INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

Lab Manual:

Microprocessors and Microcontrollers(AECB26)

Prepared by

Ms B.Lakshmi Prasanna(IARE10706)

Electronics and Communication Engineering
Institute of Aeronautical Engineering

April 7, 2022

Contents

INTRODUCTION

1.1 Introduction

The purpose of this lab is to teach the basics of Intel 8086 and 8051 assembly language. Assembly language is important because it is the principal link between the software world and the hardware world of CPU design. Assembly language is the lowest-level, human-readable programming medium we can use to express complete application programs. Assembly language gives full access to the programmable features of the hardware, so a good understanding of it will provide valuable insight into the fundamentals of CPU design, the operation of the data path, and program execution.
For understanding assembly language. First, compilers translate high-level languages into assembly language, so compiler writers must understand assembly. Operating systems also include critical components written in assembly. Furthermore, embedded and mobile device programming often require knowledge of assembly language. As these technologies become more and more important to the overall performance of computer systems, knowledge of the computer at the assembly-language level will prove to be a valuable asset. Even if you spend your entire career programming in high-level languages, a basic understanding of assembly language concepts will give you an insight into your work that will in turn make you more valuable as an electronics, electrical or computer engineer. This Lab will provide an environment for you to gain hands-on experience with the tools and concepts used in the Microprocessor and microcontroller course.

1.1.1 Student Responsibilities

1. Students are required to attend all labs.
2. Students should work individually in the hardware and software laboratories
3. Students have to follow dress code whenever they come for lab work.
4. Should take only the Worksheet, calculator (if needed) and a pen or pencil to the work area.
5. Should learn the prelab questions. Read through the lab experiment to familiarize themselves with the components and assembly sequence.
6. Should utilize 3 hour's time properly to perform the experiment and to record the readings. Do the calculations, draw the graphs and take signature from the instructor.
7. If the experiment is not completed in the stipulated time, the pending work has to be carried out in the leisure hours or extended hours.
8. Should submit the completed worksheets according to the deadlines set up by the faculty.

1.1.2 Laboratory Assistant Responsibilities

The lab assistant shall be completely familiar with each lab prior to class. The lab assistant shall provide the students with a syllabus and safety review during the first class. The syllabus shall include the lab assistant office hours, telephone number, and the name of the faculty coordinator. The lab assistant is responsible for ensuring that all the necessary equipment and/or preparations for the lab are available and in working condition. Lab experiments should be checked in advance to make sure everything is in working order. The lab assistant should fully answer any questions posed by the students and supervise the students performing the lab experiments.

1.1.3 Laboratory Faculty Responsibilities

The faculty should ensure that the laboratory is properly equipped, i.e., that the lab assistants receive any equipment necessary to perform the experiments. The faculty is responsible for resolving any questions or problems that are identified by the lab assistants or the students. The faculty may supervise the format of the final exam for the lab. They are also responsible for making any necessary corrections to this manual and ensuring that it is continually updated and available. Faculty is expected to allot marks for worksheets in a fair and timely manner. The worksheets should be returned to the students in the next lab period following submission. The lab assistant should report any errors in the lab manual to the faculty.

1.1.4 Course Coordinator Responsibilities

The course coordinator is responsible fo making any necessary corrections in Course Description and lab manual. He/She has to ensure that it is continually updated and available to the students in the CMS learning Portal.

1.2 Lab Policy and Grading

The student should understand the following policy:
ATTENDANCE: Attendance is mandatory as per the academic regulations.
LAB RECORD's: The student must:

1. Write the work sheets for the allotted experiment and keep them ready before the beginning of eachlab.
2. Keep all work in preparation of and obtained during lab.
3. Perform the experiment and record the observations in the worksheets.
4. Analyze the resultsand get the work sheets evaluated by the Faculty.
5. Upload the evaluated reports online from CMS LOGIN within the stipulated time.

Grading Policy:

The final grade of this course is awarded using the criterion detailed in the academic regulations. A large portion of the student's grade is determined in the comprehensive final exam of the Laboratory course (SEE PRACTICALS),resulting in a requirement of understanding the concepts and procedure of each lab experiment for successful completion of the lab course.

Pre-Requistes and Co-Requisties:

Co-Requisites for this lab is microprocessor and microcontroller course and pre-requisites are digital system design course. Students are required to have completed both the courses with better grade in each. Students are also assumed to have completed a programming class and be familiar with the use of a computer-based word processor. Note that the instructor reserves the right to alter any part of this information at their discretion. Any changes will be announced in class and distributed in writing to the students prior to the changes taking effect.

1.3 Course Goals and Objectives

This laboratory course will facilitates the students to program 8086 microprocessor and 8051 microcontroller. Win862 software will be used for writing and debugging assembly language programs. The course includes performing arithmetic and logical operations, string manipulations, code conversions and interfacing of I/O devices to processor/controller. The hands-on experience acquired by the student's during the course makes them to carry out processor/controller based projects and extend their knowledge on the latest trends and technologies in the field of embedded system.

More explicitly, the class objectives are:

1. Assembly language programming skills ranging from simple arithmetic operations to interfacing real time systems.
2. The usage of software tools to design, debug and test microprocessor/microcontroller based projects using assembly language programming
3. The design of microcomputer and microcontroller based real-time applications in the fields of communication systems, home based automation systems, automobiles and unmanned applications.

1.4 Use of Laboratory Instruments

One of the major goals of this lab is to familiarize the student about 8086 microprocessor, 8051 microcontroller, peripheral devices and assembly language programming. Interfacing can be done between 8086 microprocessor/8051 microcontroller and peripherals like stepper motor, ADC, DAC, Tone generator, traffic light etc and perform specific task by writing assembly language program. Serial communication cable RS232 is connected between PC and ESA86/88 trainer board. In general, all devices have physical limits. These limits are specified by the device manufacturer and are referred to as the device rating. The ratings are usually expressed in terms of voltage limits, current limits, or power limits. It is up to the engineer to make sure that in device operation, these ratings (limit values) are not exceeded. The following rules provide a guideline for instrument protection.

The following rules provide a guideline for instrument protection.

1.4.1 Instrument Protection Rules

1. Properly connect the 8086 microprocessor/8051 microcontroller kit with power supply terminals.
2. Switch on the power supply after checking connection
3. While connecting cables/bus to the devices take care of pins

4. Handle the Trainer kit carefully.

5. Switch of Trainer kit after completing the experiment.

1.5 Data Recording and Reports

1.5.1 The Laboratory Worksheets

Students must record their experimental values in the provided tables in this laboratory manual and reproduce them in the lab worksheets. Worksheets are integral to recording the methodology and results of an experiment. Make plots of data and sketches when these are appropriate in the recording and analysis of observations. Note that the data collected will be an accurate and permanent record of the data obtained during the experiment and the analysis of the results.

1.5.2 The Laboratory Files/Reports

Reports are the primary means of communicating your experience and conclusions to other professionals. In this course you will use the lab report to inform your LTF about what you did and what you have learned from the experience. Engineering results are meaningless unless they can be communicated to others. You will be directed by your LTF to prepare a lab report on a few selected lab experiments during the semester.

1.5.3 Formatting and Style

1. The lab report shall be hand written in a lab worksheet.

2. The first line of each paragraph should have a left indent.

3. All the tables should have titles and should be numbered. Tables should be labelled numerically as Table 1, Table 2, etc. Table captions appear above the table.

4. Graphs should be presented as figures. All the figures should have titles and should be numbered. Figure captions appear below the figure. Graphs should have labeled axes and clearly show the scales and units of the axes.

5. All the figures and tables must be centered on the page.

6. Do not place screenshots of your lab worksheet.

1.5.4 Order of Lab Report Components

1. Cover Page - Cover page must include lab name and number, your name and the date the lab was performed.

2. Objective - Clearly state the experiment objective in your own words.

3. Equipment Used - Indicate which equipment was used in performing the experiment.

For each part of the Lab:

1. Write the lab's part number and title.

2. Firstly, describe the problem that you studied in this part, give an introduction of the theory, and explain why you did this experiment. Do not lift the text from the lab manual; use your own words.

3. Secondly, describe the experimental setup and procedures. Do not follow the lab manual in listing out individual pieces of equipment and assembly instructions. That is not relevant information in a lab report! Instead, explain the program. Your description should take the form of a narrative, and include information not present in the manual, such as descriptions of what happened during intermediate steps of the experiment.

4. Thirdly, explain your findings. This is the most important part of your report, because here, you show that you understand the experiment beyond the simple level of completing it. Explain (compare expected results with those obtained). Analyse (analyze experimental error). Interpret (explain your results in terms of theoretical issues and relate to your experimental objectives). All the results should be presented even if there is any inconsistency with the theory.

5. Finally, provide a summary of what was learned from this part of the laboratory experiment. If the results seem unexpected or unreliable, discuss them and give possible explanations.

Conclusions - The conclusion section should provide a take-home message summing up what has been learned from the experiment:

1. Briefly restate the purpose of the experiment (the question it was seeking to answer)

2. Identify the main findings (answer to the research question)

3. Note the main limitations that are relevant to the interpretation of the results

4. Summarize what the experiment has contributed to your understanding of the problem.

Probing Further Experiments - Questions pertaining to this lab must be answered at end of laboratory report

LAB-1 Design a Program Using WIN862

2.1 Introduction

Assembly language is the most basic programming language available for any processor. With assembly language, a programmer works only with operations implemented directly on the physical CPU. assembly language is the most powerful computer programming language available, and it gives programmers the insight required to write effective code in high-level languages. Windows Driver For ESA 86/88-2 Trainer (WIN862) is used for programming, execution and debugging. The Intel 8086 is a 16-bit microprocessor that is intended to be used as the CPU in a microcomputer. The term 16-bit means that its arithmetic logic unit, its internal registers, and most of its instructions are designed to work with 16-bit binary words.

2.2 Objective

2.2.1 Educational

1. Learn about Architecture and operation of 8086 microprocessor.
2. Understand instruction set and addressing modes of 8086 microprocessor.
3. Learn to perform basic operations MOV, ADD, SUB, MUL and DIV.

2.2.2 Experimental

1. To perform arithmetical and logical operations with 8 bit data using registers.
2. To use Windows Driver for ESA 86/88-2 Trainer (WIN862) for programming, execution and debugging.
3. Observe and analyze the output in registers.

2.3 Prelab Preparation:

Reading

1. Read register organization, Instruction set, addressing modes of 8086 microprocessor.

Written

1. Prior coming to the lab complete part0 of the procedure.

2.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	RS-232		1

2.5 Background

Features of the ESA -86/88 Microprocessor Trainer:

1. 8086 CPU operating at 8 MHz MAX mode.
2. Provision for on-board 8087 (NDP) coprocessor.
3. Provision for 256 KB of EPROM and 128 KB of RAM onboard
4. Battery backup facility for RAM.
5. 48 programmable I/O lines using two 8255's
6. Timer1 and Timer2 signals are brought out to header pins
7. Priority Interrupt Controller (PIC) for eight input using 8259A
8. In standalone mode using on board keypad or with PC compatible system through its RS-232 interface
9. Display is 8 seven segment LED
10. Designed and engineered to integrate user's application specific interface conveniently at a minimum cost.
11. Powerful and user-friendly keyboard / serial monitor, support in development of application programs.
12. Software support for development of programs on Computer, the RS-232C interface cable connecting to computer from the kit facilitates transfer of files between the trainer kit and computer for development and debugging purposes.
13. High quality reliable PCB with solder mask on both sides and clear legend prints with maximum details provided for the user.

Specifications:

1. **CPU:** Intel 8086 operating at 8 MHz in MAX mode.
2. **Memory:** Total 1MB of memory is in the Kit provided.
3. **EPROM:** 4 JEDEC compatible sockets for EPROM
4. **RAM:** 4 JEDEC compatible sockets for RAM
5. **Parallel I/O:**48 I/O lines using two 8255
6. **Serial I/O:**One RS-232C compatible interface Using UART 8251A

7. **Timer:** Three 16 bit counter / timers 8253A Counter 1 is used for serial I/O Baud rate generation.

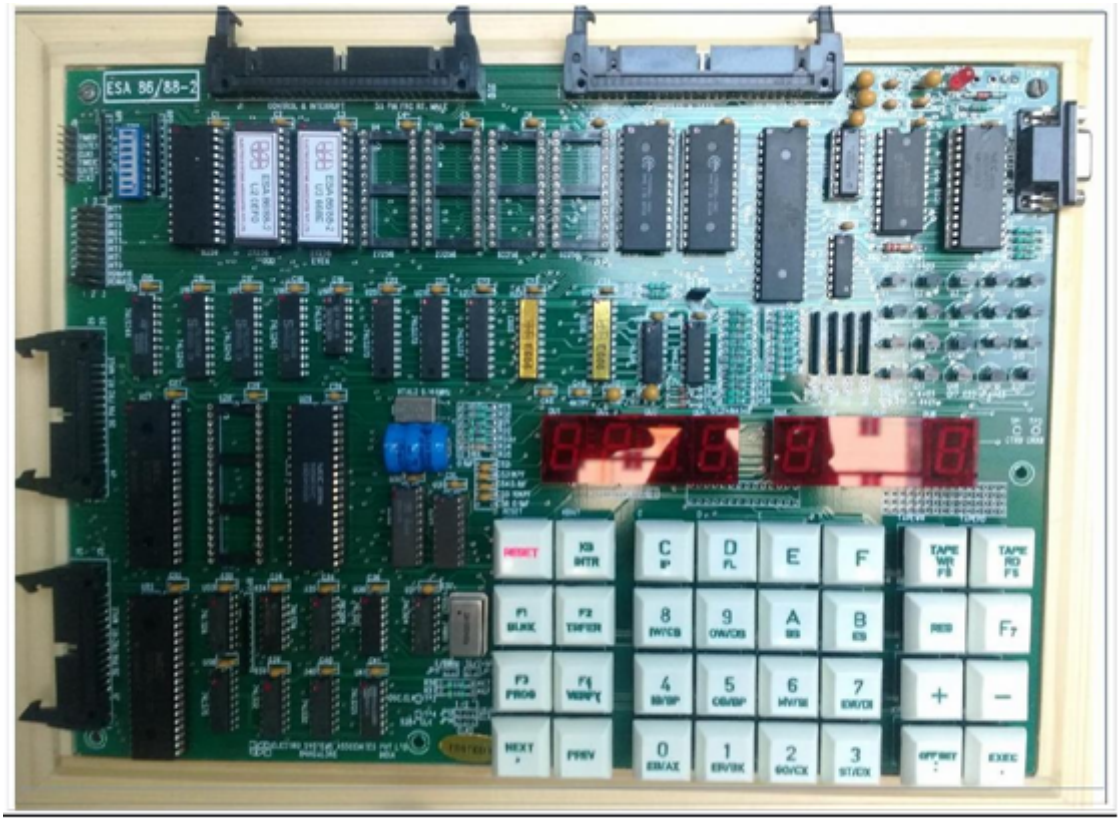


Figure 2.1: ESA86/88 trainer board

8. **PIC:** Programmable Interrupt controller using 8253A provides interrupts Vectors for 8 jumpers selectable Internal /External sources.
9. **Keyboard:** keyboard on to the trainer.
10. **Display:** 8 seven segment displays
11. **NIM:** Provision for connecting NMI to a key switch
12. **INTR:** Programmable Interrupt controller using 8259A provides Interrupt vectors for 8 jumpers selectable Internal/ External Sources.
13. **CPU Bus:** All address, data and control lines are TTL compatible and are terminated in berg strip header.
14. **Parallel I/O:**All signals are TTL compatible and Terminated in berg strip header For PPI expansion.
15. **Serial I/O:**Serial port signals are terminated in Standard 9-pin „D type connector.
16. **Monitor Software:**128KB of serial / Keyboard monitor with Powerful commands to enter verify and Debug user programs, including onboard Assemble and disassemble commands.

17. **Computer Interface**This can be interfaced to host computer System through the main serial port, also Facilitates uploading, downloading of Intel Hex files between computer and the trainer.

18. **Power requirements:**+5V DC with 1300 mA current rating (Max).

19. **Operating Configuration:** Two different modes of operation trainer are possible. They are (i) Serial operation (ii) Keypad operation The first configuration requires a computer system with an RS-232C port, can be used as the controlling device. When a computer system is interfaced to trainer, the driver program must be resident in the computer system. The second mode of operation is achieved through Onboard KEYBOARD / DISPLAY. In this mode, the trainer kit interacts with the user through a computer keyboard and 16x2 LCD Display. This configuration eliminates the need for a computer and offers a convenient way for using the trainer as a stand – alone system.

Registers:
AX:Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.
BX:Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.
Addressing modes:
Immediate addressing mode: In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes. Example: MOV AX, 0005H. In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.
Instruction set:
MOV instruction: It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing. General Format : MOV destination, source Here the source and destination needs to be of the same size, that is both 8 bit or both 16 bit. MOV instruction does not affect any flags.
ADD instruction
Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected General Format: ADD Destination, Source
SUB instruction
SUB instruction is used to subtract the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected General Format: SUB Destination, Source
MUL instruction
This instruction multiplies an unsigned byte or Word by the contents of AL. The Unsigned byte or word may be in any one of the general purpose registers or memory locations. In case of 32-bit results the most significant word of the result will be stored in DX, while the least significant of the result is stored in AX. The flags are modified depending on the result.
DIV instruction
It divides an unsigned word or double word by a 16-bit or 8-bit operand. The dividend must be in AX for 16-bit operation and divisor must be specified using any one of the addressing modes except immediate. The result will be in AL(quotient) while AH will contain the remainder. If the result is too big to fit in AL, type0(divide by zero) and an interrupt is generated. In case of double word dividend (32-bit), the higher word should be in DX and lower word should be

in AX. The divisor must be specified using any one of the addressing modes except immediate. The quotient will be in AX and the remainder will be in DX.

2.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Switch on the power supply after checking connections
3. Handle the Trainer kit carefully.

2.7 Procedure

(i)Execution procedure for 8086 (for registers):

1. Switch On Power Supply
2. Check if DIP switches board is in serial or keyboard mode (Serial mode = 1 on, Board mode = 4 On)
3. Press Reset
4. Press "EB"(Examine Byte)
5. Enter Starting Memory location (Ex: 2000)
6. Press next button, Enter OP-Code value
7. Then press next button Enter 2nd memory location and op code
8. Enter up to n values **Execution:**
9. Press Exec. Button
10. Press Go enter starting memory location
11. Press Exec.
12. Press ER (Examine Register)
13. Press AX (Now see the result in Ax)

(ii)Execution procedure for 8086 (for memory locations):

1. Switch On Power Supply
2. Check if DIP switches board is in serial or keyboard mode (Serial mode = 1 on, Board mode = 4 On)
3. Press Reset
4. Press "EB"(Examine Byte)
5. Enter Starting Memory location (Ex: 2000)
6. Press next button, Enter OP-Code value
7. Then press next button Enter 2nd memory location and op code
8. Enter up to n values **Execution:**

9. Press Exec. Button
10. Press Go enter starting memory location
11. Press Exec.
12. Press EB give input memory location and input values
13. Press Exec.
14. Press Go Give starting memory location
15. Press Exec.
16. Press Go Now observe the results in memory location

(iii)WIN862 Software procedure(for Registers):

1. Open Win862 icon on desktop and opened Window

Figure 2.2: Win862 icon

Figure 2.3: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

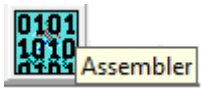


Figure 2.4: Assembler icon

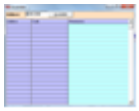


Figure 2.5: Assembler Window

3. Then write 1st Instruction then press enter button.

4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

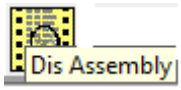


Figure 2.6: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Click on Run (check whether program is executed or not)
9. Click on view registers

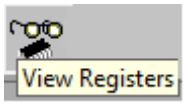


Figure 2.7: view registers

(iv)WIN862 Software procedure(for Memory locations):

1. Open Win862 icon on desktop.
2. Click on Assembler and give starting address (Like 0000:4000) then press Enter button.
3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.
7. Click on Set PC then give starting address then press Enter button.
8. Click on Run (check whether program is executed or not)
9. Click on view memory
10. Now enter input address
11. Click on Modify and Give desired input values
12. Click on Set PC. Enter initial address and press Dis-Assembler
13. Click on Run (check whether program is executed or not)

14. Now observe the result in view memory.

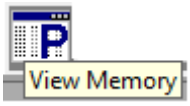


Figure 2.8: view memory

15. Click on view memory and enter destination address then press enter button
16. Now observe the result.

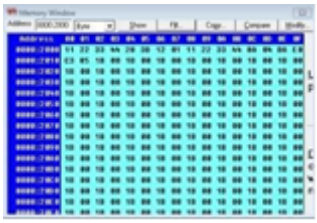


Figure 2.9: memory window

Programs: 8-bit arithmetic operations

1. **Addition:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS

			MOV AL,43
			MOV BL,11
			ADD AL,BL
			INT 3	

Observation Table:

Input		Output	
Register	Data	Register	Data
AL	43	AL	
BL	11		

2. **Subtraction:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS

			MOV AL,43
			MOV BL,11
			SUB AL,BL
			INT 03	

Observation Table:

Input		Output	
Register	Data	Register	Data
AL	43	AL	
BL	11		

3. **Multiplication:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS

			MOV AL,43
			MOV BL,11
			MUL BL
			INT 3	

Observation Table:

Input		Output	
Register	Data	Register	Data
AL	43	AX	
BL	11		

4. **Division:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS

			MOV AL,10
			MOV BL,02
			DIV BL
			INT 3	

Observation Table:

Input		Output	
Register	Data	Register	Data
AL	10	AL	
BL	02	AH	

2.8 Probing Further Experiments

1. Specify the registers used to hold quotient and remainder to perform unsigned division operation, consider 16-bit dividend and 8-bit divisor.

2. What is the procedure to change a single line in disassembler.

3. Which flag will be enabled during division error.

LAB-2 16 bit arithmetic and logical operations

3.1 Introduction

The assembly language programs for performing arithmetic and logical operations are composed by using mnemonics, various addressing modes, instructions and registers of microprocessor. The 8086 microprocessor is used to execute the instructions of assembly language program one by one. The results stored in destination registers are compared against theoretical values obtained. Arithmetic operations includes Addition, Subtraction, Multiplication, Division and logical Operations includes AND, OR, XOR.

3.2 Objective

3.2.1 Educational

1. Learn about the architecture of 8086 microprocessor.
2. Learn about Flag manipulation instructions and how they are set and reset in assembly operations.
3. Learn what registers are, why they are important and how to use them
4. Discover direct and immediate addressing and how they are used in assembly programming
5. Discover how to jump to labeled parts of code based on flags

3.2.2 Experimental

1. Write an assembly language program to perform 16-bit arithmetic operations.
2. Write an assembly language program to perform 16-bit logical operations.
3. Observe and analyze the output in registers.

3.3 Prelab Preparation:

Reading

1. Read register organization, Instruction set, addressing modes of 8086 microprocessor.

Written

1. Prior coming to the lab complete part0 of the procedure.

3.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	RS-232		1

3.5 Background

Registers:
AX:Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation. BX:Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.
CX:Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation
DX:Data register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.
Addressing modes:
Immediate addressing mode:
In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes. Example: MOV AX, 0005H. In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.
Instruction set:
MOV instruction:
It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing. General Format : MOV destination, source Here the source and destination needs to be of the same size, that is both 8 bit or both 16 bit. MOV instruction does not affect any flags.
ADD instruction:
Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected General Format: ADD Destination, Source
SUB instruction
SUB instruction is used to subtract the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected General Format: SUB Destination, Source
MUL instruction:
This instruction multiplies an unsigned byte or Word by the contents of AL. The Unsigned byte or word may be in any one of the general purpose registers or memory locations. In case of 32-bit results the most significant word of the result will be stored in DX, while the least significant of the result is stored in AX. The flags are modified depending on the result.

