

ACHIEVER'S EDUCATION CONSULTANT
08141710097
SUMMARY OF CIT344 WITH PAST QUESTIONS

CIT344: Introduction to Computer Design is a 3-credit unit course for students studying towards acquiring the Bachelor of Science in

The course is divided into 6 modules and 21 study units. What You Will Learn in This Course
The overall aim of this course is to boost your knowledge of logic designs, microprocessors and assembly language programming. In the course of your studies, you will be equipped with definitions of common terms, characteristics and applications of logic designs. You will also learn number systems and codes, memory devices, microprocessors and finally, loops and subroutines in assembly language.

Course Aim This course aims to give you an in-depth understanding of computer designs. It is hoped that the knowledge would enhance your expertise in logic designs.

Course Objectives

You should learn them carefully before proceeding to subsequent units. Therefore, it is useful to refer to these objectives in the course of your study of the unit to assess your progress. In this way, you can be sure that you have done what is required of you by the end of the unit.

These texts listed below will be of enormous benefit to you in learning this course

Digital Logic & State Machine Design. V The assignment file will be given to you in due course. In this file, you will find all the details of the work you must submit to your tutor for marking. The marks you obtain for these assignments will count towards the final mark for the course.

The presentation schedule included in this course guide provides you with important dates for completion of each tutor-marked assignment. You should therefore endeavour to meet the deadlines. You are expected to take note of the facts, information and problem solving gathered during the course. The tutor-marked assignments must be submitted to your tutor for formal assessment, in accordance to the deadline given.

The work submitted will count for 40% of your total course mark. At the end of the course, you will need to sit for a final written examination. This examination will account for 60% of your total score.

Tutor-Marked Assignments

You need to submit all the TMAs. When you have completed each assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless in extraordinary cases.

Final Examination and Grading The final examination for CIT344 will be of last for a period of 3 hours and have a value of 60% of the total course grade. The examination will consist of questions which reflect the self-assessment exercise and tutormarked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. You might find it useful to review your TMAs vi and comment on them before the examination.

[...]

100% of Course Marks Course Overview This indicates the units, the number of weeks required to complete them and the assignments.

[...]

How to Get the Most Out of This Course In distance learning, the study units replace the university lecturer. You are provided with exercises at appropriate points, just as a lecturer might give you an inclass exercise. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next to this is a set of learning objectives.

These learning objectives are meant to guide your studies. The moment a unit is finished, you must go back and check whether you have achieved the objectives. If this is made a habit, then you will increase your chances of passing the course. The main body of the units also guides you through the required readings from other sources.

Self assessment exercises are provided throughout the unit, to aid personal studies and answers are provided at the end of the unit. Working through these self tests will help you to achieve the objectives of the unit and also prepare you for tutor marked assignments and examinations. You should attempt each self test as you encounter them in the units. Note the time you are expected to spend on each unit and how the assignment relates to the units.

You need to gather together all these information in one place such as a diary, a wall chart calendar or an organiser. Whatever method you choose, you should decide on and write in your own dates for working on each unit. Once you have created your own study schedule, do everything you can to stick to it. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.

You will almost always need both the study unit you are working on and one of the materials recommended for further readings, on your desk at the same time. Work through the unit, the content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be encouraged to read from your set books. Keep in mind that you will

learn a lot by doing all your assignments carefully.

They have been designed to help you meet the objectives of the course and will help you pass the examination. Review the objectives of each study unit to confirm that you have achieved them. When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule.

When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Pay particular attention to your tutor's comments on the tutormarked assignment form and also written on the assignment when the assignment is returned to you. Check that you have achieved the unit objectives and the course objectives.

Facilitators/Tutors and Tutorials

There are 8 hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail your tutor marked assignment to your tutor well before the due date.

At least two working days are required for this purpose. They will be marked by your tutor and returned to you as soon as possible. You should try your best to attend the tutorials. This is the only chance to have face-to-face contact with your tutor and ask questions which are answered instantly.

You can raise any problem encountered in the course of your study. You will learn a lot from participating actively in tutorial discussions.

[...]

The number system is the basis of computing. It is a very important foundation for understanding the way the computer system works. In this unit, we will talk about decimal and binary number system. Endeavour to assimilate as much as possible from this unit – especially, the conversion from one number system to another.

Using these single digits, ten different values can be represented. Values greater than ten can be represented by using the same digits in different combinations. The left most digit has the highest weight and the right most digit has the lowest weight. 275 can be written in the form of an expression in terms of the base value of the number system and weights.

Fractions in Decimal Number System In a Decimal Number System the fraction part is separated from the integer part by a decimal point. The integer part of a number is written on the left hand side of the decimal point. The fraction part is written on the right side of the decimal point. Fractions in decimal number system are also represented in terms of the base value of the

number system and weights.

The weights of the fraction part are represented by $3 \times 10^2 + 8 \times 10^1 + 2 \times 10^0 + 9 \times 10^{-1} + 1 \times 10^{-2} = 300 + 80 + 2 + 0$.

[...]

Binary as the name indicates is a base-2 number system having only two numbers 0 and 1. Decimal Equivalents of Binary Number System Counting in binary number system is similar to counting in decimal number systems. In a decimal number system a value larger than 9 has to be represented by 2, 3, 4, or more digits. Similarly, in the binary number system a binary number larger than 1 has to be represented by 2, 3, 4, or more binary digits.

The number 100112 can be rewritten in terms of the expression

In a decimal number system the integer part and the fraction part of a number are separated by a decimal point. In a binary number system the integer part and the Fraction part of a binary number can be similarly represented separated by a decimal point. The binary number 1011.1012 has an integer part represented by 1011 and a fraction part 101 separated by a decimal point. The binary number 1011.1012 can be written in terms of an expression using the base value 2 and weights $2^3, 2^2, 2^1, 2^0, 2^{-1}, 2^{-2}$ and 2^{-3} .

[...]

Such numbers are represented and used in floating-point numbers notation. Digital systems use the binary number system to represent numbers. Therefore these systems should be capable of performing standard arithmetic operations on binary numbers. The only difference between the two additions is the range of numbers used.

In binary addition, four possibilities exist when two single bits are added together. The four possible input combinations of two single bit binary numbers and their corresponding sum and carry outputs are specified in Table 2. Addition of Two Single Bit Binary Numbers The first three additions give a result 0, 1 and 1 respectively which can be represented by a single binary digit. The fourth addition results in the number 2, which can be represented in binary as 102.

Thus, two digits are required. The number 2 is the sum part and 1 is the carry part. Any number of binary numbers having any number of digits can be added together. Subtracting two single bit binary numbers results in a difference bit and a borrow bit.

The four possible input combinations of two single bit binary numbers and their corresponding difference and borrow outputs are specified in Table 3. The first digit 7 cannot be subtracted from 1, therefore 10 is borrowed from the next significant digit. Four possible combinations of two single bit binary numbers and their products are listed in table 4.

Multiplication of two Single Bit Binary Numbers

the final sum expression. The
ion having weights
ctively. Weights 20 and 23
o these weights have the

non having weights
ctively. Weights 20
o these weights hav

to these weights ha

ents of binary

of 2. A quicker

of 2. A quicker

The next term becomes the weight and the binary equation

The next term becomes the weight and the binary equation

The next term becomes the weight and the binary equation

= Sum Term – Highest The sum-of-weights method requires mental arithmetic and is a quick way of converting small decimal numbers into binary. With practice large decimal numbers can be converted into binary equivalents.

Repeated Division-by-2

In this method, the decimal number to be converted into its binary equivalent is repeatedly divided by 2. The divisor is selected as 2 because the decimal number is being converted into binary a base-2 number system. In the repeated-division method the decimal number to be converted is divided by the base number, in this particular case 2. A quotient value and a remainder value is generated, both values are noted down.

The remainder value in all subsequent divisions would be either a 0 or a 1. The quotient value obtained as a result of division by 2 is divided again by 2. The new quotient and remainder values are again noted down. The process of repeated division stops when the quotient value becomes zero.

Converting Decimal to Binary using Repeated Division by 2 Method

Quotient after division
Remainder after division The process of determining the binary equivalent of a decimal number

392 is illustrated in a tabular form above. Reading the numbers in the remainder column from bottom to top 110001000 gives the binary equivalent of the decimal number 392. Explain with the aid of good examples, the different methods of converting binary numbers to decimal numbers. Binary indicates a base-2 number system having only two numbers 0 and 1. In this unit, we discussed decimal and binary number systems, manipulation of their fractions, binary arithmetic and conversion of decimal to binary and vice versa. Hoping that you understood the topics discussed, you may now attempt the questions below.

TYPES OF NUMBER SYSTEMS II

Larger decimal numbers would require lengthier binary strings. Writing such long string is tedious and prone to errors. The hexadecimal number system is a base 16 number system and therefore has 16 digits and is used primarily to represent binary strings in a compact manner. Hexadecimal number system is not used by a digital system.

The hexadecimal number system is for our convenience to write binary strings in a short and concise form. Hexadecimal Equivalents of Decimal and Binary Numbers
Decimal Binary
Hexadecimal Decimal Binary
Hexadecimal Counting in hexadecimal is similar to the other number systems already discussed. The maximum value represented by a single hexadecimal digit is F which is equivalent to decimal 15.

