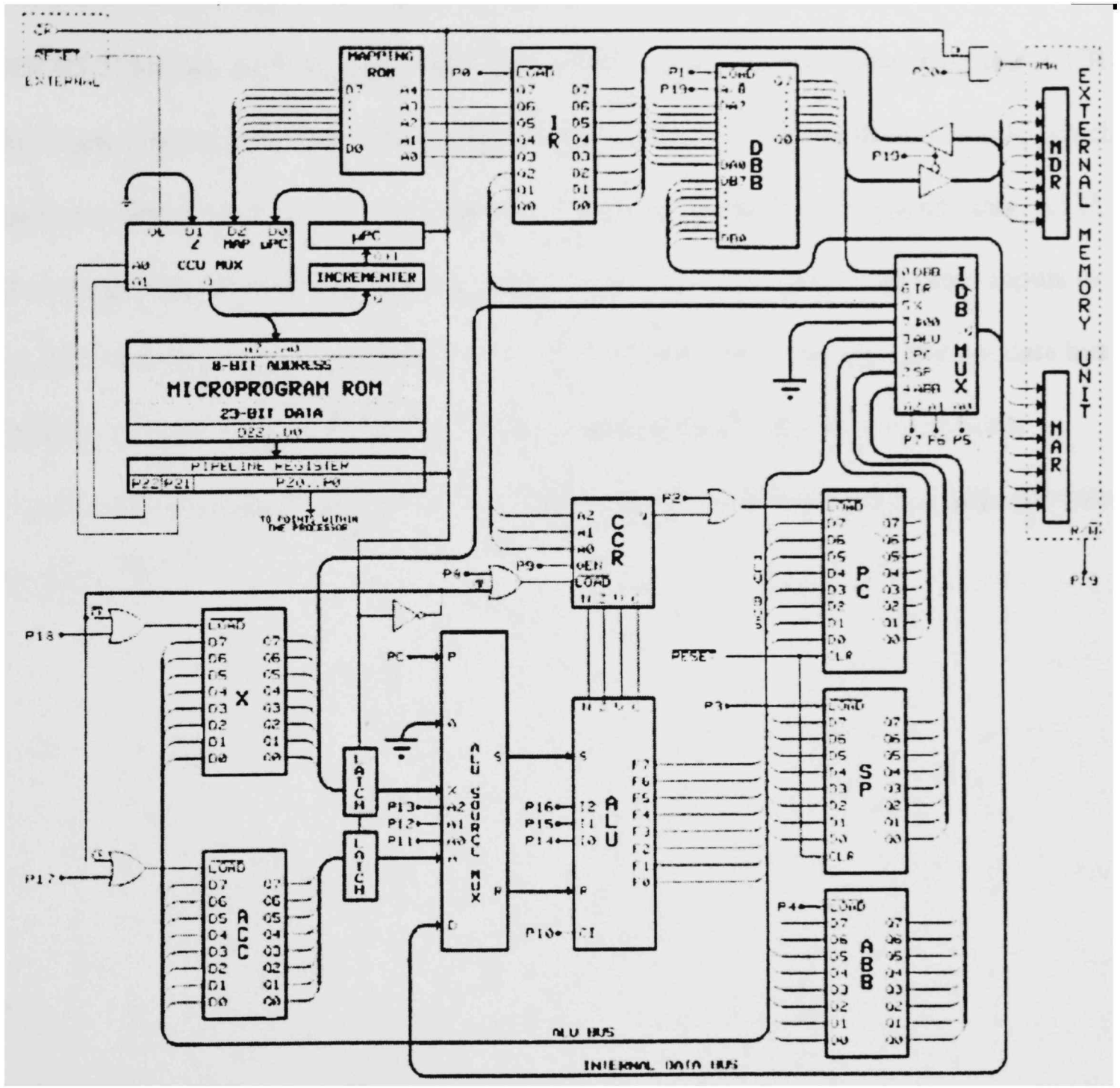


Figure 2.13 Schematic of a microprogrammable 8-bit processor [8]

Program control unit provides access to the external memory for reading of program information and for reading and writing of data information. It consists of Program Counter (PC), Stack Pointer (SP), Address Bus Buffer (ABB), Data Bus Buffer (DBB) and Internal Data Bus Multiplexer (IDB MUX).

The Arithmetic Logic Unit (ALU) provides the capacity for performing arithmetic and logic calculations on operands passing on the internal data paths of the processor. It consists of Accumulator (ACC), Index register (X), ALU Source Multiplexer (ALU SOURCE MUX), ALU function generator (ALU), and Condition Codes Register (CCR). ALU gets control lines from the pipeline register. ALU is a combinational circuit, which performs desired function (Arithmetic, logic and shift operations) on proper data. ALU source multiplexer provides inputs to the ALU function generator. These data inputs to the ALU are from any of the sources viz. Accumulator, Index register, Internal data bus, Program counter and ground. Ground, which is 8-bit 0, is for bit wise operations. Condition Codes Register is used to store the arithmetic and logical status flags generated by ALU. [8]

CHAPTER III

ALU CONTROL UNIT AND CONDITION CODES REGISTER

The arithmetic logic unit (ALU) is a combinational logic network that provides the facility for generating arithmetic, logical functions of data items presented at its inputs. The task of ALU control unit is to make sure that the proper inputs are available at the inputs of the ALU function generator, to generate status functions of data items presented to it.

This chapter discusses the implementation of the idea of using basic building blocks to realize an ALU control unit. Available sources for The ALU include the internal data bus, the accumulator, the index register and the program counter. The ALU control unit consists of the ALU source multiplexer, the accumulator, the index register and the condition code register. The ALU function generator block used in this work was designed by MAZHARUL ISLAM, a graduate student from Department of Electrical Engineering, Texas Tech University. The ALU processes two of five 8-bit operands, placing the generated function on the ALU bus, which is available to the address bus buffer, the stack pointer, the program counter and the internal data bus. The ALU control unit and Condition codes register with out ALU function generator is shown in Figure 3.1

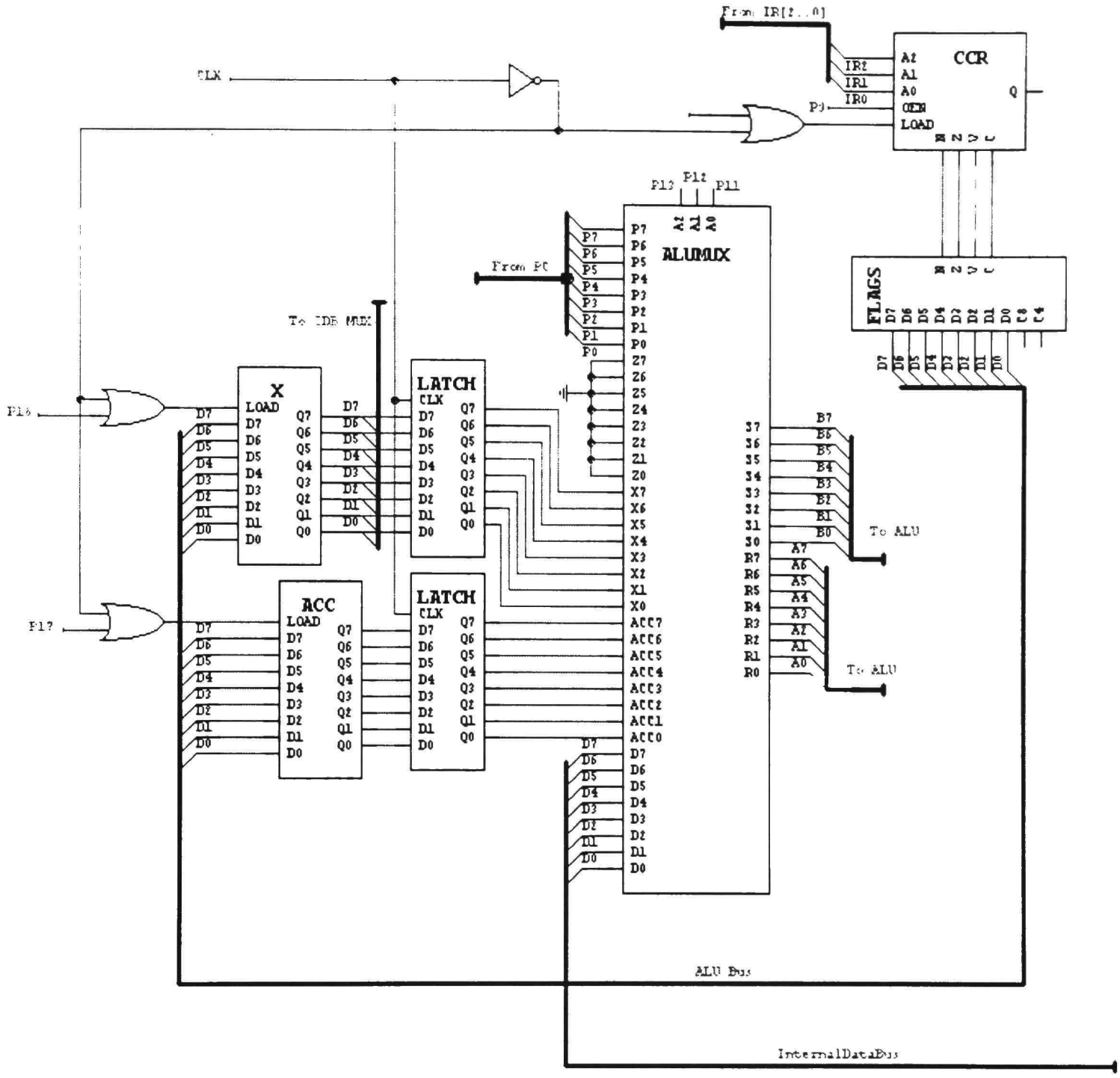


Figure 3.1 ALU control unit and CCR

3.1 ALU function generator

Figure 3.2 shows the block diagram with inputs, outputs and control lines associated with it. Table 3.1 shows the list of functions ALU can perform. Functions are selected by the pipeline register bits S0, S1, S2 and S3. The arithmetic functions also have the facility of utilizing an input carry operand, which is included as pipeline register bit C_{in}.

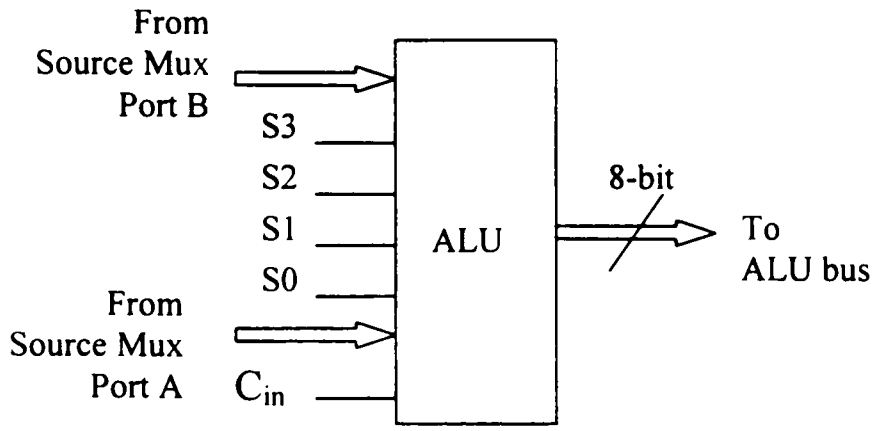


Figure 3.2. ALU function generator.

Table 3.1: Function table for ALU

S3	S2	S1	S0	Cin	Operation	Function
0	0	0	0	0	F=A+B	Add
0	0	0	0	1	F=A+B+1	Add with Carry
0	0	0	1	0	F=A+$\overline{B}$	Subtract With Borrow
0	0	0	1	1	F=A+$\overline{B}$+1	Subtract
0	0	1	0	0	F=A	Transfer A
0	0	1	0	1	F=A+1	Increment A
0	0	1	1	0	F=A-1	Decrement A
0	0	1	1	1	F=A	Transfer A
0	1	0	0	X	F=A^B	AND
0	1	0	1	X	F=AvB	OR
0	1	1	0	X	F=A⊕B	XOR
0	1	1	1	X	F=$\overline{A}$	Complement
1	0	X	X	X	F = *Shift Right* A	Shift right
1	1	X	X	X	F = *Shift Left* A	Shift left

The ALU function generator determines the output on the basis of two 8-bit operands and the input carry. These two input operands are referred to as port A and port B. There are five available sources from which the two operands are selected, and this selection is carried out by the ALU source multiplexer, which provides operands for the ALU function generator. The output function is the source for the ALU bus, an eight-bit

wide bus that provides input data for the address bus buffer, the stack pointer. the program counter, the accumulator, the index register and the internal data bus. Those ALU functions include addition, subtraction, AND, OR, EXCLUSIVE OR logic and shifting [8].

3.2 ALU Source multiplexer

ALU source multiplexer provides operands for the ALU function generator. Figure 3.3 is the block diagram for ALU source mux with control inputs, data inputs and data outputs. Table 3.2 shows the outputs with different control inputs [8].

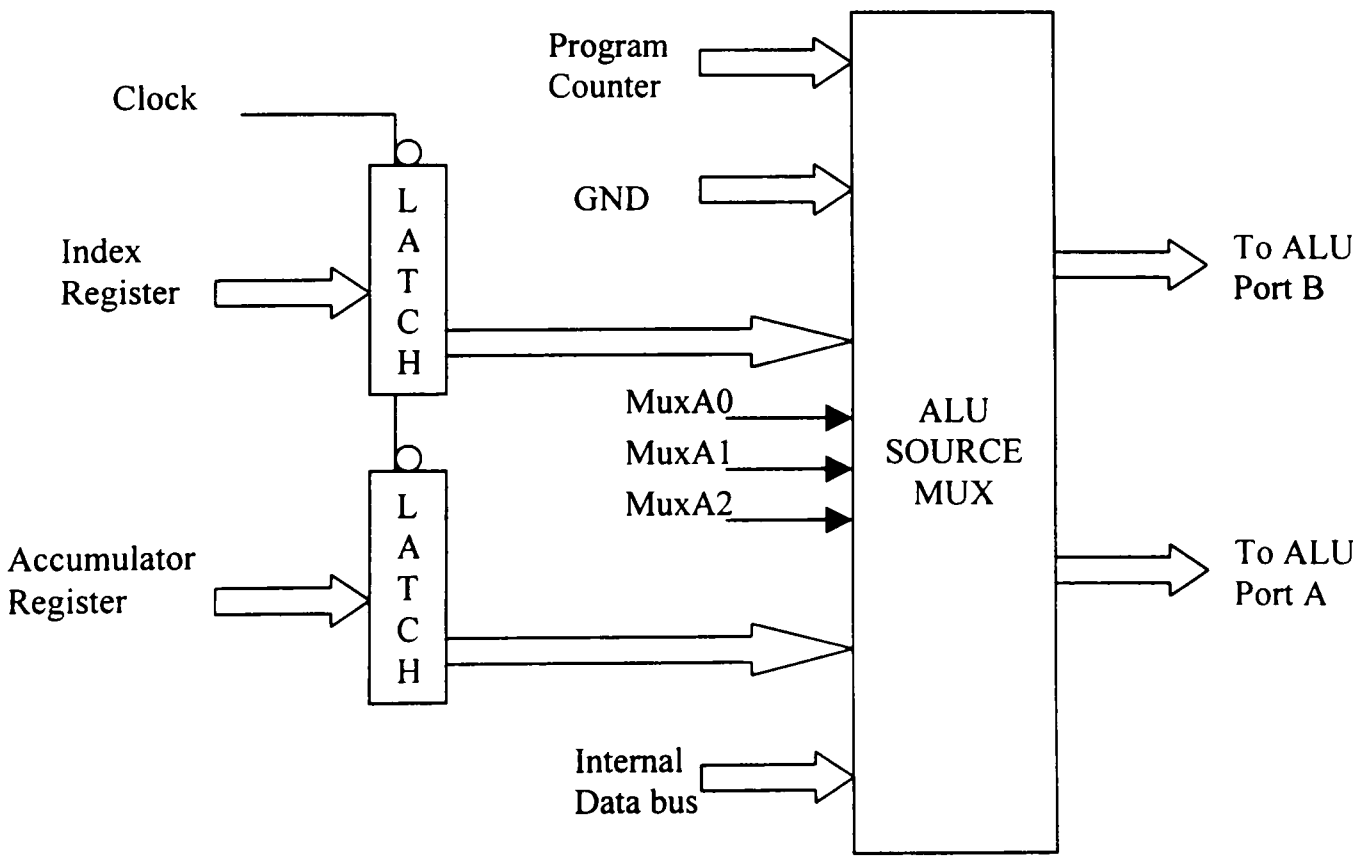


Figure 3.3. ALU Source multiplexer

Table 3.2 ALU source mux

MuxA2	MuxA1	MuxA0	Port A	Port B
0	0	0	D	Z
0	0	1	D	ACC
0	1	0	D	X
0	1	1	D	PC
1	0	0	ACC	Z
1	0	1	ACC	X
1	1	0	ACC	PC
1	1	1	X	Z

The five ALU sources include the internal data bus (D), the accumulator (ACC), the index register (X), the program counter (PC) and a logical 0 (Z). There are ten output combinations for the multiplexer. Four control lines are needed to achieve this. However ignoring the combinations of X,PC and PC,Z (which are of no use), only three control lines are needed. Pipeline register bits MuxA2, MuxA1 and MuxA0 are used to define the ALU sources.