DIV instruction:
It divides an unsigned word or double word by a 16-bit or 8-bit operand. The dividend must be in AX for 16-bit operation and divisor must be specified using any one of the addressing modes except immediate. The result will be in AL(quotient) while AH will contain the remainder. If the result is too big to fit in AL, type0(divide by zero) and an interrupt is generated. In case of double word dividend (32-bit), the higher word should be in DX and lower word should be in AX. The divisor must be specified using any one of the addressing modes except immediate. The quotient will be in AX and the remainder will be in DX.
AND instruction:
This instruction logically ANDs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location. The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined. General Format: AND Destination, Source
OR instruction:
This instruction logically ORs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location. The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined. General Format: OR Destination, Source
XOR instruction:
The XOR operation is again carried out in a similar way to the AND and OR operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero. General Format: XOR Destination, Source
INT 03 instruction:
The INT 3 instruction generates a special one byte opcode (CC) that is intended for calling the debug exception handler. (This one byte form is valuable because it can be used to replace the first byte of any instruction with a breakpoint, including other one byte instructions, without over-writing other code).
General Format: INT 03

3.6 Safety Precautions:

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Switch on the power supply after checking connections
3. Handle the Trainer kit carefully.

3.7 Procedure

WIN862 Software procedure(for Registers):

1. Open Win862 icon on desktop and opened Window

Figure 3.1: Win862 icon

Figure 3.2: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

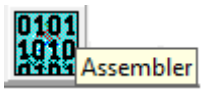


Figure 3.3: Assembler icon

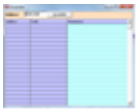


Figure 3.4: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 3.5: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Click on Run (check whether program is executed or not)
9. Click on view registers

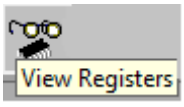


Figure 3.6: view registers

Programs: 16 Bit arithmetic and logical operations using WIN862 software and 8086 microprocessor.
Arithmetic Operations:

1. **Addition:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS

			MOV AX,4343
			MOV BX,1111
			ADD AX,BX
			INT 3	

Observation Table:

Input		Output	
Register	**Data**	**Register**	**Data**
AX	4343	AX	
BX	1111		

2. **Subtraction:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS

			MOV AX,4343
			MOV BX,1111
			SUB AX,BX
			INT 03	

Observation Table:

Input		Output	
Register	**Data**	**Register**	**Data**
AX	4343	AX	
BX	1111		

3. **Multiplication:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS

			MOV AX,4343
			MOV BX,1111
			MUL BX
			INT 3	

Observation Table:

Input		Output	
Register	**Data**	**Register**	**Data**
AX	4343	AX	
BX	1111	DX	

4. **Division:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS

			MOV AX,0080
			MOV BX,0008
			DIV BX
			INT 3	

Observation Table:

Input		Output	
Register	**Data**	**Register**	**Data**
AX	4343	AX	
BX	1111	DX	

Logical Operations :

1. **AND:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS

			MOV AX,4343
			MOV BX,1111
			AND AX,BX
			INT 3	

Observation Table:

Input		Output	
Register	**Data**	**Register**	**Data**
AX	4343	AX	
BX	1111		

2. **OR:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS

			MOV AX,4343
			MOV BX,1111
			OR AX,BX
			INT 3	

Observation Table:

Input		Output	
Register	**Data**	**Register**	**Data**
AX	4343	AX	
BX	1111		

3. **XOR:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS

			MOV AX,4343
			MOV BX,1111
			XOR AX,BX
			INT 3	

Observation Table:

Input		Output	
Register	**Data**	**Register**	**Data**
AX	4343	AX	
BX	1111		

3.8 Probing Further Experiments

1. Specify the registers used to hold double word dividend and 16-bit divisor to perform unsigned division operation, and also for quotient and remainder after division.

2. To perform logical operations like NAND, NOR which instructions should be used from the instruction set of 8086 microprocessor.

3. How to write a program to add the contents of memory location 2000H:0500H to the contents of 3000H:0600H and store the result in 5000H:0700H. Use data segment register initialization instructions.

LAB-3 Multibyte Addition and Subraction

4.1 Introduction

The multibyte data can be added either byte by byte or word by word. The number of bytes in the data can be used for the number of additions. One of the register is used to account for the final carry. Similarly the multibyte data can be Subtracted either byte by byte or word by word. The number of bytes in the data can be used for the number of subtractions. One of the register is used to account for the final borrow. To perform multibyte addition or subtraction we require three address pointers. Two pointers for input data and one pointer for output data.

4.2 Objective

4.2.1 Educational

1. Learn about the architecture of 8086 microprocessor.
2. Learn about Flags and how they are set and reset in assembly operations.
3. Learn what registers are, why they are important and how to use them
4. Discover register indirect, Indexed addressing and how they are used in assembly programming
5. Discover how to jump to labeled parts of code based on flags.

4.2.2 Experimental

1. Write an assembly language program to perform multibyte addition.
2. Write an assembly language program to perform multibyte subtraction.
3. Observe and analyze the outputs using registers and memory locations.

4.3 Prelab Preparation:

Reading

1. Read register organization, Instruction set, addressing modes of 8086 microprocessor.

Written

1. Prior coming to the lab complete part0 of the procedure.

4.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	RS-232		1

4.5 Background

Registers:
AX:Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.
BX:Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.
CX:Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation
SI:Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions are used in conjunction with the DS register to point the data locations in the data segment.
DI:Destination Index (DI) is a 16-bit register is used in conjunction with the ES register for string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data addresses in string manipulation instructions. In short, Destination Index and SI Source Index registers are used to hold address.
Addressing modes:
Immediate addressing mode:
In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.
Example: MOV AX, 0005H. In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.
Register indirect addressing mode:
Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES.
Example: MOV AX, [BX].
Indexed addressing mode:
In this addressing mode, offset of the operand is stored one of the index registers. DS and ES are the default segments for index registers SI and DI respectively.
Example: MOV AX, [SI] Here, data is available at an offset address stored in SI in DS.
Instruction set:
MOV instruction:
It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing. General Format : MOV destination, source Here the source and destination needs to be of the same size, that is both 8 bit or both

16 bit.MOV instruction does not affect any flags.
ADD instruction:
Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected
General Format: ADD Destination, Source
SUB instruction
SUB instruction is used to subtract the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected General Format: SUB Destination, Source
INC and DEC instructions
INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.
JNZ instruction:
The jnz (or jne) instruction is a conditional jump that follows a test. It jumps to the specified location if the Zero Flag (ZF) is cleared (0). jnz is commonly used to explicitly test for something not being equal to zero General Format: JNZ location
INT 03 instruction:
The INT 3 instruction generates a special one byte opcode (CC) that is intended for calling the debug exception handler. (This one byte form is valuable because it can be used to replace the first byte of any instruction with a breakpoint, including other one byte instructions, without over-writing other code).
General Format: INT 03

4.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Switch on the power supply after checking connections
3. Handle the Trainer kit carefully.

4.7 Procedure

WIN862 Software procedure(for memory locations):

1. Open Win862 icon on desktop) and opened Window

Figure 4.1: Win862 icon

Figure 4.2: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

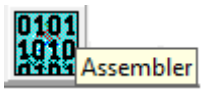


Figure 4.3: Assembler icon

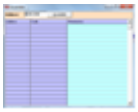


Figure 4.4: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 4.5: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Click on Run (check whether program is executed or not)
9. Click on view memory
10. Now enter input address
11. Click on Modify and Give desired input values
12. Click on Set PC. Enter initial address and press Dis-Assembler
13. Click on Run (check whether program is executed or not)

14. Now observe the result in view memory.

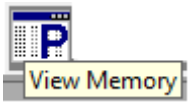


Figure 4.6: view memory

15. Click on view memory and enter destination address then press enter button
16. Now observe the result.

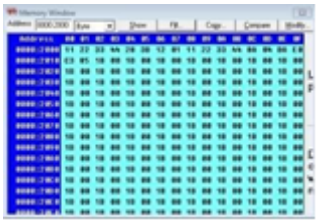


Figure 4.7: memory window

Programs: Multibyte Addition and Subtraction using WIN862 software and 8086 microprocessor.

1. **Multibyte Addition:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS
			MOV AX,0000	
			MOV SI, 2000	
			MOV DI, 3000	
			MOV BX, 2008	
			MOV CL, 04	
		UP:	MOV AL, [SI]	
			ADD AL, [BX]	
			MOV [DI], AL	
			INC SI	
			INC BX	
			INC DI	
			DEC CL	
			JNZ UP	
			INT 3	

Observation Table:

Input				Output	
MEMORY LOCATION	**Data**	**MEMORY LOCATION**	**Data**	**MEMORY LOCATION**	**Data**
2000		2008		3000	
2001		2009		3001	
2002		200A		3002	
2003		200B		3003	

2. **Multibyte Subtraction:**

MEMORY LOCATION	OP-CODE	LABLE	MNEMONIC OPERAND	COMMENTS
			MOV AX,0000	
			MOV SI, 2000	
			MOV DI, 3000	
			MOV BX, 2008	
			MOV CL, 04	
		UP:	MOV AL, [SI]	
			SUB AL, [BX]	
			MOV [DI], AL	
			INC SI	
			INC BX	
			INC DI	
			DEC CL	
			JNZ UP	
			INT 3	

Observation Table:

Input				Output	
MEMORY LOCATION	**Data**	**MEMORY LOCATION**	**Data**	**MEMORY LOCATION**	**Data**
2000		2008		3000	
2001		2009		3001	
2002		200A		3002	
2003		200B		3003	

4.8 Probing Further Experiments

1. In this experiment ADD and SUB instructions are used to perform multi-byte addition and subtraction, If we use ADC and SBB instructions what will be the result in 3000 location. Compare the obtained results.

2. What would happen if the instruction DAA is used after ADD and DAS is used after SUB instructions in this experiment, Access the obtained output.

3. What change should be done in the program to arrange the output of multi-byte addition and subtraction with lowest byte first.

LAB-4 Programs to Sort Numbers

5.1 Introduction

The string can be stored in ascending order/descending order by bubble sorting. In bubble sorting of N data, N-1 comparisons are performed by taking two consecutive data at a time. After each comparison the two data can be arranged in the ascending order/descending order in the same memory locations.

5.2 Objective

5.2.1 Educational

1. Learn about the architecture of 8086 microprocessor.
2. Learn about Flags and how they are set and reset in assembly operations.
3. Learn what registers are, why they are important and how to use them
4. Discover register indirect, Indexed addressing and how they are used in assembly programming
5. Discover how to jump to labeled parts of code based on flags.

5.2.2 Experimental

1. Write an assembly language program to arrange the numbers in ascending order.
2. Write an assembly language program to arrange the numbers in ascending order.
3. Observe and analyze the outputs using registers and memory locations.

5.3 Prelab Preparation:

Reading

1. Read register organization, Instruction set, addressing modes of 8086 microprocessor.

Written

1. Prior coming to the lab complete part0 of the procedure.

5.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	RS-232		1

5.5 Background

Registers:
AX:Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.
BX:Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.
CX:Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation
SI:Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions are used in conjunction with the DS register to point the data locations in the data segment.
DI:Destination Index (DI) is a 16-bit register is used in conjunction with the ES register for string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data addresses in string manipulation instructions. In short, Destination Index and SI Source Index registers are used to hold address.
Addressing modes:
Immediate addressing mode:
In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.
Example: MOV AX, 0005H. In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.
Register indirect addressing mode:
Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode.In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES. Example: MOV AX, [BX].
In this addressing mode, offset of the operand is stored one of the index registers. DS and ES are the default segments for index registers SI and DI respectively. Example: MOV AX, [SI] Here, data is available at an offset address stored in SI in DS.
Instruction set:
MOV instruction:
It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing. General Format : MOV destination, source Here the source and destination needs to be of the same size, that is both 8 bit or both 16 bit. MOV instruction does not affect any flags.
ADD instruction:

Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected General Format: ADD Destination, Source
INC and DEC instructions
INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.
JZ instruction
JE / JZ : Stands for 'Jump if Equal' or 'Jump if Zero'. The JZ (or JE) instruction is a conditional jump that follows a test It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, then jump.
JNZ instruction:
The jnz (or jne) instruction is a conditional jump that follows a test. It jumps to the specified location if the Zero Flag (ZF) is cleared (0). jnz is commonly used to explicitly test for something not being equal to zero General Format: JNZ location
JNC instruction:
The JNC instruction(jump if no carry) is a conditional jump that follows a test. It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump. General Format: JNC location
INT 03 instruction:
The INT 3 instruction generates a special one byte opcode (CC) that is intended for calling the debug exception handler. (This one byte form is valuable because it can be used to replace the first byte of any instruction with a breakpoint, including other one byte instructions, without over-writing other code). General Format: INT 03
XCHG instruction:
The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations. General Format : XCHG Destination, Source
CMP instruction:
The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset. General Format : CMP Destination, Source

5.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Switch on the power supply after checking connections
3. Handle the Trainer kit carefully.

5.7 Procedure

WIN862 Software procedure(for memory locations):

1. Open Win862 icon on desktop and opened Window

Figure 5.1: Win862 icon

Figure 5.2: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

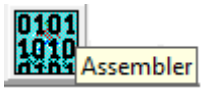


Figure 5.3: Assembler icon

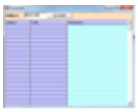


Figure 5.4: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 5.5: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Click on Run (check whether program is executed or not)
9. Click on view memory

10. Now enter input address
11. Click on Modify and Give desired input values
12. Click on Set PC. Enter initial address and press Dis-Assembler
13. Click on Run (check whether program is executed or not)
14. Now observe the result in view memory.

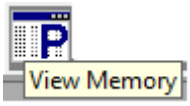


Figure 5.6: view memory

15. Click on view memory and enter destination address then press enter button
16. Now observe the result.

Figure 5.7: memory window

Programs:Sorting of numbers

1. **Ascending order:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AX,0000	
			MOV CH,0004	
			DEC CH	
		UP1	MOV CL,CH	
			MOV SI,2000	
		UP	MOV AL,[SI]	
			INC SI	
			CMP AL,[SI]	
			JC DOWN	
			XCHG AL,[SI]	
			DEC SI	
			MOV [SI],AL	
			INC SI	
		DOWN	DEC CL	
			JNZ UP	
			DEC CH	
			JNZ UP1	
			INT 03	

Observation Table:

Input		Output	
MEMORY LOCATION	**Data**	**MEMORY LOCATION**	**Data**
2000		2000	
2001		2001	
2002		2002	
2003		2003	

2. **Descending order:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AX,0000	
			MOV CH,0004	
			DEC CH	
		UP1	MOV CL,CH	
			MOV SI,2000	
		UP	MOV AL,[SI]	
			INC SI	
			CMP AL,[SI]	
			JNC DOWN	
			XCHG AL,[SI]	
			DEC SI	
			MOV [SI],AL	
			INC SI	
		DOWN	DEC CL	
			JNZ UP	
			DEC CH	
			JNZ UP1	
			INT 03	

Observation Table:

Input		Output	
MEMORY LOCATION	**Data**	**MEMORY LOCATION**	**Data**
2000		2000	
2001		2001	
2002		2002	
2003		2003	

5.8 Probing Further Experiments

1. How to write a program to identify the largest number from a series of numbers in the array.

2. Identify the instruction that you have to change in the program to find smallest number from a series of numbers in the array.

3. Specify the instruction to modify ascending order program to descending order program.

LAB-5 Programs for String Manipulations operations

6.1 Introduction

String is s series of data byte or word available in memory at consecutive locations. It is either referred as byte string or word string. Their memory is always allocated in a sequential order. Instructions used to manipulate strings are called string manipulation instructions.

6.2 Objective

6.2.1 Educational

1. Learn about the architecture of 8086 microprocessor.
2. Learn about Flags and how they are set and reset in assembly operations.
3. Learn what registers are, why they are important and how to use them
4. Discover register indirect, Indexed addressing and how they are used in assembly programming
5. Discover how to jump to labeled parts of code based on flags.

6.2.2 Experimental

1. Write an assembly language program to insert a byte in a string.
2. Write an assembly language program to delete a byte in a string.
3. Write an assembly language program to move a block of data from one memory location to other memory location.
4. Write an assembly language program to reverse a string.
5. Write an assembly language program to search a number/character in a string.
6. Observe and analyze the outputs using registers and memory locations.

6.3 Prelab Preparation:

Reading

1. Read register organization, Instruction set, addressing modes of 8086 microprocessor.

Written

1. Prior coming to the lab complete part0 of the procedure.

6.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	RS-232		1

6.5 Background

Registers:
AX:Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.
BX:Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.
CX:Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation
SI:Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions are used in conjunction with the DS register to point the data locations in the data segment.
DI:Destination Index (DI) is a 16-bit register is used in conjunction with the ES register for string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data addresses in string manipulation instructions. In short, Destination Index and SI Source Index registers are used to hold address.
Addressing modes:
Immediate addressing mode:
In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.
Example: MOV AX, 0005H. In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.
Register indirect addressing mode:
Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES.
Example: MOV AX, [BX].
In this addressing mode, offset of the operand is stored one of the index registers. DS and ES are the default segments for index registers SI and DI respectively.
Example: MOV AX, [SI]
Here, data is available at an offset address stored in SI in DS.
Indexed addressing mode:
In this addressing mode, offset of the operand is stored one of the index registers. DS and ES are the default segments for index registers SI and DI respectively.
Example: MOV AX, [SI]
Here, data is available at an offset address stored in SI in DS.

Instruction set:
MOV instruction:
It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing.
General Format : MOV destination, source
Here the source and destination needs to be of the same size, that is both 8 bit or both 16 bit.MOV instruction does not affect any flags.
ADD instruction:
Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected
General Format: ADD Destination, Source
INC and DEC instructions
INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.
JZ instruction
JE / JZ : Stands for 'Jump if Equal' or 'Jump if Zero'. The JZ (or JE) instruction is a conditional jump that follows a test It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, then jump.
JNZ instruction:
The jnz (or jne) instruction is a conditional jump that follows a test. It jumps to the specified location if the Zero Flag (ZF) is cleared (0). jnz is commonly used to explicitly test for something not being equal to zero
General Format: JNZ location
JNC instruction:
The JNC instruction(jump if no carry) is a conditional jump that follows a test. It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump.
General Format: JNC location
INT 03 instruction:
The INT 3 instruction generates a special one byte opcode (CC) that is intended for calling the debug exception handler. (This one byte form is valuable because it can be used to replace the first byte of any instruction with a breakpoint, including other one byte instructions, without over-writing other code).
General Format: INT 03
XCHG instruction:
The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations.
General Format : XCHG Destination, Source
CMP instruction:
The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset.
General Format : CMP Destination, Source
CLD instruction:
This instruction is used to reset the direction flag to zero so that SI and/or DI can be incremented automatically after execution of string instruction. CLD does not affect any other flag.

MOVSB instruction:
These instructions copy a word or byte from a location in the data segment to a location in the extra segment. The offset of the source is in SI and that of destination is in DI. For multiple word/byte transfers the count is stored in the CX register.When direction flag is 0, SI and DI are incremented and when it is 1, SI and DI are decremented.MOVSB is used for byte sized movements while MOVSW is for word sized.
REP/REPE/REP2/REPNE/REPNZ
REP is used with string instruction; it repeats an instruction until the specified condition becomes false.

6.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Switch on the power supply after checking connections
3. Handle the Trainer kit carefully.

6.7 Procedure

WIN862 Software procedure(for memory locations):

1. Open Win862 icon on desktop and opened Window

Figure 6.1: Win862 icon

Figure 6.2: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

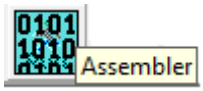


Figure 6.3: Assembler icon

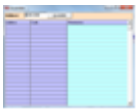


Figure 6.4: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 6.5: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Click on Run (check whether program is executed or not)
9. Click on view memory
10. Now enter input address
11. Click on Modify and Give desired input values
12. Click on Set PC. Enter initial address and press Dis-Assembler
13. Click on Run (check whether program is executed or not)
14. Now observe the result in view memory.

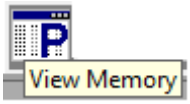


Figure 6.6: view memory

15. Click on view memory and enter destination address then press enter button
16. Now observe the result.

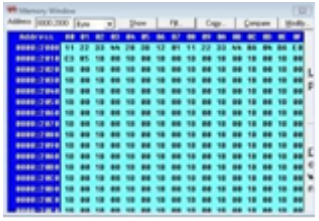


Figure 6.7: memory window

textbfPrograms:String manipulations

1. **Inserting a byte in a string:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV SI,2000	
			MOV DI,3000	
			MOV BX,5000	
			MOV CX,0005	
			CLD	
		L1:	MOV AL,[SI]	
			CMP AL,[BX]	
			JZ L2	
			MOVSB	
			LOOP L1	
			JMP L3	
		L2:	MOVSB	
			MOV BX,7000	
			MOV AL,[BX]	
			MOV [DI],AL	
			DEC CX	
			INC DI	
			REP	
			MOVSB	
		L3:	INT 03	

Observation Table:

Input		Output	
MEMORY LOCATION	**Data**	**MEMORY LOCATION**	**Data**
2000		3000	
2001		3001	
2002		3002	
2003		3003	
2004		3004	
5000		3005	
7000			

2. **Deleting a byte in a string:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV SI,2000	
			MOV DI,3000	
			MOV BX,5000	
			MOV CX,0005	
			CLD	
		L1	MOV AL,[SI]	
			CMP AL,[BX]	
			JZ L2	
			MOVSB	
			LOOP L1	
			JMP L3	
		L2	INC SI	
			DEC CX	
			REP	
			MOVSB	
		L3	INT 03	

Observation Table:

Input		Output	
MEMORY LOCATION	Data	MEMORY LOCATION	Data
2000		3000	
2001		3001	
2002		3002	
2003		3003	
2004			
5000			

3. **Moving a block of data from one location to other location:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV SI, 2000	
			MOV DI, 2008	
			MOV CX, 0008	
			REP	
			MOVSB	
			INT 03	

Observation Table:

Input		Output	
MEMORY LOCATION	**Data**	**MEMORY LOCATION**	**Data**
2000		2008	
2001		2009	
2002		200A	
2003		200B	
2004		200C	
2005		200D	
2006		200E	
2007		200F	

4. **Reversing a string:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV SI,2000	
			MOV DI,5000	
			MOV CX,0008	
			ADD SI,07	
		UP :	MOV AL,[SI]	
			MOV [DI],AL	
			DEC SI	
			INC DI	
			DEC CX	
			JNZ UP	
			INT 03	

Observation Table:

Input		Output	
MEMORY LOCATION	**Data**	**MEMORY LOCATION**	**Data**
2000		2008	
2001		2009	
2002		200A	
2003		200B	
2004		200C	
2005		200D	
2006		200E	
2007		200F	

5. **Search a number/character in a string:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV CX,0004	
			MOV AX,0000	
			MOV SI,2000	
			MOV BX,3000	
		UP	MOV AL,[SI]	
			CMP AL,[BX]	
			JZ DOWN	
			INC SI	
			DEC CL	
			JNZ UP	
			MOV AH,00	
			JMP L3	
		DOWN	DEC CL	
			MOV AH,01	
			MOV [DI], AH	
		L3	INT 03	

Observation Table:

Input		Output	
MEMORY LOCATION	Data	MEMORY LOCATION	Data
2000		3000	
2001			
2002			
2003			

6.8 Probing Further Experiments

1. How to write a program to find the number of even and odd numbers from a given series of 16-bit hexadecimal numbers.

2. Write a program to move a string of data words from offset 2000H to offset 3000H the length of the string id OFH.

3. How to find out positive and negative numbers from a given series of signed numbers.

4. What is the effect of direction flag for string manipulation operation.

5. Do required Modifications in the program to support word comparison instead of byte comparison

LAB-6 Code Conversions

7.1 Introduction

Code conversion allows user to translate a number that is represented using one coding system to other coding system. Data in the form of text and numbers are used for programming the electronic devices. But computers cannot understand human language. They can only understand the data in the form of 0's and 1's. To make data interpretable by computer many number formats are being used. Some of them are the Binary number system, Octal number system, Hexadecimal number system, etc. To make the text understandable by computers ASCII codes are used. Internal converters are used for converting data from one format to another. In this the code conversion involves operations like Packed BCD to Unpacked BCD, BCD to ASCII, Hexadecimal number to ASCII number. The microprocessor understands the binary/hex number system. In byte-oriented systems, the term unpacked BCD usually implies a full byte for each digit (often including a sign), whereas packed BCD typically encodes two digits within a single byte by taking advantage of the fact that four bits are enough to represent the range 0 to 9. To convert packed BCD to ASCII, it must first be converted to unpacked BCD. Then the unpacked BCD is tagged with 30H. ASCII uses a one-byte word for representing a character. So, split the hexadecimal into the pairs, as each digit of hexadecimal is 4-bits. For each pair, find the specified ASCII character from the ASCII lookup table.