Counting using Hexadecimal Numbers

The binary string is divided into small groups of 4-bits starting from the least significant

bit. Each 4-bit binary group is replaced by its hexadecimal.

[...]

Each digit of the hexadecimal number is replaced by an equivalent binary string of 4-bits.

3 hexadecimal number

There are two methods to convert from decimal to hexadecimal. The first method is the indirect method and the second method is the repeated division method. A decimal number can be converted into its hexadecimal equivalent indirectly by first converting the decimal number into its binary equivalent and then converting the binary to Hexadecimal. Convert decimal numbers to binary by repeatedly dividing the decimal number by 2.

A decimal number can be directly converted into hexadecimal by using repeated division. The decimal number is continuously divided by 16. The conversion of decimal 2096 to hexadecimal using the repeated division-by-16 method is illustrated in Table 3.

Hexadecimal Equivalent of Decimal Numbers Using Repeated Division Number Quotient after division Remainder after division

Converting hexadecimal numbers to decimal is done using two methods. The first method is the indirect method and the second method is the The indirect method of converting hexadecimal number to decimal number is to first convert hexadecimal number to binary and then binary to decimal. A hexadecimal number can be directly converted into decimal by using the sum of weights method. The conversion steps using the sum-ofweights method are shown.

CA02 hexadecimal number $C \times 16^3 + A \times 16^2 + 0 \times 16^1 + 2 \times 16^0$ Writing the number in an expression

51714 Decimal equivalent Numbers represented in hexadecimal can be added and subtracted directly without having to convert them into decimal or binary equivalents. The rules of addition and subtraction that are used to add and subtract numbers in decimal or binary number systems apply to hexadecimal addition and subtraction. Hexadecimal addition and subtractions allows large binary numbers to be quickly added and subtracted.

[...]

The octal number is a base 8 number system with digits ranging from 0 to 7. Octal number system was prevalent in earlier digital systems and is not used in modern digital systems especially when the hexadecimal number is available. Each octal and the octal equivalents are listed below Octal Equivalents of Decimal and Binary Numbers Counting in octal is similar to counting in any other number system. The maximum value represented by a single octal digit is 7.

For representing larger values a combination of two or more octal digits has to be

used. Thus, decimal 8 is represented by a combination of 108.

Counting using Octal Numbers

The binary string is divided into small groups of 3-bits starting from the least significant bit. Each 3bit binary group is replaced by its octal equivalent.

[...]

Each digit of the octal number is replaced by an equivalent binary string of 3-bits.

001 111 010 110 Replacing each octal digit by its 3-bit binary

There are two methods to convert from decimal to octal. A decimal number can be converted into its octal equivalent indirectly by first converting the decimal number into its binary equivalent and then converting the binary to octal. Convert decimal numbers to binary and hexadecimal by repeatedly dividing the decimal number by 2 and 16 respectively.

A decimal number can be directly converted into octal by using repeated division. The decimal number is continuously divided by 8. The conversion of decimal 2075 to octal using the repeated division-by8 method is illustrated in Table 6. Converting octal numbers to decimal is done using two methods.

An octal number can be directly converted into decimal by using the sum of weights method. The conversion steps using the sum-of-weights method are shown.

4 x 83 + 0 x 82 + 3 x 81 + 3 x 80 Writing the number in an expression

Decimal equivalent Numbers represented in octal can be added and subtracted directly without having to convert them into decimal or binary equivalents. The rules of addition and subtraction that are used to add and subtract numbers in decimal or binary number systems apply to octal addition and subtraction. Octal addition and subtractions allows large binary numbers to be quickly added and subtracted.

Difference 1 6 1 1

Explain how you can convert hexadecimal and octal numbers to binary and decimal numbers and vice versa.

Convert these binary numbers to hexadecimal and octal and back to binary

A code in computer parlance is a generic term for program instructions, used in two general senses. The first sense refers to human-readable source code, which include the instructions written by the programmer in a programming language. Consider the decimal number range +7 to -8. These positive and negative decimal numbers can be represented by the 2's complement representation.

The decimal number range +7 to -8 is represented using an excess-8 code that assigns 0000 to -8 the lowest number in the range and 1111 to

Excess-8 code is obtained by adding a number to the lowest number -8 in the range such that the result is zero. The number 8 is added to all the remaining decimal numbers from -7 up to the highest number +7. The excess-8 represented is presented below.

Range 7 to 8

Binary Coded Decimal code is used to represent decimal digits in binary. The remaining six 4-bit combinations 1010, 1011, 1100, 1101, 1110 and 1111 are considered to be invalid and do not exist.

BCD Decimal BCD

The two digits are considered to be separate. A telephone keypad having the digits 0 to 9 generates BCD codes for the keys pressed. Most digital systems display a count value or the time in decimal on 7-segment LED display panels. Since the numbers displayed are in decimal, therefore the BCD Code is used to display the decimal numbers.

Consider a 2-digit 7-segment display that can display a count value from 0 to 99.

BCD Addition

Multi-digit BCD numbers can be added together. The two 2-digit BCD numbers are added and generate a result in BCD. In the example, the least significant digits 3 and 5 add up to 8 which is a valid BCD representation. Similarly, the most significant digits 2 and 4 add up to 6 which also is a valid BCD representation.

Consider the next example where the least significant numbers add up to a number greater than 9 for which there is no valid BCD code

If a carry results, it is added to the next most significant digit.

0110 6 is added A carry is generated which is added to the result of the next most significant digits The answer is 0111 0001

The Gray code does not have any weights assigned to its bit positions. The Gray code is not a positional code. The Gray code is different from the unsigned binary code as successive values of Gray code differ by only one bit. Table 3 shows the Gray code representation of decimal numbers 0 to 9.

The ASCII code is a 7-bit code representing 128 unique codes which represent the alphabet characters A to Z in lower case and upper case, the decimal numbers 0 to 9, punctuation marks and control characters.

ASCII codes 011 0000 to 011 1001 represents numbers 0 to ASCII codes 1100001 to 1111010 represent lower case alphabets a to z ASCII codes 100 0001 to 101 1010 represent upper case alphabets A to Z

The 7-bit ASCII code only has 128 unique codes which are not enough to represent some graphical characters displayed on computer screens. An 8-bit code extended ASCII code gives 256 unique codes. The extended 128 unique codes represent graphic symbols which have become an unofficial standard as vendors use their own interpretation of these graphic codes. Binary Coded Decimal code is used to represent decimal digits in binary. The ASCII Code is a 7-bit code representing 128 unique codes which represent the alphabet characters A to Z in lower case and upper case, the decimal numbers 0 to 9, punctuation marks and control characters. In this unit, we talked about various codes, their characteristics and how they can be used.

COMBINATIONAL LOGIC CIRCUIT

A combinational circuit does not require memory so the output depends only on the current inputs. Combinational circuits help in reducing design complexity and reduce the chip count in a circuit. It does not require memory so the output depends only on the current inputs. In this unit, we discussed combinational logic circuit, its analysis and how it can be designed.

Briefly explain the term combinational logic circuit, how it is analysed and designed. The output of the single bit adder circuit generates a sum bit. 3 Half-adder Sum & Carry Out Boolean Expression The sum and carry out expressions of the half-adder can be determined from the function table. The single bit binary adder has two bit output.

One bit represents the Sum between numbers A and B. The other bit represents the carry bit generated due to addition. The function table of the full-adder has three input columns representing the two single bit numbers A, B and the carry in bit. The function table also has two output columns representing the sum bit and carry out bit. Full-adder Function Table and Circuit Implementation Full-adder Sum & Carry out Boolean Expression The sum and carry out expressions of the full-adder can be determined from the function table.

The sum output of the second half-adder is The carry out of the second half-adder is Cin
The output of the OR gate is in $AB + C_{in}$ $C_{in} =$

Four single bit full-adders are connected together to form a 4-bit parallel adder capable of adding two 4-bit binary numbers.

Connection Diagram Logic Symbol

Explain with the aid of diagrams how you can form a full-adder from half-adders. State what a multiplexer is state how a multiplexer is designed discuss the applications of multiplexers state what a demultiplexer is discuss the applications of demultiplexers. The main use of the Multiplexer is to select data from multiple sources and to route it to a single destination. In a

computer, the ALU combinational circuit has two inputs to allow arithmetic operations to be performed on two quantities.

The two quantities are usually stored in different set of registers. The inputs of the two multiplexers are connected to the output of each of the multiple registers. The outputs of the two multiplexers are connected to the two inputs of the ALUs. The multiplexers are used to route the contents of any two registers to the ALU inputs.

A 4 X 1 Multiplexers and its Truth Table

Data input multiplexer selects one of many inputs to be directed to an output.

Data Input MUX Implementation The 4 x 1 MUX can be implemented with pass gates as follows

Implementation of the 4x1 MUX with Pass Gates Multiplexers can be efficiently implemented using the majority function and even- parity function.