The ACC and X (accumulator and index register) sources for the ALU source multiplexer are latched at the beginning of the clock cycle. This is done so that the output of the two registers will be stable during the entire clock cycle. The accumulator, index register and condition code register are loaded on the rising edge of the clock. Because the outputs of the registers are latched at the beginning of the cycle, the changes that take place within them in the middle of the cycle will not be present at the input of ALU source mux. This allows us to change the registers in the same clock cycle in which their initial values are used [8].

3.3 Accumulator

The accumulator (ACC) is an active-low edge-triggered eight-bit latch used by the microprogrammer as a hardware storage resource. Figure 3.4 shows a block diagram of the accumulator.

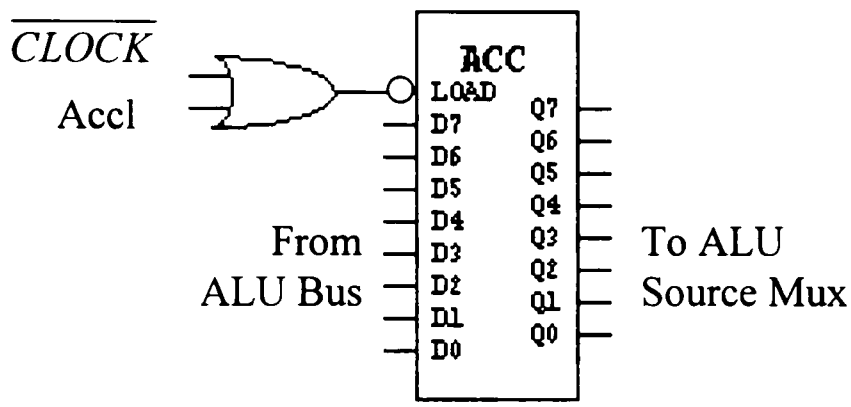


Figure 3.4 Accumulator

The accumulator is available for temporary storage of byte-length data for any purpose the programmer desires. It is loaded by pipeline register bit "Accl". The control line is gated through OR logic with the inverted clock. This causes the ACC-LOAD line to remain high for the first half of the clock cycle, delaying its high to low transition until half way through the clock cycle. This delay provides enough propagation time so that the processor status can settle following some ALU operation, and the results can be loaded into the accumulator in the same clock cycle.

For example, the contents of the accumulator can be incremented in one clock cycle. At the beginning of the clock cycle, the accumulator contents are passed through the ALU source mux and incremented by the ALU function generator. The resulting function is fed back to the accumulator. All of this takes place within the first half of the

system clock cycle. Midway through the cycle, the accumulator can be loaded with the data. The latch on the output of the accumulator prevents the data change from propagating to the ALU source mux, so the output of the ALU function generator is stable through the clock cycle [8].

3.4 Index Register

The index register (X) is an active edge-triggered eight-bit latch used by the microprogrammer as a hardware storage resource. Figure 3.5 shows the block diagram of index register.

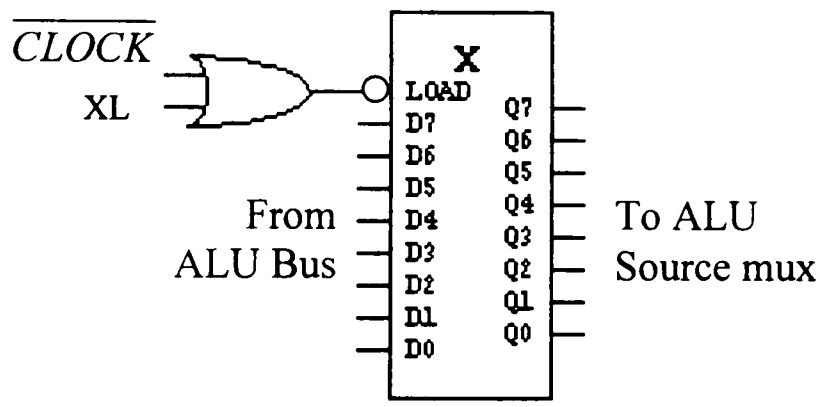


Figure 3.5 Index Register

The index register is available for temporary storage of byte-length data for any purpose the programmer designs. However, its intended use is to store a number that can be added to an address in order to provide indexed addressing. It is loaded by pipeline register bit "XL". The control line is gated through OR logic with the inverted clock. This causes the X-LOAD line to remain high for the first half of the clock cycle, delaying its high to low transition until half way through the clock cycle. This delay provides enough propagation time so that the processor status can settle following some ALU operation, and the results can be loaded into the index register in the same clock cycle.

Consider the following example. The contents of the index register can be decremented in one clock cycle. At the beginning of the clock cycle, the index register contents are passed through the ALU source mux and decremented by the ALU function generator. The resulting function is fed back to the index register. All of this takes place within the first half of the system clock cycle. Midway through the cycle, the index register can be loaded with the data. The latch on the output of the index register prevents the data change from propagating to the ALU source mux, so the output of the ALU function generator is stable through the clock cycle. [8]

3.5 Flags

The flags are affected by the arithmetic and logic operations in the ALU. In most of these operations, the result is stored in the accumulator. Therefore, the flags generally reflect data conditions in the accumulator – with some exceptions.

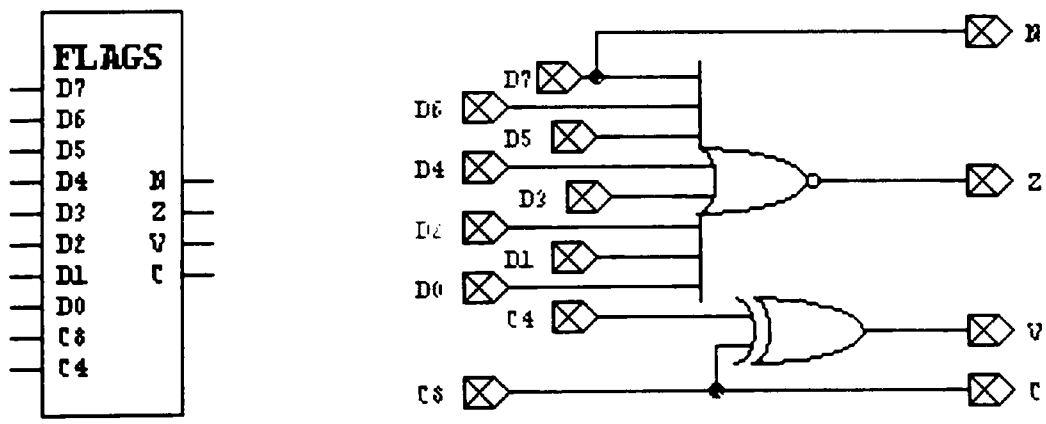


Figure 3.6 Flags

The descriptions and the conditions of the flags are as follow [5]:

- N – Negative: After the arithmetic or logic operation, if bit D7 of the result (usually in the accumulator) is 1, the negative flag is set. This flag is used with signed numbers. In a given byte, if D7 is 1, the number is viewed as a negative number; if it

is 0, the number is considered positive. In arithmetic operations with signed numbers, bit D7 is reserved for indicating the sign and the remaining seven bits are used to represent the magnitude of a number [5].

- Z – Zero flag: The zero flag is set if the ALU operation results in zero, and the flag is reset if the result is not 0. This flag can be modified by the results in the other registers as well. This can be implemented simply by applying all the data bits to a NOR gate, which gives an output of 1 when all the inputs are 0s [5].
- C – Carry flag: If the arithmetic operation results in carry, the Carry flag is set; otherwise it is reset. The carry flag also serves as a borrow flag for subtraction. In the ALU function generator, designed by ISLAM, this bit is designated by C8 [5].
- V – Overflow flag: The Overflow flag is set if there is an overflow after an arithmetic operation and the flag is reset if there is no overflow. The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned. When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position. In the case of signed numbers, the left most bit always represents the sign and signed numbers are in 2's complement form. When two signed numbers are added, the sign bit is treated as part of the number and end carry does not indicate an overflow.

An overflow may occur if the two numbers added are both positive or both negative. For example:

Carries: 0 1		Carries: 1 0	
+65	01000001	−65	10111111
+70	01000110	−70	10111010
+130	10000111	−135	1 01111001

Here the 8-bit result that should have been positive has a negative sign and the result that should have been negative has a positive sign. An overflow condition can be detected by observing the carry in to the sign bit and the carry out of the sign bit position. If these two carries are not equal, an overflow is produced. A simple XOR gate can detect this condition. When these two carries are applied to an XOR gate and an overflow is there, the output is 1. In ALU function generator, C7 and C8 designate these two bits [7].

Figure 3.6 shows the hardware implementation of the flags. All the flags generated by this circuit are stored in condition codes register.

3.6 Condition Codes Register

The condition codes register (CCR) is an active-low edge triggered four-bit latch that is used to store the arithmetic and logical status flags generated by the flags. Figure 3.7 shows the block diagram for the condition codes register. The circuit includes a multiplexer, which allows any one of the four bits to be selected for output. In addition, each logical status of each bit is available for output, providing for the selection of one of eight possible combinations.

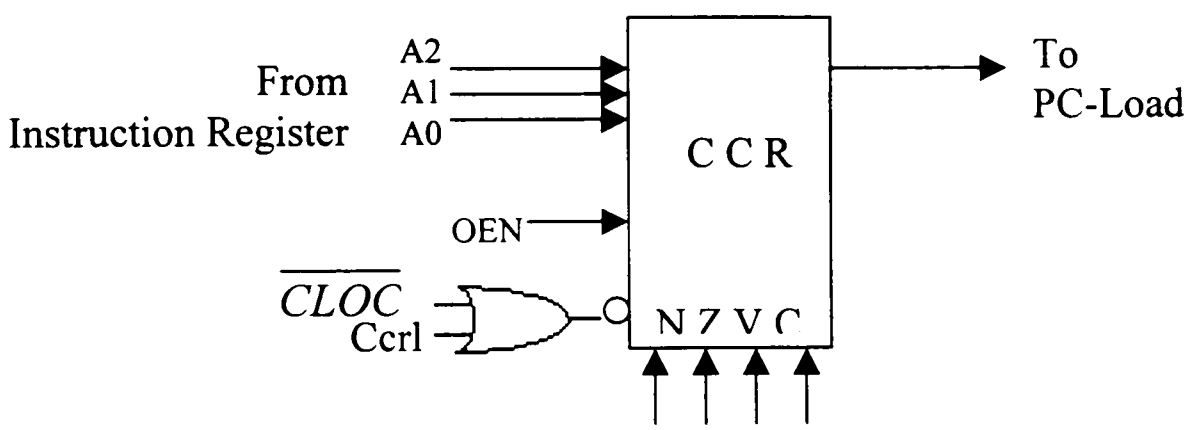


Figure: 3.7 Condition Codes Register

The CCR is loaded with the status flags from the Flag register on high to low transition on pipeline register bit "CCRL". However, the control line is gated using OR logic with the system clock. The result is to delay the drop of the control line until the middle of the clock cycle. This provides enough delay so that the flags can be loaded in the same clock cycle in which they are generated. This same principle is used in loading the accumulator and index register.

The CCR is used to conditionally load the program counter so that the processor can execute conditional jumps and conditional branches. The logic is shown in Figure 3.8. The different combinations of inputs for required outputs are shown in Table 3.3 [8].

Table 3.3 Condition Codes Register

A2	A1	A0	OEN	Q
X	X	X	0	0
0	0	0	1	C
0	0	1	1	$\overline{C}$
0	1	0	1	V
0	1	1	1	$\overline{V}$
1	0	0	1	Z
1	0	1	1	$\overline{Z}$
1	1	0	1	N
1	1	1	1	$\overline{N}$

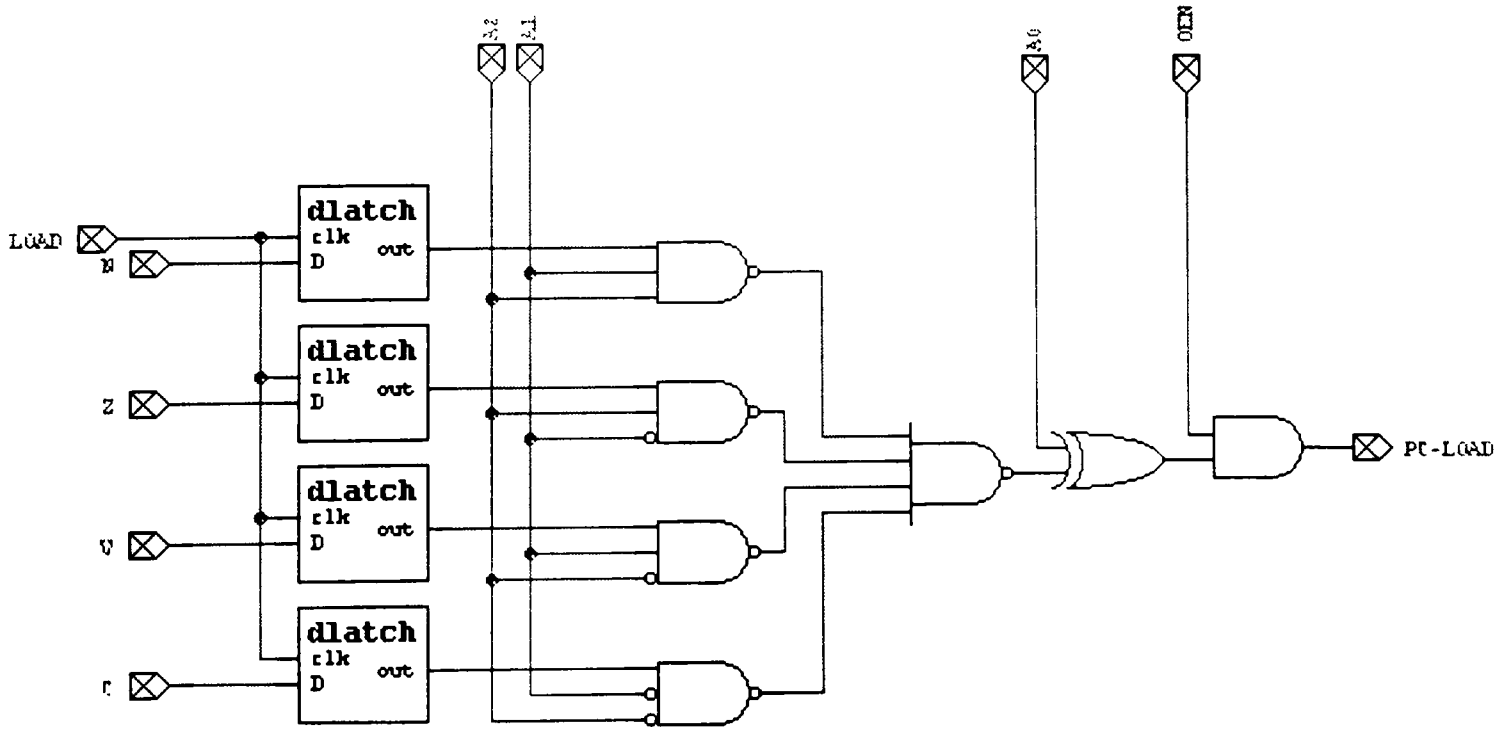


Figure 3.8 Condition Codes Register Logic

If the load of the program counter is not conditional, the "CCR-OEN" line of pipeline register is set to a logical 0. This sets the Q out put to a logical 0. The PC-LOAD

line is active-low edge triggered and when Q is gated with the PC-LOAD line using OR logic, the Q line has no effect on PC-LOAD. However, if the Q output of the CCR is a logical 1, it holds the PC-LOAD line in a logical 1 state, disabling the load of PC. Therefore, the load of the PC is conditional on the value of Q.