7.2 Objective

7.2.1 Educational

1. Learn about the architecture of 8086 microprocessor.
2. Learn to identify proper instructions to be used for conversion.
3. Learn fundamentals of number systems.
4. Understand the concepts and techniques associated with the number systems and codes.

7.2.2 Experimental

1. To Write an assembly language program to convert packed BCD number to Unpacked BCD number.
2. To Write an assembly language program to convert packed BCD number to ASCII number.
3. To Write an assembly language program to convert hexadecimal number to ASCII number.

7.3 Prelab Preparation:

Reading

1. Study the fundamentals of number systems, code conversions in Digital Logic Design, Register organization, Instruction set and Addressing modes of 8086 microprocessor.

Written

1. Prior coming to the lab complete part0 of the procedure.

7.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	RS-232		1

7.5 Background

Registers:
AX:Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.
BX:Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.
CX:Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation
SI:Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions are used in conjunction with the DS register to point the data locations in the data segment.
DI:Destination Index (DI) is a 16-bit register is used in conjunction with the ES register for string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data addresses in string manipulation instructions. In short, Destination Index and SI Source Index registers are used to hold address.
Addressing modes:
Immediate addressing mode:
In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes. Example: MOV AX, 0005H.
In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.
Register indirect addressing mode:
Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode.In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES.
Example: MOV AX, [BX].
In this addressing mode, offset of the operand is stored one of the index registers. DS and ES are the default segments for index registers SI and DI respectively.

Example: MOV AX, [SI]
Here, data is available at an offset address stored in SI in DS.
Indexed addressing mode:
In this addressing mode, offset of the operand is stored one of the index registers. DS and ES are the default segments for index registers SI and DI respectively.
Example: MOV AX, [SI]
Here, data is available at an offset address stored in SI in DS.
Instruction set:
MOV instruction:
It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing.
General Format : MOV destination, source
Here the source and destination needs to be of the same size, that is both 8 bit or both 16 bit. MOV instruction does not affect any flags.
ADD instruction:
Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected
General Format: ADD Destination, Source
INC and DEC instructions
INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.
CMP instruction:
The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset.
General Format : CMP Destination, Source
AND instruction:
This instruction logically ANDs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.
General Format: AND Destination, Source
OR instruction:
This instruction logically ORs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.
General Format: OR Destination, Source
SHR instruction:
This instruction shifts each bit in the specified destination to the right and 0 is stored in the MSB position. The LSB is shifted into the carry flag. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count. All flags are affected
General Format: SHR destination, count
JNZ instruction:
The JNZ (or JNE) instruction is a conditional jump that follows a test. It jumps to the specified location if the Zero Flag (ZF) is cleared (0). JNZ is commonly used to explicitly test for

something not being equal to zero
General Format: JNZ location
JC instruction:
The JC instruction(jump if carry) is a conditional jump that follows a test. It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, then jump.
General Format: JC location
INT 03 instruction:
The INT 3 instruction generates a special one byte opcode (CC) that is intended for calling the debug exception handler. (This one byte form is valuable because it can be used to replace the first byte of any instruction with a breakpoint, including other one byte instructions, without over-writing other code).
General Format: INT 03
XCHG instruction:
The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations.
General Format : XCHG Destination, Source
CMP instruction:
The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset.
General Format : CMP Destination, Source
CLD instruction:
This instruction is used to reset the direction flag to zero so that SI and/or DI can be incremented automatically after execution of string instruction. CLD does not affect any other flag.
MOVSB instruction:
These instructions copy a word or byte from a location in the data segment to a location in the extra segment. The offset of the source is in SI and that of destination is in DI. For multiple word/byte transfers the count is stored in the CX register.When direction flag is 0, SI and DI are incremented and when it is 1, SI and DI are decremented.MOVSB is used for byte sized movements while MOVSW is for word sized.
REP/REPE/REP2/REPNE/REPNZ:
REP is used with string instruction; it repeats an instruction until the specified condition becomes false.

7.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Switch on the power supply after checking connections
3. Handle the Trainer kit carefully.

7.7 Procedure

WIN862 Software procedure(for memory locations):

1. Open Win862 icon on desktop and opened Window

Figure 7.1: Win862 icon

Figure 7.2: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

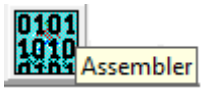


Figure 7.3: Assembler icon

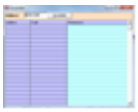


Figure 7.4: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 7.5: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Click on Run (check whether program is executed or not)
9. Click on view memory

10. Now enter input address
11. Click on Modify and Give desired input values
12. Click on Set PC. Enter initial address and press Dis-Assembler
13. Click on Run (check whether program is executed or not)
14. Now observe the result in view memory.

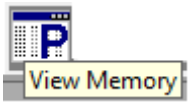


Figure 7.6: view memory

15. Click on view memory and enter destination address then press enter button
16. Now observe the result.

Figure 7.7: memory window

Programs:Code conversions

1. **Packed BCD number to Unpacked BCD number:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AX,0000	
			MOV AL,72	
			MOV AH,AL	
			AND AL,0F	
			MOV CL,04	
			SHR AH,CL	
			INT 03	

Observation Table:

Input		Output	
REGISTER	**Data**	**REGISTER**	**Data**
AL		AX	

2. **Packed BCD number to ASCII number:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AL,56	
			MOV AH,AL	
			AND AL,0F	
			MOV CL,04	
			SHR AH,CL	
			OR AX,3030	
			INT 03	

Observation Table:

Input		Output	
REGISTER	**Data**	**REGISTER**	**Data**
AL		AX	

3. **Hexadecimal number to ASCII number:**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV SI,2000	
			MOV DI,3000	
			MOV CX,0003	
		UP	MOV AL,[SI]	
			CMP AL,0A	
			JC FWD	
			ADD AL,07	
		FWD	OR AL,30	
			MOV [DI], AL	
			INC SI	
			INC DI	
			DEC CX	
			JNZ UP	
			INT 03	

Observation Table:

Input		Output	
MEMORY LOCATION	**Data**	**MEMORY LOCATION**	**Data**
2000		3000	
2001		3001	
2002		3002	
2003		3003	
2004		3004	

7.8 Probing Further Experiments

1. Use RCR and XOR instructions and write a program to convert 8 bit binary number in to equivalent grey code.

2. How to write a program to convert binary number into equivalent BCD number.

3. Distinguish packed BCD and unpacked BCD.

4. If the content of 8-bit register rotated 4 times what is the change that can be observed in the register.

LAB-7 Interfacing Stepper Motor to 8086 microprocessor

8.1 Introduction

This laboratory explores the Interfacing of stepper motor to 8086 microprocessor and rotates it in clockwise and anticlock wise direction. A stepper motor is a type of DC motor that rotates in steps. When electrical signal is applied to it, the motor rotates in steps and the speed of rotation depends on the rate at which the electrical signals are applied and the direction of rotation is dependent on the pattern of pulses that is followed.

8.2 Objective

8.2.1 Educational

1. Learn about the architecture of 8086 microprocessor.
2. Learn the working principle of stepper motor.
3. Learn to calculate speed of stepper motor.
4. Understand the driving sequence of stepper motor.
5. Learn to calculate the step angle.

8.2.2 Experimental

1. To rotate the stepper motor in clock wise direction.
2. To rotate the stepper motor in anticlock wise direction.

8.3 Prelab Preparation:

Reading

1. Study the Instruction set of 8086 microprocessor, addressing modes and working principle of stepper motor.

Written

1. Prior coming to the lab complete part0 of the procedure.

8.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	RPS	+5v	1
4	Stepper motor interfacing card		1
5	Stepper motor		1
6	FRC Connector,RS-232 cable		1

8.5 Background

A stepper motor is a device used to obtain an accurate position control of rotating shafts. It employs rotation of its shaft in terms of steps, rather than continuous rotation as in case of AC or DC motors. To rotate the shaft of the stepper motor, a sequence of pulses is needed to be applied to the windings of the stepper motor, in a proper sequence.
The number of pulses required for one complete rotation of the shaft of the stepper motor is equal to its number of internal teeth on its rotor. The stator teeth and the rotor teeth lock with each other to fix a position of the shaft.
With a pulse applied to the winding input, the rotor rotates by one teeth position or an angle x. The angle x may be calculated as:
X=3600/no. of rotor teeth
After the rotation of the shaft through angel x, the rotor locks itself with the next tooth in the sequence on the internal surface of stator.
The internal schematic of a typical stepper motor with four windings.
The stepper motors have been designed to work with digital circuits. Binary level pulses of 0-5V are required at its winding inputs to obtain the rotation of shafts. The sequence of the pulses can be decided, depending upon the required motion of the shaft.

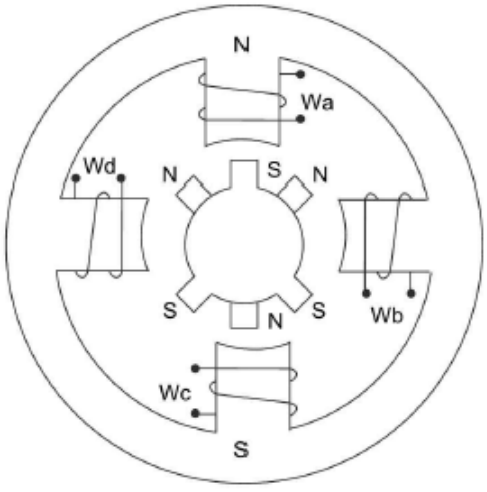


Figure 8.1: Internal schematic of a four winding stepper motor

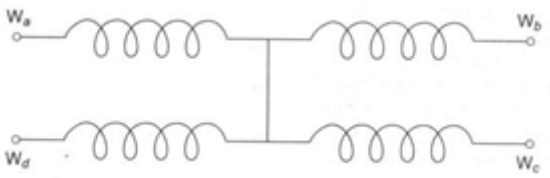


Figure 8.2: Winding arrangement of a stepper motor

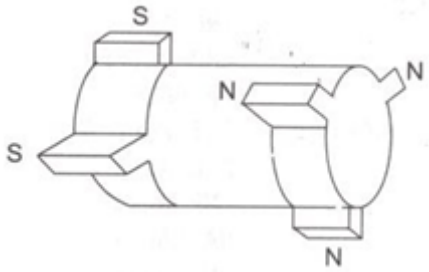


Figure 8.3: Stepper motor rotor

The circuit for interfacing a winding Wn with an I/O port. Each of the windings of a stepper motor needs this circuit for its interfacing with the output port. A typical stepper motor may have parameters like torque 3 Kg-cm, operating voltage 12V, current rating 0.2 A and a step angle 1.80 i.e. 200 steps/revolution (number of rotor teeth).
A simple schematic for rotating the shaft of a stepper motor is called a wave scheme. In this scheme, the windings Wa, Wb, Wc and Wd are applied with the required voltages pulses, in a cyclic fashion. By reversing the sequence of excitation, the direction of rotation of the stepper motor shaft may be reversed.
Table.1 shows the excitation sequences for clockwise and anticlockwise rotations. Another popular scheme for rotation of a stepper motor shaft applies pulses to two successive windings at a time but these are shifted only by one position at a time. This scheme for rotation of stepper motor shaft is shown in table2.

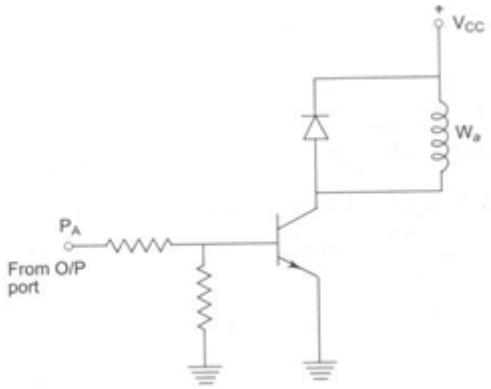


Figure 8.4: Interfacing stepper motor winding

Motion	step	A	B	C	D
Clock Wise Direction	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
	5	1	0	0	0
Anti clock wise Direction	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0

Motion	step	A	B	C	D
Clock Wise Direction	1	0	0	1	1
	2	0	1	1	0
	3	1	1	0	0
	4	1	0	0	1
	5	0	0	1	1
Anti clock wise Direction	1	0	0	1	1
	2	1	0	0	1
	3	1	1	0	0
	4	0	1	1	0
	5	0	0	0	0

8.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Carefully connect stepper motor to stepper motor interfacing card.
3. Switch on the power supply after checking connections
4. Handle the Trainer kit carefully.

8.7 Procedure

WIN862 Software procedure:

1. Open Win862 icon on desktop and opened Window

Figure 8.5: Win862 icon

Figure 8.6: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

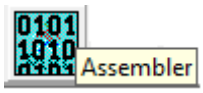


Figure 8.7: Assembler icon

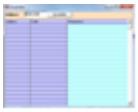


Figure 8.8: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 8.9: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Connect stepper motor interfacing card along with stepper motor to the trainer board.
9. Click on Run (check whether program is executed or not)

Programs:Stepper motor interfacing

1. **rotating stepper motor in clockwise direction :**

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AL,80	
			MOV DX,0FFE6	
			OUT DX	
			MOV BX,1770	
			MOV AL,33	
			MOV DX,0FFE0	
		BACK	OUT DX	
			MOV CX,1262	
		SELF	LOOP SELF	
			ROR AL,1	
			DEC BX	
			JNZ BACK	
			INT 03	

2. **rotating stepper motor in anticlockwise direction :**

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AL,80	
			MOV DX,0FFE6	
			OUT DX	
			MOV BX,1770	
			MOV AL,33	
			MOV DX,0FFE0	
		BACK	OUT DX	
			MOV CX,1262	
		SELF	LOOP SELF	
			ROL AL,1	
			DEC BX	
			JNZ BACK	
			INT 03	

8.8 Probing Further Experiments

1. Indicate all the possible bit sequences that should be applied to the windings of stepper motor to rotate in anticlock wise direction.

2. In this program we used Port A as the input port, so the value of CWR is 80H, If port B is used as input port What is the value of CWR.

3. Identify the instruction that should be changed in the clock wise direction program to rotate in anti clock wise direction.

LAB-8 Interfacing ADC and DAC Devices to 8086 microprocessor

9.1 Introduction

Many events monitored and controlled by the microprocessor are analog events. The ADC and DAC devices are used to interface the microprocessor to the analog world. Analog-to-Digital Converters (ADC's) convert analog signals to digital data. They are a common peripheral used with microprocessors for applications such as monitoring analog circuitry (voltages, temperature sensors, etc), digitizing audio and video signals, digitizing radio signal, etc. Digital-to-Analog converters (DAC's) convert digital data to analog signals. They are a common peripheral used with microprocessors for applications such as controlling analog circuitry, audio and video generation, radio signal generation, etc.

9.2 Objective

9.2.1 Educational

1. Learn about the architecture of 8086 microprocessor
2. Learn about conversion techniques in ADC.
3. Learn features and pin configuration of ADC0808/0809
4. Learn ADC0808/0809 Block diagram and its operation.
5. Learn about DAC0800 Digital to Analog Converter.
6. Gain experience in writing assembly language programs on generating different waveforms.

9.2.2 Experimental

1. To configure ADC module and read analog signals.
2. To measure digital output for given analog input.
3. To generate square, sawtooth and triangular waveforms at desired frequencies.

9.3 Prelab Preparation:

Reading

1. Read and study about features, pin configuration and operationADC0808/0809 and DAC0800.

Written

1. Prior coming to the lab complete part0 of the procedure.

9.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	RPS	+5v	1
4	A/D Interfacing module		1
5	D/A Interfacing modules		1
6	Power mate connector		1
7	FRC Connector, RS-232 cable		1
8	CRO		1

9.5 Background

Analog to Digital Data Converters Interfacing: In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
The analog to digital converters is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
The process of analog to digital conversion is a slow.
Process and the microprocessor have to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. The set asks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.
The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
It may range anywhere from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.
General algorithm for ADC interfacing contains the following steps:
•Ensure the stability of analog input, applied to the ADC.
•Issue start of conversion pulse to ADC.
•Read end of conversion signal to mark the end of conversion processes.
•Read digital data output of the ADC as equivalent digital output.
Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by as ample and hold circuit which samples the analog signal and holds it constant for specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.
ADC 0808/0809:
The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation

converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.
These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C, as shown. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.
There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do not contain any internal sample and hold circuit.
If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

Analog I/P selected	Address lines		
	C	B	A
I/P 0	0	0	0
I/P 1	0	0	1
I/P 2	0	1	0
I/P 3	0	1	1
I/P 4	1	0	0
I/P 5	1	0	1
I/P 6	1	1	0
I/P 7	1	1	1

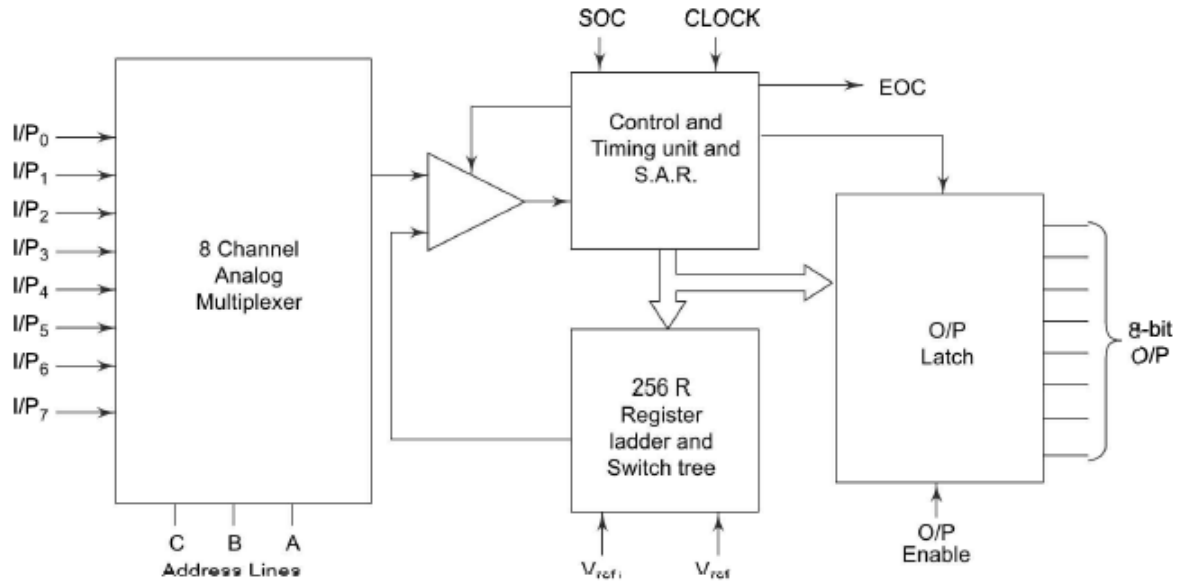
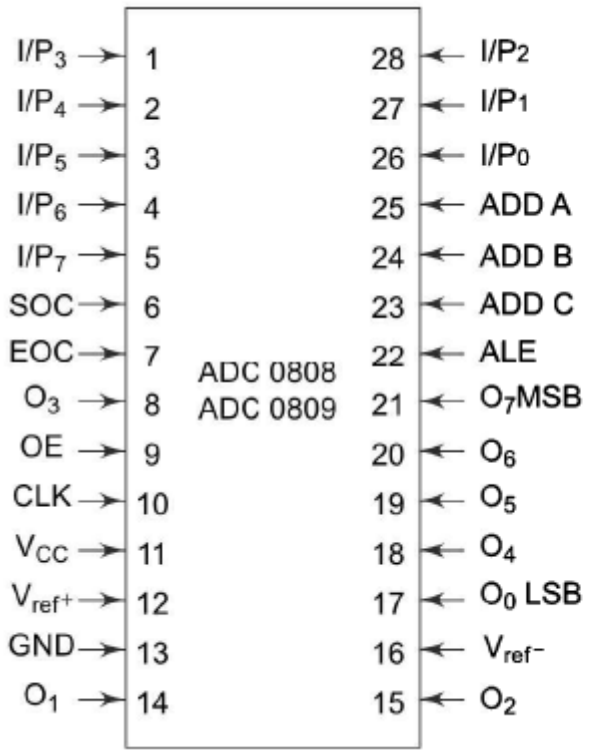


Figure 9.1: Block Diagram of ADC 0808



I/P_0–I/P_7	Analog inputs
ADD A, B, C	Address lines for selecting analog
O_7–O_0	Digital 8-bit output with O_7 MSB a
SOC	Start of conversion signal pin
EOC	End of conversion signal pin
OE	Output latch enable pin, if high ena
CLK	Clock input for ADC
V_{CC}, GND	Supply pins +5V and GND
V_{ref+} and V_{ref-}	Reference voltage positive (+5 Volt and Reference voltage negative (0

Figure 9.2: Pin Diagram of ADC 0808

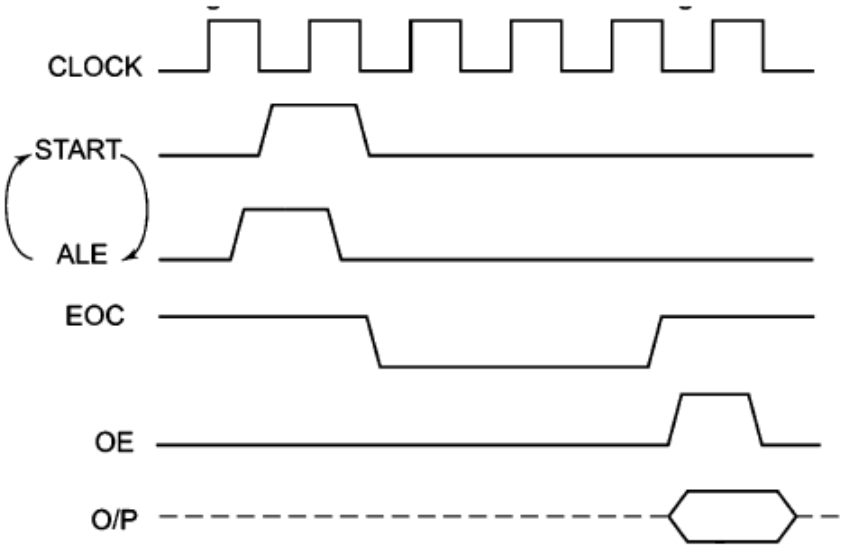


Figure 9.3: Timing Diagram Of ADC 0808

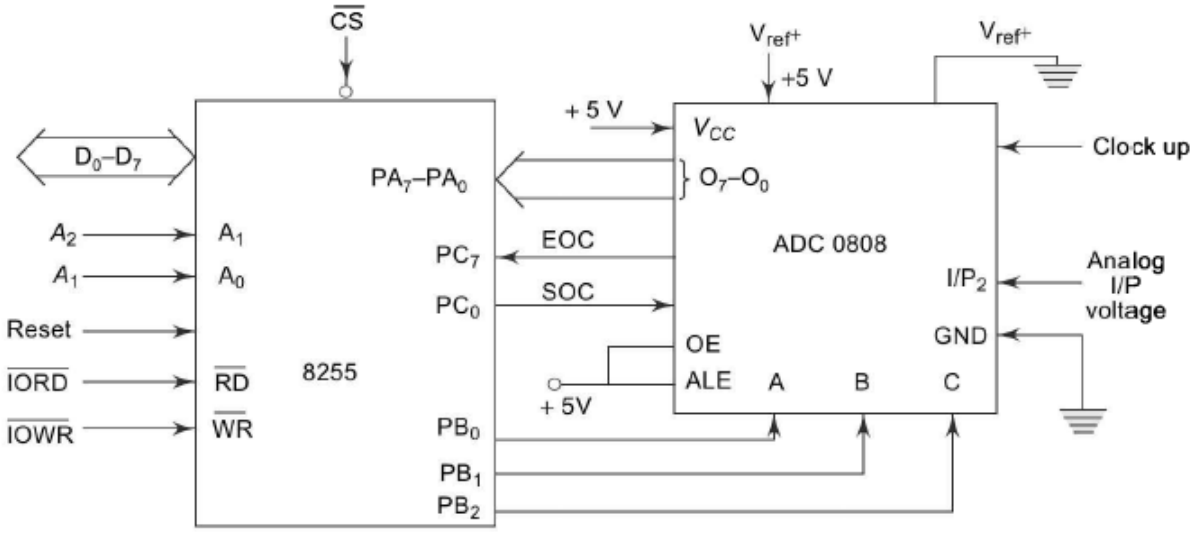


Figure 9.4: Interfacing ADC0808 with 8086

Interfacing Digital To Analog Converters:
The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.
DAC0800 8-bit Digital to Analog Converter:
The DAC 0800 is a monolithic 8-bit DAC manufactured by National Semiconductor.
It has settling time around 100ms and can operate on a range of power supply voltages i.e. from 4.5V to +18V.
Usually the supply V+ is 5V or +12V.
The V-pin can be kept at a minimum of -12V.