Even-Parity Function

Other than its use as a data router, they are used as a parallel to serial converter, logic function generator and also for operation sequencing. Data Routing A two - digit 7-Segment display uses two 7-Segments Display digits connected to two BCD to 7-Segment display circuits. Similarly, the BCD input 1001 representing the numbers 9 is applied at the inputs of the LSD display circuit. The circuit uses two BCD to 7-Segment decoder circuits to decode each of the two BCD inputs to the respective 7-Segment display outputs.

The display circuit can be implemented using a single CD to 7Segment IC and a multiplexer. Parallel to Series Conversion In a digital system, binary data is used and represented in parallel. When two binary numbers are added, the two numbers are represented in parallel and the parallel adder works and generates a sum term which is also in parallel. Transmission of information to remote locations through a piece of wire requires that the parallel information be converted into serial form.

In a serial data representation, data is represented by a sequence of single bits. All communication systems set up across remote locations use serial transmission. An 8-bit parallel data can be converted into serial data by using an 8-to1 multiplexer such as 74X151 which has 8 inputs and a single output. The 8-bit data which is to be transmitted serially is applied at the 8 inputs I0-7 of the multiplexer.

A three bit counter which counts from 0 to 7 is connected to the three select inputs S0, S1 and S2. The counter is connected to a clock which sends a clock pulse to the counter every 1 millisecond. Initially, the counter is reset to 000, the I0 input is selected and the data at input I0 is routed to the output of the multiplexer. On receiving the clock signal after 1 millisecond the counter increments its count from 000 to 001 which selects I1 input of the multiplexer and routes the data present at the input to the output.

Similarly, at the next clock pulse the counter increments to 010, selecting I2 input and routing the data to the output. The output of the multiplexer is connected to the wire through which the serial data is transmitted. Serial to Parallel Conversion Logic Function Generator Multiplexers can be used to implement a logic function directly from the function table without the need for simplification. The select inputs of the multiplexer are used as the function variables.

The inputs of the multiplexer are connected to logic 1 and 0 to represent the missing and available terms.

Logic Function Generator Based on 3-Variable Logic Function Table Operation Sequencing

A paint manufacturing plant might have a four step process to manufacture paint. The second step cannot start before the first step has completed. Similarly, the third and fourth steps of the paint manufacturing process cannot proceed unless steps two and three have completed. It is not necessary that each of the manufacturing steps is of the same duration.

On receiving the signal, the next step of the manufacturing process is initiated. The entire sequence of operations is controlled by a multiplexer and a decoder circuit.

The counter output is connected to the select input of the multiplexer and the inputs of the decoder which selects the multiplexer input I0 is and activates the Decoder output Y

2 its output is set to logic 1, which is routed by the multiplexer to the clock input of the 2-bit counter which increments to the next count. This continues until Process 4 signals its completion after which the Decoder and the Multiplexer is deselected completing the manufacturing process. Control of Manufacturing Process through Operation Sequencing Demultiplexer A multiplexer has several inputs. It selects one of the inputs and routes the data at the selected input to the single output.

The demultiplexer selects one of the several outputs and routes the data at the single input to the selected output. A demultiplexer is also known as a data distributor. A 1 x 4 Demultiplexer The circuit if compared to that of the 2-to-4 decoder. The decoder enable input is used as the demultiplexer data input.

A demultiplexer is not available commercially. The only difference between the two is the addition of the data input line, which is used as enable line in the 2-to-4 decoder circuit.

Multiple Destinations

The output of the ALU has to be stored in one of the multiple registers or storage units. The Data input of the demultiplexer is connected to the output of the ALU. Each output of the demultiplexer is connected to each of the multiple registers. By selecting the appropriate output data from the ALU is routed to the appropriate register for storage.

The second use of the demultiplexer is the reconstruction of parallel data from the incoming serial data stream. Serial data arrives at the data input of the demultiplexer at fixed time intervals. A counter attached to the Select inputs of the demultiplexer routes the incoming serial bits to successive outputs where each bit is stored. When all the bits have been stored, data can be read out in parallel.

Multiplexers are used in a wide variety of applications. It selects one of the several outputs and routes the data at the single input to the selected output. It is used to connect a single source to multiple destinations. It is also used for the reconstruction of parallel data from the incoming serial data stream. In this unit, we explained about multiplexers and demultiplexers, how they are designed and their applications.

III

In this unit, we shall discuss decoders and encoders, designing with them and their applications in our everyday lives. Outputs to indicate the presence of the multi-bit code. Often, decoders have an enable line that turns on outputs or leaves them off.

Implementation of a Boolean Function

We can use multiple decoders to form a larger decoder. Below is a 3-to-8 decoder implemented with two 2-to-4 decoders. A 3-to-8 Decoder Implemented with Two 2-to-4 Decoders Decoders have two major uses in Computer Systems.

Computers have different internal and external devices like the Hard

Disk, CD Drive, Modem, Printer, etc. Each of these different devices is selected by specifying different codes. A decoder is used to uniquely select or deselect the appropriate devices. Instruction Decoder Computer programs are based on instructions which are decoded by the computer hardware and implemented. These instruction codes are decoded by an instruction decoder to generate signals that control different logic circuits like the ALU and memory to perform these operations.

Representing the selected input.

Example: An input of 00010000 in an encoder will give

Encoders are useful when the occurrence of one of several disjoint events needs to be represented by an integer identifying the event. An example is in resolving interrupt requests. In this unit, we examined decoders and encoders and also how to design with them.

SEQUENTIAL LOGIC CIRCUITS

The outputs of the combinational circuits also cannot be stored. The use of a memory element which is capable of storing digital inputs and outputs is an important part of all practical digital circuits. The result is temporarily stored in one of the two memory elements used to store the

numbers 2 and 3.

We have two types of sequential circuits namely

Synchronous sequential circuits – Their behaviour is determined by the values of the signals at only discrete instants of time. Asynchronous sequential circuits – Their behaviour is immediately affected by the input signal changes. In this unit we explained sequential logic circuit and its types.

Extensively discuss sequential logic circuit and its types using diagrams

4 The Gated S-R Latch. Describe how a latch and flip-flop are designed explain the various types of latch and flip-flop explain how the types of latch and flip-flop are designed state the various uses of latches and flip-flops explain the various applications of the latch and flip-flop. At the latch input allows the latch to change its state. When the latch is set to a certain state it retains its state unless the inputs are changed to set the latch to a new state.

An S-R Latch is implemented by connecting two NAND gates together. The output of each NAND gate is connected to the input of the other NAND gate. The outputs of the two NAND gates are the Q and its complement $\bar{Q}$. The circuit diagram of the NAND based S-R latch is NAND Based S-R Latch A truth-table shows the operation of the S-R NAND based latch. When inputs are $S = 1$ and $R = 1$ the next state output Q_{t+1} remains the same as the previous state output Q_t .

When inputs are $S = 0$ and $R = 1$ the output $\bar{Q}$ is set to 1. When inputs are $S = 1$ and $R = 0$ the output Q is set to 0. Inputs $S = 0$ and $R = 0$ are not applied as they place the latch in an invalid state. The NAND gate based S-R latch has active-low inputs.

Truth-Table of NAND based S-R Latch. Connections are identical to that of the NAND based latch. The S and R inputs have been switched. NOR Based S-R Latch The truth table of the NOR gate based latch is shown.

$S = 0$ and $R = 0$ the next state output Q_{t+1} remains the same as the previous state output Q_t . When inputs are $S = 0$ and $R = 1$ the output $\bar{Q}$ is set to 0. When inputs are $S = 1$ and $R = 0$ the output Q is set to 1. Inputs $S = 1$ and $R = 1$ are not applied as they place the latch in an invalid state.

The NOR gate based S-R latch has active-high inputs.

Truth-Table of NOR based S-R Latch

Comparing the operation of the NOR based and NAND based S-R latches. The NAND based latch has active-low inputs, whereas NOR based latch has active-high inputs. Both the S-R latches are set to logic 1 when the set input is activated and the reset input is inactive. Both the latches are set to logic 0 when the reset input is activated and the set input is inactive.

The latches maintain the output state when both the set and reset inputs are inactive. For both the

latches both the set and reset inputs cannot be activated simultaneously as this leads to invalid output states. The logic symbols of the two latches are shown below. NOR Based Active-High and NAND Based Active-Low S-R Latches.

Digital systems use switches to input values and to control the output. For example, a keypad uses 10 switches to enter decimal numbers 0 to 9.

When a switch is closed the switch contacts physically vibrate or

The switch bounce causes the voltage at the output of the switch to vary between logic low and high for a very short duration before it settles to a steady state. The variation in the voltage causes the digital circuit to operate in an erratic manner. An S-R latch connected between the switch and the digital circuit prevents the varying switch output from reaching the digital circuit. The output voltage settles to logic 0 when a solid contact is made.

The active-low input S-R latch shown in figure 4 prevents the output signal from varying between logic 1 and 0. The 74LS279 IC has four S-R latches which can be used independently. The circuit diagram of the gated S-R latch is shown in figure 6. In the gated S-R circuit, the S and R inputs are applied at the inputs of the NAND gates 1 and 2 when the enable input is set to activehigh.