3.7 Logic Circuits

Till now every individual part is explained with block diagrams. Logical circuits associated with them are shown here.

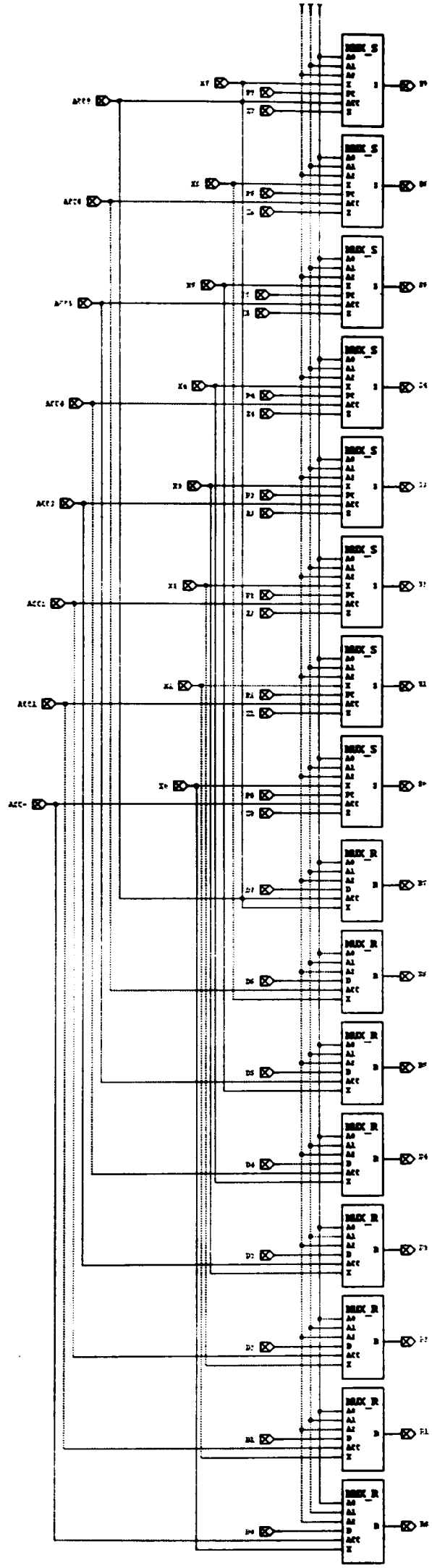


Figure 3.9 ALU Source mux

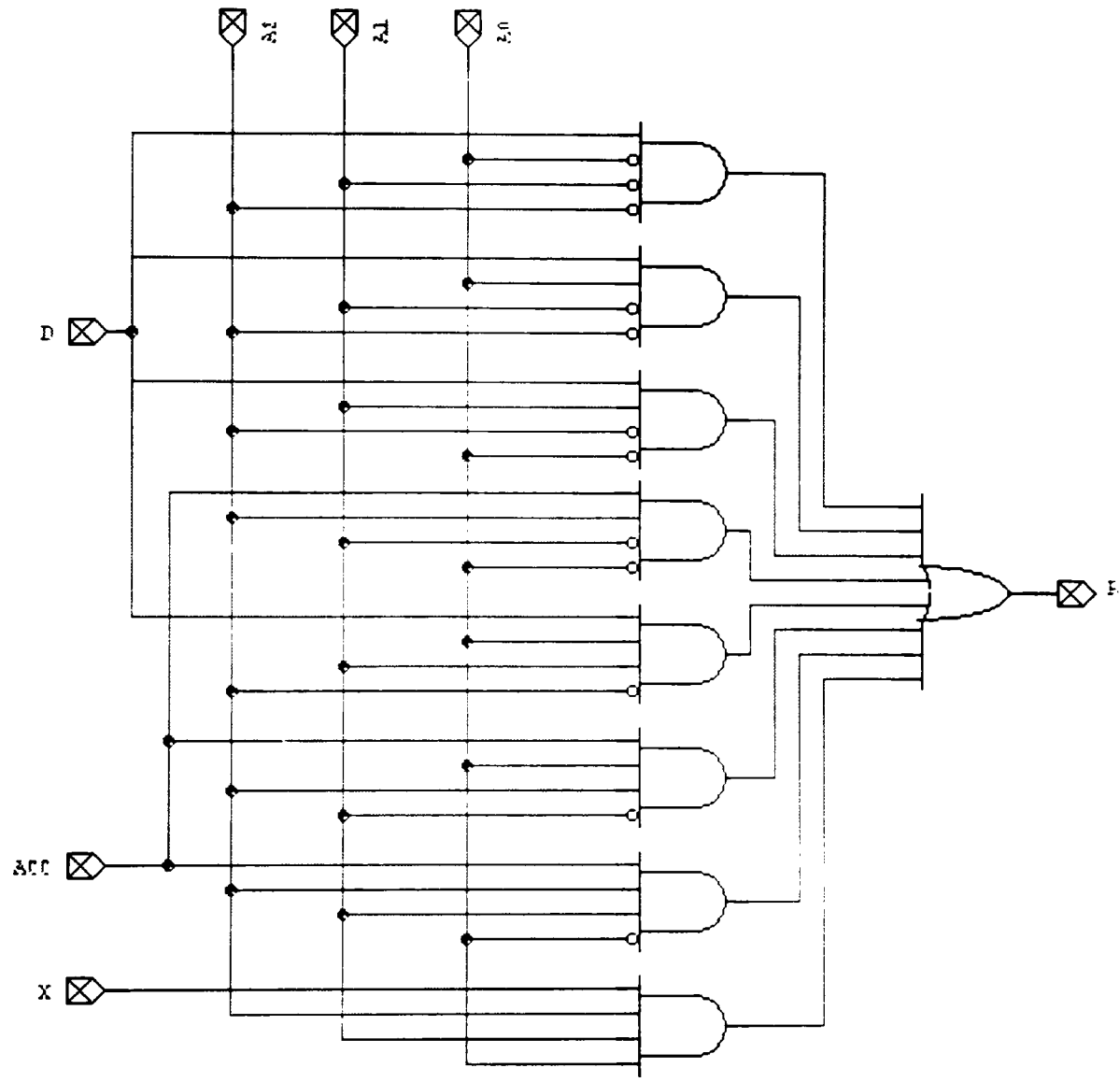


Figure 3.10 MUX_R

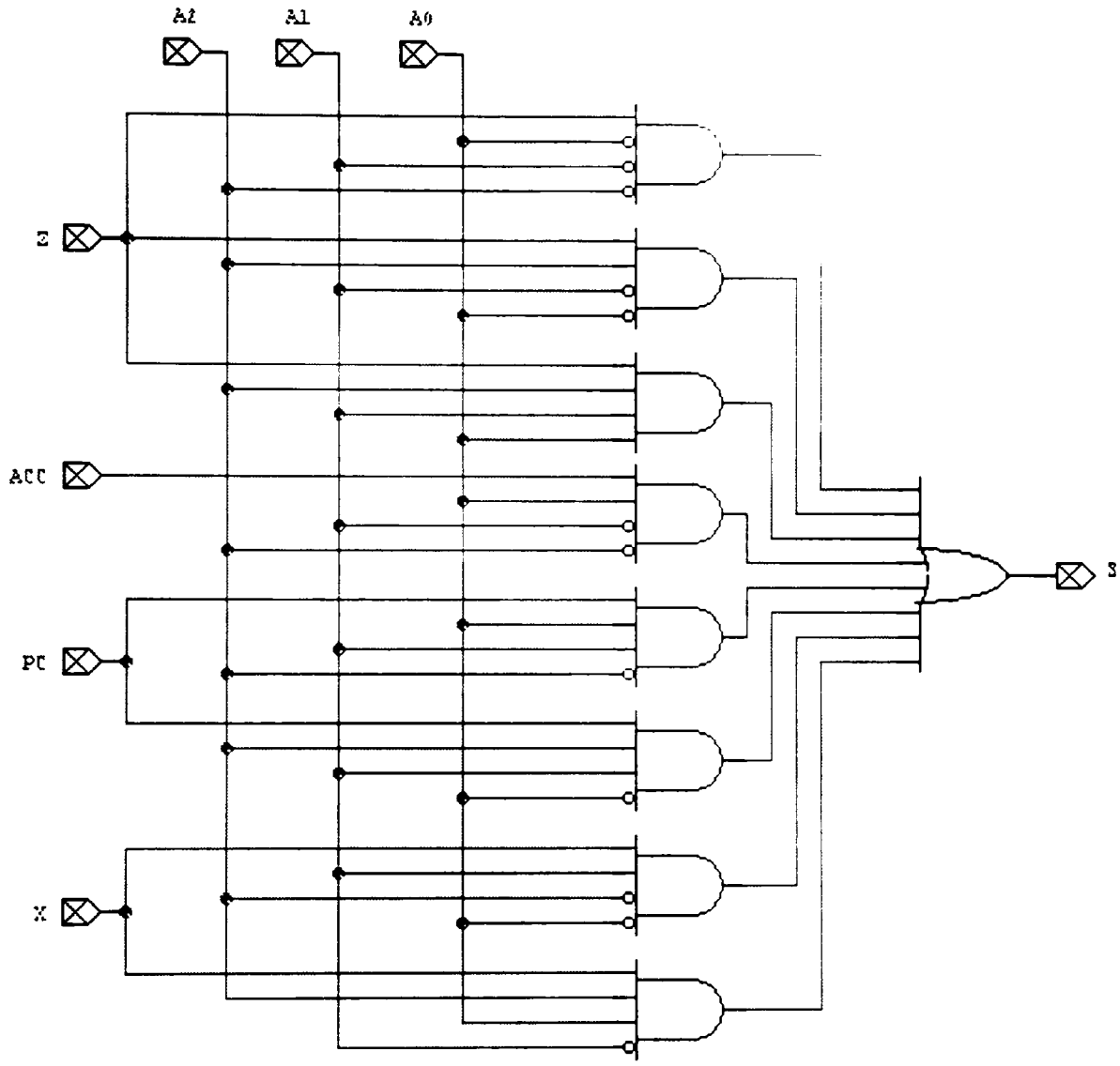


Figure 3.11 MUX_S

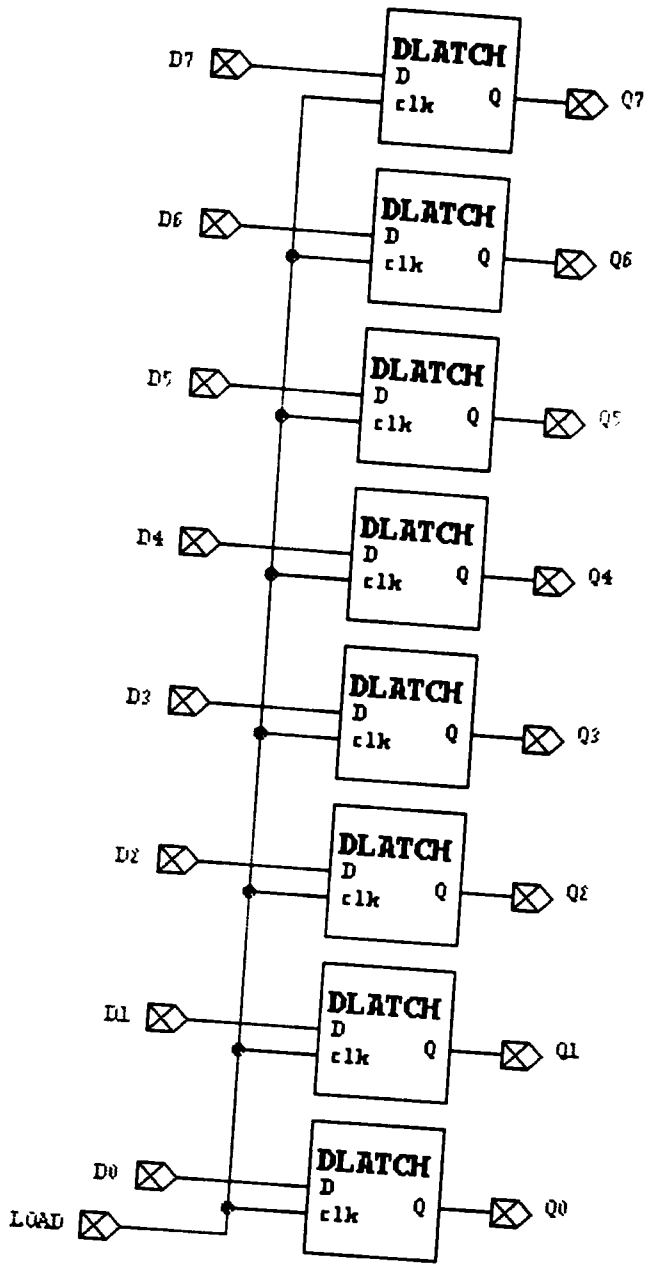


Figure 3.12 Accumulator / Index register / Latch

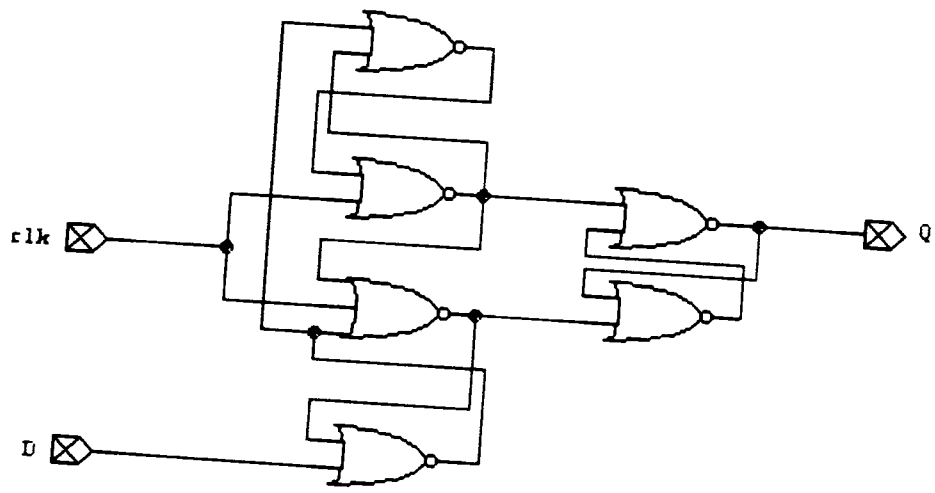


Figure 3.13 DLATCH

CHAPTER IV

LAYOUT OF 8-BIT ALU CONTROL UNIT

4.1 Introduction to layouts

There are several approaches that can be used to describe an integrated circuit (IC). In the basic sense, an IC is an electronic network that has been fabricated on a single piece of a semiconductor material such as silicon. The silicon surface is subjected to various processing steps in which impurities and other material layers are added with specific geometrical patterns. The steps are sequenced to form three dimensional regions that act as transistors for use in switching and amplification. Passive elements, such as resistors and capacitors, are not always included as elements in the circuit, but arise as parasitic elements due to the electrical properties of the materials. The wiring among the devices is achieved using interconnects, which are patterned layers of low-resistance materials such as aluminum. The resulting structure is equivalent to creating a conventional electronic circuit using discrete components and copper wires [6].