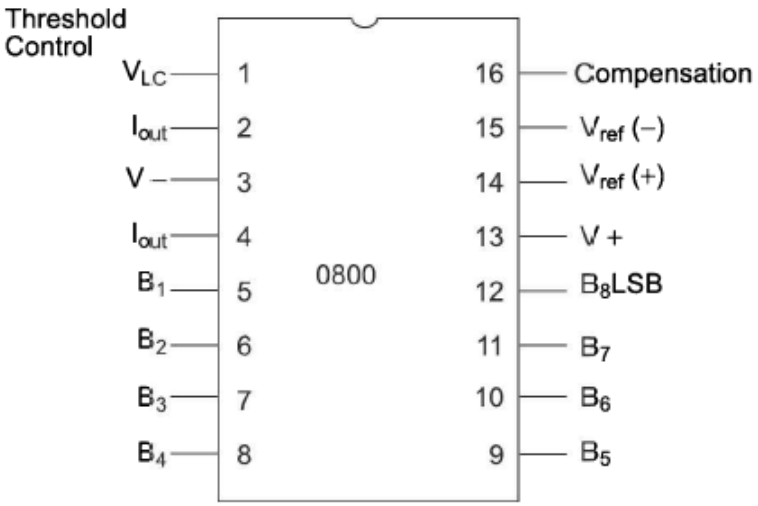


Figure 9.5: Pin Diagram of DAC 0800

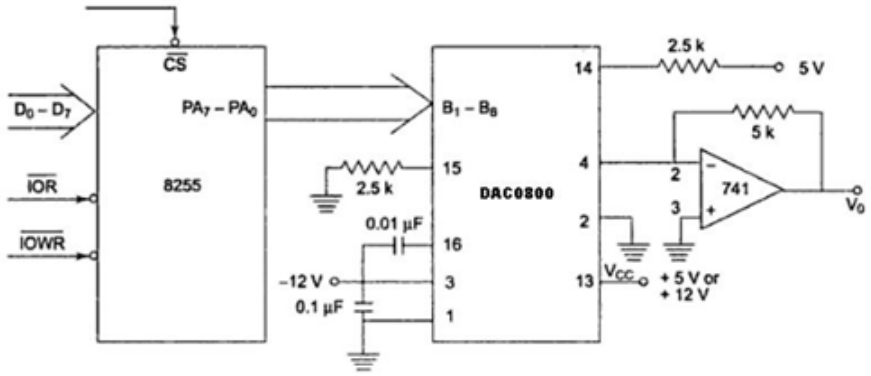


Figure 9.6: Interfacing DAC0800 with 8086

Digital to Analog Converter Interfacing :
Intersil's AD 7523 is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder(R=10KΩ) for digital to analog conversion along with single pole double through NMOS switches to connect the digital inputs to the ladder.

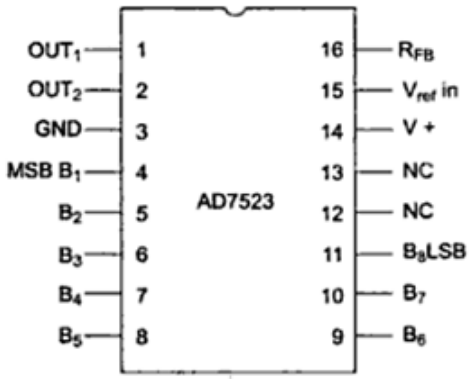


Figure 9.7: Pin Diagram of AD7523

The supply range extends from +5V to +15V , while Vref may be anywhere between -10V to +10V. The maximum analog output voltage will be +10V, when all the digital inputs are at logic high state. Usually a Zener is connected between OUT1 and OUT2 to save the DAC from negative transients.
An operational amplifier is used as a current to voltage converter at the output of AD 7523 to convert the current output of AD7523 to a proportional output voltage.
It also offers additional drive capability to the DAC output. An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.

9.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Carefully connect CRO to DAC interfacing card.
3. Switch on the power supply after checking connections
4. Handle the Trainer kit carefully.

9.7 Procedure

WIN862 Software procedure:

1. Open Win862 icon on desktop and opened Window

Figure 9.8: Win862 icon

Figure 9.9: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

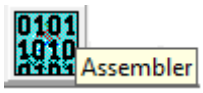


Figure 9.10: Assembler icon

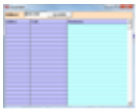


Figure 9.11: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 9.12: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Connect ADC or DAC interfacing card to the trainer board.
9. Click on Run (check whether program is executed or not)

Programs:ADC and DAC interfacing

1. **Analog to Digital Converter :**

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AL, 98H	
			MOV DX, 0FFE6	
			OUT DX,AL	
			MOV AL, 01H	
			OUT DX,AL	
			MOV AL, 00H	
			OUT DX,AL	
			MOV AL, 02H	
			MOV DX, 0FFE2H	
			OUT DX,AL	
			MOV DX, 0FFE4H	
			IN AL,DX	
			ROR AL, 1H	
			JNC BACK	
			MOV DX, 0FFE0H	
		BACK:	IN AL,DX	
			MOV DI, 2000H	
			MOV [DI], AL	
			INT 03H	

Observation Table:

Input		Output	
REGISTER	**Data**	**MEMORY LOCATION**	**Data**
AL		2000	

2. **Digital to Analog Converter:**
   Generation of Square wave :

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AL,80	
			MOV DX,0FFE6	
			OUT DX	
			MOV DX,0FFE0	
		BACK	MOV AL,00	
			OUT DX	
			MOV CX,0147	
		SELF1	LOOP SELF1	
			MOV AL,0FF	
			OUT DX	
			MOV CX,0147	
		SELF2	LOOP SELF2	
			JMP BACK	

Generation of Triangular wave :

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AL,80	
			MOV DX,0FFE6	
			OUT DX	
			MOV AL,00	
		L3	MOV DX,0FFE2	
		L1	OUT DX	
			INC AL	
			CMP AL,0FF	
			JB L1	
		L2	OUT DX	
			DEC AL	
			CMP AL,00	
			JNBE L2	
			JMP L3	

Generation of Sawtooth wave :

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AL,80	
			MOV DX,0FFE6	
			OUT DX	
		L2	MOV AL,00	
			MOV DX,0FFE2	
		L1	OUT DX	
			INC AL	
			CMP AL,0FF	
			JB L1	
			OUT DX	
			JMP L2	

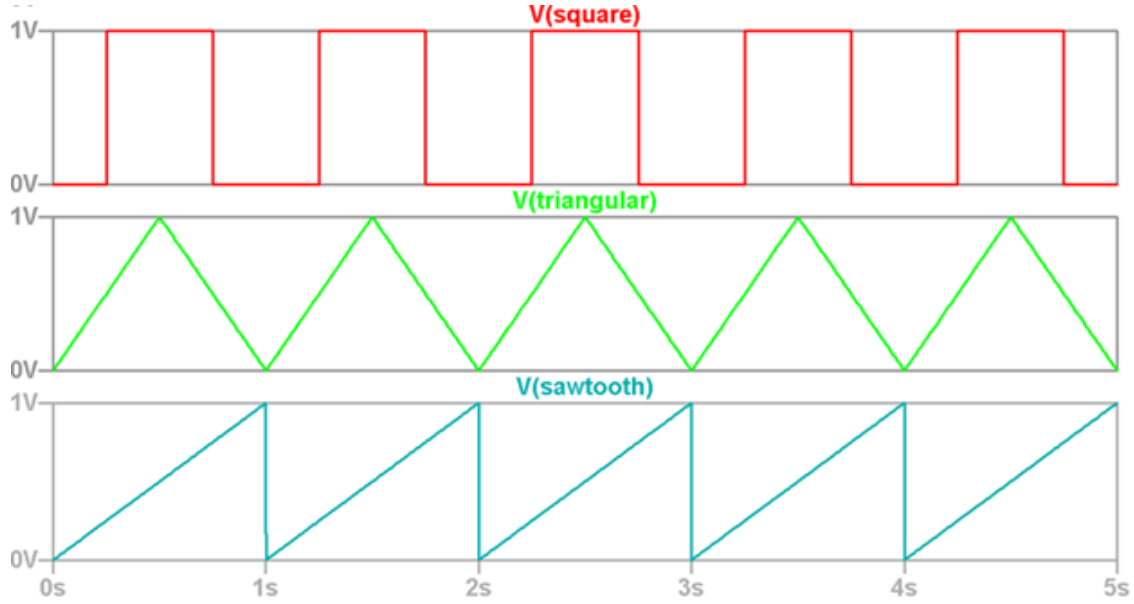


Figure 9.13: Analog Waveforms

9.8 Probing Further Experiments

1. In this A/D converter program for 5V analog input voltage range, the A/D converter is producing digital output in the range of 00H to FFH. If the input analog voltage is 0V, what is the corresponding digital output.

2. For 0-5V analog input voltage range, the A/D converter is producing digital output in the range of 00H to FFH. If the input analog voltage is 0.196V, what is the corresponding digital output.

3. If Iref =2mA, and digital input to D/A converter IC 0800 is 10000000, Find the output current of 0800.

LAB-9 Interfacing Keyboard to 8086 Microprocessor

10.1 Introduction

8279 programmable keyboard/display controller is designed by Intel that simultaneously drives the display of the system and interfaces a keyboard with the CPU, leaving it free for its routine task. The keyboard-display interface first scans the keyboard and identifies if any key has been pressed. It then sends their relative response of the pressed key to the CPU. It also transmits the data received from the CPU, to the display device. Both of these functions are performed by the controller without involving CPU. The Keyboard can be interfaced either in the interrupt or the polled mode. In the Interrupt mode, the processor is requested service only if any key is pressed, otherwise the CPU will continue with its main task. In the Polled mode, the CPU periodically reads an internal flag of 8279 to check whether any key is pressed or not with key pressure.

10.2 Objective

10.2.1 Educational

1. Learn about the architecture of 8086 microprocessor
2. Learn about features and pin configuration of 8279 keyboard/display controller.
3. Learn different ways of making the switches.
4. Learn 8279 Architecture and its operation.
5. Gain experience in Keyboard Circuit Connections and Interfacing.

10.2.2 Experimental

1. To Interface keyboard module to 8086 microprocessor and identify the keypress.

10.3 Prelab Preparation:

Reading

1. Study about features, pin configuration and operation of 8279 keyboard/display controller.

Written

1. Prior coming to the lab complete part0 of the procedure.

10.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	Keyboard Interfacing module		1
4	FRC Connector, RS-232 cable		1

10.5 Background

Keyboard Interfacing: The keyboard consists of maximum 64 keys, which are interfaced with the CPU by using the key-codes. These key-codes are de-bounced and stored in an 8-byte FIFORAM, which can be accessed by the CPU. If more than 8 characters are entered in the FIFO, then it means more than eight keys are pressed at a time. This is when the overrun status is set.

If a FIFO contains a valid key entry, then the CPU is interrupted in an interrupt mode else the CPU checks the status in polling to read the entry. Once the CPU reads a key entry, then FIFO is updated, and the key entry is pushed out of the FIFO to generate space for new entries.

Architecture and Description:

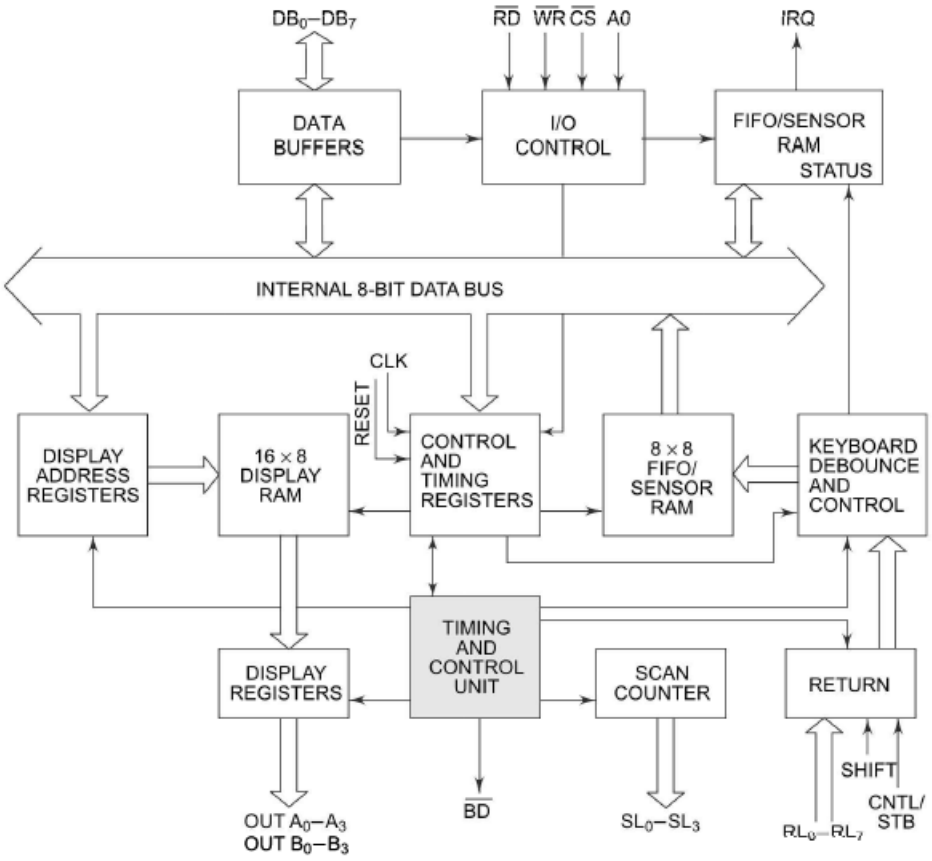


Figure 10.1: 8279 Internal architecture

I/O Control and Data Buffer:

This unit controls the flow of data through the microprocessor. It is enabled only when D is low. Its data buffer interfaces the external bus of the system with the internal bus of the microprocessor. The pins A0, RD, and WR are used for command, status or data read/write operations.

Control and Timing Register and Timing Control:

This unit contains registers to store the keyboard, display modes, and other operations as programmed by the CPU. The timing and control unit handles the timings for the operation of the circuit.

Scan Counter:

It has two modes i.e. Encoded mode and Decoded mode. In the encoded mode, the counter provides the binary count that is to be externally decoded to provide the scan lines for the keyboard and display. In the decoded scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL0-SL3

Return Buffers, Keyboard Debounce, and Control:

This unit first scans the key closure row-wise, if found then the keyboard debounce unit debounces the key entry. In case, the same key is detected, then the code of that key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.

FIFO/Sensor RAM and Status Logic:

This unit acts as 8-byte first-in-first-out (FIFO) RAM where the key code of every pressed key is entered into the RAM as per their sequence. The status logic generates an interrupt request after each FIFO read operation till the FIFO gets empty.

In the scanned sensor matrix mode, this unit acts as sensor RAM where its each row is loaded with the status of their corresponding row of sensors into the matrix. When the sensor changes its state, the IRQ line changes to high and interrupts the CPU.

Display Address Registers and Display RAM:

This unit consists of display address registers which holds the addresses of the word currently read/written by the CPU to/from the display RAM

8279 Pin Description:

Data Bus Lines, DB0 - DB7

These are 8 bidirectional data bus lines used to transfer the data to/from the CPU.

CLK

The clock input is used to generate internal timings required by the microprocessor.

RESET

As the name suggests this pin is used to reset the microprocessor.

CS

When this pin is set to low, it allows read/write operations, else this pin should be set to high.

A0

This pin indicates the transfer of command/status information. When it is low, it indicates the transfer of data.

RD, WR

This Read/Write pin enables the data buffer to send/receive data over the data bus.

IRQ

This interrupt output line goes high when there is data in the FIFO sensor RAM. The interrupt line goes low with each FIFO RAM read operation. However, if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

Vss, Vcc

These are the ground and power supply lines of the microprocessor.

SL0 to SL3

These are the scan lines used to scan the keyboard matrix and display the digits. These lines can be programmed as encoded or decoded, using the mode control register.

RL0 to RL7

These are the Return Lines which are connected to one terminal of keys, while the other terminal of the keys is connected to the decoded scan lines. These lines are set to 0 when any key is pressed.

SHIFT

The Shift input line status is stored along with every key code in FIFO in the scanned keyboard mode. Till it is pulled low with a key closure, it is pulled up internally to keep it high

CNTL/STB - CONTROL/STROBED I/P Mode

In the keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line is a strobe line that enters the data into FIFO RAM, in the strobed input mode. It has an internal pull up. The line is pulled down with a key closure.

BD

It stands for blank display. It is used to blank the display during digit switching.

OUTA0 – OUTA3 and OUTB0 – OUTB3

These are the output ports for two 16x4 or one 16x8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and the keyboard.

Operational Modes of 8279

There are two modes of operation on 8279:

Input Mode and Output Mode.

Input Mode

This mode deals with the input given by the keyboard and this mode is further classified into 3 modes.

Scanned Keyboard Mode :In this mode, the key matrix can be interfaced using either encoded or decoded scans. In the encoded scan, an 8×8 keyboard or in the decoded scan, a 4×8 keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.

Scanned Sensor Matrix: In this mode, a sensor array can be interfaced with the processor using either encoder or decoder scans. In the encoder scan, 8×8 sensor matrix or with decoder scan 4×8 sensor matrix can be interfaced.

Strobed Input:In this mode, when the control line is set to 0, the data on the return lines is stored in the FIFO byte by byte.

Output Mode

This mode deals with display-related operations. This mode is further classified into two output modes.

Display Scan: This mode allows 8/16 character multiplexed displays to be organized as dual 4-bit/single 8-bit display units.

Display Entry: This mode allows the data to be entered for display either from the right side/left side.

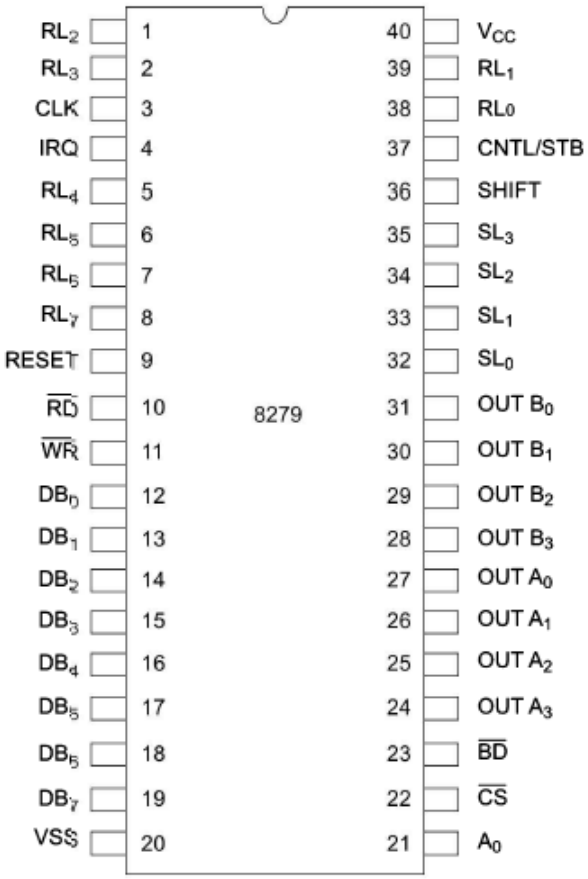


Figure 10.2: pin diagram of 8279

10.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Carefully connect Keyboard interfacing card to 8086 trainer kit.
3. Switch on the power supply after checking connections
4. Handle the Trainer kit carefully.

10.7 Procedure

WIN862 Software procedure:

1. Open Win862 icon on desktop and opened Window

Figure 10.3: Win862 icon

Figure 10.4: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

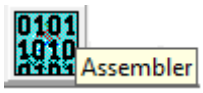


Figure 10.5: Assembler icon

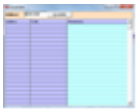


Figure 10.6: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 10.7: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Connect keyboard interfacing card to the trainer board.
9. Click on Run (check whether program is executed or not)

Program:Keyboard interfacing

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AX,0000	
			MOV AL,90H	
			MOV DX,0FFE6H	
			OUT DX,AL	
			CALL CLR	
		BACK:	CALL KESCN	
			MOV AL,BH	
			MOV SI,2100H	
			MOV AH,00H	
			ADD AX,SI	
			MOV SI,AX	
			MOV CL,[SI]	
			CALL CLR	
			MOV CH,08	
		D0:	RCL CL,1	
			JNB D1	
			MOV AL,02	
			MOV DX,0FFE4	
			OUT DX	
			MOV AL,03	
			MOV DX,0FFE4	
			OUT DX	
			JMP D2	
		D1:	MOV AL,00	
			MOV DX,0FFE4	
			OUT DX	
			MOV AL,03	
			MOV DX,0FFE4	
			OUT DX	
		D2:	DEC CH	
			JNE D0	
			JMP BACK	
		CLR:	MOV BL,06	
		S2:	MOV BH,08	
		S1:	MOV AL,02	
			MOV DX,0FFE4	
			OUT DX	
			MOV AL,03	
			MOV DX,0FFE4	
			OUT DX	
			MOV AL,03	
			MOV DX,0FFE4	
			OUT DX	
			DEC BH	
			JNE S1	
			DEC BL	
			JNE S2	
			RET	

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
		KESCN:	MOV BL,03	
			MOV BH,0F	
			MOV CL,08	
		NXTGRP:	MOV AL,CL	
			MOV DX,0FFE2	
			OUT DX	
			RCR AL,01	
			MOV CL,AL	
			MOV DX,0FFE0	
			INW DX	
			AND AL,1F	
			CMP AL,00	
			JNE NXTKEY	
		NXTGRP:	SUB BH,05	
			DEC BL	
			CMP BL,FF	
			JNE NXTGRP	
			JMP KESCN	
		NXTKEY:	RCR AL,01H	
			JNB NZ	
			RET	
		NZ:	ADD BH,01	
			JMP NXTKEY	

10.8 Probing Further Experiments

1. How to handle key debouncing in keyboard controller.

2. In the column scanning if it returns 1 in all bit positions what it signifies.

3. What is Two-key lockout.

LAB-10 Serial and Parallel Communication

11.1 Introduction

Data is to be sent from the source to the destination, and it is necessary for the source and destination formats to be similar for compatability between them. In parallel communication all the bits are sent and received together. Data transfer between registers in a processor is done this way. This is fine as long as the source and destination are in close proximity, but when they are placed far apart say two computers in two separate buildings, a lot of problems occurs. If we are sending 8 bits 8 long wires are required, problem occurs if the bit size increases. So serial communication is preferred compared to parallel communication. In serial communication we send only q bit at a time, one after the other. So 8 bits need 8 times the time required, compared to the previous case. The advantage is only one physical wire is required for transmission.

11.2 Objective

11.2.1 Educational

1. Learn about the architecture of 8086 microprocessor
2. Identify the difference between serial communication and Parallel communication.
3. Learn about serial data transmission modes.
4. Learn about Asynchronous and Synchronous data transfer schemes.
5. Understand the operation of 8251 USART and 8255 PPI.
6. Gain experience on how to interface I/O devices using peripherals.

11.2.2 Experimental

1. To establish Parallel communication between two microprocessors using 8255.
2. To establish Serial communication between two microprocessor kits using 8251.

11.3 Prelab Preparation:

Reading

1. Read and study the various types and modes of communication.
2. Study about features, pin configuration and operation of 8251 USART, 8255 PPI.

Written

1. Prior coming to the lab complete part0 of the procedure.

11.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	Serial communication module		1
4	Parallel communication module		1
5	FRC Connector, RS-232 cable		1

11.5 Background

Serial and Parallel Transmission: Let us now try to have a comparative study on parallel and serial communications to understand the differences and advantages and disadvantages of both in detail. We know that parallel ports are typically used to connect a PC to a printer and are rarely used for other connections. A parallel port sends and receives data eight bits at a time over eight separate wires or lines. This allows data to be transferred very quickly. However, the setup looks more bulky because of the number of individual wires it must contain. But, in the case of a serial communication, as stated earlier, a serial port sends and receives data, one bit at a time over one wire. While it takes eight times as long to transfer each byte of data this way, only a few wires are required. Although this is slower than parallel communication, which allows the transmission of an entire byte at once, it is simpler and can be used over longer distances. So, at first sight it would seem that a serial link must be inferior to a parallel one, because it can transmit less data on each clock tick. However, it is often the case that, in modern technology, serial links can be clocked considerably faster than parallel links, and achieves a higher data rate.