If the enable input is disabled by setting it to logic low the output of NAND gates 3 and 4 remains logic 1, whatever the state of S and R inputs. Thus, logic 1 applied at the inputs of NAND gates 1 and 2 keeps the Q and Q outputs to the previous state. The logic symbol of a gated SR latch is shown in figure 6.

EN

Truth-Table of a Gated S-R Latch One way to eliminate the undesirable condition of the indeterminate state in the SR latch is to ensure that inputs S and R are never equal to 1 at the same time. This is done by the D latch. Thus, the D latch has the ability to hold data in its internal storage. The output follows changes in the data input as long as the control input is gated.

The circuit is often called a transparent latch. Because of this the output of a latch cannot be applied directly to the input of the same or another latch when all the latches are triggered by a common clock source. Flip-flops are used to overcome this difficulty. Flip-flops are synchronous bi-stable devices, known as bi-stable multivibrators.

Flip-flops have a clock input instead of a simple enable input. The output of the flip-flop can only change when appropriate inputs are applied at the inputs and a clock signal is applied at the clock input. Flip-flops with enable inputs can change their state at any instant when the enable input is active. Digital circuits that change their outputs when the enable input is active are difficult to design and debug as different parts of the digital circuit operate at different times.

In synchronous systems, the output of all the digital circuits changes when a clock signal is

applied instead of the enable signal. The change in the state of the digital circuit occurs either at the low-to-high or high-to-low transition of the clock signal. The low to high or high to low transition of the clock is considered to be an edge. Three different types of edge-triggered flip-flops are generally used in digital logic circuits.

J-K edge-triggered flip-flop

Each flip-flop has two variations, that is, it is either positive edge triggered or negative edge triggered. The edge-detection circuit which allows a flip-flop to change its state on either the positive or the negative transition of the clock is implemented using a simple combinational circuit.

Truth-Table of Positive and Negative Edge -Triggered The logic symbols of a positive edge and a negative edge triggered D

Logic Symbol of Positive and Negative Edge Triggered The truth tables of the two D flip-flops are also shown. Truth-Table of Positive and Negative Edge Triggered The J-K flip-flop is widely used in digital circuits.

The circuit diagram of a J-K edge-triggered flip-flop is shown in figure Edge-Triggered J-K Flip-Flop

Four set of inputs are applied at J and K, the effect on the outputs is as follows. With $Q=1$ and $Q=0$, on a clock transition the outputs of NAND gates 3 and 4 are set to logic 1. The output of the NAND gate 4 is set to 0 as all three of its inputs are at logic 1. The output of the NAND gate 4 is also set to 1 as the input of the NAND gate 4 is connected to $Q=0$.

The output of the NAND gate 3 is also set to 1 as its input connected to Q is at logic 0. The output of the NAND gate 3 is set to 0 as all its input are at logic 1.

With $Q=1$ and $Q=0$, on a clock transition the output of the NAND gates

The truth tables of the positive and negative edge triggered J-K flipflops are shown in Table 7.

Truth-Table of Positive and Negative Edge Triggered J-K

The logic symbols of the J-K flip-flops are shown in figure 15. The flip-flops are set-reset to some initial state by using asynchronous inputs known as Preset and Clear inputs. Since these inputs change the output to a known logic level independently of the clock signal therefore these inputs are known as asynchronous inputs. The circuit diagram of a J-K flip-flop with Preset and Set is shown below.

Asynchronous inputs are shown. To preset the flipflop to $Q=1$ and $Q=0$ the PRE input is set to 0 which sets the Q output to 1 and the output of NAND gate 4 to 1. The CLR input is set to 1 which sets the Q output to 0 as all three inputs of the NAND gate 2 are set to 1. The flip-flop is cleared to $Q=0$ and $Q=1$ by setting the PRE input is set to 1 and the CLR input is to 0.

The CLR input set to 0 sets $Q=1$ it also sets the output of NAND gate 3 to 1. The PRE input set to 1 sets the output Q to 0. When the PRE and the CLR inputs are used inputs J and K have no effect on the operation of the flip-flop.

J-K Flip-Flop with Asynchronous Preset and Clear

Logic Symbol of a J-K Flip-Flop with Asynchronous The truth table of a J-K flip-flop with asynchronous inputs is shown in Table 8. Truth Table of J-K Flip-Flop with Asynchronous The 74HC74 Dual Positive-Edge triggered D Flip-Flop an Integrated Circuit. The 74HC74 has dual D-flip-flops with The 74HC112 Dual Positive-Edge triggered J-K flip-flop an Integrated Circuit. The 74HC112 has dual J-K-flip-flops with Master-Slave flip-flops have become obsolete and are being replaced by edge triggered flip-flops.

Master-Slave flips have two stages each stage works in one half of the clock signal. The inputs are applied in the first half of the clock signal. The outputs do not change until the second half of the clock signal. As mentioned earlier the use of edge-triggered flipflop is to synchronise the operation of a digital circuit with a common clock signal.

The master-slave setup also allows digital circuits to operate in synchronisation with a common clock signal. The circuit diagram of the master-slave J-K flip-flop is shown in figure 18 below. The Master-Slave flip-flop is composed of two parts the Master and the Slave. Both the Master and the Slave are Gated S-R flip-flops.

Master-Slave Flip-Flop The truth-table of the master-slave flip-flop is shown in table 9 below. 6 Flip-Flop Operating Characteristics The performance of the flip-flop is specified by several operating characteristics. Pulse width The propagation delay time is the interval of time when the input is applied and the output changes.

Maximum Clock frequency

Delays are measured. Propagation delay t_{PLH} measured with respect to the triggering edge of the clock to the low-to-high transition of the output. On a low-to-high transition of the output that occurs due to the clock Propagation delay t_{PHL} measured with respect to the triggering edge of the clock to the high-to-low transition of the output. On a high-to-low transition of the output that occurs due to the clock Propagation Delay t_{PLH} measured with respect to the leading edge of the preset input to the low-to-high transition of the output.

On a high-to-low transition of the preset signal the flip-flop changes its output state to logic high. On a high-to-low transition of the clear signal the flip-flop changes its output state to logic low. The propagation delay is measured at.

50% transition mark on the triggering edge of the clear signal and the 50% mark on the high-to-low transition of the output that

Set-Up Time When a clock transition occurs at the clock input of a flip-flop the output of the

flip-flop is set to a new state based on the inputs. For the flip-flop to change its output to a new state at the clock transition, the input should be stable. Maximum Clock Frequency A flip-flop can be operated at a certain clock frequency. Each signal has to be of a specified duration for correct operation of the flipflop.

The clock signal is specified by minimum high time and minimum low time. A flip-flop consumes power during its operation. The power consumed by a flip-flop is defined by $P = V_{cc} \times I_{cc}$. The flip-flop is connected to +5 volts and it draws 5 mA of current during its operation, therefore the power dissipation of the flip-flop is 25 mW.

A digital circuit is made of a number of gates, functional units and flipflops. Latches are used to maintain stable data at the input of the multiplexer. The time required to convert parallel data to serial data depends upon the number of parallel bits. A byte parallel data requires 8-bit storage and 8 clocks are required to convert it into serial data.

The demerit in a gated D-latch based circuit is the extended enable time. D-latches are enabled data applied at the input of the latches can change. D-latch is said to work in transparent mode when the enable signal is activated. D-latch operates in the latched mode when the enable signal is inactive.

The conversion should only start when the enable signal has been deactivated and the 8-bit data has been stored in the latches. A better and a precise parallel to serial converter circuit uses edge triggered D-flip-flops. The 8-bit data to be converted into serial data is stored precisely at the clock transition. Thus, if the data changes after the clock transition it has no effect on the data stored in the D flip-flop.

D-Flip-Flops used for Parallel Data Storage

Microprocessor use multi-bit flip-flops to store information. These multi-bit flip-flops are known as registers. These registers for example, can store data generated at the output of the ALU. The registers can also be used to exchange or copy data, see figure 21.

A register is a set of flip-flops connected in parallel to store multi-bit binary information. Some digital applications require that the inputs be applied in a certain sequence to activate an output.

J-K Flip-Flop Connected to Respond to a Particular Input Sequence

Same clock source should be used to maintain synchronization between the three parts. A clock frequency can be divided by 2 using a J-K flip flop. The J-K inputs of the flip-flop are connected to logic high . At each clock transition the output of the flip-flop toggles to the alternate state.

MHZ and 1 MHZ signal using two J-K flip-flops connected together.

Bit Right Shift Register J-K Flip-Flop used as a Counter

Counters which count up or countdown are commonly used in digital circuits. An up counter

counts up from 0 to 10 increments to the next higher count value on the application of each clock signal. Similarly, a down-counter counts down to the next lower count value on the application of each clock pulse. Bit Up-Counter at the latch input allows the latch to change its state.

Three different types of edge-triggered flip-flops generally used in digital logic circuits are

In this unit we explained about latches and flip-flops. When a switch is closed the switch contacts physically vibrate or 'bounce' before making a solid contact.

REGISTERS

We shall look at the shift register, shift register counters and the applications of shift registers. The CPU's control unit coordinates and times the CPU's functions, and it uses the program counter to locate and retrieve the next instruction from memory. In digital circuits multi-bit data has to be stored temporarily until it is processed. A flip-flop is able to store a single binary bit of information.