So we can define an integrated circuit as a set of patterned layers. Each layer has specific electrical characteristics, such as sheet resistance, and is patterned according to layers above and below. Stacking different material patterns results in geometrical objects that function electrically as devices or interconnects.

A layout editor such as L-Edit is used to design the patterns on each layer and accomplish the physical design of the chip. The drawings represent the patterning of each layer, and the overall image can be interpreted as the top view of the chip. Each layer is

distinguished by a separate color on the computer monitor so that three-dimensional structures such as transistors can be distinguished [6].

4.1.1 Design Philosophies

Digital VLSI can be implemented at several levels depending upon the starting point. The most common divisions are as follows [6]:

1. Full Custom: In full custom design every detail of the integrated circuit layout needs to be completed. At this level, all gates must be designed, drawn and simulated.
2. Cell-based: Cell-based designs are based on existing cells stored in a library, which is a collection of pre-designed gates and modules. The properties of each cell such as speed and lay-out dimensions are provided to the system designer, who provides the arrangement and interconnect to implement the system. Application-specific integrated circuits (ASICs) are usually constructed in this manner.
3. Gate arrays: Gate arrays consist of arrays of MOSFETs that can be wired using inter-connect lines to implement the desired functions. Logic circuits can be prototyped very quickly using this approach.

CMOS standard cells were used to layout the 8-bit ALU control unit. CMOS is recognized as a leading contender for existing and future VLSI systems. CMOS provides an inherently low power static circuit technology that exhibits a of lower power-delay product than other comparable design-rule nMOS or pMOS technologies. It also provides high density, primarily because the transistors can be made very small.

Standard cells are designed to fit together like bricks in a wall. Figure 4.1 shows an example of a simple standard cell. Power and ground buses (Vdd and GND) run

horizontally on metal lines inside the cell. Standard cell design allows the automation of the process of assembling an ASIC. Groups of standard cells fit horizontally together to form rows. The rows stack vertically to form rectangular blocks. The important ideas that arise in the process of building complex chips from cells are:

i. Designing basic cells that can be used to construct more complex functions.

ii. Employing cell geometries with standardized electrical parameters that can be easily interfaced with other cells.

iii. Building a hierarchy of cells.

4.1.2 Cells and Hierarchy

In this design cell is the basic unit. At the logic level this may be a simple logic function or a complex boolean operation. The circuit equivalent of a cell is a defined layout pattern with given dimensions, input output ports and specified electrical performance. On the other hand cell may be defined as a set of HDL declarations or timing diagrams. A system is constructed by interconnecting cells together. At the chip design level the most important factors are [6]:

- Area and dimensions: Every cell consumes chip area and has a particular geometrical shape associated with it. Both are important for high-density integration.
- Ports: The location of the input and output ports is very important for routing the data path. Also the power supply and ground are usually needed to provide electrical energy to the cell.

- Interconnect strategy: The cells must be wired together using the interconnect layers. Even with three or four separate conducting layers, this can be the limiting factor in the logic density.

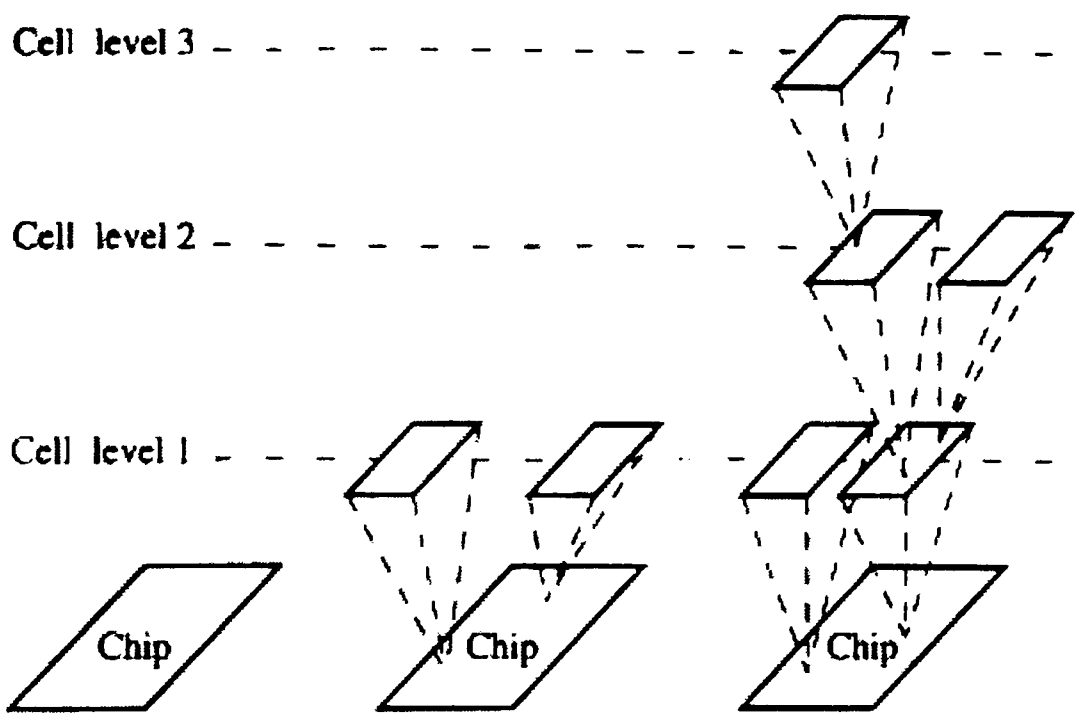


Figure 4.1: Cells and Hierarchy [6]

4.1.3 Signal Groups

Digital signals may be grouped into two main categories: data and control. In general data bits are encoded information segments such as numbers or symbols being processed by the system. Control bits on the other hand determine the operations that the data bits are subjected to. For example, in a 4:1 multiplexer, four input lines and a single output port transmit data bits while two control bits determine the operation, i.e., which of 4 lines is directed to the output.

The design of digital network can usually be divided into two related sub problems: the design of data path logic and the design of the associated data path control

logic. Determining the data path is centered on creating the functions that process the data stream. For example, logic operations, such as NANDs and NORs or arithmetic operations, such as addition, are classified as data path functions. The control circuit on the other hand determines the root that the data follows the current operation. Control signals are used to switch the data stream through the gates in proper order. These concepts are quite general and can be used to describe systems of arbitrary complexity. For example, in microprocessor, the program controls the data path sequence, while the data path itself contains the function [6].

Although there are many exceptions, data path logic tends to be localized and can be contained within cell units. Control logic, on the other hand, is usually applied to groups of functional blocks, and must be routed to various locations. This illustrates the basis of hierarchical design. Cells provide defined logic functions with input and output ports. Once the basic functions are defined, the cells may be placed and wired as needed to provide the system level functions [6].

4.1.4 The Floor Plan

The floor plan of the chip shows the replacement and area consumption of the major logic functions in the finished chips. Routing of data lines, clocks and control signals is determined by comparing the logical design with the physical design of the floor plan [6].

Floor plan design is performed at the architectural level. All major operations groups are identified and the interconnect requirements are studied. The placement of cells is not arbitrary, since the large-scale system performance is directly affected. Initial

real estate allocations are based on the complexity of the cells, the limitations of the fabrication technology, and past experience. The size, shape and placements of the cells are adjusted as needed as the design evolves. This often requires trade-offs among the various functional units to obtain a functional design [6].

In Figure 4.2 the blocks labeled unit A, unit B etc., represent distinct cells, each with a specific logic function. Input and output lines specified by port locations on the blocks. In addition power supply and ground connections must be included. Interconnections among units (both data and control signals) must be routed within the limits imposed by the layout design rules. The lines may go around the functional units, or may pass through (or over) them [6].

The floor plan is important to designers at all levels. At the system level, the distance between functional blocks can limit the data transmission speed and, hence the performance of the chip. Circuit and cell design is often based on satisfying the electrical specifications using the minimum amount of real estate consumption. Interconnect routing effects every aspect of the design and operation of the network [6].

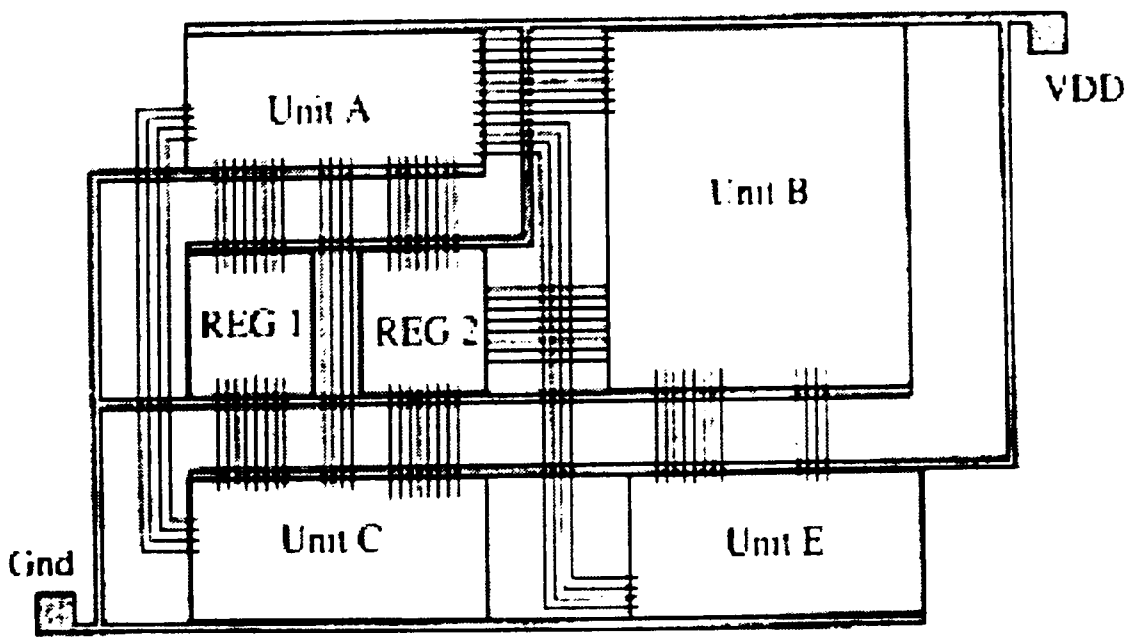


Figure 4.2: Floor Plan [6]

4.1.5 Interconnects

The limiting factor in high-density system design is the interconnect routing and connections. One reason for the situation is the existence of basic layout rules such as: [6]

- The minimum width and spacing rules for wires on the same layer and
- Surround design rules that are required for contacts and vias.

These automatically limit the density of the wiring. Since, each layer is intrinsically two-dimensional, wires on the same layer cannot cross without creating an electrical short-circuit. Routing must be implemented using cross-overs and cross-unders, and the problem can become very complex. It is interesting to note that, with lithographic resolutions below 1 micron, the number of MOSFETs in a circuit is not of much concern because each transistor is so small. However, the number of contact cuts needed can be a problem due to surround-type design rule specifications [6].

Interconnect routing is complicated by parasitic electrical coupling among lines that are physically close to each other. This is termed "crosstalk", and can cause data transmission errors. Both logic '0' and logic '1' voltage levels can be effected due to crosstalk [6].

Crosstalk problems can be difficult to isolate, particularly in high-density layouts. It is therefore best to avoid in the original design. The effects can be minimized by obeying all design rule spacings, avoiding long lengths of parallel lines and purposely introducing "kinks" into the lines to disturb the coupling.

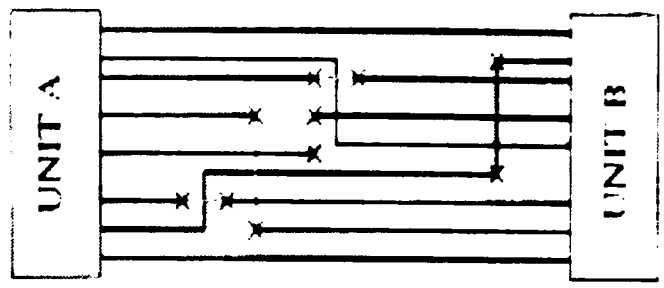


Figure 4.3: Inter connects

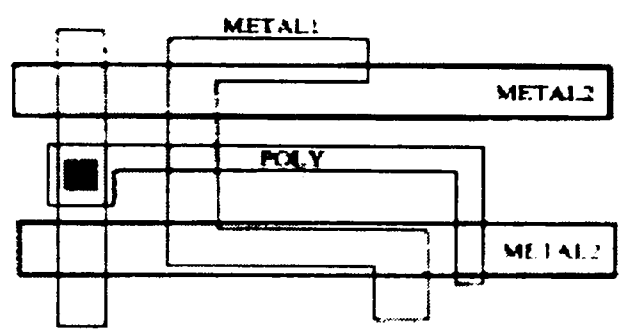


Figure 4.4: Crossovers

Interconnect wiring usually requires that signals be routed through different layers as shown in Figure 4.3. In this drawing unit A and unit B are to be connected by an 8-bit data bus. However, the location of the input and output ports requires that two levels of interconnect be used to achieve the desired results. When employing different material layers, the layers can be crossed with only minimal coupling. In Figure 4.4, we can see that [6]

- POLY can cross METAL1 and METAL2.
- A POLY contact may be used to create a current flow path to METAL1.
- METAL1 and METAL2 can cross over.

These rules help solve the interconnect problems in complex situations. However, layer-to-layer coupling capacitance is introduced whenever lines cross. The value of this capacitance is proportional to the overlapping area. For example, the METAL1-

METAL2 overlap shown in the drawing actually couples the two horizontal METAL1lines to each other. Although the coupling is very weak, it may cause problems in high-speed systems [6].

4.1.6 Pad Frames

Bonding pads are large metal regions that allow wiring between the silicon circuitry and the pins on the IC package. The pad frame layout gives the placement of bonding pads and sets the minimum die size and the number of connections to the outside world. These are pre-defined for a given die size. Figure 4.4 shows the pad frame for a

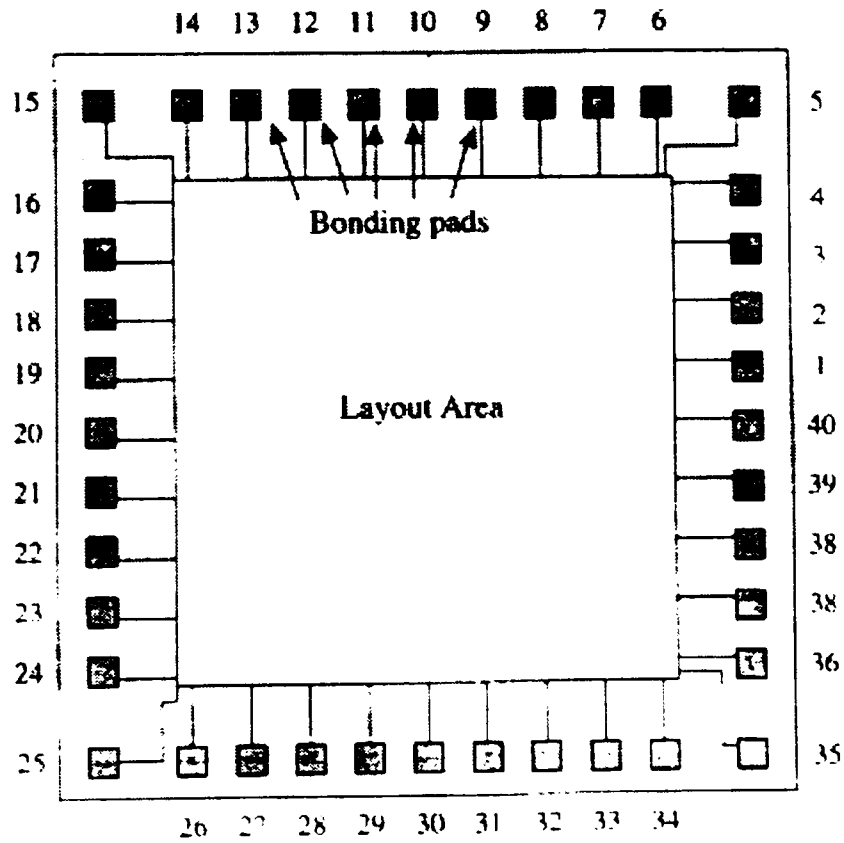


Figure 4.5: Pad frame

40-pin design. These have been numbered according to their pin connection when the die is placed in a dual in-line (DIP) package. Two bonding pads must be reserved for the power supply and ground connections [6].