Even in shorter distance communications, serial computer buses are becoming more common because of a tipping point where the disadvantages of parallel busses (clock skew, interconnect density) outweigh their advantage of simplicity. The serial port on your PC is a full-duplex device meaning that it can send and receive data at the same time. In order to be able to do this, it uses separate lines for transmitting and receiving data.

From the above discussion we could understand that serial communications have many advantages over parallel one like:

•Requires fewer interconnecting cables and hence occupies less space.

•"Cross talk" is less of an issue, because there are fewer conductors
compared to that of parallel communication cables.

Many IC s and peripheral devices have serial interfaces.

•Clock skew between different channels is not an issue.

•Cheaper to implement.

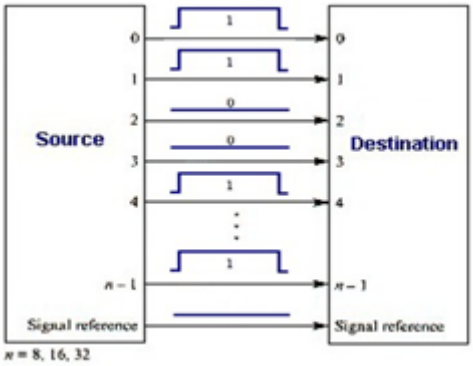


Figure 11.1: Parallel transmission

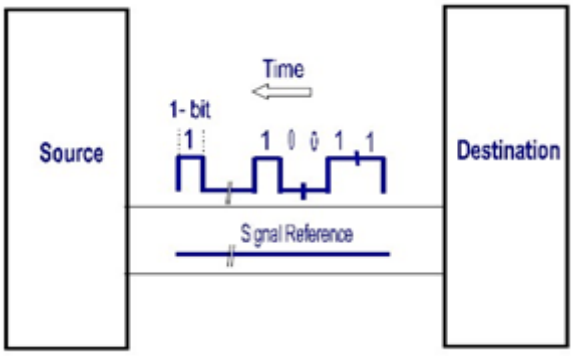


Figure 11.2: Serial transmission

Clock skew:
Clock skew is a phenomenon in synchronous circuits in which the clock signal sent from the clock circuit arrives at different components at different times, which can be caused by many things, like:
•Wire-interconnect length
•Temperature variations
•Variation in intermediate devices
•capacitive coupling
•Material imperfections

Serial Data Transmission Modes
When data is transmitted between two pieces of equipment, three communication modes of operation can be used.
Simplex:In a simple connection, data is transmitted in one direction only. For example, from a computer to printer that cannot send status signals back to the computer.
Half-duplex:In a half-duplex connection, two-way transfer of data is possible, but only in one direction at a time.
Full duplex:In a full-duplex configuration, both ends can send and receive data simultaneously, which technique is common in our PCs.

Serial Data Transfer Schemes
Like any data transfer methods, Serial Communication also requires coordination between the sender and receiver. For example, when to start the transmission and when to end it, when one particular bit or byte ends and another begins, when the receiver's capacity has been exceeded,

and so on. Here comes the need for synchronization between the sender and the receiver. A protocol defines the specific methods of coordinating transmission between a sender and receiver. For example a serial data signal between two PCs must have individual bits and bytes that the receiving PC can distinguish. If it doesn't, then the receiving PC can't tell where one byte ends and the next one begin or where one bit ends and begins. So the signal must be synchronized in such a way that the receiver can distinguish the bits and bytes as the transmitter intends them to be distinguished.
There are two ways to synchronize the two ends of the communication.
1.Synchronous data transmission
2.Asynchronous data transmission

Asynchronous And Synchronous Data Transfer Schemes:

Synchronous Data Transmission

The synchronous signaling methods use two different signals. A pulse on one signal line indicates when another bit of information is ready on the other signal line.
In synchronous transmission, the stream of data to be transferred is encoded and sent on one line, and a periodic pulse of voltage which is often called the "clock" is put on another line, that tells the receiver about the beginning and the ending of each bit.

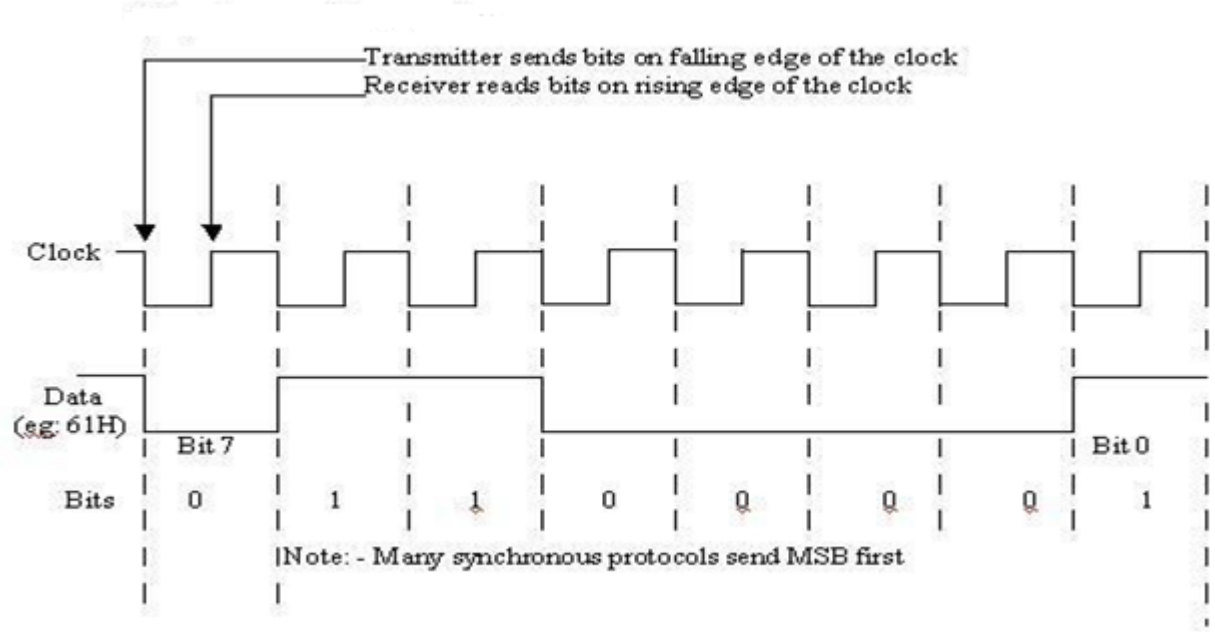


Figure 11.3: Synchronous transmission

Advantages: The only advantage of synchronous data transfer is the Lower overhead and thus, greater throughput, compared to asynchronous one.

Disadvantages: •Slightly more complex
•Hardware is more expensive

Asynchronous data transmission

The asynchronous signaling methods use only one signal. The receiver uses transitions on that signal to figure out the transmitter bit rate (known as auto baud) and timing. A pulse from the local clock indicates when another bit is ready. That means synchronous transmissions use an external clock, while asynchronous transmissions are use special signals along the transmission medium. Asynchronous communication is the commonly prevailing communication method in the personal computer industry, due to the reason that it is easier to implement and has the

unique advantage that bytes can be sent whenever they are ready, a no need to wait for blocks of data to accumulate.

Advantages:

•Simple and doesn't require much synchronization on both communication sides.

•The timing is not as critical as for synchronous transmission; therefore hardware can be made cheaper.

•Set-up is very fast, so well suited for applications where messages are generated at irregular intervals, for example data entry from the keyboard.

disadvantages:

One of the main disadvantages of asynchronous technique is the large relative overhead, where a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information.

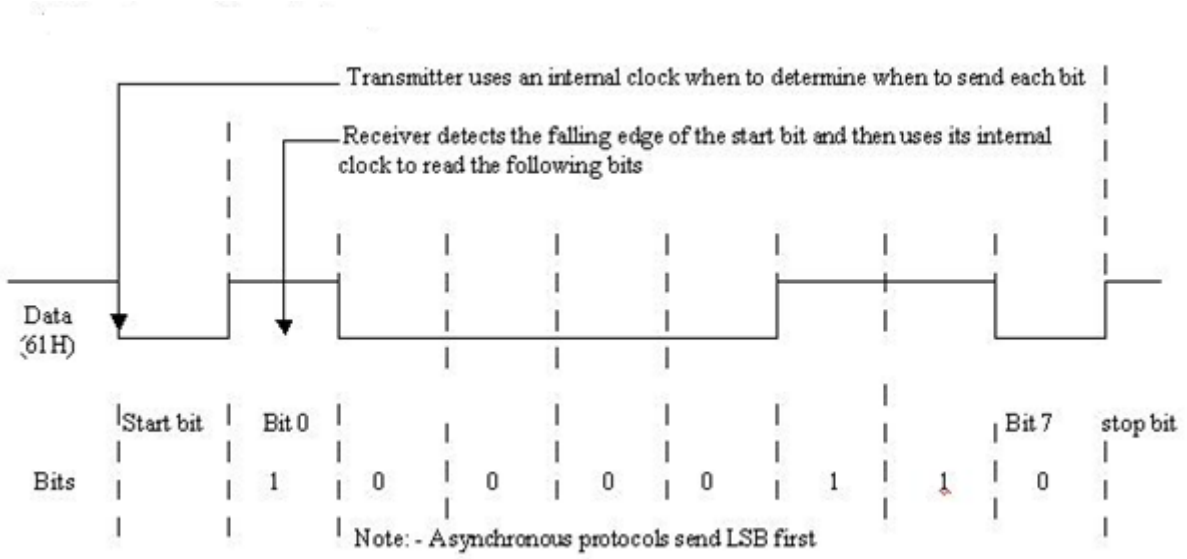


Figure 11.4: Asynchronous transmission

8251 USART Architecture and Interfacing:

8251A-Programmable Communication Interface:

(8251A-USART-Universal Synchronous/Asynchronous Receiver/Transmitter):

Introduction:

A USART is also called a programmable communications interface (PCI). When information is to be sent by 8086 over long distances, it is economical to send it on a single line. The 8086 has to convert parallel data to serial data and then output it. Thus lot of microprocessor time is required for such a conversion.

Similarly, if 8086 receives serial data over long distances, the 8086 has to internally convert this into parallel data before processing it. Again, lot of time is required for such a conversion. The 8086 can delegate the job of conversion from serial to parallel and vice versa to the 8251A USART used in the system.

The Intel8251A is the industry standard Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel microprocessor families such as 8080, 85, 86 and The 8251A converts the parallel data received from the processor on the D7-0 data pins into serial data, and transmits it on TxD (transmit data) output pin of 8251A. Similarly, it converts the serial data received on RxD (receive data) input into parallel data, and the processor reads it using the data pins D7-0.

Features:
•Compatible with extended range of Intel microprocessors.
•It provides both synchronous and asynchronous data transmission.
•Synchronous 5-8 bit characters.
•Asynchronous 5-8 bit characters.
•It has full duplex, double buffered transmitter and receiver.
•Detects the errors-parity, overrun and framing errors.
•All inputs and outputs are TTL compatible.
•Available in 28-pin DIP package.

Architecture:
The 8251A is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251 receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion. The internal block diagram of 8251A is shown in fig below.

Data Bus Buffer:
This bidirectional, 8-bit buffer used to interface the 8251A to the system data bus and also used to read or write status, command word or data from or to the 8251A.

Read/Write control logic:
The Read/Write Control logic interfaces the 8251A with microprocessor, determines the functions of the 8251A according to the control word written into its control register and monitors the data flow. This section has three registers and they are control register, status register and data buffer.
•When the reset is high, it forces 8251A into the idle mode.
•The clock input is necessary for 8251A for communication with microprocessor and this clock does not control either the serial transmission or the reception rate.

Transmitter section:
The transmitter section accepts parallel data from microprocessor and converts them into serial data. The transmitter section is double buffered, i.e., it has a buffer register to hold an 8-bit parallel data and another register called output register to convert the parallel data into serial bits. When output register is empty, the data is transferred from buffer to output register. Now the processor can again load another data in buffer register.
•If buffer register is empty, then TxRDY is goes to high.
•If output register is empty then TxEMPTY goes to high.
•The clock signal controls the rate at which the bits are transmitted by the USART.
•The clock frequency can be 1,16 or 64 times the baud rate.

Receiver Section:
The receiver section accepts serial data and converts them into parallel data. The receiver section is double buffered, i.e., it has an input register to receive serial data and convert to parallel, and a buffer register to hold the parallel data. When the RxD line goes low, the control logic assumes it as a START bit, waits for half a bit time and samples the line again. If the line is still low, then the input register accepts the following bits, forms a character and loads it into the buffer register.

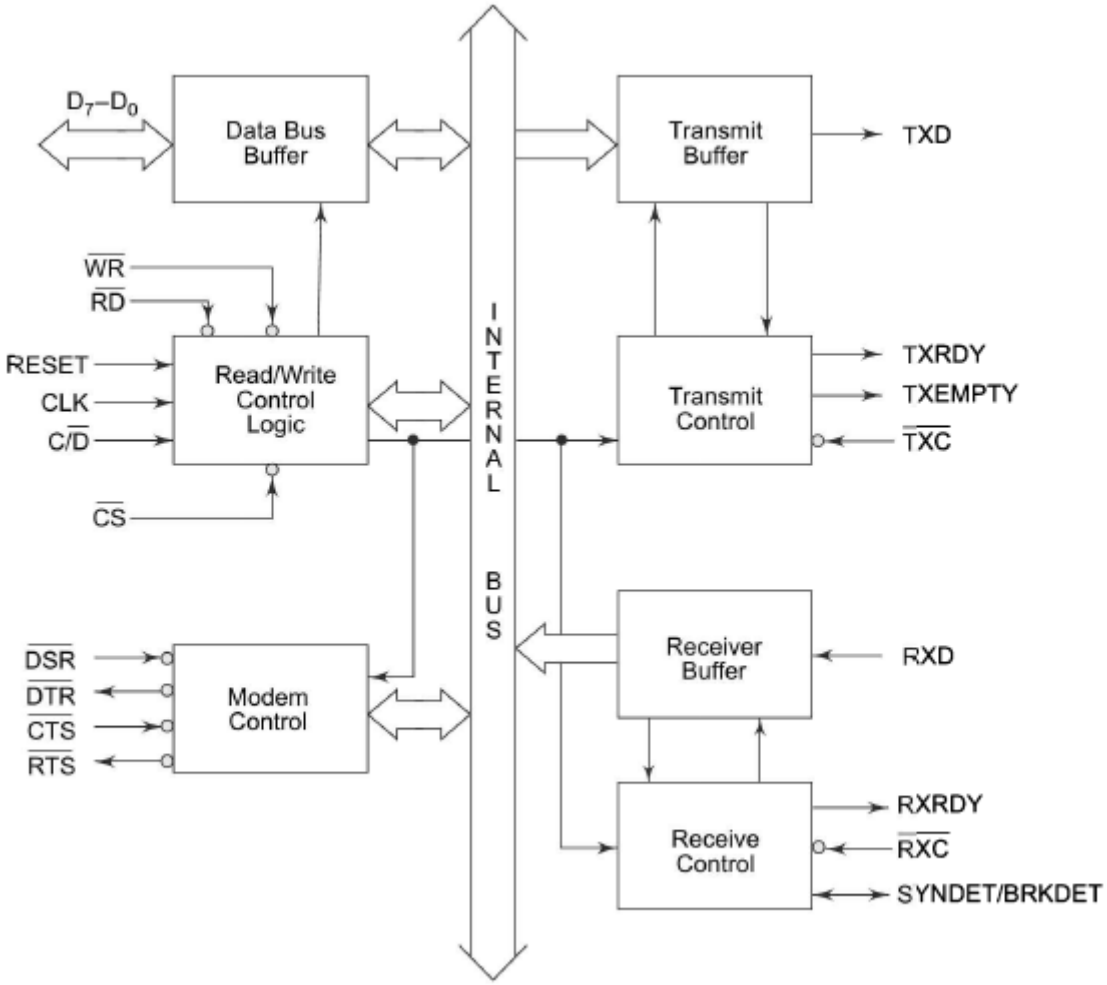


Figure 11.5: 8251A Internal architecture

The microprocessor reads the parallel data from the buffer register.

•When the input register loads a parallel data to buffer register, the RxRDY line goes high.

•The clock signal controls the rate at which bits are received by the USART.

•During asynchronous mode, the signal SYNDET/BRKDET will indicate the break in the data transmission. During synchronous mode, the signal SYNDET/BRKDET will indicate the reception of synchronous character.

MODEM Control:

The MODEM control unit allows to interface a MODEM to 8251A and to establish data communication through MODEM over telephone lines. This unit takes care of handshake signals for MODEM interface.

8251A Pin configuration:

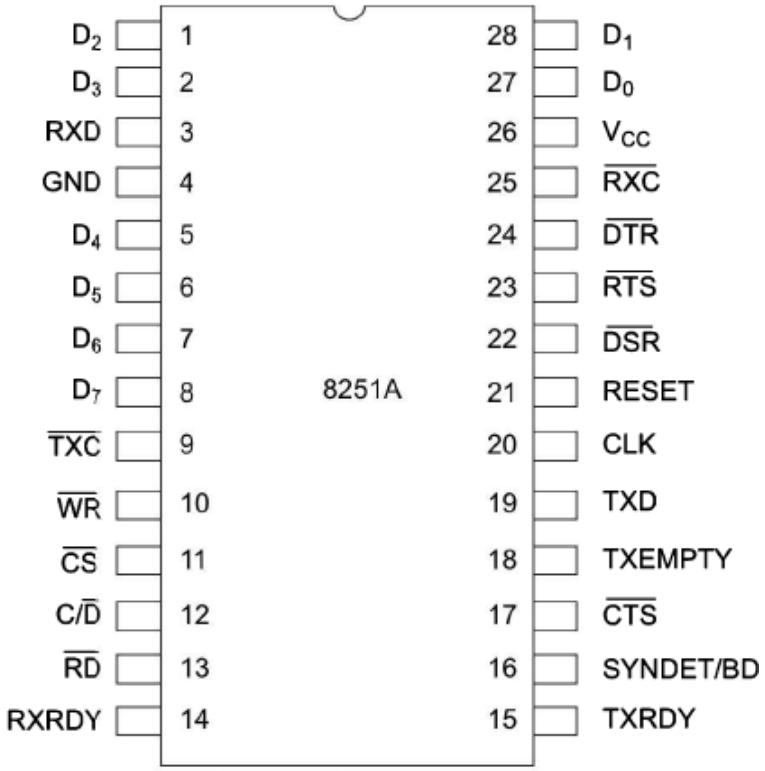


Figure 11.6: 8251A Pin Configuration.

D0 to D7 (l/O terminal):
This is bidirectional data bus which receives control words and transmits data from the CPU and sends status words and received data to CPU.
RESET (Input terminal):
A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction." The min. reset width is six clock inputs during the operating status of CLK.
CLK (Input terminal) :
CLK signal is used to generate internal device timing. CLK signal is independent of RXC or TXC. However, the frequency of CLK must be greater than 30 times the RXC and TXC at Synchronous mode and Asynchronous "x1" mode, and must be greater than 5 times at Asynchronous "x16" and "x64" mode.
WR (Input terminal) :
This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.
RD (Input terminal) :
This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.
C/ D (Input terminal) :
This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU. If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed.
CS (Input terminal) :
This is the "active low" input terminal which selects the 8251 at low level when the CPU ac-

cesses. Note: The device won't be in "standby status"; only setting CS = High.

TXD (output terminal) :

This is an output terminal for transmitting data from which serial-converted data is sent out. The device is in "mark status" (high level) after resetting or during a status when transmit is disabled. It is also possible to set the device in "break status" (low level) by a command.

TXRDY (output terminal):

This is an output terminal which indicates that the 8251is ready to accept a transmitted data character. But the terminal is always at low level if CTS = high or the device was set in "TX disable status" by a command. Note: TXRDY status word indicates that transmit data character is receivable, regardless of CTS or command. If the CPU writes a data character, TXRDY will be reset by the leading edge or WR signal.

TXEMPTY (Output terminal):

This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode," the terminal is at high level, if transmit data characters are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note : As the transmitter is disabled by setting CTS "High" or command, data written before disable will be sent out. Then TXD and TXEMPTY will be "High". Even if a data is written after disable, that data is not sent out and TXE will be "High". After the transmitter is enabled, it sent out. (Refer to Timing Chart of Transmitter Control and Flag Timing)

TXC (Input terminal):

This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16 or 1/64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

RXD (input terminal):

This is a terminal which receives serial data.

RXRDY (Output terminal) :

This is a terminal which indicates that the 8251 contains a character that is ready to READ. If the CPU reads a data character, RXRDY will be reset by the leading edge of RD signal. Unless the CPU reads a data character before the next one is received completely, the preceding data will be lost. In such a case, an overrun error flag status word will be set.

RXC (Input terminal) :

This is a clock input signal which determines the transfer speed of received data. In "synchronous mode," the baud rate is the same as the frequency of RXC. In "asynchronous mode," it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16, 1/64 the RXC.

SYNDET/BD (Input or output terminal):

This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode, "this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters.

In "asynchronous mode," this is an output terminal which generates "high level "output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

DSR (Input terminal) :

This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.

DTR (Output terminal):

This is an output port for MODEM interface. It is possible to set the status of DTR by a

command.

CTS (Input terminal):

This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmittable if the terminal is at low level.

RTS (Output terminal):

This is an output port for MODEM interface. It is possible to set the status RTS by a command.

8251A USART Interfacing with 8086:

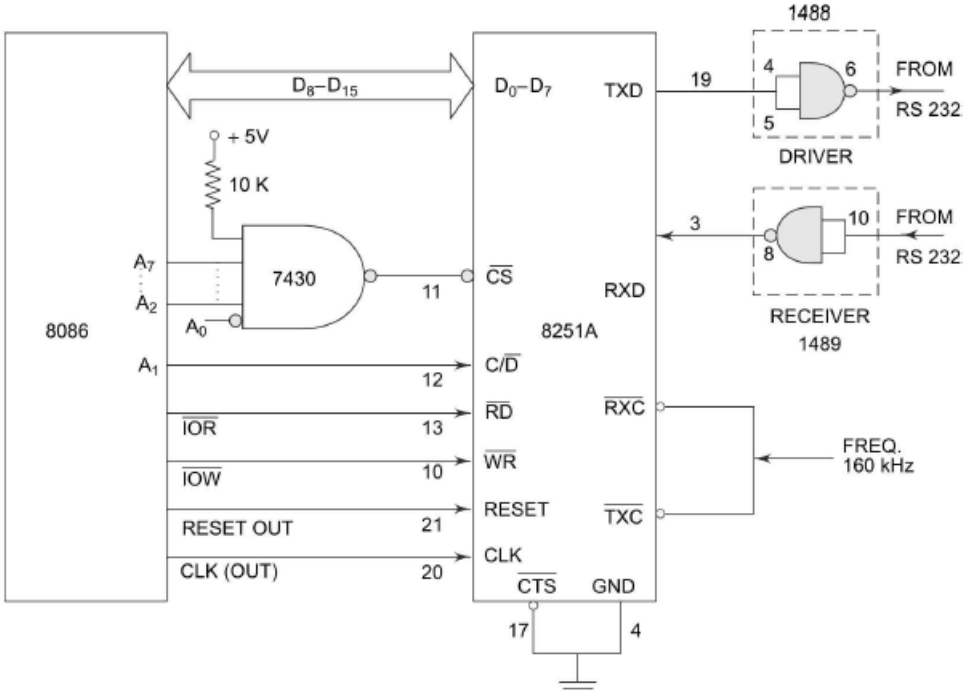


Figure 11.7: Interfacing 8251A USART with 8086 microprocessor

Programming 8251A :

Prior to starting a data transmission or reception, the 8251A must be loaded with a set of control words generated by the microprocessor. These control signals define the complete functional definition of the 8251A and must immediately follow a reset operation (internal or external). The control words are split into two formats.

1.Mode instruction

2.Command instruction

Mode instruction:

Mode instruction is used for setting the function of the 8251A. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."