Multiple bits of data are stored by using multiple flip-flops which have their clock inputs connected together. Thus, by activating the clock signal multiple-bits of data are stored. The shifting of data involves shifting of bits from one flip-flop to the other within the register or moving data in and out of the register. The shift operation of the binary data is carried out by applying clock signals.

Several different kinds of shift operations can be identified. At each clock transition 1-bit of serial data is shifted in and at the same instant 1-bit of serial data is shifted out. For a 4-bit shift register, 8 clock transitions are required to shift in 4-bit data and completely shift out the 4-bit data. Serial In/Shift Right/Serial Out Register The shift left and shift right shift registers are identical in their working.

They are connected differently for shift left and shift right operations. Bidirectional Shift Registers are available which allow data to be shifted left or right. The 4-bit register is configured to shift left or right by setting the RIGHT /LEFT signal to logic high or low respectively. When the register is configured to shift right, the AND gates marked 1 are enabled.

The input of the first flip-flop is connected to the serial Input, the inputs of the next three flip-flops are connected to the Q outputs of the previous flip-flops. Thus, on a clock transition data is shifted 1-bit towards the right. The serial data is shifted out of the register through output Q3. The data is shifted out in parallel by the application of a single clock signal.

The shift register has 4 parallel outputs. The circuit diagram of the Serial In/Parallel Out register is shown. The register has parallel inputs, data bits are loaded into the register in parallel by activating a load signal. The data is shifted out serially by application of clock signals.

Thus, in a 4-bit shift register, after 4 clock signals the 4-bit data is completely shifted out of the shift register. The internal circuit of a 4-bit Parallel In/Serial Out Shift register is shown. The 4-bit data is initially loaded in Parallel into the shift register by setting the SHIFT /LOAD input to logic low. The AND gates marked 2 are enabled allowing data to be applied at the inputs of the respective D flip-flops.

On a positive clock transition the data is latched by the respective flip-flops. Bit Parallel In/Serial Out Shift Register The register has parallel inputs and parallel outputs. Data is entered in parallel by applying a single clock pulse. Data is latched by the flipflops on the clock transition and is available in parallel form at the flipflop outputs.

The internal circuit of 4-bit Parallel In/Parallel Out Register is shown. A D-Flip-Flop based 4-Bit Parallel In/Parallel Out Register register. By applying clock pulses data is shifted right. By applying clock pulses data is shifted left.

When data is rotated through a register counter a specific sequence of states is repeated. Two commonly used register counters in digital logic are the Johnson Counter and the Ring Counter. Johnson Counter In a Johnson counter, the Q output of the last flip-flop of the shift register is connected to the data input of the first flip-flop. The circuit of a 4-bit, D flip-flop based Johnson Counter is shown in the figure below.

The sequence of states that are implemented by a n-bit Johnson counter are $2n$.

Bit Ring Counter

After the initialisation of the counter, the logic high set at the output of the first flip-flop is shifted right at each clock transition. With a Ring Counter circuit no decoding gates are required. Each state of the ring counter has a unique output. Sequence of States of a 4-Bit Ring Counter The major application of a shift register is to convert between parallel and serial data.

Shift registers are also used as keyboard encoders. The two applications of the shift registers are discussed.

Serial-to-Parallel Converter

A Parallel In/Serial Out shift register offers a better solution instead of using a Multiplexer-Register combination to convert parallel data into serial data. Similarly, a Serial In/Parallel Out shift register replaces a Demultiplexer-Register combination. In Asynchronous Serial data transmission mode, a character which is constituted of 8-bits is transmitted. A character is preceded by a logic low start bit.

The logic low start bit is an indication that 8 character bits are to follow and the transmission line is no longer in an idle state. After 8-character bits have been transmitted, the end of the character is indicated by two stop bits that are at logic high. The two logic bits indicate the end of the character and also set the transmission line to the idle state. Therefore, a total of 11 bits are

transmitted to send one character from one end to the other.

The 11-bit serial character format is shown. The serial data is preceded by a logic low start bit which triggers the J-K flip-flop. The output of the flip-flop is set to logic high which enables the clock generator. On each clock transition, the Serial In/Parallel Out shift register shifts in one bit data.

The One-shot resets the J-K flip-flop output Q to logic 0 disabling the clock generator and also clears the 8-bit counter to count 000.

Keyboard Encoder

We have a simple keypad encoder circuit where the 0 to 9 digit keypad was connected through a decade to BCD encoder. Pressing any keypad key enables the corresponding input of the encoder circuit which encodes the input as a 4-bit BCD output. Computer keyboards which have more keys employ a keyboard encoder circuit that regularly scans the keyboard to check for any key press. The scanning is done by organising the keys in the form of rows and columns.

With the help of a shift register based ring counter one row is selected at a time. The two counters are connected as an 8-bit Ring counter which sequences through a bit pattern having all 1's and a single.

The 8 state sequence selects one row at a time by setting it to logic

When a key is pressed, the selected column which is set to logic 0 sets the output of the NAND gate to logic 1 which triggers two One Shots. The first One Shot inhibits the clock signal to the ring counters for a short interval until the Key Code is stored. The One Shot also triggers the second One-Shot that sends a pulse to the clock input of the Key Code register. The Key Code Register stores the key ID represented as 3-bit column and 3-bit row code.

In this unit we discussed registers. Information from an input device or from the computer's memory is communicated via the bus to the central processing unit, which is the part of the computer that translates commands and runs programs.

Finite State Machines as a Restrictive Turing Machines

Modeling the Behaviour of Finite State Machine Classifiers, Acceptors, Transducers & Sequencers This unit introduces finite state machines, a primitive, but useful computational model for both hardware and certain types of software. If we restrict the head to move in only one direction, we have the general case of a finite-state machine. Much simpler languages, such as the sequences of well-balanced parenthesis strings, also cannot be recognised by finite-state machines. Simple forms of pattern matching.

Models for sequential logic circuits, of the kind on which every present-day computer and many device controllers is based. An intimate relationship with directed graphs having arcs labeled with symbols from the input alphabet. Even though each of these models can be depicted in a

different setting, they have a common mathematical basis.

The Interrelationship of Various Models with Respect to

Finite State Machines are considered deterministic if all transitions to next states are unique to a given state and its inputs. Most useful FSMs are fully deterministic, making them ideal for embedded systems software and the process of validation and verification. The difference between the two machines, as we shall explore later, are noteworthy, but not necessarily paramount to successful Finite State Machine design for software application architecture, but likely important in sequential circuit design.

Finite State Machines are defined as sharing the following characteristics

The term Mealy machine, after George H. Mealy, is also often used for transducer.

"Black Box" Processing an Input Sequence to Produce an Output Sequence

This simpler model is called a classifier, or Moore machine, after E.F. Moore. Classifier Finite State Machine. For a Mealy machine, the transition will be associated with an output. Although the Mealy Machine model may be more flexible than Moore Machine, it is the need of the system being analysed or designed that determines which of the two is most suitable.

Modeling the Behaviour of Finite State Machines

By convention, the edge detector always outputs 0 after reading the very first symbol. The initial state is identified with the overall function of the machine.

We make the assumptions that

The result of cons, in particular the first argument, becomes partially available even before its second argument is computed. Each function will return NIL if the input list is NIL, and we do not show this explicitly.

Functional code example for the edge-detector

We will use three functions, f, g, and h. Notice that f is never called after its initial use. Its only purpose is to provide the proper output for the first symbol in the input. An alternative representation is to use a single function, say k, with an extra argument, treated as just a symbol.

This argument represents the name of the function that would have been called in the original representation. The states can be represented as separate "functions", as in the functional view, or just as the value of one or more variables. However the allowable values must be restricted to a finite set. No stacks or other extendible structures can be used, and any arithmetic must be restricted to a finite range.

The program is started by calling f. Here we assume that read is a method that returns the next

character in the input stream and write character c to the output stream.

F: States $\times$ Symbols $\rightarrow$ States state-transition function

F is the state to which the machine goes when currently in state q G is the output produced when the machine is currently in state q The relationship of these functions is expressible by the following diagram.

[...]

Symbols extended output function where Symbols^* denotes the set of all sequences of symbols.

[...]

In the last equation, juxtaposition is like cons'ing on the right. In other words, F^* is the state of the machine after all symbols in the sequence x have been processed, whereas G^* is the sequence of outputs produced along the way. In essence, G^* can be regarded as the function computed by a transducer. These definitions could be transcribed into rex rules by representing the sequence x as a list with corresponding to .

This description is similar to the one used for Turing machines, except that the motion is left unspecified, since it is implicitly one direction.

Such a 4-tuple means the following

If the machine's control is in State1 and reads Symbol1, then machine will write Symbol2 and the next state of the controller will be State2.

State Transition Table for the Edge-Detector Machine

Unlike the case of Turing machines, there is no particular halting convention. Instead, the machine is always read to proceed from whatever current state it is in. Instead, we are only interested in categorizing each input sequence into one of a finite set of possibilities. Often these possibilities can be made to derive from the current state.

The states are divided up into the "accepting" states and the "rejecting" states . The machine in this case is called an acceptor or recogniser. The sequences it accepts are those given by $cF^* = 1$ that is, the sequences x such that, when started in state q_0 , after reading x , the machine is in a state q such that $c = 1$.