4.1.7 Global Signal Distribution

Logic networks use both local and global signals for their operation. Local signals are most often data streams, and can be treated at the cell level. However, global broadcasts, such as system timing clocks, must be accounted for in the floor plan in order to ensure proper distribution. Interconnect network topologies are often based on geometrical symmetries in an effort to ensure that each line delivers the same signal to the receiver, regardless of location [6].

4.1.8 Power Distribution

All integrated circuits require power distribution bus lines to supply current to the gates. In CMOS, we usually have a positive VDD and a ground voltage that must be routed across a die. It is important to use a regular geometrical structure that accommodates the shapes of the logic cells. There are several geometries that are used for power supply distribution. The simple example is shown in Figure 4.5. This provides VDD and GND rails with space in between for the placement of circuits. It is important to remember that power distribution lines must be capable of handling relatively large current levels. Since, low resistance lines are essential, wide metal lines are used [6].

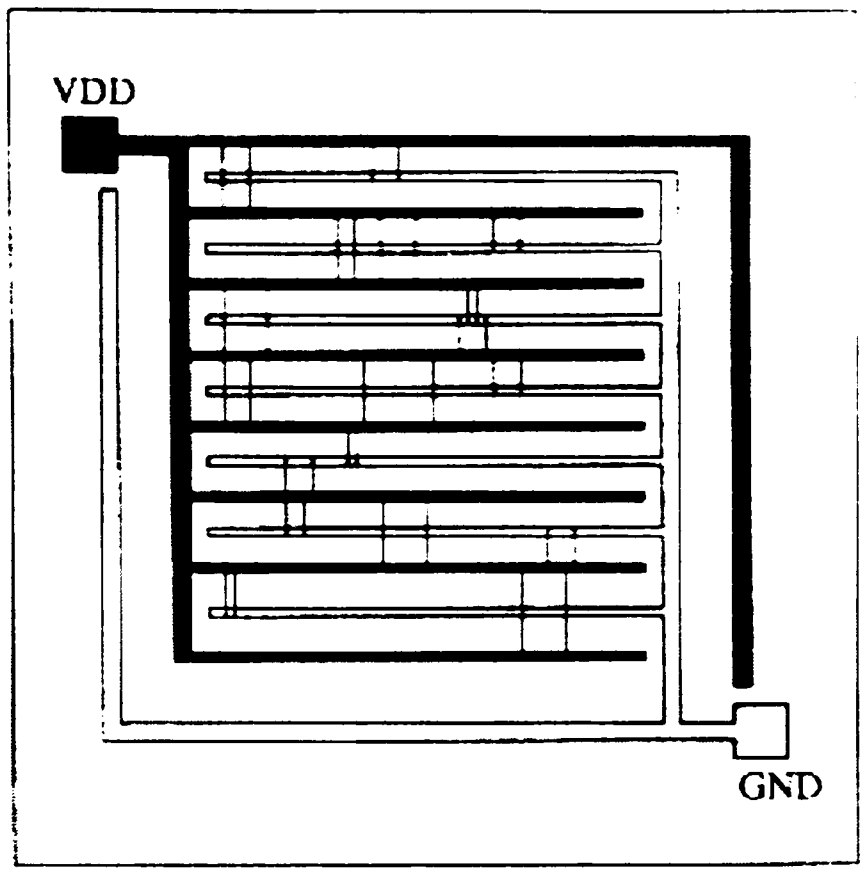


Figure 4.6: power and ground distribution

4.2 ALU control unit chip

The 8-bit ALU control unit developed in logic works is laid out using Tanner L-Edit version 7. SCNA Technology is used with LAMBDA = 0.6 micron. Actual layout of ALU control unit with data-path, control logic and Interconnections is shown in Figure 4.12. Figure 4.13 shows the floor plan and pin diagram of the ALU control unit. Pin names of all pins are shown in Table 4.1. Actual layout of chip is given in Figure 4.14.

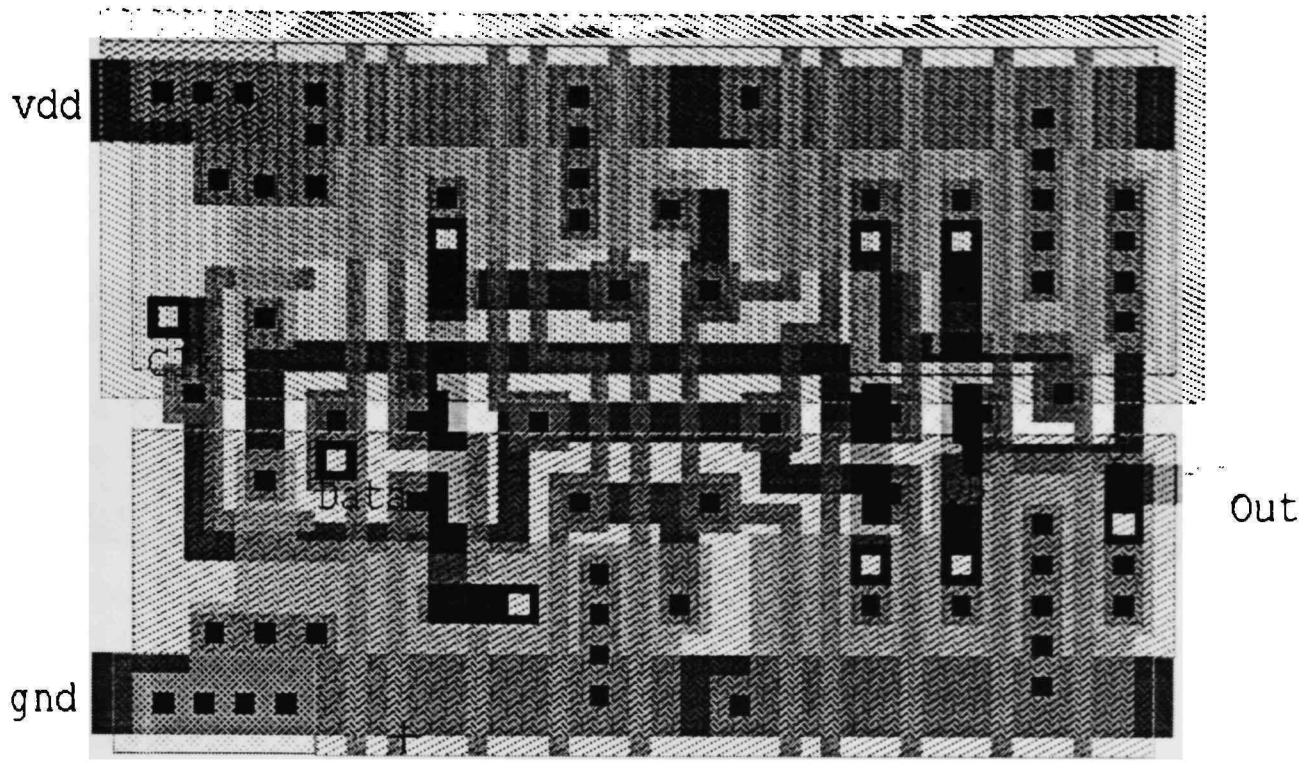


Figure 4.7 Layout of D Latch.

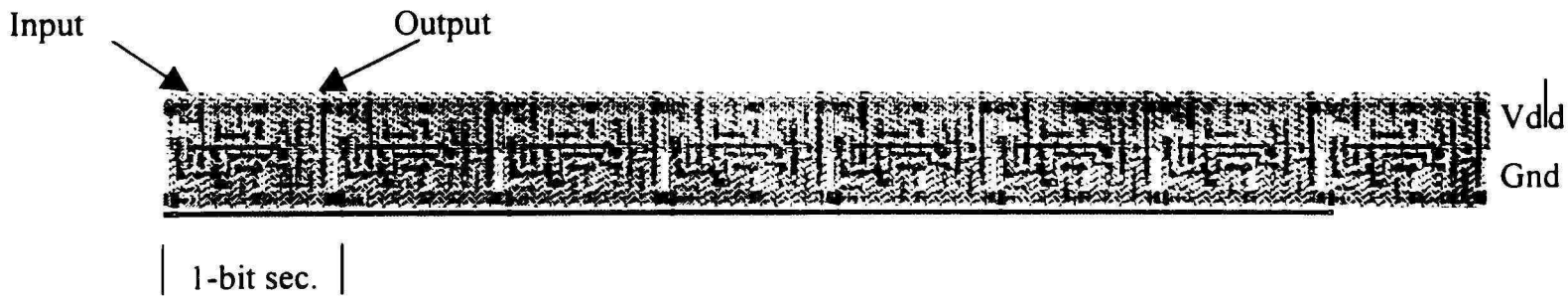


Figure 4.8 layout of 8-bit Latch.

Placing 8 D Flip-flops cells side by side and connecting all clocks to form a load line forms 8-bit latch. Inputs are data inputs of D flip-flops and out puts are "OUT"s of D flip-flops.

Figure 4.9 1-Bit Section of MUX_R.

Figure 4.10 1-Bit Section of MUX_S.

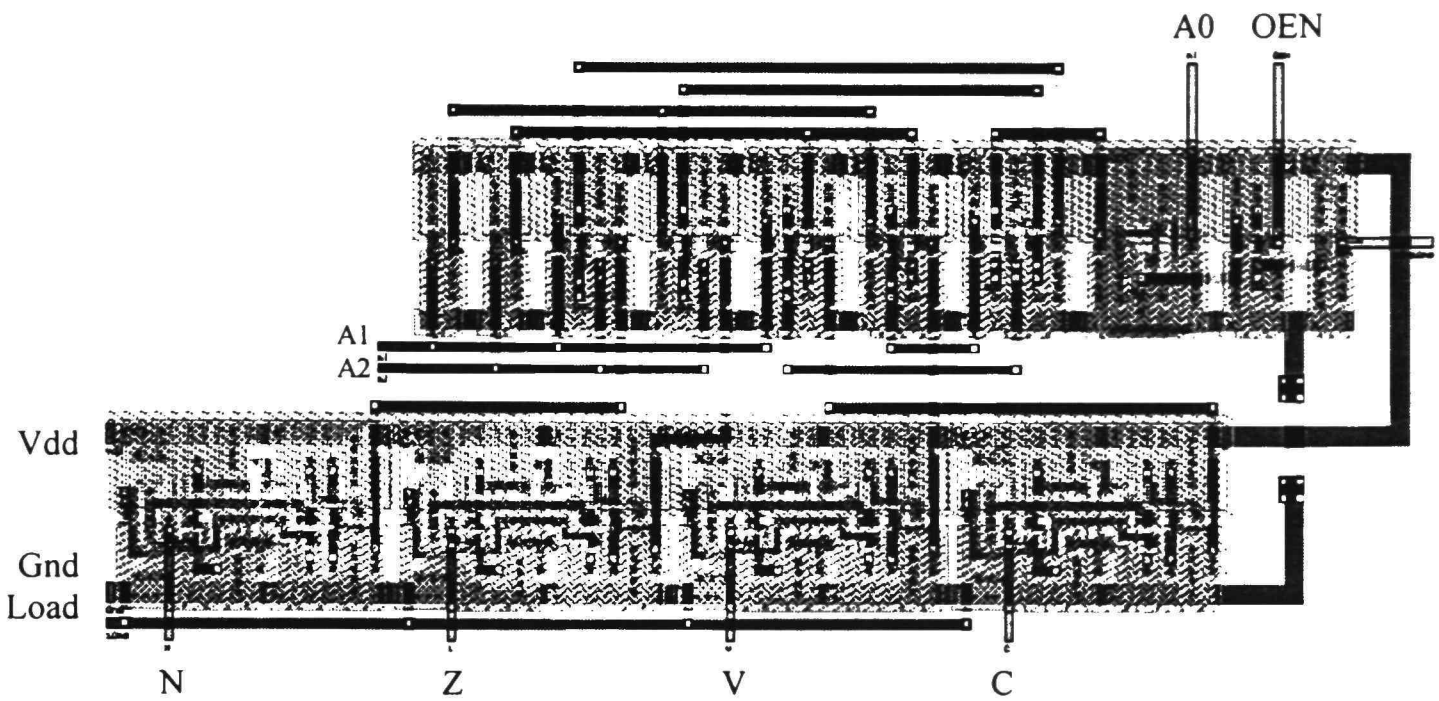


Figure 4.11 Layout of CCR.

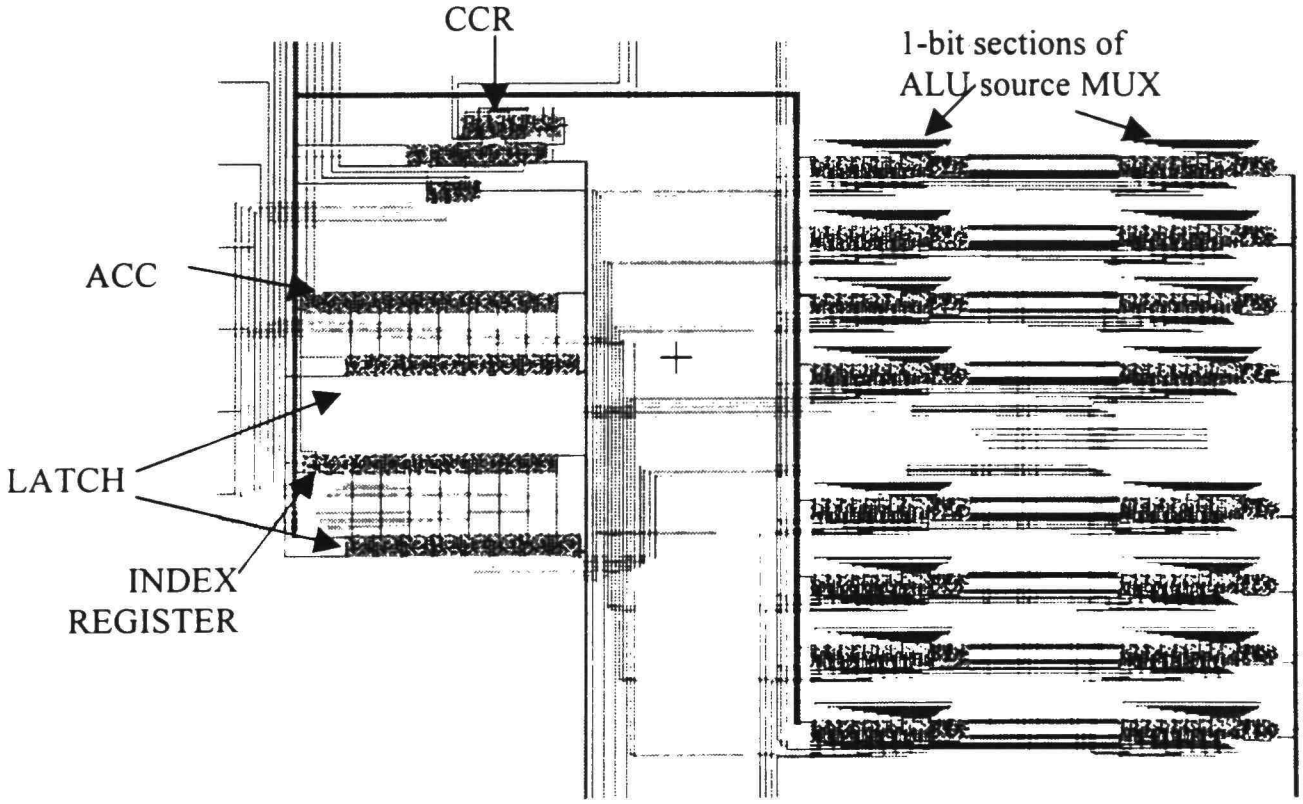


Figure 4.12 ALU control unit and CCR

All the individual layout diagrams are shown in Figures 4.7 through 4.11. All the layouts are laid out using standard inverter, nand, nor, xor and Dflip-flops. Figure 4.12 shows the overall ALU control unit and CCR.