Items set by mode instruction are as follows:

•Synchronous/asynchronous mode

•Stop bit length (asynchronous mode)

•Character length

•Parity bit

•Baud rate factor (asynchronous mode)

•Internal/external synchronization (synchronous mode)

•Number of synchronous characters (Synchronous mode)

The bit configuration of mode instruction format is shown in Figures below. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

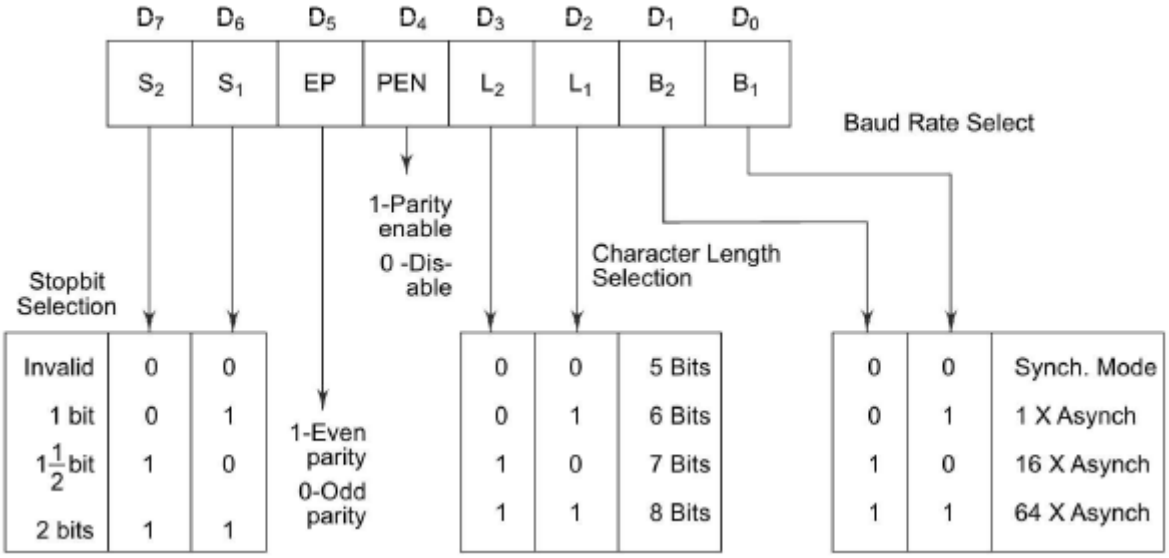


Figure 11.8: Mode instruction format Asynchronous mode

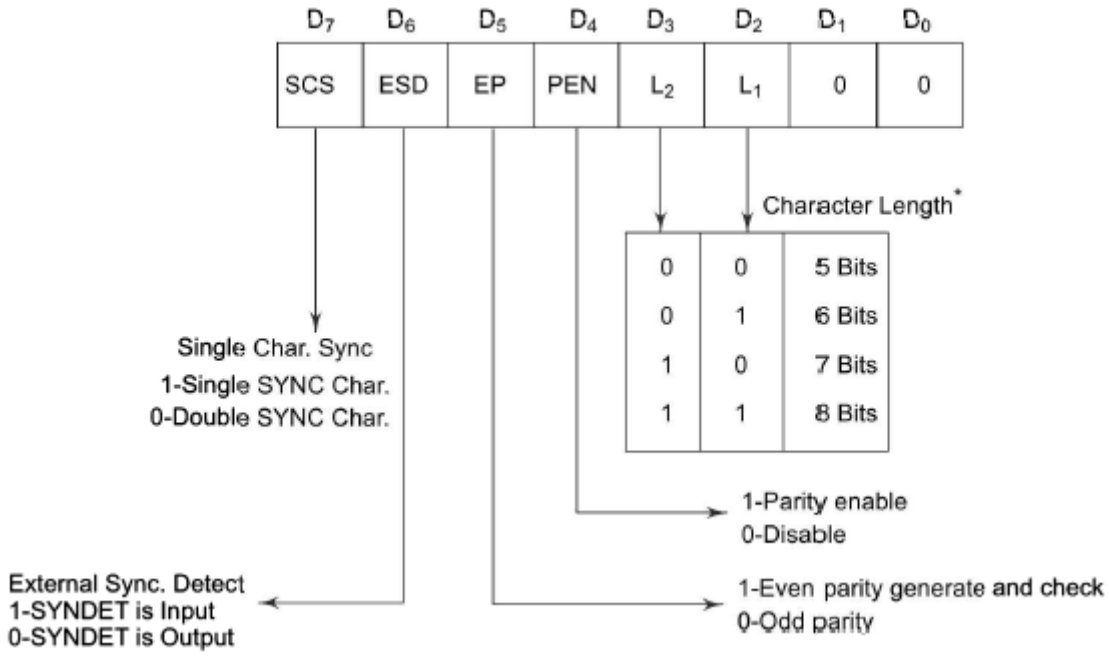


Figure 11.9: Mode instruction format Synchronous mode

Command Instruction:

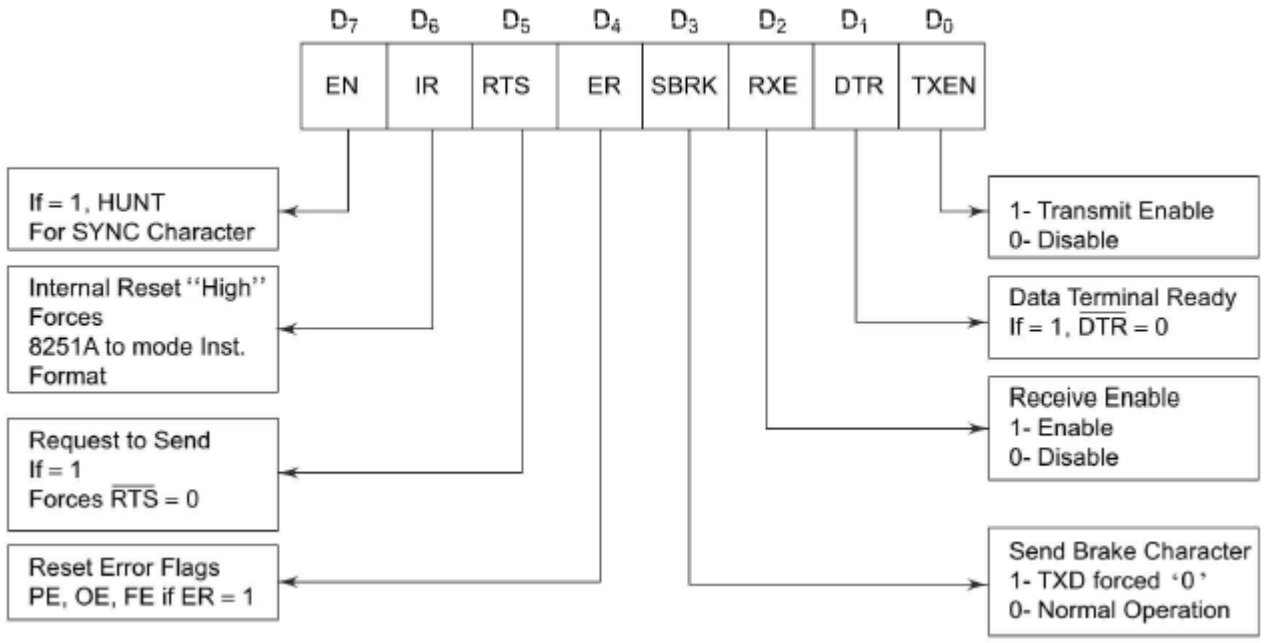


Figure 11.10: Command Instruction format

Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.
Items to be set by command are as follows:
•Transmit Enable/Disable
•Receive Enable/Disable
•DTR, RTS Output of data.
•Resetting of error flag.
•Sending to break characters
•Internal resetting
•Hunt mode (synchronous mode)

Status Word:

It is possible to see the internal status of the 8251 by reading a status word. The format of status word is shown below.

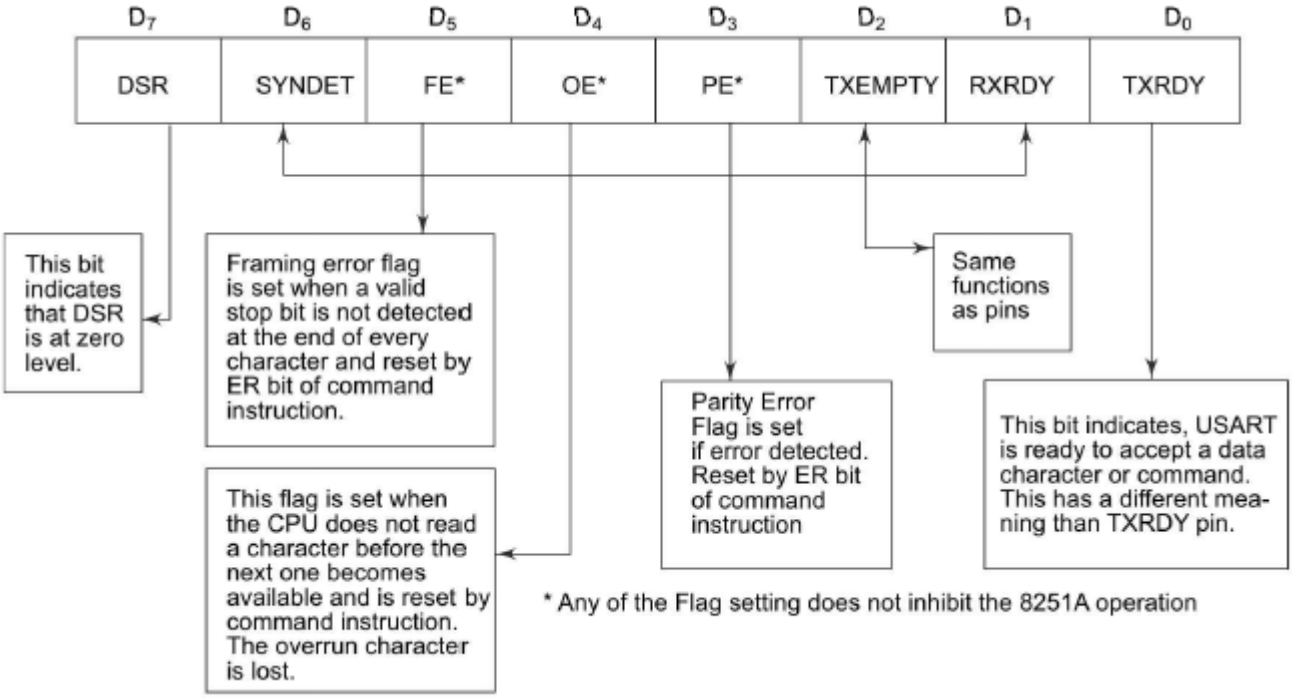


Figure 11.11: Status read instruction format

11.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Carefully connect Serial communication/Parallel communication interfacing card to 8086 trainer kit.
3. Switch on the power supply after checking connections
4. Handle the Trainer kit carefully.

11.7 Procedure

WIN862 Software procedure:

1. Open Win862 icon on desktop and opened Window

Figure 11.12: Win862 icon

Figure 11.13: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

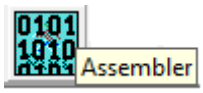


Figure 11.14: Assembler icon

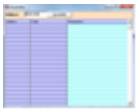


Figure 11.15: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

Figure 11.16: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Connect keyboard interfacing card to the trainer board.
9. Click on Run (check whether program is executed or not)

Program:Parallel communication between two microprocessors using 8255.

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
			MOV AL,90	
			MOV DX,3006	
			OUT DX	
		BACK:	MOV DX,3000	
			IN AL,DX	
			NOT AL	
			MOV DX,3002	
			OUT DX	
			MOV AL,02	
			MOV DX,3006	
			OUT DX	
			CALL DELAY	
			MOV AL,03	
			MOV DX,3006	
			OUT DX	
			CALL DELAY	
			MOV AL,0A	
			MOV DX,3006	
			OUT DX	
			CALL DELAY	
			MOV AL,0B	
			MOV DX,3006	
			OUT DX	
			CALL DELAY	
			MOV AL,0E	
			MOV DX,3006	
			OUT DX	
			CALL DELAY	
			MOV AL,0F	
			MOV DX,3006	
			OUT DX	
			CALL DELAY	
			JMP BACK	

Delay Program

MEMORY LOCATION	OP-CODE	LABEL	MNEMONIC OPERAND	COMMENTS
4500			MOV CX,7FFF	
			LOOP NEXT	
			RET	

11.8 Probing Further Experiments

1. Write a program to initialize 8251 in synchronous mode with even parity, single SYNCH character, 7-bit data character. Then receive FFH bytes of data from a remote terminal and store it in the memory at address 5000H: 2000H.

LAB-11 Interfacing traffic light controller and tone generator

12.1 Introduction

Traffic Light interface module is designed to simulate the function of four way traffic light controller. Combination of Red, Yellow, Green LED's are provided to indicate Halt, Wait, Go. Combination of Red and Green LED's are provided for pedestrian crossing. All LED's are arranged ion the form of an intersection. Junction will be printed on the PCB with printing. At the left corner of each road a group of LED's are arranged in the form of T-section to control traffic on the road. Each road is named as North(N), South(S), East(E), West(W). 24 LED's are controlled through 24 port lines of 8255. This interface allows user to write programs for simulating a variety of traffic situations. Tone generator interface consists of transistor and a speaker. Transistor is driven by a port line and the transistor in turn amplifies the signal and drives the speaker. By controlling ON/OFF periods of the port line through software, user can generate required musical note.

12.2 Objective

12.2.1 Educational

1. Learn about the architecture of 8086 microprocessor.
2. Learn about operating modes in 8255 PPI.
3. Understand the port lines in 8255 PPI.
4. Gain experience on how to interface I/O devices using peripherals.

12.2.2 Experimental

1. To Write an assembly language program to interface Traffic light to 8086 microprocessor using 8255.
2. To Write an assembly language program to interface Tone generator to 8086 microprocessor using 8255.

12.3 Prelab Preparation:

Reading

1. Read and study operating modes and I/O lines of 8255 PPI.
2. Study about traffic light and tone generator modules.

Written

1. Prior coming to the lab complete part0 of the procedure.

12.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8086 microprocessor kit/Win862 with PC		1
2	Keyboard		1
3	Traffic Light Interfacing Kit		1
4	Tone generator Interfacing Kit		1
5	FRC Connector, RS-232 cable		1

12.5 Background

Traffic Light Control:
Traffic light controller interface module is designed to simulate the function of four way traffic light controller.
Combinations of red, amber and green LED's are provided to indicate Halt, Wait and Go signals for vehicles.
Combination of red and green LED's are provided for pedestrian crossing. 36 LED's are arranged in the form of an intersection.
A typical junction is represented on the PCB with comprehensive legend printing. At the left corner of each road, a group of five LED's (red, amber and 3 green) are arranged in the form of a T-section to control the traffic of that road. Each road is named North (N), South(S), East (E) and West (W). LED's L1, L10, L19 and L28 (Red) are for the stop signal for the vehicles on the road N, S, W, and E respectively.
L2, L11, L20 and L29 (Amber) indicates wait state for vehicles on the road N, S, W, and E respectively. L3, L4 and L5 (Green) are for left, strait and right turn for the vehicles on road S. similarly L12-L13-L14, L23-L22-L21 and L32-L31-L30 simulates same function for the roads E, N, W respectively. A total of 16 LED's (2 Red and 2 Green at each road) are provided for pedestrian crossing.
L7-L9.L16-L18, L25-L27 and L34-L36 (Green) when on allows pedestrians to cross and L6-L8, L15-L17, L24-L26 and L33-L35 (Red) when on alarms the pedestrians to wait.
To minimize the hardware pedestrian's indicator LED's (both red and green are connected to same port lines (PC4 to PC7) with red inverted.
Red LED's L10 and L28 are connected to port lines PC2 and PC3 while L1 and L19 are connected to lines PC0 and PC1 after inversion. All other LED's (amber and green) are connected to port A and B.
Working:
8255 is interfaced with 8086 in I/O mapped I/O and all ports are output ports. The basic operation of the interface is explained with the help of the enclosed program. The enclosed program assumes no entry of vehicles from North to West, from road East to South.
At the beginning of the program all red LED's are switch ON, and all other LED's are switched OFF. Amber LED is switched ON before switching over to proceed state from Halt state.
The sequence of traffic followed in the program is given below.
a) From road north to East From road east to north From road south to west From road west to south From road west to north
b) From road north to East From road south to west From road south to north From road south to east From road north to south c)From road south to north Pedestrian crossing at roads west

and east
d) From road east to west From road west to east Pedestrian crossing at roads north and south

Tone Generator:

A tone generator is a signal generator circuit which converts applied electrical signals to audio signals. It can be used to produce dial tones in telephones or produce sirens in ambulances or VIP vehicles etc or to generate melody tunes in toys, door bells etc. It can send electrically generated audio pulses to specific components. It can also be used to test the audio equipment. It basically creates an electric signal and converts it into sound. Different type of tone generators generates different audio signals depending on the application. The source from which the electronic signal is applied also varies with the application.

12.6 Safety Precautions

1. Properly connect the 8086 microprocessor kit with power supply terminals.
2. Carefully connect Traffic light controller/Tone generator interfacing card to 8086 trainer kit.
3. Switch on the power supply after checking connections
4. Handle the Trainer kit carefully.

12.7 Procedure

WIN862 Software procedure:

1. open Win862 icon on desktop and opened Window

Figure 12.1: Win862 icon

Figure 12.2: win862 opened window

2. Click on Assembler and give starting address (0000:4000), then press enter button

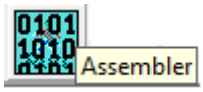


Figure 12.3: Assembler icon

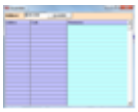


Figure 12.4: Assembler Window

3. Then write 1st Instruction then press enter button.
4. Then write 2nd Instruction then press enter button.
5. Then write up to nth Instruction then press enter button and close the Assembler window.
6. Now click on Dis Assembler and give starting address (Like 0000:4000) then press enter button.

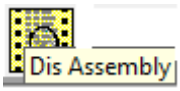


Figure 12.5: Disassembler Window

7. Click on Set PC then give starting address then press Enter button.
8. Connect traffic light/tone generator interfacing card to the trainer board.
9. Click on Run (check whether program is executed or not)

Program:interface traffic light control to 8086 microprocessor

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT
			MOV AL,80	
			MOV DX,0FFE6	
			OUT DX	
		AGAIN	MOV SI,2038	
		NEXTST	MOV AL,[SI]	
			MOV DX,0FFE0	
			OUT DX	
			INC SI	
			ADD DX,02	
			MOV AL,[SI]	
			OUT DX	
			INC SI	
			ADD DX,02	
			MOV AL,[SI]	
			OUT DX	
			INC SI	
			CALL DELAY	
			CMP SI,0056	
			JNZ NEXTST	
			JMP SHORT AGAIN	
		DELAY	MOV CX,0FF	
		DELAY5	PUSH CX	
			MOV CX,03FF	
		DELAY10	NOP	
			LOOP DELAY10	
			POP CX	
			LOOP DELAY5	
			RET	

Program:interface tone generator to 8086 microprocessor

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT
			MOV DX,0FFE6	
			MOV AL,80	
			OUT DX	
		GETKEY	MOV SI,2100	
			CALLS 0B1C,0FF00	
			CMP AL,0F	
			JMP GETKEY	
			MOV BH,00	
			MOV BL,AL	
			MOV CL,4F	
			MOV DX,0FFE4	
		FREQ	MOV AL,00	
			OUT DX	
			MOV CH,[BX][SI]	
		NXTPL	NOP	
			NOP	
			NOP	
			NOP	
			DEC CH	
			JNZ NXTPL	
			MOV AL,0FE	
			OUT DX	
			MOV CH,[BX][SI]	
		NXTPH	NOP	
			NOP	
			NOP	
			NOP	
			DEC CH	
			JNZ NXTPH	
			DEC CL	
			JNZ FREQ	
			JMP SHORT GETKEY	

12.8 Probing Further Experiments

1. To allow vehicles in all the four directions how many sequences need to send for the traffic signals.

2. How to vary the on time for any traffic signal.

3. How to vary the beep duration in tone generator.

LAB-12 Arithmetic And Logical operations using 8051 Microcontroller

13.1 Introduction

The assembly language programs for performing arithmetic and logical operations are composed by using mnemonics, various addressing modes, instructions and registers of microcontroller. The 8051 microcontroller is used to execute the instructoions of assembly language program one by one. The results stored in destination registers are compared against theoretical values obtained. Arithmetic operations includes Addition, Subtraction, Multiplication, Division and logical Operations includes AND, OR, XOR.

13.2 Objective

13.2.1 Educational

1. Learn about the architecture of 8051 microcontroller.
2. Learn about addressing modes and instruction set of 8051 Microcontroller.

13.2.2 Experimental

1. To write an assembly language program to perform arithmetic and logical operations using 8051 microcontroller.

13.3 Prelab Preparation:

Reading

1. Study about features, pin configuration, register organization and operation of 8051 microcontroller.

Written

1. Prior coming to the lab complete part0 of the procedure.

13.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8051 trainer kit		1
2	keyboard		1
3	RPS	+5v	1

13.5 Background

Registers:
Accumulator(A):The accumulator, register A, is used for all arithmetic and logic operations. If the accumulator is not present, then every result of each calculation (addition, multiplication, shift, etc.) is to be stored into the main memory. Access to main memory is slower than access to a register like the accumulator because the technology used for the large main memory is slower (but cheaper) than that used for a register.
"R" Registers:The "R" registers are a set of eight registers, namely, R0, R1 to R7. These registers function as auxiliary or temporary storage registers in many operations. Consider an example of the sum of 10 and 20. Store a variable 10 in an accumulator and another variable 20 in, say, register R4. To process the addition operation, execute the following command
ADD A,R4
After executing this instruction, the accumulator will contain the value 30. Thus "R" registers are very important auxiliary or helper registers. The Accumulator alone would not be very useful if it were not for these "R" registers. The "R" registers are meant for temporarily storage of values.
Let us take another example. We will add the values in R1 and R2 together and then subtract the values of R3 and R4 from the result.
MOV A,R3 ;Move the value of R3 into the accumulator
ADD A,R4 ;Add the value of R4
MOV R5,A ;Store the resulting value temporarily in R5
MOV A,R1 ;Move the value of R1 into the accumulator
ADD A,R2 ;Add the value of R2
SUBB A,R5 ;Subtract the value of R5 (which now contains R3 + R4)
As you can see, we used R5 to temporarily hold the sum of R3 and R4. Of course, this is not the most efficient way to calculate (R1 + R2) – (R3 + R4), but it does illustrate the use of the "R" registers as a way to store values temporarily.

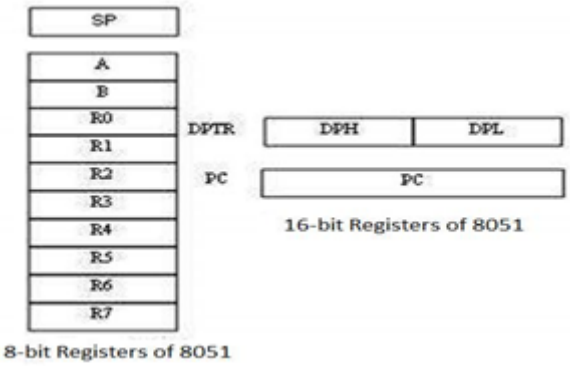


Figure 13.1: Registers of 8051

"B" Register:The "B" register is very similar to the Accumulator in the sense that it may hold an 8-bit (1-byte) value. The "B" register is used only by two 8051 instructions: MUL AB and DIV AB. To quickly and easily multiply or divide A by another number, you may store the other number in "B" and make use of these two instructions. Apart from using MUL and DIV instructions, the "B" register is often used as yet another temporary storage register, much like a ninth R register.
Data Pointer(DPTR):
The Data Pointer (DPTR) is the 8051's only user-accessible 16-bit (2-byte) register. The Accu-

mulator, R0–R7 registers and B register are 1-byte value registers. DPTR is meant for pointing to data. It is used by the 8051 to access external memory using the address indicated by DPTR. DPTR is the only 16-bit register available and is often used to store 2-byte values.

Addressing modes:

Immediate Addressing:In Immediate Addressing mode, the operand, which follows the Opcode, is a constant data of either 8 or 16 bits. The name Immediate Addressing came from the fact that the constant data to be stored in the memory immediately follows the opcode.