The structure is shown as follows

Since the input symbols are unchanging, this machine generates a fixed sequence, interpreted as either the output sequence of a transducer or the sequence of classifier outputs. An example of a sequencer is a MIDI sequencer, used to drive electronic musical instruments. The output alphabet of a MIDI sequencer is a set of 16-bit words, each having a special interpretation as

pitch, note start and stop, amplitude, etc. Although most MIDI sequencers are programmable, the program typically is of the nature of an initial setup rather than a sequential input. Any finite state machine can be shown as a graph with a finite set of nodes.

There is no other memory implied other than the state shown. The start state is designated with an arrow directed into the corresponding node, but otherwise unconnected. In the case of a transducer, the arcs are labeled / as shown below, where is the input symbol and is the output symbol. The state transition is designated by virtue of the arrow going from one node to another.

Transducer Transition from q_1 to q_2 , based on Input , Giving Output In the case of a classifier, the arrow is labeled only with the input symbol. The categories are attached to the names of the states after /. 0 and 1, we sometimes use a double-lined node for an accepting state and a single-lined node for a rejecting state.

Classifier Example

The appropriate machine type is classifier, with the outputs being in the set $\{0, 1, \text{more}\}$. The name "more" is chosen arbitrarily. We don't know how many states there will be initially. Let us use a, b, c, ... as the names of the states, with a as the start state.

Each state is labeled with the corresponding class as we go. The idea is to achieve a finite closure after some number of states have been added.

Acceptor Example

We can use the state transitions from the previous classifier. We need only designate those states that categorise there being one edge as accepting states and the others as rejecting states.

Inter-Convertibility of Transducers and Classifiers

We can describe a relationship between classifiers and transducers, so that most of the theory developed for one will be applicable to the other. Those outputs could be the outputs of an appropriately defined transducer. However, classifiers are a little more general in this sense, since they give output even for the empty sequence, whereas the output for a transducer with input is always just . A transducer T started in state q_0 is equivalent to a classifier C started in state q_0 if, for any non-empty sequence x , the sequence of outputs emitted by T is the same as the sequence of outputs of the states through which C passes.

With this definition in mind, the following would be a classifier equivalent to the edge detector transducer.

A Classifier Formally Equivalent to the Edge-Detector

This makes it impossible to attach a fixed input to either g or h . Call these resulting states g_0, g_1, h_0, h_1 . Now we can construct an output consistent classifier from the transducer. We don't need to split f , since it has a very transient character.

Its output can be assigned arbitrarily without spoiling the equivalence of the two machines. The procedure for converting a classifier to a transducer is simpler. When the classifier goes from state q to q' , we assign to the output transition the state output value c . In this unit we discussed extensively about finite state machines.

MEMORY ORGANISATION

Sequential circuits cannot operate without a memory element. Memory elements used in Sequential circuits are relatively small and store few binary bits of information. Large memories capable of storing very large amounts of information are used in computer systems. A complete unit of information is sometimes called a Word and consists of one or more bytes.

The first eight bits of binary information 11001010 in the first row are stored in eight cells. The addresses of the eight consecutive cells starting from the left most cell are , , , , , and representing the first row and columns 1 to 8 respectively.

Individual cells at row 5 and column 3 have a binary 1 and a cell at row

The 4 x 16 memory array allows data to be accessed in the form of 4-bit nibbles. The 1 x 64 array allows data to be accessed in units of 1 bit.

4 memory is used as a 16 Nibble memory and the 64 x 1 is known as a

The total number of cells in each case is 131072. Memory density on the other hand specifies the number of bits stored per unit area. More the number of bits stored in a unit area more dense the memory, that is, more bits are stored in less space. The capacity and the density of a memory are determined by the total number of cells implemented in a unit area.

When the signal is set to high it allows data to be read from the memory. When the signal is set to low data is written into the memory. Some memory chips have two separate Read and Write signals. The read and write signals are separately asserted to control the Read and Write operation.

Address signals are required to specify the location in the memory from which information is accessed . A set of parallel address lines known as the address bus carry the address information. The number of bits comprising the address bus depends upon the size of the memory. For example, a memory having four locations to store data has four unique addresses specified by a 2-bit address bus.

Generally, the wider the data bus more data can be accessed at each read or write operation. In a computer system there are more than one memory chips to store program information. At any particular instant a read or write operation is carried out on a single addressable location. The Read signal is asserted to configure the memory circuitry for reading data from the memory.

An address is applied on the Address Lines. The internal address decoder of the memory decodes

the address and selects one unique row from which data is read. The data read from the memory is stored in a data buffer inside the microprocessor. In the diagram shown, a microprocessor places an address 100 on its external address bus connected to the address lines of the memory.

The Write signal is asserted to configure the memory circuitry for writing data to the memory. An address is placed on the Address Lines by the microprocessor. The internal address decoder of the memory decodes the address and selects one unique row select line which selects the row location 3. In this unit we discussed about memory organisation.

The memory array can be organised in several ways depending on the unit of data. The 64cell array organised as 8 x 8 cell array is considered as an 8 byte memory, that is, it has eight locations and each location stores a single byte. The 64-cell array organised as 4 x 16 cell array stores 16 nibbles and if organised as 1 x 64 stores 64 single bit values. Write short notes on the following memory organisation memory signals read & write signals memory read operation memory write operation.

MEMORY TYPES

It goes further to explain them and their applications. RAM allows a read or write operation to be carried out at any address. All locations are accessible in equal time. RAM memories do not store permanent data.

Since ROM chips are read only, therefore user cannot write any information to ROM chips. ROM chips are programmed by the manufacturer and contain important information which is required to start the computer. In a Static RAM each cell which is capable of storing a binary 0 or 1 is made up of a flip-flop which retains information as long as power continues to be supplied to the flip-flop. Dynamic RAM on the other hand uses a capacitor to store a single bit of data.

External power is required to operate the transistors. As long as the external power is applied the static memory cell retains the data. The circuit of a single flipflop based cell which can store a binary 0 or 1 is shown. Circuit Diagram of a Static Memory Cell based on a FlipFlop The flip-flop used to store a binary bit works like a latch.

When both the SEL and W signals are activated the latch is configured in the transparent mode and the data applied at the Data In line flows through the latch to the output. The flip-flop based cells are combined to form an array. Additional logic is added to select cells at appropriate locations and to read and write data. A 3 x 8 decoder decodes a 3-bit address to select any one of the eight locations comprising of a group of cells.

3 x 8 decoder is activated which is connected to the SEL input of the four latches in the first row. Similarly, address 111 activates the eighth output line of the 3 x 8 decoder which selects the four latches in the last row. The memory array has four Data In lines to store the 4-bit data values at the eight locations. When data is to be read from a memory array, the memory chip is selected and the output enabled.

The Write signal along with the CS signal are used to write data into any 4-bit location. The data 1001 is placed at the four Data In lines respectively. The three CS, WE and OE are shown to be active low.

Internal Structure of an 8 x 4 Static RAM

RAM chips are subdivided into Asynchronous RAM and Synchronous Burst RAM. The Static memory described is an Asynchronous SRAM, the operation of which does not depend upon the clock signal. The read and write operations are carried out asynchronously. Synchronous SRAM uses a clock signal which is used by the microprocessor to synchronise its activities to synchronise the read and write operations for faster operation.

The block diagram of a Synchronous Burst SRAM is shown.

Block Diagram of a Synchronous Burst RAM

In the diagram, the external address, the WE and the CS external signals are latched in on a positive clock transition simultaneously. The data that is to be written into the memory is also latched into the Data Input Register at the same positive clock transition. For a read operation the data is latched in the Data Output register on the positive clock transition. In the Flow-through SRAM there is no Data Output Register so the data is asynchronously available on the data lines during a read operation.

A static RAM uses a latch to store a single bit of information. The drawback of DRAM is the discharging of the capacitor over a period of time. The extra circuitry required to refresh the capacitor complicates the operation of the DRAM. The circuit diagram of a single DRAM capacitor based memory cell is shown.

The capacitor is connected through a MOSFET which connects or disconnects the column line at B to the capacitor at D. If the row is set at logic high the MOSFET connects the column line to the capacitor. If the row line is set to logic low the MOSFET disconnects the column line from the capacitor. Writing a 1 or 0 into the DRAM Cell A write operation allows a logic 1 or 0 to be stored in a DRAM cell. The R/W signal is set to logic low indicating a write operation which enables the tri-state Input Buffer.

The logic 1 which is to be stored in the memory cell is applied at the DIN data line which is available at A on the column line. The row line is selected which allows the MOSFET to connect column B to capacitor D. The capacitor is charged to logic 1 voltage level via ABD. A Write operation to store logic 0 in a DRAM cell is similar. The appropriate row is selected by specifying the storage location address.

The R/W signal is set to logic low which enables the Input Buffer. The logic 0 to be stored in the DRAM cell is applied at the D IN which is stored on the capacitor via ABD. The read operation is accomplished by specifying the address of the location from which data is to be read. The DRAM address decoder activates the appropriate row.