Table 4.1 Pin names

Pin #	**Pin name**	**Description**	**type**
1	Vdd		power
2 - 9	A0 – A7	ALU Bus	Input
10 - 17	X0 – X7	Index Register	Output
18 – 25	PC0 – PC7	Program counter	Input
26 – 33	D0 – D7	Internal Data Bus	Input
34	Gnd		Ground
35 - 42	R7 – R0	ALU Port1	Output
43 – 50	S7 – S0	ALU Port2	Output
51	PCLOAD	To PC Load	Output
52 – 54	MUXA2 – A0	ALU MUX control lines	Input
55	OEN	OEN Bit of CCR	Input
56 – 58	CCRA0 – A2	CCR control lines	Input
59	CCRLD	CCR Load	Input
60	C8	Carry out from ALU	Input
61	C7	From ALU	Input
62	ACCL	Accumulator load	Input
63	XL	Index register load	Input
64	CLOCK	System clock	Input

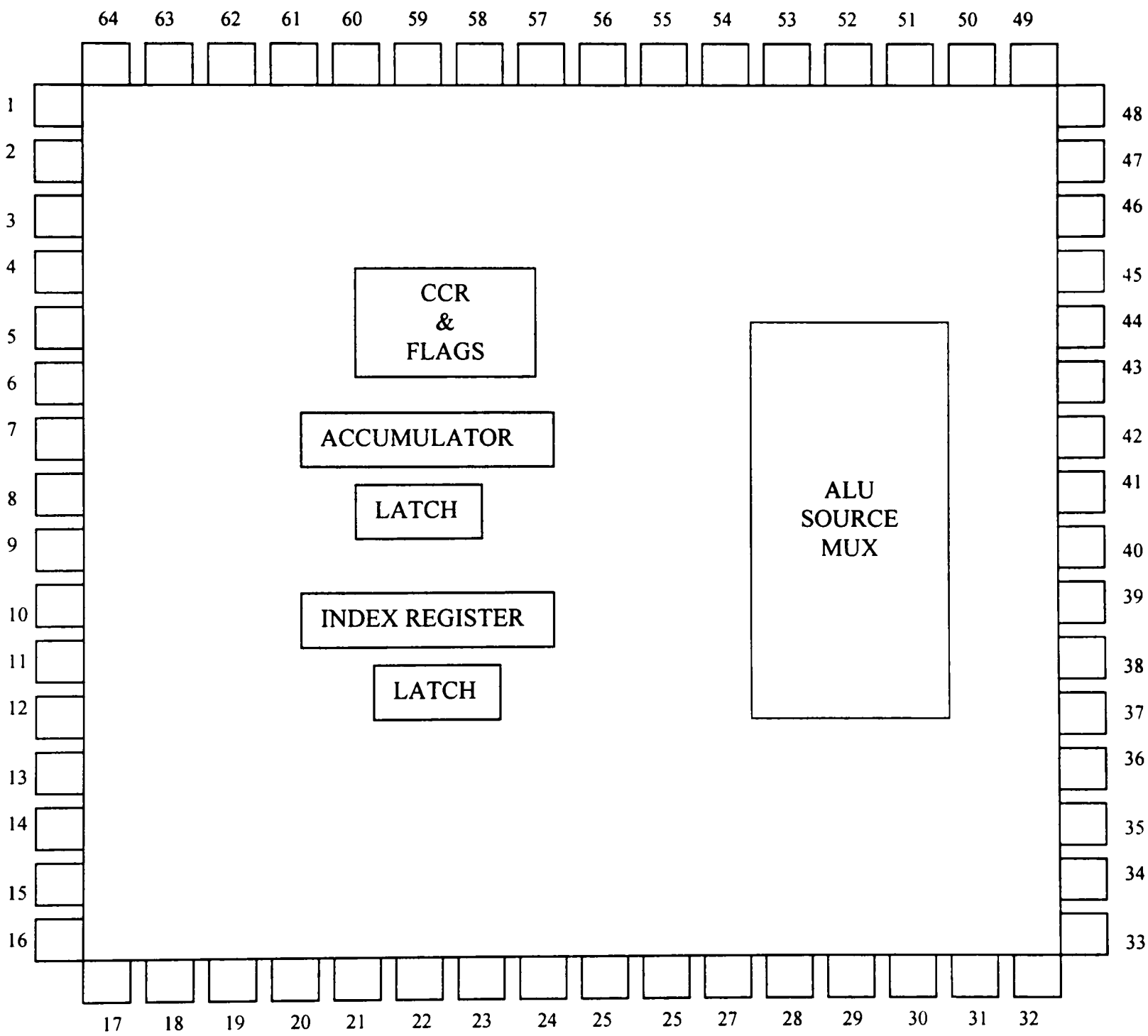


Figure 4.13 Pin diagram and Floor plan of ALU Control unit

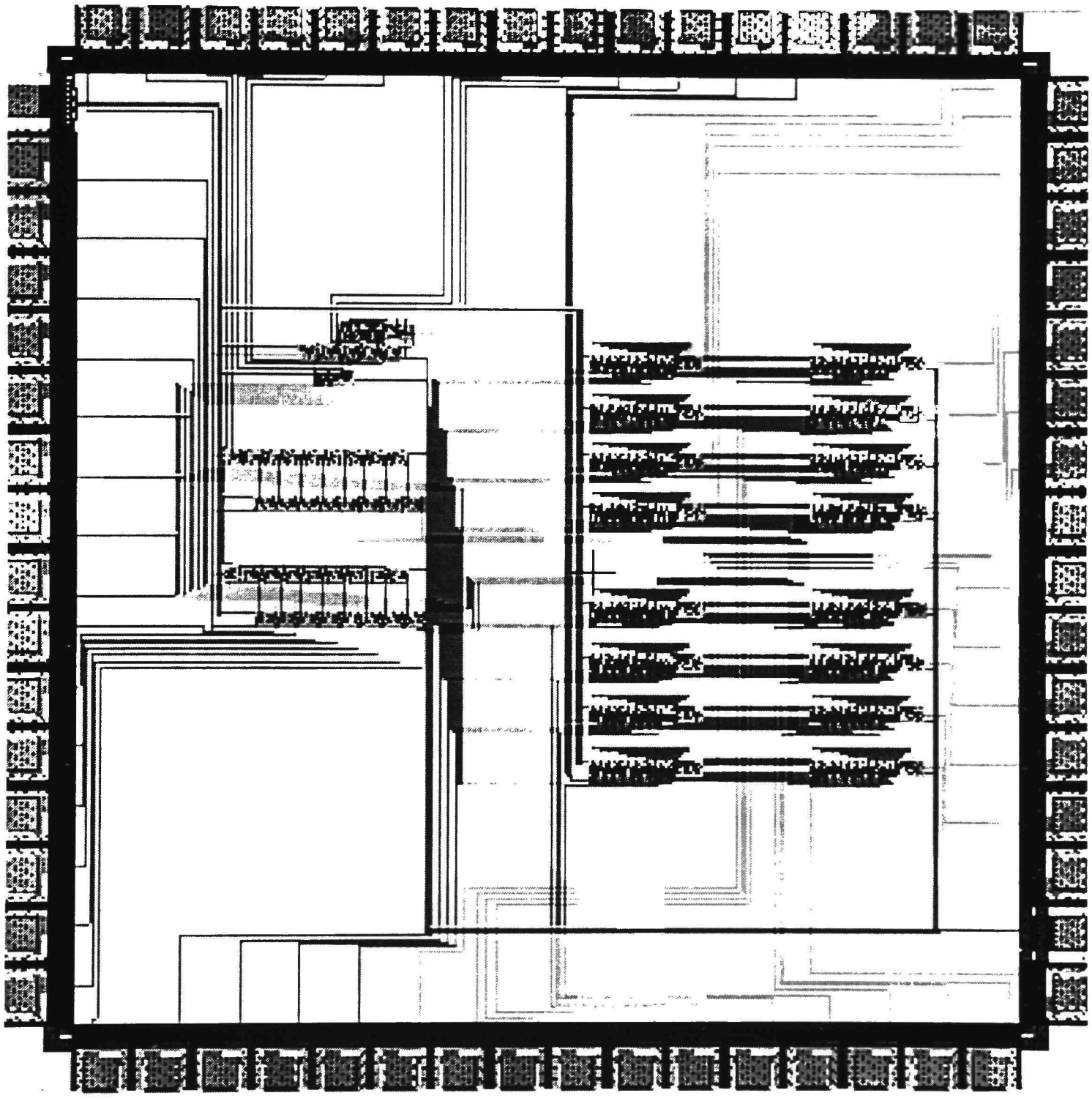


Figure 4.14 Layout of ALU Control Unit

Standard cells were placed horizontally to construct CCR, LATCH and MULTIPLEXER. 8-bit latch is used as accumulator, index register. These latches are placed vertically on left side of the chip because accumulator and index register take external inputs. Multiplexer is placed towards right of these because the inputs to multiplexer come from the output of the latches and multiplexer outputs are taken out as output of the chip. Condition Codes Register checks the flags of ALU operation. So ALU bus bits are needed for CCR, which are available on the left side of chip. So CCR is kept on the top of accumulator and index register.

All design rules (minimum length, width and space, etc.,) were followed to make an electrically correct layout. Care was also taken to minimize the parasitic capacitance and resistance with appropriate interconnection methods as described earlier. The control logic signals were routed both vertically and horizontally and vertically to connect those with all necessary functional blocks. The power supply and ground distribution were made in a manner similar to Figure 4.6. Care was taken to minimize clock skew.

ALU control unit has total of 64 pins, among them 37 are input pins, 25 are output pins and 2 are supply pins. Pin names and types are explained in Table 4.1.

Inverters drive all the outputs coming out of ALU control unit and CCR. Drive strength of inverter is 1X (minimum sized inverter). The delay between output and input of multiplexer is 0.6nsec.

4.3 Cascading

ALU control unit can be cascaded to make a larger unit. Just simply connect required number of units parallel and tie ALU Source MUX control lines together. Figure 4.15 shows the block diagram for 32-bit operation.

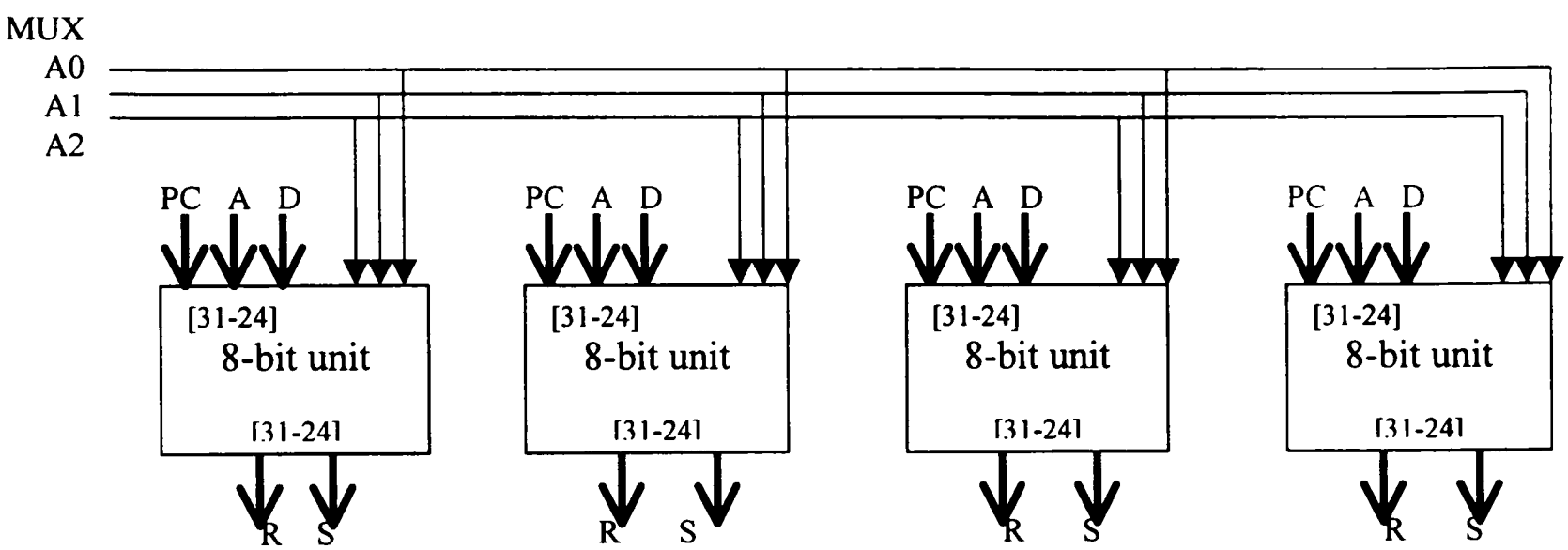


Figure 4.15 Cascading of 8-bit units

Left most block acts as most significant bits block and right most block acts as least significant bits block. R and S out puts go to the respective ALU input ports. Numbers with in each block indicate bit numbers. Separate logic should be implemented for calculating zero flag (simple NOR of all the bits of ALU bus). C32, C31 (carry bits) and A31 (MSB of ALU bus bit) are applied to CCR. CCR does not need any cascading.

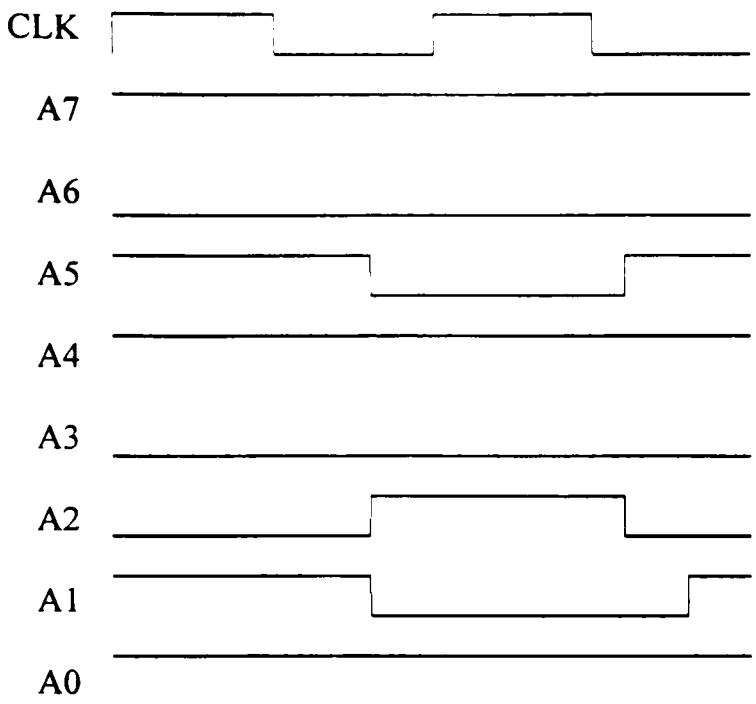
CHAPTER V

RESULTS

An 8-bit "MODULAR ALU CONTROL UNIT" was successfully implemented using a cell based implementation technique in Logic Works 3.0.3. The physical layout was laid out in L-Edit (SCNA Technology is used with LAMBDA = 0.6 micron) and extracted into PSpice. Successful simulations were run on extracted files in PSpice. The simulation results are shown in Figures 5.1 and 5.2 for two sets of input data. All the inputs are given as two sets, first in first clock cycle and second in second clock cycle.