The constant value to be stored is specified in the instruction itself rather than taking from a register. The destination register to which the constant data must be copied should be the same size as the operand mentioned in the instruction.

Example: MOV A, #030H

Here, the Accumulator is loaded with 30 (hexadecimal). The # in the operand indicates that it is a data and not the address of a Register. Immediate Addressing is very fast as the data to be loaded is given in the instruction itself.

Register Addressing:In the 8051 Microcontroller Memory Organization Tutorial, we have seen the organization of RAM and four banks of Working Registers with eight Registers in each bank. In Register Addressing mode, one of the eight registers (R0 – R7) is specified as Operand in the Instruction.

It is important to select the appropriate Bank with the help of PSW Register. Let us see a example of Register Addressing assuming that Bank0 is selected.

Example: MOV A, R5

Here, the 8-bit content of the Register R5 of Bank0 is moved to the Accumulator.

Direct Addressing:In Direct Addressing Mode, the address of the data is specified as the Operand in the instruction. Using Direct Addressing Mode, we can access any register or on-chip variable. This includes general purpose RAM, SFRs, I/O Ports, Control registers.

Example: MOV A, 47H

Here, the data in the RAM location 47H is moved to the Accumulator.

Register Indirect Addressing:In the Indirect Addressing Mode or Register Indirect Addressing Mode, the address of the Operand is specified as the content of a Register. This will be clearer with an example.

Example: MOV A, @R1

The @ symbol indicates that the addressing mode is indirect. If the contents of R1 is 56H, for example, then the operand is in the internal RAM location 56H. If the contents of the RAM location 56H is 24H, then 24H is moved into accumulator.

Only R0 and R1 are allowed in Indirect Addressing Mode. These register in the indirect addressing mode are called as Pointer registers.

Register Indirect Addressing:With Indexed Addressing Mode, the effective address of the Operand is the sum of a base register and an offset register. The Base Register can be either Data Pointer (DPTR) or Program Counter (PC) while the Offset register is the Accumulator (A).

In Indexed Addressing Mode, only MOVC and JMP instructions can be used. Indexed Addressing Mode is useful when retrieving data from look-up tables.

Example: MOVC A, @A+DPTR

Instruction Set:

Data transfer InstructionsMOV (Move): Copy the content of source to destination

General format: MOV dest, src

Examples: MOV A,B ; Copy the content of B to A

MOV A,56H ; Copy the content of RAM location 56H to A

MOV @R0,B ; Copy the content of B to the RAM address pointed by R0

MOV P0,A ; Copy the content of A to port 0

MOV R1,#45 ; Copy 45(decimal) to R1

MOV P1,R1 ; Copy the content of R1 to port 1

MOV DPTR,#4567H ; This register is 16 bit and it should contain a 16 bit address
Arithmetic instructions
ADD (Addition): This instruction adds the source and A, and puts the sum in A. The CY, OV, AC flags are affected.
General format: ADD A, src
Example:
ADD A,32H ; add the content of RAM address 32H to A(sum in A)
ADD A,@R1 ; add the content of RAM address pointed by R1(sum in A)
ADD A,R2 ; add the content of R2 to A(sum in A)
ADD A, #67 ; add 67 (decimal) to A(sum in A)
ADDC (Add with carry):In this instruction source, carry flag and A are added and the sum is put in A. The CY, OV, AC flags are affected.
General format: ADDC A, src
Example:
ADDC A, #76H ; add 76H and CY to A (sum in A)
ADDC A,56H ; add the content of 56H and CY and A (sum in A)
ADDC A, R4 ;add content of R4 and CY and A (sum in A)
INC:This instruction adds 1 to the destination. Destination can be any register or memory location.
None of the flags are affected
General format: INC dest
Example:
INC R5 ; add 1 to the number in R5
INC @R0 ; add 1 to the number in the address pointed by R0
INC A ; add 1 to the number in A
INC 43H ; add 1 to the content in address 43H
SUBB (subtract with borrow)
This instruction subtracts the source byte and the carry flag from A and puts the result in A.
The OV, CY and AC flags are affected
For normal subtraction instruction we have to clear the carry and do the operation.
General format: SUBB A, src
Example:
MOV A,#78H ; A=78H
CLR C ; C=0
SUBB A,#23H ; A= A-23H
DEC (decrement)This instruction subtracts 1 from the destination, which can be any register or RAM location.
No flags are affected
General format: DEC dest
Example:
DEC R3 ; subtract 1 from the number in R3
DEC @ R1 ; subtract 1 from the number pointed by R1
MUL (multiplication):This instruction multiplies two unsigned numbers.one is to be in A and the other is to be in B.
The product can be two bytes long (maximum), and it will have its lower byte in A and its upper byte in B.
OV, CY flags are affected
The multiply instruction clears the carry flag and sets the OV flag if the product is greater than FFH.
General format: MUL AB
Example:

MOV A,#89H ; A=89H
MOV B,#97H ; B=97H
MUL AB ; The Product Is 50CFH with A=CFH, B=50H, and OV=1
DIV (Division):This instruction divides the content of A by the content of B. The quotient will be in A and the remainder will be in B
OV, CY flags are affected
General format: DIV AB
Example:
MOV A,#245H ; A=245H
sa ‘ MOV B,#17H ; B=17H
DIV AB ; A=14(quotient), B=7(remainder), CY=0 and OV=0
Logical instructions
ANL (logical AND):This instruction performs the logical AND operation on the source and destination operands and stores the result in the destination variable No flags are affected
General format: ANL dest, source
Example:
ANL A,R2
If ACC=D3H (11010011) and R2=75H (01110101), the result of the instruction is ACC=51H (01010001)
ORL (logical OR):This instruction performs the logical OR operation on the source and destination operands and stores the result in the destination variable No flags are affected
General format: ORL dest, source
Example:
ORL A,R2
If ACC=D3H (11010011) and R2=75H (01110101), the result of the instruction is ACC=F7H (11110111)
XRL (logical XOR)
This instruction performs the logical XOR (Exclusive OR) operation on the source and destination operands and stores the result in the destination variable
No flags are affected
General format: XRL dest, source
Example:
XRL A,R0
If ACC=C3H (11000011) and R0=AAH (10101010), then the instruction results in ACC=69H (01101001)
Subroutine Instructions
LCALL
LONG CALL:LCALL calls a program subroutine. LCALL increments the program counter by 3 and pushes that value onto the stack (low byte first, high byte second). The Program Counter is then set to the 16-bit value which follows the LCALL opcode, causing program execution to continue at that address.
General Format: LCALL SUBROUNAME
Example: LCALL DELAY

13.6 Safety Precautions

1. Properly connect the 8051 microcontroller kit with power supply terminals.
2. Switch on the power supply after checking connections
3. Handle the Trainer kit carefully.

13.7 Procedure

Programs:Arithmetic and logical Operations using 8051 microcontroller:
Arithmetic Operations using 8051 microcontroller :
Addition :

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT

			MOV A,#02
			MOV B,#02
			ADD A,B
			LCALL 03	

Observation Table :

Input		output	
REGISTER	Data	REGISTER	Data
A	02	A	04
B	02		

Subtraction :

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT

8000			MOV A,#04
			MOV B,#02
			SUBB A,B
			LCALL 03	

Observation Table :

Input		output	
REGISTER	Data	REGISTER	Data
A	04	A	02
B	02		

Multiplication :

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT
			MOV DPTR,#9000	
			MOVX A,@DPTR	
			MOV F0,A	
			INC DPTR	
			MOVX A,@DPTR	
			MUL AB	
			LCALL 03	

Observation Table :

Input		output	
MEMORY LOCATION	**Data**	**REGISTER**	**Data**
9000	03	A	06
9001	02		

Division :

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT
			MOV DPTR,#9000	
			MOVX A,@DPTR	
			MOV R0,A	
			INC DPTR	
			MOVX A,@DPTR	
			MOV F0,A	
			MOV A,R0	
			DIV AB	
			INC DPTR	
			MOV @DPTR,A	
			LCALL 03	

Observation Table :

Input		Output	
MEMORY LOCATION	**Data**	**REGISTER**	**Data**
9000	03	A	06
9001	02		

Logical Operations using 8051 microcontroller :
AND:

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT
			MOV R0,#15	
			MOV A,#23	
			ANL A,R0	
			MOV R1,A	
			LCALL 03	

Observation Table :

Input		Output	
Register	**Data**	**Register**	**Data**
R0		R1	
A			

OR:

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT
			MOV R0,#15	
			MOV A,#23	
			ORL A,R0	
			MOV R1,A	
			LCALL 03	

Observation Table :

Input		Output	
Register	**Data**	**Register**	**Data**
R0		R1	
A			

XOR:

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERAND	COMMENT
			MOV R0,#15	
			MOV A,#23	
			XRL A,R0	
			MOV R1,A	
			LCALL 03	

Observation Table :

Input		Output	
Register	**Data**	**Register**	**Data**
R0		R1	
A			

13.8 Further Probing Experiments

1. What will be the status of carry flag (CY) and Auxiliary carry (AC) flags after execution of these instructions
   MOV A, #97H

ADD A, #80H

2. What will be the contents of accumulator after executing the instructions
   MOV A, #0FH
   ANL A, #2CH

3. If we use ADC instruction instead of ADD instruction in this addition program what difference can be observed in the output.

4. 4. In this program we performed 8-bit addition operation, how to write program to perform 16-bit operation.

LAB-13 Timer/Counter

14.1 Introduction

A timer is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a stopwatch. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer.
A counter is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal. It is used to count the events happening outside the microcontroller. In electronics, counters can be implemented quite easily using register-type circuits such as a flip-flop.

14.2 Objective

14.2.1 Educational

1. Learn about TMOD and TCON register in 8051 microcontroller.
2. Learn about different types of timers in 8051 Microcontroller.
3. Identify the difference between timers and counters.

14.2.2 Experimental

1. To verify Timer/Counter operations in timer0 and timer1 modes using 8051 microcontroller

14.3 Prelab Preparation:

Reading

1. Study about different types of timers and their modes, TCON register, TMOD register, Counters of 8051 microcontroller.

Written

1. Prior coming to the lab complete part0 of the procedure.

14.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8051 trainer kit with keyboard		1
2	Timer/Counter Interface Module		1
3	RPS	+5v	1
4	RS – 232		1
5	FRC cables		1

14.5 Background

8051 Timers/Counters:
Timers/Counters of the 8051 micro controller. The 8051 has two counters/timers which can be used either as timer to generate a time delay or as counter to count events happening outside the microcontroller. Many of the microcontroller applications require counting of external events such as frequency of the pulse trains and generation of precise internal time delays between computer actions. Both these tasks can be implemented by software techniques, but software loops for counting, and timing will not give the exact result rather more important functions are not done. To avoid these problems, timers and counters in the micro-controllers are better options for simple and low-cost applications. These timers and counters are used as interrupts in 8051 microcontroller.
There are two 16-bit timers and counters in 8051 microcontroller: timer 0 and timer 1. Both timers consist of 16-bit register in which the lower byte is stored in TL and the higher byte is stored in TH. Timer can be used as a counter as well as for timing operation that depends on the source of clock pulses to counters A timer is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a stopwatch. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer. A counter is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal. It is used to count the events happening outside the microcontroller. In electronics, counters can be implemented quite easily using register-type circuits such as a flip-flop.
Clock :
Every Timer needs a clock to work, and 8051 provides it from an external crystal which is the main clock source for Timer. The internal circuitry in the 8051 microcontrollers provides a clock source to the timers which is 1/12th of the frequency of crystal attached to the microcontroller, also called Machine cycle frequency.

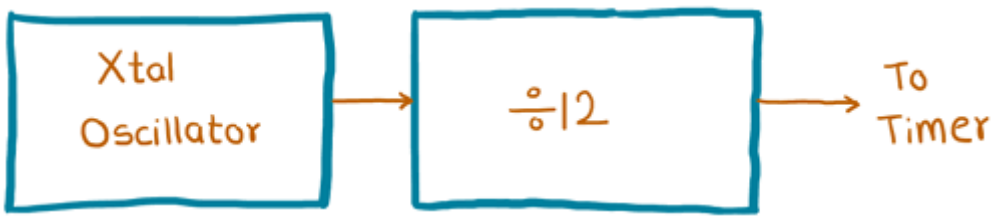


Figure 14.1: 8051 Timer Clock

For example, suppose we have a crystal frequency of 11.0592 MHz then the microcontroller will provide 1/12th i.e.
Timer clock frequency= (Xtal Osc.frequency)/12 = (11.0592 MHz)/12 = 921.6 KHz
period T= 1/(921.6 kHz)=1.085 micro seconds

Timer:

8051 has two timers Timer0 (T0) and Timer1 (T1), both are 16-bit wide. Since 8051 has 8-bit architecture, each of these is accessed by two separate 8-bit registers as shown in the figure below. These registers are used to load timer count.

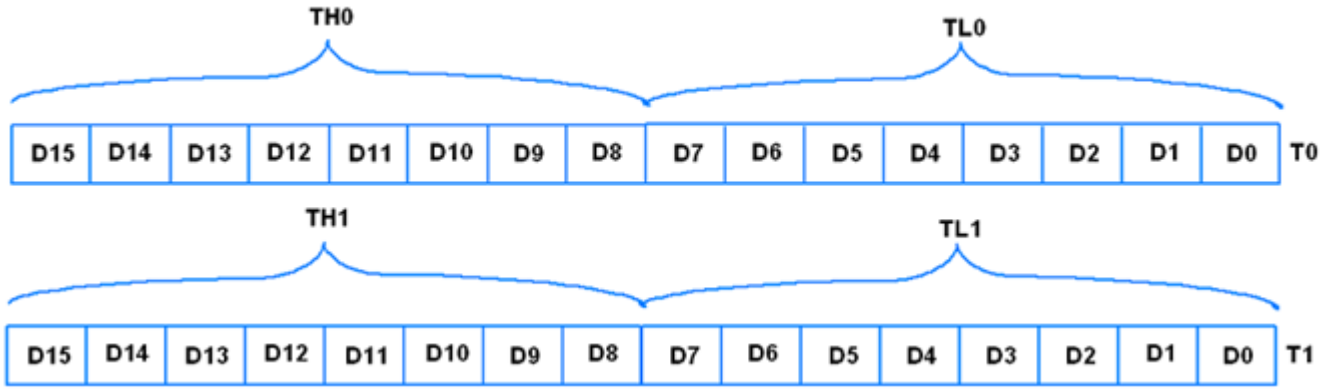


Figure 14.2: Timers of 8051

8051 has a Timer Mode Register and Timer Control Register for selecting a mode of operation and controlling purpose.

TMOD register:

TMOD is an 8-bit register used to set timer mode of timer0 and timer1.

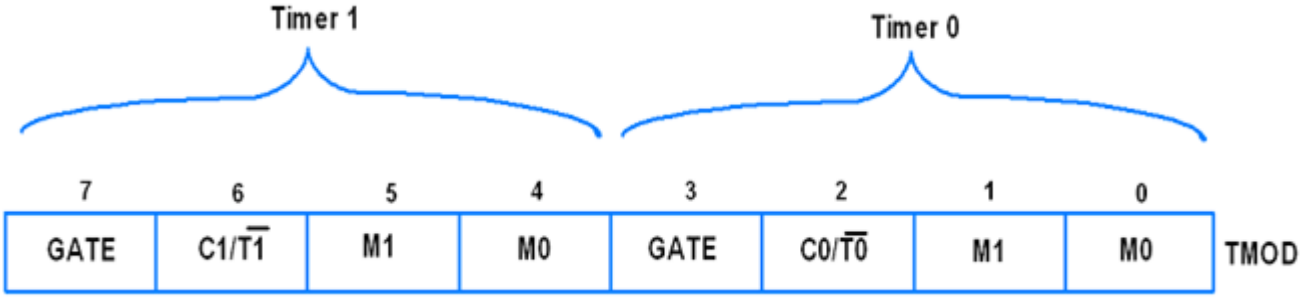


Figure 14.3: TMOD Register

Its lower 4 bits are used for Timer0 and the upper 4 bits are used for Timer1

Bit 7,3 – GATE:

1 = Enable Timer/Counter only when the INT0/INT1 pin is high and TR0/TR1 is set.
0 = Enable Timer/Counter when TR0/TR1 is set.

Bit 6,2 - C/T (Counter/Timer):

Timer or Counter select bit
1 = Use as Counter
0 = Use as Timer

Bit 5:4 and 1:0 - M1:M0:

Timer/Counter mode select bit These are Timer/Counter mode select bit as per the below table

M1	M0	Mode	Operation
0	0	0 (13-bit timer mode)	13-bit timer/counter, 8-bit of THx & 5-bit of TLx
0	1	1 (16-bit timer mode)	16-bit timer/counter, THx cascaded with TLx
1	0	2 (8-bit auto-reload mode)	8-bit timer/counter (auto-reload mode), TLx reload with the value held by THx each time TLx overflow
1	1	3 (split timer mode)	Split the 16-bit timer into two 8-bit timers i.e. THx and TLx like two 8-bit timer

TCON Register:

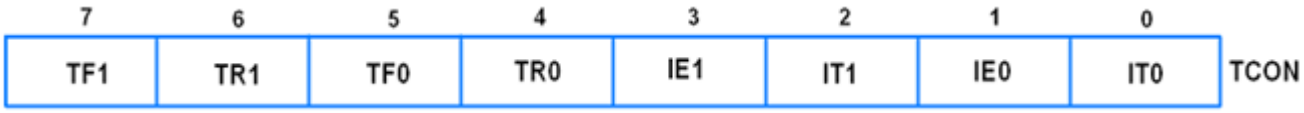


Figure 14.4: TCON Register

TCON is an 8-bit control register and contains a timer and interrupt flags.

Bit 7 - TF1:
Timer1 Overflow Flag
1 = Timer1 overflow occurred (i.e. Timer1 goes to its max and roll over back to zero).
0 = Timer1 overflow not occurred.
It is cleared through software. In the Timer1 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

Bit 6 - TR1:
Timer1 Run Control Bit
1 = Timer1 start.
0 = Timer1 stop.
It is set and cleared by software.

Bit 5 – TF0:
Timer0 Overflow Flag
1 = Timer0 overflow occurred (i.e. Timer0 goes to its max and roll over back to zero).
0 = Timer0 overflow not occurred.
It is cleared through software. In the Timer0 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

Bit 4 – TR0:
Timer0 Run Control Bit
1 = Timer0 start.
0 = Timer0 stop.
It is set and cleared by software.

Bit 3 - IE1:
External Interrupt1 Edge Flag
1 = External interrupt1 occurred.
0 = External interrupt1 Processed.

It is set and cleared by hardware.
Bit 2 - IT1:
External Interrupt1 Trigger Type Select Bit
1 = Interrupt occurs on falling edge at INT1 pin.
0 = Interrupt occur on a low level at the INT1 pin.
Bit 1 – IE0:
External Interrupt0 Edge Flag
1 = External interrupt0 occurred.
0 = External interrupt0 Processed.
It is set and cleared by hardware.
Bit 0 – IT0:
External Interrupt0 Trigger Type Select Bit
1 = Interrupt occurs on falling edge at INT0 pin.
0 = Interrupt occur on a low level at INT0 pin.
Timer Modes:
Mode 0 (13-Bit Timer Mode):
Both Timer 1 and Timer 0 in Mode 0 operate as 8-bit counters (with a divide-by-32 prescaler). Timer register is configured as a 13-bit register consisting of all the 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the register. The timer interrupt flag TF1 is set when the count rolls over from all 1s to all 0s. Mode 0 operation is the same for Timer 0 as it is for Timer 1.
Mode 1 (16-Bit Timer Mode):
Timer mode "1" is a 16-bit timer and is a commonly used mode. It functions in the same way as 13-bit mode except that all 16 bits are used. TLx is incremented starting from 0 to a maximum 255. Once the value 255 is reached, TLx resets to 0 and then THx is incremented by 1. As being a full 16-bit timer, the timer may contain up to 65536 distinct values and it will overflow back to 0 after 65,536 machine cycles.
Mode 2 (8 Bit Auto Reload):
Both the timer registers are configured as 8-bit counters (TL1 and TL0) with automatic reload. Overflow from TL1 (TL0) sets TF1 (TF0) and also reloads TL1 (TL0) with the contents of Th1 (TH0), which is preset by software. The reload leaves TH1 (TH0) unchanged.
The benefit of auto-reload mode is that you can have the timer to always contain a value from 200 to 255. If you use mode 0 or 1, you would have to check in the code to see the overflow and, in that case, reset the timer to 200. In this case, precious instructions check the value and/or get reloaded. In mode 2, the microcontroller takes care of this. Once you have configured a timer in mode 2, you don't have to worry about checking to see if the timer has overflowed, nor do you have to worry about resetting the value because the microcontroller hardware will do it all for you. The auto-reload mode is used for establishing a common baud rate.
Mode 3 (Split Timer Mode):
Timer mode "3" is known as split-timer mode. When Timer 0 is placed in mode 3, it becomes two separate 8-bit timers. Timer 0 is TL0 and Timer 1 is TH0. Both the timers count from 0 to 255 and in case of overflow, reset back to 0. All the bits that are of Timer 1 will now be tied to TH0.
When Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be set in modes 0, 1 or 2, but it cannot be started/stopped as the bits that do that are now linked to TH0. The real timer 1 will be incremented with every machine cycle.

14.6 Safety Precautions

1. Properly connect the 8051 microcontroller kit with power supply terminals.
2. Switch on the power supply after checking connections

3. Handle the Trainer kit carefully.

14.7 Procedure

Programs:To Verify Timer '0' and Timer '1' in Counter Mode:

To Verify Timer '0'- Counter Mode :

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERANDS	COMMENT
			MOV A, #89 (TMOD=89)	
			ORL A,#05H	
			MOV TMOD,A	
			SETB 8C (TRO=8C)	
			LCALL 68EAH	
		LOOP:	MOV DPTR,#0194H	
			MOV A,8A (TLO=8A)	
			MOVX @DPTR,A	
			INC DPTR	
			MOV A,8C (THO=8C)	
			MOVX @DPTR,A	
			LCALL 6748H	
			SJMP LOOP	

To Verify Timer '1'- Counter Mode :

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERANDS	COMMENT
			MOV A, #89 (TMOD=89)	
			ORL A,#50H	
			MOV TMOD,A	
			SETB 8E (TR1=8E)	
			LCALL 68EAH	
		LOOP:	MOV DPTR,#0194H	
			MOV A,8B (TL1=8B)	
			MOVX @DPTR,A	
			INC DPTR	
			MOV A,8D (TH1=8D)	
			MOVX @DPTR,A	
			LCALL 6748H	
			SJMP LOOP	

14.8 Further Probing Experiments

1. In this program the address of TMOD register is 89, If timer 1 is operated as an interval timer in mode 2 and timer 0 is operated as an interval timer in mode 0, start/stop op-

erations of both timers are controlled by software, what will the address of TMOD register.

2. In the instruction "MOV TH1,#-3", what is the value that is being loaded in the TH1 register?

3. What is the frequency of the clock that is being used as the clock source for the timer.

4. What is the maximum delay that can be generated with the crystal frequency of 22MHz?

5. Which special function register play a vital role in the timer/counter mode selection process by allocating the bits in it?

LAB-14 Interfacing Keyboard to 8051 Microcontroller

15.1 Introduction

The key board here we are interfacing is a matrix keyboard. The advantage of matrix keypad is that it will allow the programmer to reduce the number of pins to be used. In a 4×4 matrix keypad, there are four rows and four columns connected to 16 push button switches. It may look like one needs 16 pins for the microcontroller to be connected to the matrix keypad but practically 16 inputs of keypad interface are possible with the 8 pins of a microcontroller port. All 8 lines can be connected to the same port or different ports based on the application requirements. In fact, 8 port pins of a microcontroller are sufficient for a 4×4 keypad interface using row and column matrix connection technique by saving other 8 bits of the port. Whenever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a key is pressed only a bit in the port goes high. Which indicates microcontroller that the key is pressed. By this high on the bit key in the corresponding column is identified. Once we are sure that one of key in the key board is pressed next our aim is to identify that key. To do this we firstly check for particular row and then we check the corresponding column the key board.

15.2 Objective

15.2.1 Educational

1. Learn about the working principle of HEX Keypad.
2. Learn about the architecture of 8051 Microcontroller.
3. Learn about the instruction set and addressing modes of 8051 Microcontroller.