The R/W signal is set to logic high which enables the output buffer. The Refresh Buffer is enabled by setting the Refresh signal to high. The input of the Refresh Buffer is connected to the output buffer/sense amplifier. The CAS signal when de-asserted, disables the DOUT data line, therefore the next column address cannot occur unless the data at the current address is latched by the external system reading data from the DRAM.

ROMs retain data when the power is turned off.

General Architecture of a ROM

A 16 x 8 ROM is shown. A 4-bit address is decoded by a 4 x 16 decoder which selects the appropriate row line. The MOSFETs connected to the selected row output logic 1 on the respective column lines. The MOSFETs that are not connected output logic 0.

The terminating resistor connected to the end of each column line ensures that the output line stays low when a MOSFET outputs logic 0.

ROMs are of different types

Data is erased by exposing the PROM to Ultraviolet light. The 4-bit code which is to be converted is applied as an address at the 4-bit address input of the ROM. At each of the 256 locations corresponding to the 256 addresses 256 Gray Code values are stored. The 4-bit Gray Code contents stored at the first 16 locations of the ROM are shown.

ROM can also be used as a simple table. Each location in the ROM stores a value which can be accessed by specifying the location address. LookUp tables used in computers can be implemented using ROMs.

ROM Programmed to Convert 4-Bit Binary to 4-Bit Gray Code

The high density FLASH memory cell is implemented using a single floating-gate MOS transistor. A data bit is stored as a charge or the absence of a charge on the floating gate.

MOS Transistor with Charge and no Charge FLASH memory operations are classified into

The programming operation adds charge to the floating gate of those cells that are to store a logic 0. No charge is added to those gates that are to store logic 1. Once the gate is charged it retains the charge for years. During the read operation a positive voltage is applied to the MOS transistor control gate.

During the erase operation charge is removed from the memory cell. A sufficiently large positive voltage is applied at the source with respect to the control gate. If charges are present on the gate, the positive voltage supply at the source attracts the electrons depleting the gate. A flash memory is erased prior to programming.

In this unit, we discussed about the types of memory, their applications and operations.

MEMORY EXPANSION

This unit exposes us to the various ways and reasons memory is expanded. State why memory is expanded explain the roles of microprocessors in a computer system. RAM and ROM memory depending upon specific applications. In a computer, part of the RAM is reserved to support the Video Memory, Stack and I/O buffers.

The ROM used by a computer is relatively very small as it stores few bytes of code used to Boot the Computer system on power up. The total amount of memory that is supported by any digital system depends upon the size of the address bus of the microprocessor or a micro-controller.

A microprocessor having an address bus of 16 bits can generate 216 or

65536 unique addresses to access 65536 locations which allows either a single 65536 location RAM or a combination of RAM and ROM totaling 65536 memory locations to be connected to the microprocessor. In this unit we talked about ways and reasons why we should expand our memory. State and explain 5 reasons we need to expand memory.

MEMORY SUMMARY

This unit summarises all that we have talked about memory. It does it in a clear and concise manner for easy understanding. The Static Ram is non-volatile and is not a high density memory as a latch is required to store a single bit of information. The Static Ram is non-volatile and is not a high density memory as a latch is required to store a single bit of information.

Write a detailed essay on what you have learned in this module.

INTRODUCTION TO

Central Processing Unit & Arithmetic and Logical Unit In this unit, we shall be discussing about microprocessors, their history, design, speed and finally architecture. 8080, a complete 8-bit computer on one chip, introduced in 1974.

The first microprocessor to make it into a home computer was the Intel

8088, introduced in 1979 and incorporated into the IBM PC . If you are familiar with the PC market and its history, you know that the PC market moved from the 8088 to the 80286 to the 80386 to the 80486 to the Pentium to the Pentium II to the Pentium III to the Pentium 4. All of these microprocessors are made by Intel and all of them are improvements on the basic design of the 8088.

The Pentium 4 can execute any piece of code that ran on the original

The following table helps you to understand the differences between the different processors that Intel has introduced over the years.

Microprocessor Components

Floating-point unit The floating-point unit is a specialised logic unit optimised to perform non-integer calculations much faster than the general-purpose logic unit can perform them.

Bus interfaces

Some instructions require one tick, others multiple ticks, and some processors execute multiple instructions during one tick. The number of ticks per instruction varies according to microprocessor architecture, its instruction set, and the specific instruction. Each requires many clock cycles to execute, but accomplishes a lot of work. AMD Athlon XP 3000+, which actually runs at 2.167 GHz, may be faster than an Intel Pentium 4 running at 3.06 GHz, depending on the application.

The comparison is complicated because different CPUs have different strengths and weaknesses. The only safe use of direct clock speed comparisons is within a single family. Also, even within a family, processors with similar names may differ substantially internally. Clock speeds increase every year, but the laws of physics limit how fast CPUs can run.

If designers depended only on faster clock speeds for better performance, CPU performance would have hit the wall years ago. Instead, designers have improved internal architectures while also increasing clock speeds. Recent CPUs run at more than 650 times the clock speed of the PC/XT's 8088 processor, but provide 6,500 or more times the performance.

Wider data busses and registers

Early CPUs processed data in 4-bit or 8-bit chunks, whereas current CPUs process 32 or 64 bits per operation.

Superscalar architecture

If one pipeline is good, more are better. Using multiple pipelines allows multiple instructions to be processed in parallel, an architecture called superscalar.

Central Processing Unit & Arithmetic and Logical Unit This unit exposes us to the Central Processing Unit and the Arithmetic

Explain what the CPU is state the various terms associated with the CPU explain what the ALU is. Central Processing Unit & Arithmetic & Logical Unit A computer is a complex system consisting of many different components. But at the heart or the brain of the computer is a single component that does the actual computing. This is the Central Processing Unit .

The job of the CPU is to execute programs. A computer is built to carry out instructions that are written in a very simple type of language called machine language. Each type of computer has its

own machine language, and it can directly execute a program only if it is expressed in that language. When the CPU executes a program, that program is stored in the computer's main memory .

In addition to the program, memory can also hold data that is being used or processed by the program. These locations are numbered, and the sequence number of a location is called its address. An address provides a way of picking out one particular piece of information from among the millions stored in memory.

[...]

Accessing data in a register usually takes zero clock cycles. Therefore, you should try to keep variables in the registers. Register sets are very small and most registers have special purposes which limit their use as variables, but they are still an excellent place to store temporary data. The CPU spends almost all its time fetching instructions from memory and executing them.

A scanner that converts images into coded binary numbers that can be stored and manipulated on the computer. Somehow the CPU has to communicate with and control all these devices. The CPU can only do this by executing machine language instructions . So, for each device in a system, there is a device driver, which consists of software that the CPU executes when it has to deal with the device.

Without the device driver, the actual physical device would be useless, since the CPU would not be able to communicate with it. A computer system consisting of many devices is typically organised by connecting those devices to one or more busses. A bus is a set of wires that carry various sorts of information between the devices connected to those wires. Control signals can be used, for example, by one device to alert another that data is available for it on the data bus.

Now, devices such as keyboard, mouse, and network interface can produce input that needs to be processed by the CPU. This method is called polling, since the CPU polls the input devices continually to see whether they have any input data to report. The CPU can waste an awful lot of time just waiting for input. Those instructions make up an interrupt handler that does the processing necessary to respond to the interrupt.

Interrupts allow the CPU to deal with asynchronous events. Interrupts make it possible for the CPU to deal efficiently with events that happen "asynchronously", that is, at unpredictable times. As another example of how interrupts are used, consider what happens when the CPU needs to access data that is stored on the hard disk. Data on the disk has to be copied into memory before it can be accessed.

When the CPU needs data from the disk, it sends a signal to the disk drive telling it to locate the data and get it ready. Then, instead of just waiting the long and unpredictable amount of time the disk drive will take to do this, the CPU goes on with some other task. When the disk drive has the data ready, it sends an interrupt signal to the CPU. The interrupt handler can then read the requested data.

All modern computers use multitasking to perform several tasks at once. Some computers can be used by several people at once. Since the CPU is so fast, it can quickly switch its attention from one user to another, devoting a fraction of a second to each user in turn. This application of multitasking is called timesharing.

Each of the individual tasks that the CPU is working on is called a thread. At any given time, only one thread can actually be executed by a CPU. When a timer interrupt occurs, the CPU has a chance to switch from one thread to another, whether the thread that is currently running likes it or not. Ordinary users, and indeed ordinary programmers, have no need to deal with interrupts and interrupt handlers.

In fact, most users, and many programmers, can ignore threads and multitasking altogether. Indeed, threads are built into the Java programming language as a fundamental programming concept. While programmers don't actually deal with interrupts directly, they do often find themselves writing event handlers, which, like interrupt handlers, are called asynchronously when specified events occur. Such "event-driven programming" has a very different feel from the more traditional straight-through, synchronous programming.

By the way, the software that does all the interrupt handling and the communication with the user and with hardware devices is called the operating system. The operating system is the basic, essential software without which a computer would not be able to function. Other programs, such as word processors and World Wide Web browsers, are dependent upon the operating system. The operation to be performed is selected by set of inputs known as function select inputs.