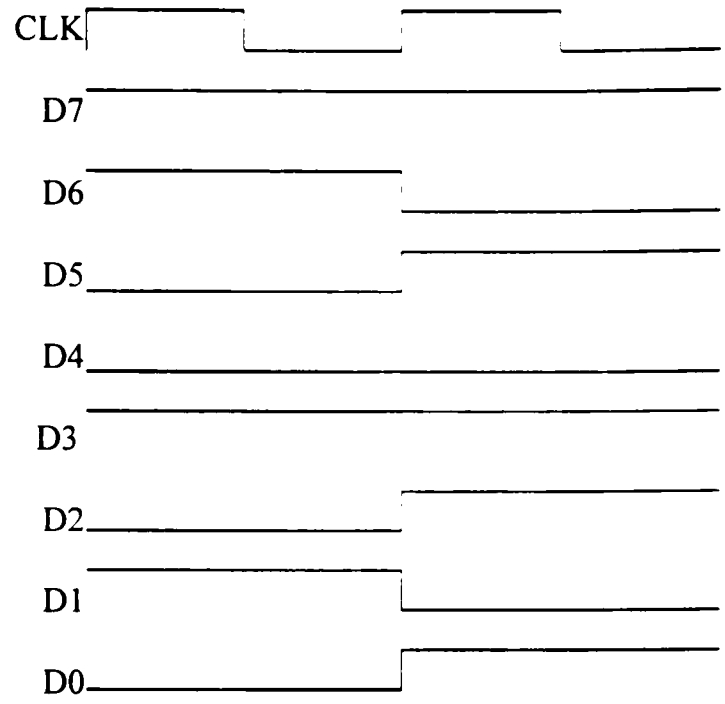
For the first set, test vectors are as shown below. The ALU bus for the first clock cycle is [A7-A0]=[10110011] and for the second clock cycle it is set to [10010101]. The Internal data bus for the first clock cycle is [D7-D0]=[11001010] and it is changed to [10101101] in the second clock cycle. Similarly [PC7-PC0] is set to [10100010] in first cycle and it is changed to [11101001] in the second clock cycle. The System clock starts with a rising edge. Multiplexer control bits MuxA2, MuxA1, MuxA0 are set to 001 for the first clock cycle and 111 for the first clock cycle. Accumulator and Index register load bits are set to 0 for both the clock cycles. Table 3.2 shows that for multiplexer control bits of 001, the outputs should be the Data bus and Accumulator bus. Figures 5.1(f) and 5.1(g) confirm that output "R" =[11001010], which is D, and the output "S" = [10110011], which is ACC, at the falling edge of the first clock cycle. For the second clock cycle multiplexer control lines are 111. Table 3.2 shows that the outputs should be X and 0. Figure 5.2 confirms that during the second clock cycle falling edge, R=[10010101], which is X, and S=[00000000], which is 0. At CCR, control signals are

given as OEN = 1, [A2-A0]=[011], CCRL = 0 and carry signals as C7=0, C8=1 for the first clock cycle. Table 3.3 shows that when OEN=1 and control lines are 011 then PCLOAD should be $\overline{V}$. Overflow V can be calculated as V=C8 (XOR) C7 = 1, so $\overline{V}$ =0. Figure 5.1 confirms the result. For the second clock cycle OEN = 1, [A2-A0]=[111], CCRL =0. From table 3.3 PCLOAD should be $\overline{N}$. N (negative flag) is set with A7 bit of accumulator (here A7 = 1). So PCLOAD should be 0. Figure 5.1 confirms the result.



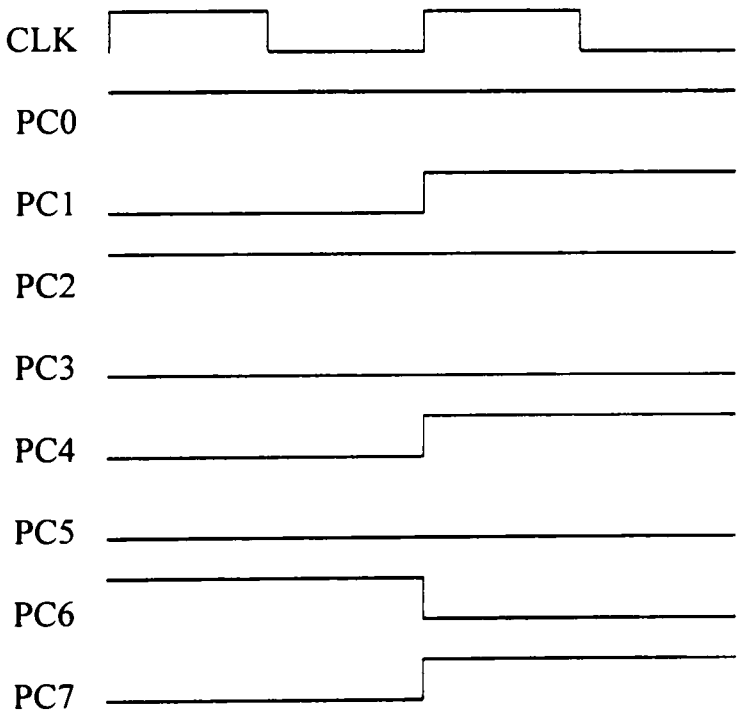
A = 10110011 A = 10010101

(a) ALU Bus



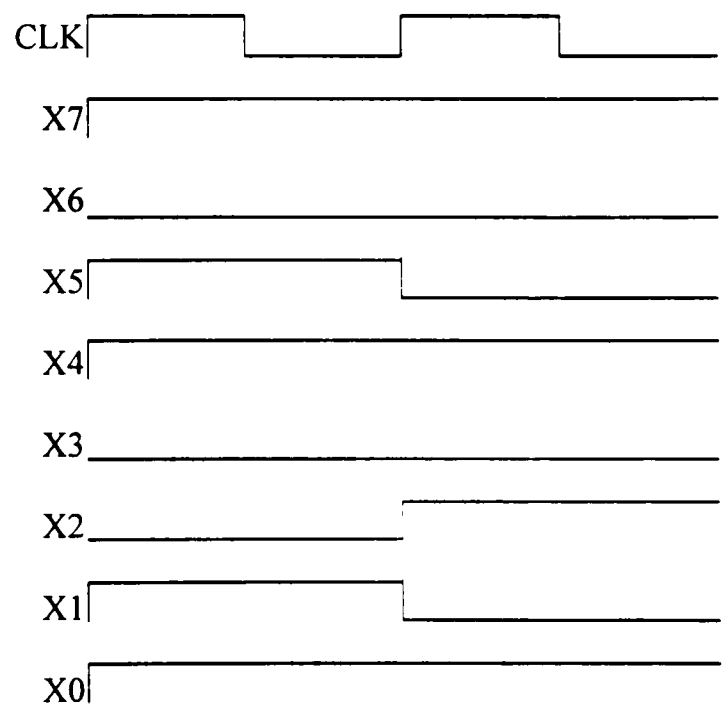
D = 11001010 D = 10101101

(b) Internal Data Bus



PC = 10100010 PC = 11101001

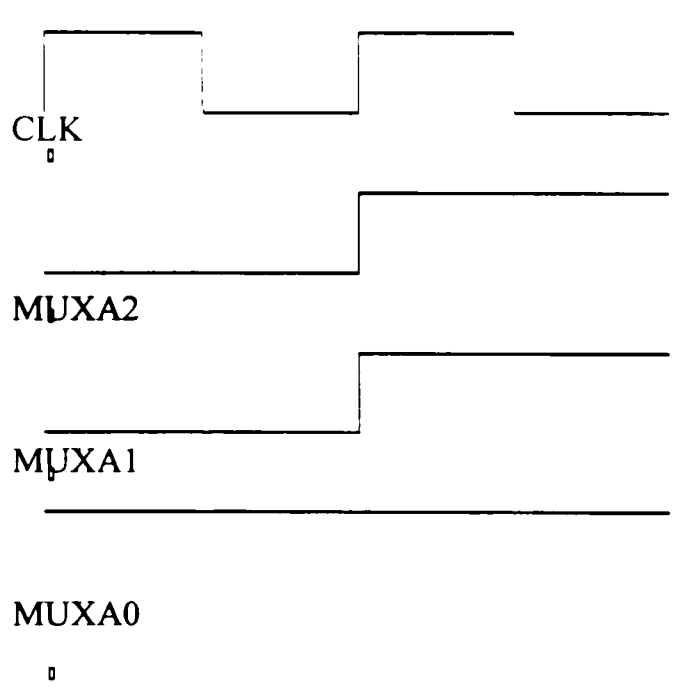
(c) Program counter



X = 10110011 X = 10010101

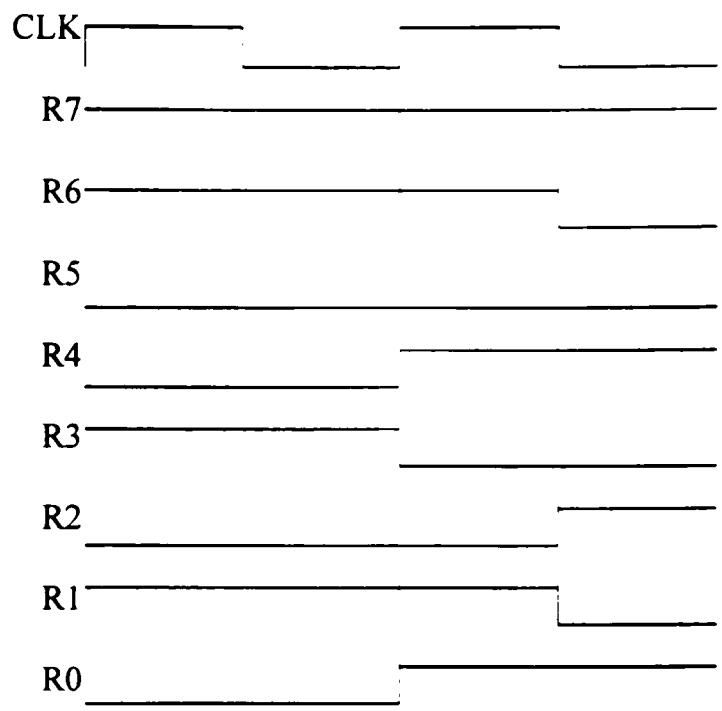
(d) Index Register

Fig: 5.1Simulation results for first set of data



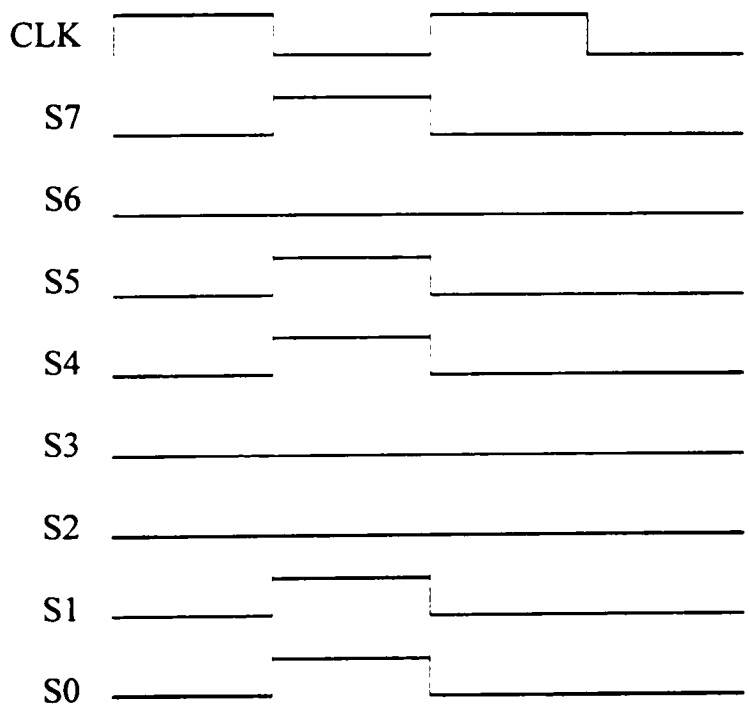
A2,A1,A0 = 001 A2,A1,A0 = 111

(e) Multiplexer Control lines



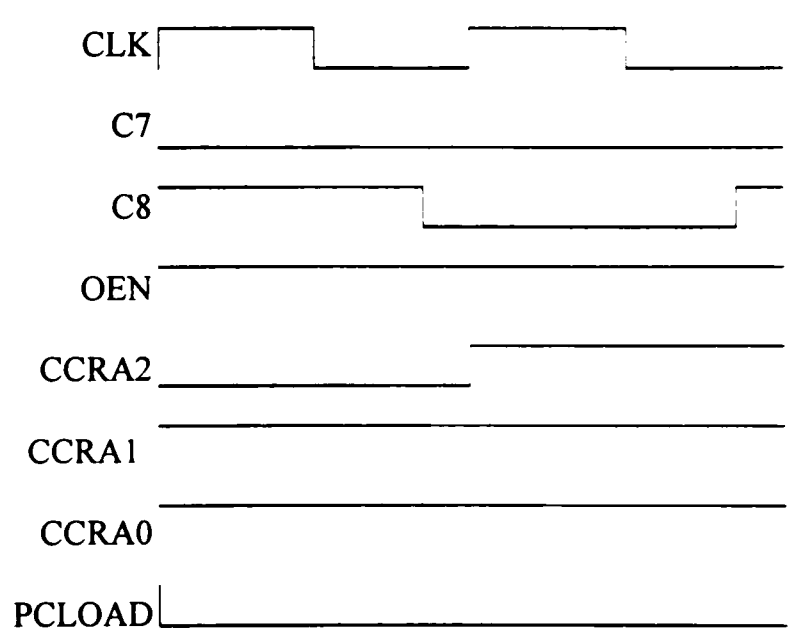
R = 11001010 R = 10010101

(f) Output R



S = 10110011 S = 00000000

(g) Output S



OEN = 1 OEN = 1

A2,A1,A0 = 011 A2,A1,A0 = 111

C7 = 0, C8 = 1

(h) CCR inputs and output

Fig 5.1 Continued

For the second test the test vectors are given as follows. Here clock starts with 0

1. First clock cycle:

   Inputs:

   A = [10010011], C7 = 1, C8 = 1

   D = [11001010]

   PC = [10100010]

   MUXA2 –A0 = 001

   ACCL = 0, XL = 0

   OEN = 0, CCRL = 0

   CCRA2 –A1 =111;

   Expected outputs:

   R = D (From Table 2.2, when MUXA2-A0 = 001, R=D and S=A)

   S = A

   X = A (X is simply loaded with "A" at rising edge of clock if XL=0)

   PCLOAD = 0 (From Table 2.3, when OEN = 0, PCLOAD =0);

   Simulation outputs:

   R = [11001010] = D

   S = [10010011] = A

   X = [10010011] = A

   PCLOAD = 0.

2. Second clock cycle:

   Inputs:

   A = [10110101], C7 = 1, C8 = 0

D = [10101101]

PC = [11101001]

MUXA2 –A0 = 110

ACCL = 0, XL = 0

OEN = 1, CCRL = 0

CCRA2 –A1 =110;

Expected outputs:

R = A (From Table 2.2, when MUXA2-A0 = 110 then R=A and S=PC.)