There are different MSI ALUs available that have two 4-bit inputs a 4bit output and three to five function select inputs that allows up to 32 different functions to be performed.

Register Indirect Addressing Mode Based Index Addressing Mode

80386 Scaled Indexed Addressing Mode Processors let you access memory in many different ways. Memory addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types. Mastery of the addressing modes is the first step towards mastering assembly language. Most 80386 instructions can operate on the general purpose register set.

By specifying the name of the register as an operand to the instruction, you may access the contents of that register. The only restriction is that both operands must be the same size. In addition to the general purpose registers, many instructions allow you to specify one of the segment registers as an operand. You cannot move data from one segment register to another with a single mov instruction.

They should only contain segment addresses. The key to good assembly language programming is the proper use of memory addressing modes. Beyond the increase to 32 bits, the new addressing modes on the 80386 are probably the biggest improvement to the chip over earlier processors. The most common addressing mode, and the one that's easiest to understand, is the

displacement-only addressing mode.

The displacement-only addressing mode is perfect for accessing simple variables. Intel named this the displacement-only addressing mode because a 16 bit constant follows the mov opcode in memory. In that respect it is quite similar to the direct addressing mode on the x86 processors. On the x86, a direct address can be thought of as a displacement from address zero.

8086 processors

If you want to provide an offset into a different segment, you must use a segment override prefix before your address. The CPU uses the data segment register by default. As with the x86 addressing mode, these four addressing modes reference the byte at the offset found in the bx, bp, si, or di register, respectively. Intel refers to and as base addressing modes and bx and bp as base registers .

And addressing modes as indexed addressing modes . This text will call these forms register indirect modes to be consistent. Note that while running in 16 bit real mode on the 80386, offsets in these 32 bit registers must still be in the range 0...0FFFFh. You cannot use values larger than this to access more than 64K in a segment.

Also note that you must use the 32 bit names of the registers. You cannot use the 16 bit names. The offsets generated by these addressing modes are the sum of the constant and the specified register. Intel's literature does not differentiate between these modes with or without the constant.

From the programmer's point of view, however, these addressing modes are useful for entirely different things. The addressing modes that do not involve bp use the data segment by default. Those that have bp as an operand use the stack segment by You substitute di in the figure above to obtain the addressing mode. You substitute di in the figure above for the addressing mode.

These addressing modes are a slight modification of the base/indexed addressing modes with the addition of an eight bit or sixteen bit constant.

You may substitute di in the figure above to produce the You may substitute di in the figure above to produce the

The base/indexed addressing modes let you pair up two 32 bit registers. Finally, the base/indexed/displacement addressing modes let you combine a constant and two registers to form the effective address. Keep in mind that the offset produced by the effective address computation must still be 16 bits long when operating in real mode. On the 80386 the terms base register and index register actually take on some meaning.

Note that the 80386 allows you to use the same register as both a base and index register, which is actually useful on occasion. Note that the choice of index register does not affect the choice of the default segment. There is one restriction the 80386 places on the index register. You cannot

use the esp register as an index register.

It's okay to use esp as the base register, but not as the index register. These addressing modes are particularly useful for accessing elements of arrays, though they are not limited to such purposes. These modes let you multiply the index register in the addressing mode by one, two, four, or eight. The general syntax for these addressing modes is `disp` where "base" and "index" represent any 80386 32 bit general purpose registers and "n" is the value one, two, four, or eight.

The 80386 computes the effective address by adding `disp`, base, and `index*n` together. The effective address is the final offset produced by an addressing mode computation. There is even a special instruction `load effective address` that computes effective addresses. Different addressing modes may take differing amounts of time to compute the effective address.

The exact difference varies from processor to processor. Generally, though, the more complex an addressing mode is, the longer it takes to compute the effective address. Complexity of an addressing mode is directly related to the number of terms in the addressing mode. If your offset is in the range -128...+127 the instruction will be shorter than an instruction with a displacement outside that range.

The size of the value in the register does not affect the execution time or size. So if you can arrange to put a large number in the register and use a small displacement that is preferable over large constant and small values in the register. If the effective address calculation produces a value greater than 0FFFFh, the CPU ignores the overflow and the result wraps around back to zero. In this unit, we discussed addressing modes, their various types and how they are used.

Mention and explain the different types of addressing modes we have.

[...]

Unlike the other programming languages, assembly language is not a single language, but rather a group of languages. In contrast to high level languages, data structures and program structures in assembly language are created by directly implementing them on the underlying hardware. So, instead of cataloguing the data structures and program structures that can be built, we will compare and contrast the hardware capabilities of various processor families. Assembly languages have the same structure and set of commands as machine languages, but they enable a programmer to use names instead of numbers.

Each type of CPU has its own machine language and assembly language, so an assembly language program written for one type of CPU won't run on another. In the early days of programming, all programs were written in assembly language. Now, most programs are written in high-level language. Programmers still use assembly language when speed is essential or when they need to perform an operation that isn't possible in a high-level language.

Assembly language teaches how a computer works at the machine level. Another reason is that the assembly programs are quicker, smaller, and have larger capacities than ones created with

other languages. Labels are used in assembly language, just as in HLLs, to mark lines as the targets of GOTOs. You can also specify variable names, procedure names, and other entities using symbolic labels.

Most of the time the label field is optional, meaning a label need be present only if you want a label on that particular line. Some mnemonics, however, require a label, others do not allow one. In general, you should always begin your labels in column one as it makes your programs easier to read. Assembler directives are special instructions that provide information to the assembler but do not generate any code.

A pseudo-opcode is a message to the assembler, just like an assembler directive, however a pseudo-opcode will emit object code bytes. These instructions emit the bytes of data specified by their operands but they are not true 80X86 machine instructions. When the assembler is processing a line of text, it completely ignores everything on the source line following a semicolon. You can use blank lines anywhere in your source file.

Blank lines are useful for spacing out certain sections of code, making them easier to read. Any number of spaces or tabs can separate the various fields in the statement. The first code sequence is much easier to read than the second. With respect to readability, the judicious use of spacing within your program can make all the difference in the world.

Assembly language programs are hard enough to read as it is. Formatting your listings to help make them easier to read will make them much easier to maintain. You may have a comment on the line by itself.

Y, 0

In assembly language code lines have two parts, the first one is the name of the instruction which is to be executed, and the second one are the parameters of the command. In the above example, we are using the instruction mov, it means move the value 25 to al register. The name of the instructions in this language is made of two, three or four letters. These instructions are also called mnemonic names or operation codes, since they represent a function the processor will perform.

For the creation of a program it is necessary to follow five steps

Translation to machine language is the creation of the object program, in other words, the written program as a sequence of zeros and ones that can be interpreted by the processor. The last stage is the elimination of detected faults on the program on the test stage.



NATIONAL OPEN UNIVERSITY OF NIGERIA

University Village, 91 Cadastral Zone, Nnamdi Azikwe Expressway, Jabi, Abuja

FACULTY OF SCIENCES

NOVEMBER, 2018 EXAMINATIONS

COURSE CODE: CIT344

COURSE TITLE: Introduction to Computer Design

CREDIT: 3 Units

TIME ALLOWED: 2½ Hours

INSTRUCTION: Answer Question ONE (1) and any other FOUR (4) Questions

1a) State the major ways by which finite state machines may be classified. (1 mark)

b) Draw the block diagram of a Read-Write memory. (4 marks)

c) Complete the cells in the table below with either "YES" or "NO". (6 marks)

Characteristics Memory Type	Non-Volatile	High Density	One-Transistor	In-System Write Ability
SRAM				
DRAM				
ROM				
EPROM				
EEPROM				
FLASH				

d) State the application areas or uses of multiplexer. (2 marks)

e) What are the major uses of decoders in computer systems? (2 marks)

f) Write short notes on latch. (5 marks)

g) Complete the cells in the Truth-Table below for a NAND-based S-R Latch. (2 marks)

Input		Output
S	R	Q_{t+1}
0	0	
0	1	
1	0	
1	1	

2a) Write short notes on Flip-Flops (5 marks)

b) Briefly describe a Half-Adder. Use diagram to illustrate your answer. (3 marks)

c) How many types of sequential circuits do we have? State and briefly describe them. (4 marks)

3a) Briefly describe a Full-Adder. (4 marks)

b) Draw the block representation of a Full-Adder. (2 marks)

c) State the two basic operations performed on memories and the signals typically used to support them. (6 marks)

4a) State the classifications of FLASH Memory operations. (1½ marks)

b) State any three (3) benefits of using high level programming language. (1½ marks)

c) Convert the following binary numbers to decimal:

(i) 10110.101 (ii) 1111.111 (iii) 110100100

(7½ marks)

d) Convert 1101001000001_2 to hexadecimal ($1\frac{1}{2}$ marks)

5a) State the two major functions of a register and its different shift operations. (7 marks)

b) State the defining characteristics of finite state machines. (5 marks)

6a) State the similarities in the operations of a NAND-based S-R latch and a NOR-based S-R latch (6 marks)

b) Enumerate any six (6) commands in assembly language that allows instructions to be processed when debugging a program. (6 marks)

ACHIEVER'S EDUCATIONAL CONSULTANT 08141710097