S = PC

X = A (X is simply loaded with "A" at rising edge of clock if XL=0)

PCLOAD = 1 (From Table 2.3, when OEN = 1, CCRA2-A0 = 110, then

PCLOAD = N. With A7 bit of ALU bus as 1, PCLOAD should be 1.);

Simulation outputs:

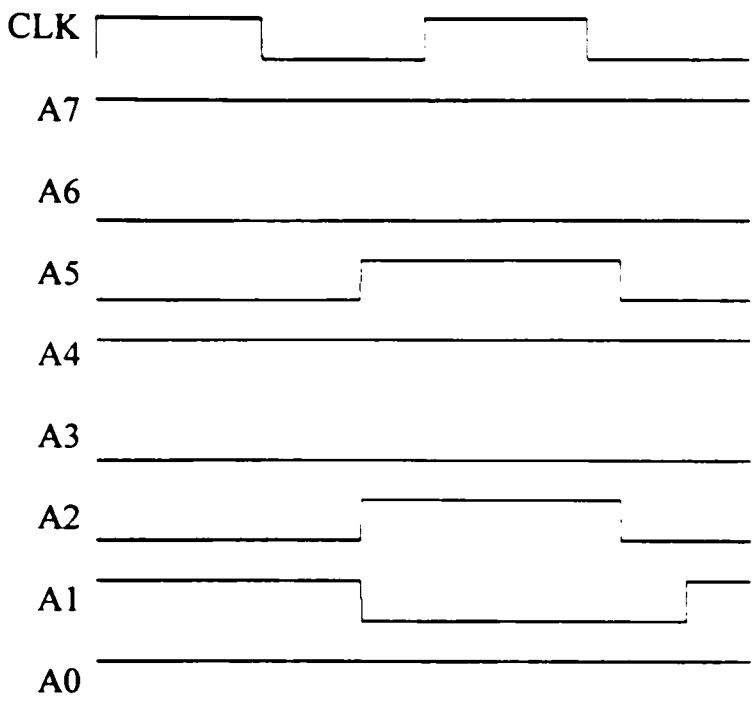
R = [10110101]= A

S = [11101001] = PC

X = [10110101] = A

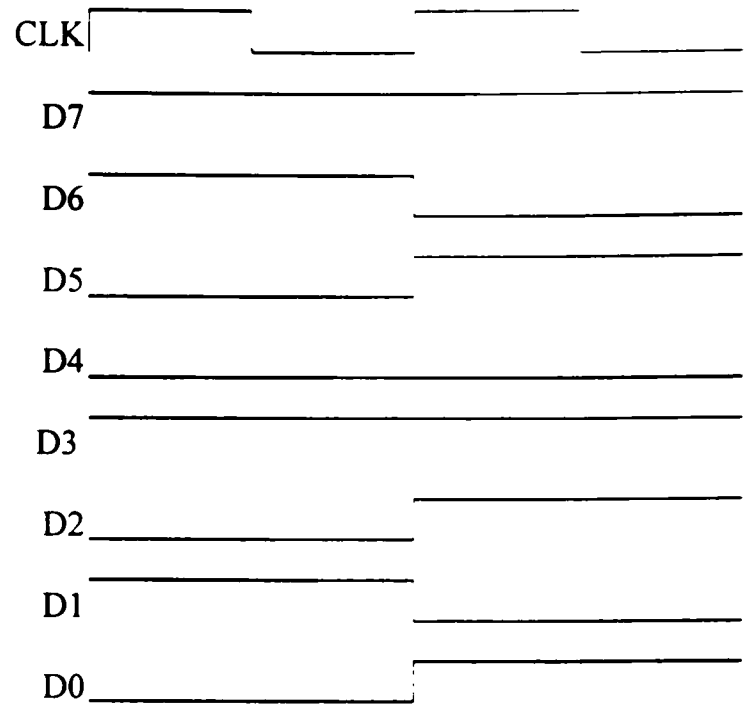
PCLOAD = 1.

There are some glitches in the waveforms of outputs. For example consider the case of output R1. For the first half of the waveform R1 = D1 = 1. At the end of the first clock cycle D1 changes from 1 to 0. So waveform R1 goes from 1 to 0 at the end of first clock cycle. But at the beginning of the second clock cycle output R1 = A1. A1 is 1 during the first half of the second clock cycle. So the waveform comes back to 1 in no time. So there is a glitch.



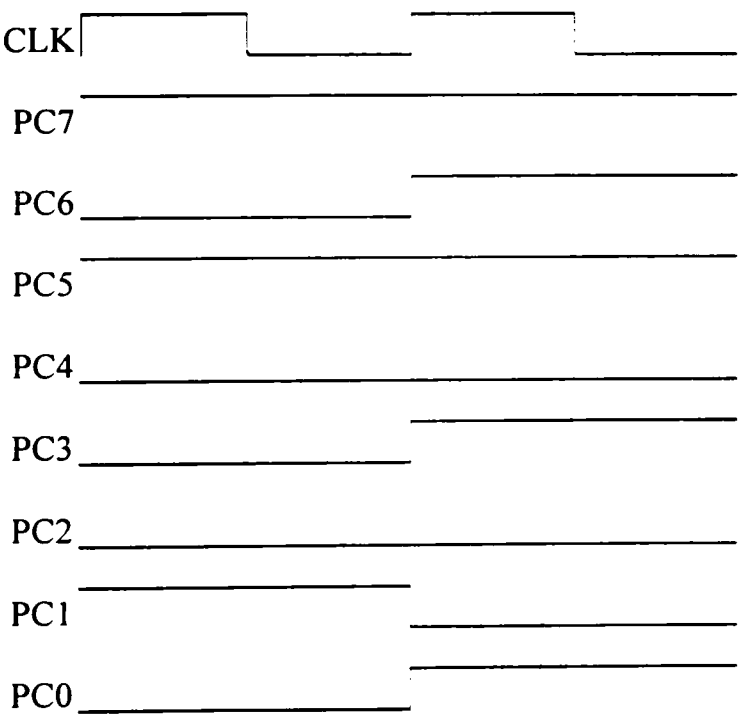
A = 10010011 A = 10110101

(a) ALU Bus



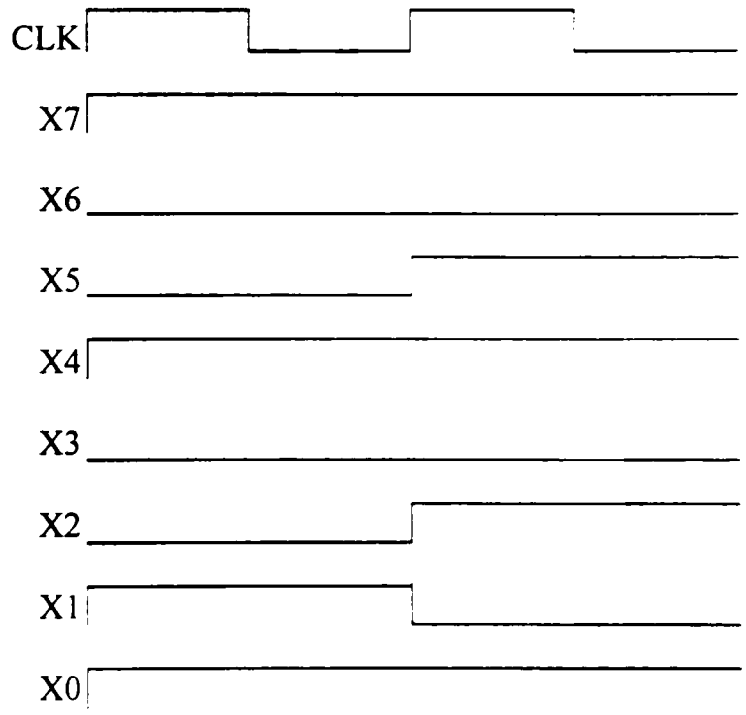
D = 11001010 D = 10101101

(b) Internal Data Bus



PC = 10100010 PC = 11101001

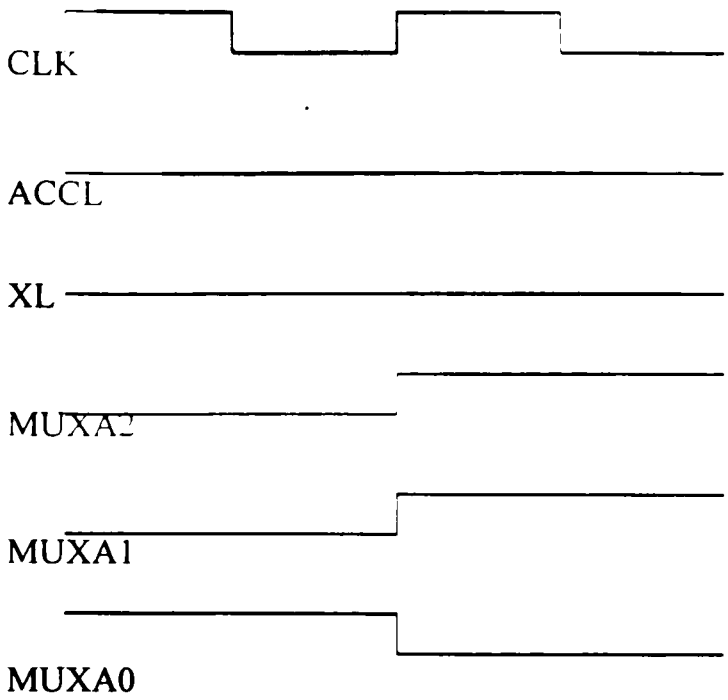
(c) Program counter



X = 10010011 X = 10110101

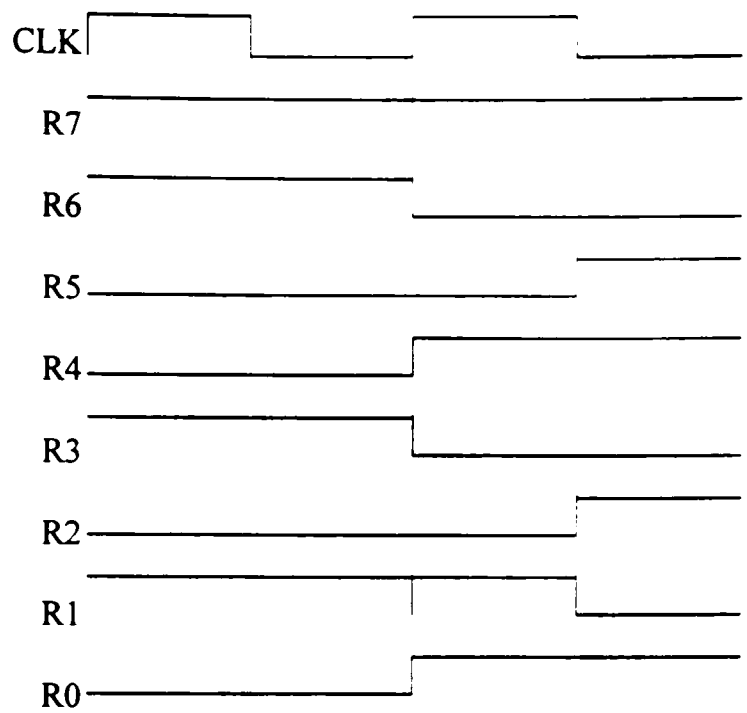
(d) Index Register

Fig: 5.2 Simulation results for second set of data



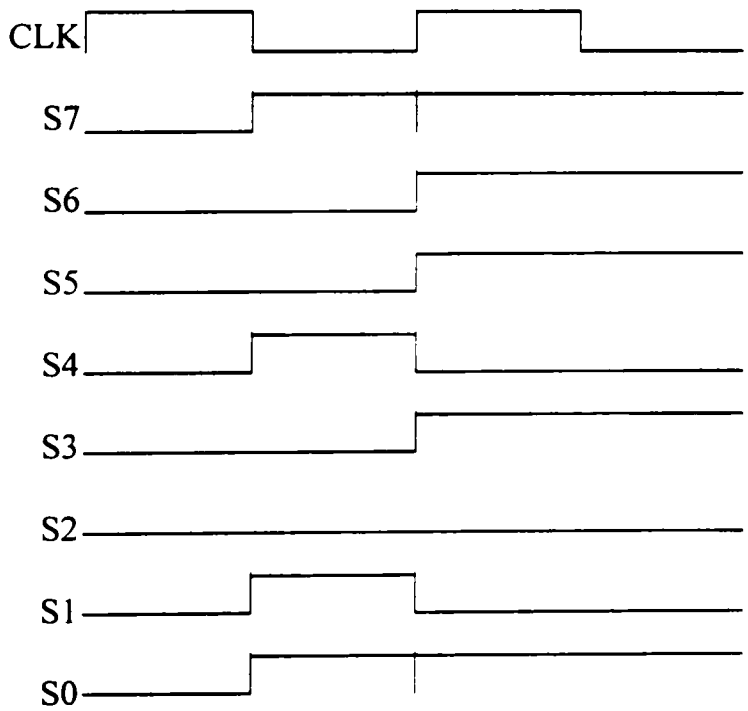
A2,A1,A0 = 001 A2,A1,A0 = 110

(e) Multiplexer Control lines



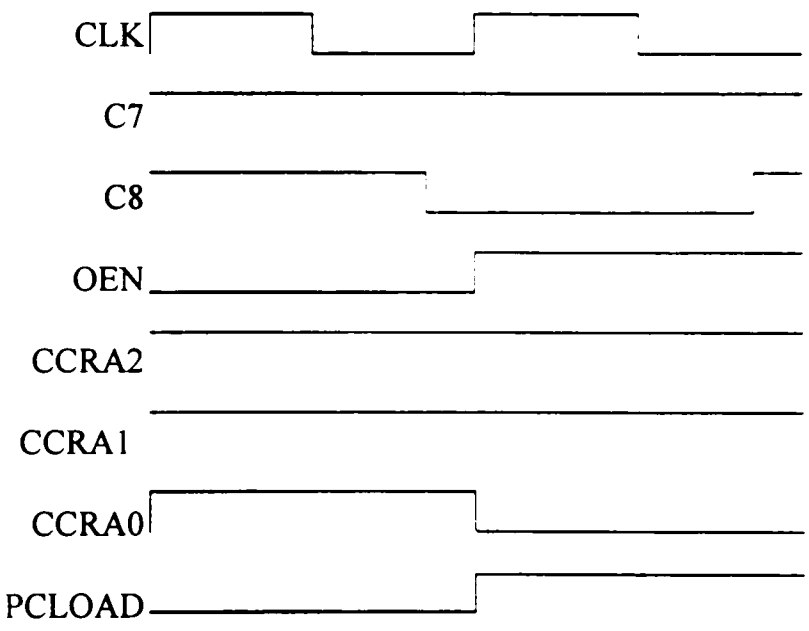
R = 11001010 R = 10110101

(f) Output R



S = 10010011 S = 11101001

(g) Output S



OEN = 0 OEN = 1

A2,A1,A0 = 111 A2,A1,A0 = 110

C7 = 1, C8 = 1 C7 = 1, C8 = 0

(h) CCR inputs and output

Fig 5.2 Continued

CHAPTER VI

CONCLUSIONS

The proposed 8-bit ALU control unit has been designed, laid out and simulated successfully. Simulation results confirm the expected results. This design has been developed as a cell and saved in a custom library of L-Edit for any further hierarchical design.

Any large-scale digital design expects to achieve the following objective:

- Simplicity in design,
- Lower cost,
- Higher reliability,
- Flexibility,
- Short design time.

In designing ALU control unit all the above points are considered. Logic design, floor plan and layout have been made as simple as possible because simplicity is one of important criteria of building blocks. Data-path, control logic, interconnections, power supply and ground connections have been done efficiently to minimize the real estate. The layout and simulation tools used are of low cost.

The ALU control unit meets most of the advantages of a typical building block. This is a LSI level design because about 2900 transistors have been employed. This offers flexibility in various applications. This can be cascaded for higher number of bits.

This design has used total of 64 pins. As number of pins is more, more chip area is needed which is nothing but more silicon area. Cost per transistor increases if it is

manufactured as chip. Also cascading of ALU control unit has some problem. Flag circuit for zero flag should be designed again for specific number of bits, though accumulator, index register and multiplexer can be used for higher number of bits. CCR can be used without any changes for higher number of bits.

REFERENCES

1. M. Morris Mano, Computer System Architecture, 1993, Prentice-Hall, Inc., Englewood Cliffs, N J.

2. Neil Weste, and Kamran Eshraghian, Principles of CMOS VLSI Design, 1988, Addison-Wesley Publishing Company, Menlo Park, California.

3. Michael John Sebastian Smith, Application-Specific Integrated Circuits, 1997, Addison-Wesley Publishing Company, Menlo Park, California.

4. Jan M Rabaey, Digital Integrated circuits: a design perspective, 1996, Prentice-Hall Inc., Englewood Cliffs, N J.

5. Ramesh S. Gaonkar Micro Processor Architecture, Programming and Applications with 8085, Third edition, Penram International

6. John P. Uyemura, Physical design of CMOS Integrated Circuits Using L-Edit™, 1998, PWS Publishing Company, Boston Massachusetts.

7. "Modular Arithmetic Logic Shift Unit" by Mazharul Islam.

8. John W. Carter, Microprocessor Architecture and Microprogramming: A state machine approach, 1996, Prentice Hall, Englewood Cliffs, NJ.

9. D.E. White, Logic Design for Array-Based Circuit, 1992, Academic Press Inc., San Diego, CA.

10. "8-bit Micro Controller" by Bobby K Abraham

PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University or Texas Tech University Health Sciences Center, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copyright infringement.

Agree (Permission is granted.)

Student's Signature _____ Date _____

Disagree (Permission is not granted.)

Student's Signature _____ Date _____