15.2.2 Experimental

1. To Write an assembly language program to interface keyboard to 8051 Microcontroller.

15.3 Prelab Preparation:

Reading

1. Study about working principle of matrix keyboard, architecture, addressing modes, instruction set of 8051 microcontroller.

Written

1. Prior coming to the lab complete part0 of the procedure.

15.4 Equipment needed

S.No	Device	Range / Rating	Quantity (in No's)
1	8051 trainer kit with keyboard		1
2	Key board module		1
3	RPS	+5v	1
4	FRC cables		1
5	RS-232 cable		1

15.5 Background

keypad is used as a input with 8051 microcontroller. Matrix Keypads are mostly used in calculators, mobile phones, telephones, ATM etc. It is used when a number of input switches are required. In this article we will study how to interface keypad with 8051 microcontroller. An experiment will show the keypad interfacing. User will give input through keypad and then that input will be displayed on LCD. **Keypad structure:**
In a keypad, push button switches are arranged in rows and columns. For a 4×4 keypad 16 switches are used and to connect to microcontroller we need 16 inputs pins. But the arrangement is changed by connecting switches in a special way. Now we need only 8 pins of microcontroller to connect keypad to it.

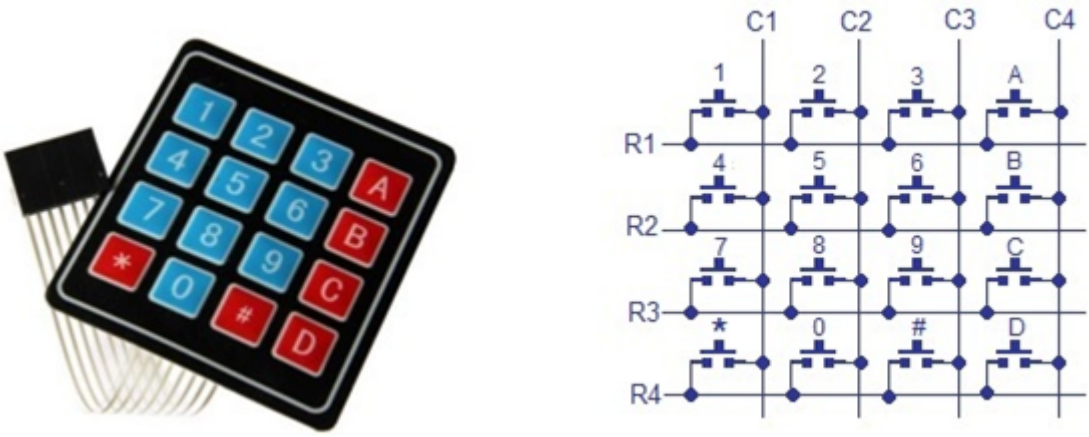


Figure 15.1: 4X4 Keypad

The status of each key/switch is determined by Scanning the rows or columns. The column pins (Col 1–Col4) are connected to the microcontroller as the inputs pinsand the rows pins (Row 1–Row 4) are connected to the output pins of the microcontroller. Normally, all the column pins are pulled high by internal or external pull up resistors. Now we can read the status of each switch through scanning.
Reading Data:
Scanning is done in a different way. Columns pins are used as input pins, and rows pins are used as output. If a low logic is given to all the Rows and high logic is given to each Column.
For finding Column number:
•When a switch/key is pressed, the corresponding row and column will get short.
•Output of the corresponding column goes to go low.
•Since we have made all the rows zero so this gives the column number of the pressed key.

For Finding Row number:

•After the detection of column number, the controller set's all the rows to high.

•Each row is one by one set to zero by the microcontroller and the earlier detected column is checked and obviously it becomes zero.

•The row due to which the column gets zero is the row number of the pressed key.

•keypad interfacing 8051 microcontroller

For the interfacing of keypad with the microcontroller, it is good to connect LCD also, so that we can observe specific changes if the keypad is pressed. Keypad 4×4 (having 4 rows and 4columns) is connected to Port 2 of 8051 microcontroller to scan input. LCD is connected to Port 1 of the microcontroller for displaying output. Port3 pin0 and pin2 of microcontroller is connected to RS and EN pins of LCD respectively.

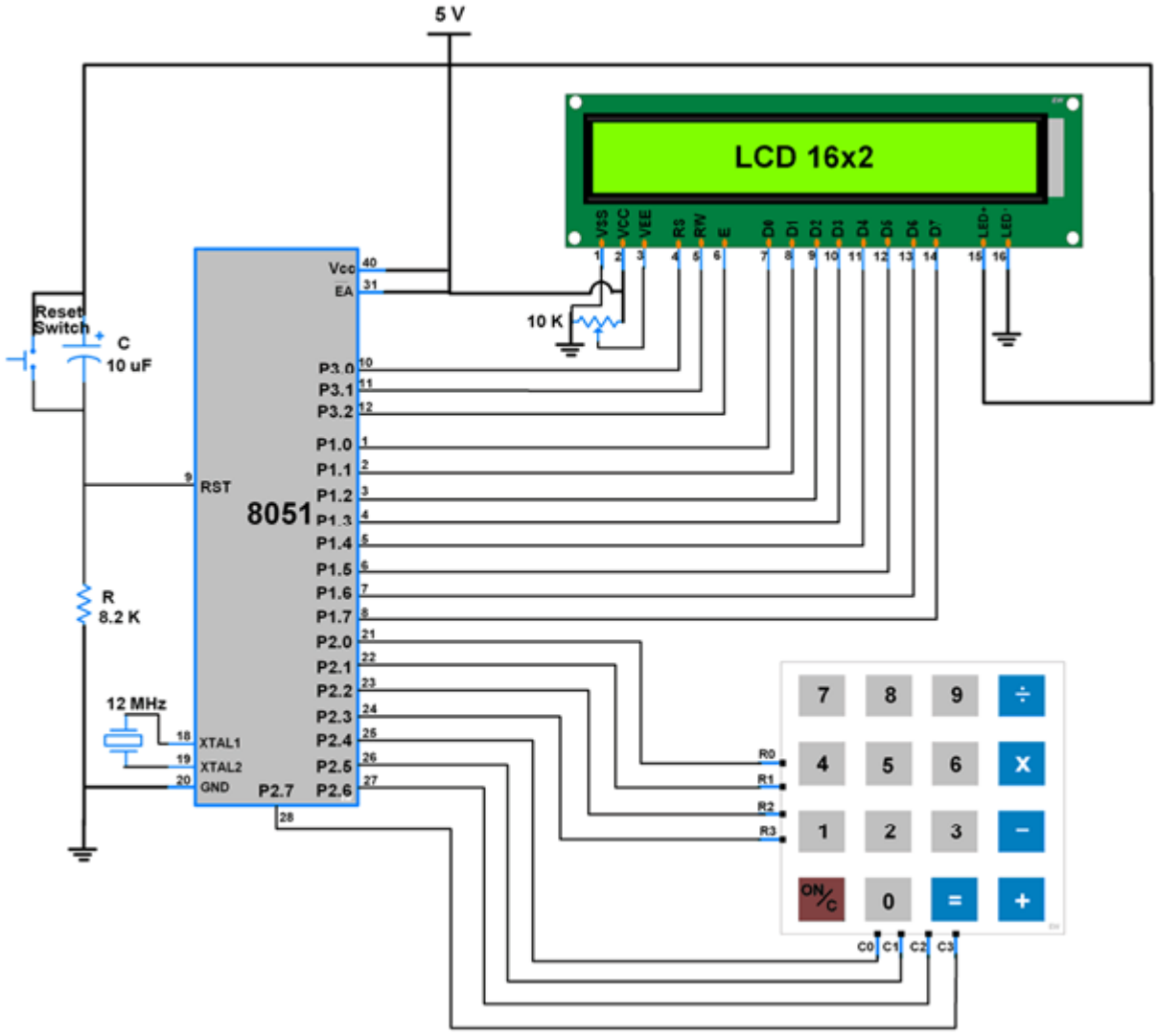


Figure 15.2: 4X4 Keypad interfacing with 8051

15.6 Safety Precautions

1. Properly connect the 8051 microcontroller kit with power supply terminals.
2. Switch on the power supply after checking connections

3. Handle the Trainer kit carefully.

15.7 Procedure

Programs:Interfacing keyboard to 8051 microcontroller:

MEMORY LOCATION	OPCODE	LABEL	MNEMONIC OPERANDS	COMMENT
			MOV A,#90H	
			MOV DPTR,#CNTRL	
			MOVX @DPTR,A	
			MOV B,#20H	
		BLINK2:	MOV DPTR,#PORTB	
			MOV A,#FFH	
			MOVX @DPTR,A	
			MOV DPTR,#PORTC	
			MOV A,#00H	
			MOVX @DPTR,A	
			MOV A,#F0H	
			MOVX @DPTR,A	
			DJNZ B,BLNK2	
			MOV A,#FEH	
		BACK:	MOV B,#21H	
			MOV DPTR,#PORTB	
		BLINK1:	MOVX @DPTR,A	
			MOV DPTR,#PORTC	
			MOV A,#00H	
			MOVX @DPTR,A	
			MOV A,#F0H	
			MOVX @DPTR,A	
			LCALL DELAY	
			RL A	
			DJNZ B,BLNK1	
			SJMP BACK	
			MOV R0,#F7H	
			MOV R1,#FFH	
		DELAY:	DJNZ R1,ILOOP	
		OLOOP:	DJNZ R0,OLOOP	
		ILOOP:	RET	

15.8 Further Probing Experiments

1. If we need to operate a key of a keyboard in an interrupt mode, then it will generate what kind of interrupt?

2. What is described by this command: CJNE A,#00001111b, ROW1

3. How to detect that in which column, the key is placed?

4. When reading the columns of a matrix, if no key is pressed what should we get all in binary notation

Appendix A :Instruction set of 8086 Microprocessor

Table A.1 *Data Copy/Transfer Instructions*

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags*								
				O	*D*	*I*	*T*	*S*	*Z*	*A*	*P*	*C*
MOV	Move			–	–	–	–	–	–	–	–	–
	Accumulator to memory	10	3									
	Memory to accumulator	10	3									
	Register to register	2	2									
	Memory to register	8 + AM	2–4									
	Register to memory	9 + AM	2–4									
	Immediate to register	4	2–3									
	Immediate to memory	10 + AM	3–6									
	Register to SS, DS or ES	2	2									
	Memory to SS, DS or ES	8 + AM	2–4									
	Segment register to register	2	2									
	Segment register to memory	9 + AM	2–4									
PUSH	Push word onto stack			–	–	–	–	–	–	–	–	–
	Register	11	1									
	Segment register	10	1									
	Memory	16 + AM	2–4									
POP	Pop word off stack			–	–	–	–	–	–	–	–	–
	Register	8	1									
	Segment register SS, DS, or ES	8	1									
	Memory	17 + AM	2–4									
XCHG	Exchange			–	–	–	–	–	–	–	–	–
	Register with accumulator	3	1									
	Register with memory	17 + AM	2–4									
	Register with register	4	2									
IN	Input from I/O port			–	–	–	–	–	–	–	–	–
	Fixed port	10	2									
	Variable port	8	1									
OUT	Output to I/O port			–	–	–	–	–	–	–	–	–
	Fixed port	10	2									
	Variable port	8	1									
XLAT/ XLATB	Translate	11	1									
LEA	Load effective address	2 + AM	2–4	–	–	–	–	–	–	–	–	–
LDS/LES	Load pointer using DS/ES	16 + AM	2–4	–	–	–	–	–	–	–	–	–
LAHF	Load AH from flags	4	1	–	–	–	–	–	–	–	–	–
SAHF	Store AH in to flags	4	1	–	–	–	–	r	r	r	r	r
PUSHF	Push flags onto stack	10	1	–	–	–	–	–	–	–	–	–
POPF	Pop flags from stack	8	1	r	r	r	r	r	r	r	r	r

Table A.2 *Arithmetic Instructions*

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags*								
				O	*D*	*I*	*T*	*S*	*Z*	*A*	*P*	*C*
ADD	Addition			m	–	–	–	m	m	m		m	m
	Register to register	3	2										
	Memory to register	9 + AM	2–4										
	Register to memory	16 + AM	2–4										
	Immediate to register	4	3–4										
	Immediate to memory	17 + AM	3–6										
	Immediate to accumulator	4	2–3										
ADC	Add with carry			m	–	–	–	m	m	m		m	m
	Register to register	3	2										
	Memory to register	9 + AM	2–4										
	Register to memory	16 + AM	2–4										
	Immediate to register	4	3–4										
	Immediate to memory	17 + AM	3–6										
	Immediate to accumulator	4	2–3										
INC	Increment by 1			m	–	–	–	m	m	m		m	–
	16-bit register	2	1										
	8-bit register	3	2										
	Memory	15 + AM	2–4										
DEC	Decrement by 1			m	–	–	–	m	m	m		m	–
	16-bit register	2	1										
	8-bit register	3	2										
	Memory	15 + AM	2–4										
SUB	Subtraction			m	–	–	–	m	m	m		m	m
	Register from register	3	2										
	Memory from register	9 + AM	2–4										
	Register from memory	16 + AM	2–4										
SUB	Immediate from accumulator	4	2–3										
	Immediate from register	4	3–4										
	Immediate from memory	17 + AM	3–6										
SBB	Subtract with borrow			m	–	–	–	m	m	m		m	m
	Register from register	3	2										
	Memory from register	9 + AM	2–4										
	Register from memory	16 + AM	2–4										
	Immediate from accumulator	4	2–3										
	Immediate from register	4	3–4										
	Immediate from memory	17 + AM	3–6										

Figure A.2: Table A.2

Table A.2 (*Contd.*)

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags*								
				O	*D*	*I*	*T*	*S*	*Z*	*A*	*P*	*C*
CMP	Compare			m	–	–	–	m	m	m	m	m
	Register to register	3	2									
	Memory to register	9 + AM	2–4									
	Register to memory	16 + AM	2–4									
	Immediate to register	4	3–4									
	Immediate to memory	4	3–6									
	Immediate to accumulator	17 + AM	2–3									
AAA	ASCII adjust after addition	4	1	d	–	–	–	d		d	m	d m
AAS	ASCII adjust after subtraction	4	1	d	–	–	–	d	d	m	d	m
AAM	ASCII adjust after multiplication	83	2	d	–	–	–	m	m	d	m	d
AAD	ASCII adjust after division	60	2	d	–	–	–	m	m	d	m	d
DAA	Decimal adjust accumulator	4	1	d	–	–	–	m	m	m	m	m
DAS	Decimal adjust subtraction	4	1	d	–	–	–	m	m	m	m	m
NEG	Negate			m	–	–	–	m	m	m	m	m
	Register	3	2									
	Memory	16 + AM	2–4									
MUL	Unsigned multiplication			m	–	–	–	d	d	d	d	m
	8-bit register	70–77	2									
	16-bit register	118–133	2									
	8-bit memory	(76–83) + AM	2–4									
	16-bit memory	(124–139) + AM	2–4									
IMUL	Integer multiplication			m	–	–	–	d	d	d	d	m
	8-bit register	80–98	2									
	16-bit register	128–154	2									
IMUL	8-bit memory	(86–104) + AM	2–4									
	16-bit memory	(134–160) + AM	2–4									
CBW	Convert byte to word	2	1	–	–	–	–	–	–	–	–	–
CWD	Convert word to double word	5	1	–	–	–	–	–	–	–	–	–
DIV	Unsigned division			d	–	–	–	d	d	d	d	d
	8-bit register	80–90	2									
	16-bit register	144–162	2									
	8-bit memory	(86–96) + AM	2–4									
	16-bit memory	(150–168) + AM	2–4									
IDIV	Integer division			d	–	–	–	d	d	d	d	d
	8-bit register	101–112	2									
	16-bit register	165–184	2									
	8-bit memory	(107–118) + AM	2–4									
	16-bit memory	(171–190) + AM	2–4									

Figure A.3: Table A.2(contd)

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags*								
				O	*D*	*I*	*T*	*S*	*Z*	*A*	*P*	*C*
AND	Logical AND			0	–	–	–	m	m	d	m	0
	Register to register	3	2									
	Memory to register	9 + AM	2–4									
	Register to memory	16 + AM	2–4									
	Immediate to register	4	3–4									
	Immediate to memory	17 + AM	3–6									
	Immediate to accumulator	4	2–3									
OR	Logical OR			0	–	–	–	m	m	d	m	0
	Register to register	3	2									
	Memory to register	9 + AM	2–4									
	Register to memory	16 + AM	2–4									
	Immediate to accumulator	4	2–3									
	Immediate to register	4	3–4									
	Immediate to memory	17 + AM	3–6									
NOT	Logical NOT			–	–	–	–	–	–	–	–	–
	Register	3	2									
	Memory	16 + AM	2–4									
XOR	Logical exclusive OR			0	–	–	–	m	m	d	m	0
	Register with register	3	2									
	Memory with register	9 + AM	2–4									
	Register with memory	16 + AM	2–4									
	Immediate with accumulator	4	2–3									
	Immediate with register	4	3–4									
	Immediate with memory	17 + AM	3–6									
TEST	TEST			0	–	–	–	m	m	d	m	0
	Register with register	3	2									
	Memory with register	9 + AM	2–4									
	Immediate with accumulator	4	2–3									
	Immediate with register	5	3–4									
	Immediate with memory	11 + AM	3–6									

Figure A.4: Table A.3 Logical Instructions

Table A.4 *Branching Instructions*

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags*								
				O	*D*	*I*	*T*	*S*	*Z*	*A*	*P*	*C*
CALL	Call a procedure			–	–	–	–	–		–	–	–	–
	Intrasegment direct	19	3									
	Intrasegment indirect through register	16	2									
	Intrasegment indirect through memory	21 + AM	2–4									
	Intrasegment direct	28	5									
	Intrasegment indirect	37 + AM	2–4									
RET	Return from a procedure			–	–	–	–	–		–	–	–	–
	Intrasegment	8	1									
	Intrasegment with constant	12	3									
	Intersegment	18	1									
	Intersegment with constant	17	3									
INT N	Interrupt			–	–	0	0	–	–	–	–	–	
	Type = 3	52	1									
	Type π 3	51	2									
INTO	Interrupt if overflow			–	–	0	0	–		–	–	–	–
	Interrupt if taken	53										
	Interrupt if not taken	4										
JMP	Jump			–	–	–	–	–		–	–	–	–
	Intrasegment direct short	15	2									
	Intrasegment direct	15	3									
	Intersegment direct	15	5									
	Intrasegment indirect through memory	18 + AM	2–4									
	Intrasegment indirect through register	11	2									
	Intersegment indirect	24 + AM	2–4									
IRET	Return from Interrupt	24	1	r	r	r	r	r		r	r	r	r
JZ/JE	Jump if not zero/ Jump if not equal	16/4	2	–	–	–	–	–		–	–	–	–
JNX/ JNE	Jumps if not zero/ Jumps if not equal	16.4	2	–	–	–	–	–		–	–	–	–
JS	Jump if sign	16/4	2	–	–	–	–	–		–	–	–	–
JNS	Jump if not sign	16/4	2	–	–	–	–	–		–	–	–	–
JO	Jump of overflow	16/4	2	–	–	–	–	–		–	–	–	–
JNO	Jump if not overflow	16/4	2	–	–	–	–	–		–	–	–	–

Figure A.5: Table A.4 Branching Instructions

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags* O	D	I	T	S	Z	A	P	C
JP/JPE	Jump if parity/ Jump if parity even	16/4	2	–	–	–	–	–	–	–	–	–
JNP/JPO	Jump if not parity/Jump if parity odd	16/4	2	–	–	–	–	–	–	–	–	–
JB/ JNAE/JC	Jump if below/Jump if not above or equal/Jump if carry	16/4	2	–	–	–	–	–	–	–	–	–
JNB/JAE/JNC	Jump if not below/Jump if above or equal/Jump if not carry	16/4	2	–	–	–	–	–	–	–	–	–
JBE/JNA	Jump if below or equal/ Jump if not above	16/4	2	–	–	–	–	–	–	–	–	–
JNBE/JA	Jump if not below or equal/ Jump if above	16/4	2	–	–	–	–	–	–	–	–	–
JL/ JNGE	Jump if less/ Jump if not greater or equal	16/4	2	–	–	–	–	–	–	–	–	–
JNL/JGE	Jump if not less/ Jump if greater or equal	16/4	2	–	–	–	–	–	–	–	–	–
JLE/JNG	Jump if less or equal/ Jump if not greater	16/4	2	–	–	–	–	–	–	–	–	–
JNLE/JG	Jump if not less or equal/ Jump if greater	16/4	2	–	–	–	–	–	–	–	–	–
JCXZ	Jump if CX is zero	18/6	2	–	–	–	–	–	–	–	–	–

Figure A.6: Table A.4

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags* O	D	I	T	S	Z	A	P	C
LOOP	Loop	17/5	2	–	–	–	–	–	–	–	–	–
LOOPE/ LOOPZ	Loop if equal/ Loop if zero	18/6	2	–	–	–	–	–	–	–	–	–
LOO- PNZ/ LOOPNE	Loop if not zero/ Loop if not equal	19/5	2	–	–	–	–	–	–	–	–	–

Figure A.7: Table A.5 Loop instructions

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags* *O D I T S Z A P C*
NOP	No operation	3	1	– – – – – – – – –
HLT	Halt	2	1	– – – – – – – – –
WAIT	Wait while $\overline{\text{TEST}}$ pin not asserted	3+5n	1	– – – – – – – – –
LOCK	Lock Bus	2	1	– – – – – – – – –
ESC	Escape			– – – – – – – – –
Register	2	2		
	Memory	8+AM	2–4	

Figure A.8: Table A.6 Machine Control instructions

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags* *O D I T S Z A P C*
CLC	Clear carry flag	2	1	– – – – – – – – 0
CMC	Complement carry flag	2	1	– – – – – – – – m
STC	Set carry flag	2	1	– – – – – – – – 1
CLD	Clear direction flag	2	1	– 0 – – – – – – –
STD	Set direction flag	2	1	– 1 – – – – – – –
CLI	Clear interrupt flag	2	1	– – 0 – – – – – –
STI	Set interrupt flag	2	1	– – 1 – – – – – –

Figure A.9: Table A.7 Flag Manipulation instructions

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags*								
				O	*D*	*I*	*T*	*S*	*Z*	*A*	*P*	*C*
SHL/SAL	Shift Logical Left/ Shift arithmetic Left			m	–	–	–	m		m	d	m	m
	Register with single shift	2	2									
	Register with variable shift	8 + 4/bit	2									
	Memory with single shift	15 + AM	2–4									
	Memory with variable shift	(20 + AM) + 4/bit	2–4									
SHR	Shift logical right			m	–	–	–	m		m	d	m	m
	Register with single shift	2	2									
	Register with variable shift	8 + 4/bit	2									
	Memory with single shift	15 + AM	2–4									
	Memory with variable shift	(20 + AM) + 4/bit	2–4									
SAR	Shift arithmetic right			m	–	–	–	m		m	d	m	m

Figure A.10: Table A.8 Shift and Rotate instructions

Mnemonic	*Description*	*Clock Cycles*	*Number of bytes*	*Flags* O	D	I	T	S	Z	A	P	C
	Register with single shift	2	2									
	Register with variable shift	8 + 4/bit	2									
	Memory with single shift	15+AM	2–4									
	Memory with variable shift	(20+AM) + 4/bit	2–4									
ROR	Rotate right without carry			m	–	–	–	–	–	–	–	m
	Register with single shift	2	2									
	Register with variable shift	8 + 4 bit	2									
	Memory with single shift	15 + AM	2–4									
	Memory with variable shift	(20 + AM) + 4/bit	2–4									
ROL	Rotate left			m	–	–	–	–	–	–	–	m
	Register with single shift	2	2									
	Register with variable shift	8 + 4/bit	2									
	Memory with single shift	15 + AM	2–4									
	Memory with variable shift	(20 + AM) + 4/bit	2–4									
RCR	Rotate right through carry			m	–	–	–	–	–	–	–	m
	Register with single shift	2	2									
	Register with variable shift	8 + 4/bit	2									
	Memory with single shift	15 + AM	2–4									
	Memory with variable shift	(20 + AM) + 4/bit	2–4									
RCL	Rotate left through carry			m	–	–	–	–	–	–	–	m
	Register with single shift	2	2									
	Register with variable shift	8 + 4/bit	2									
	Memory with single shift	15 + AM	2–4									
	Memory with variable shift	(20 + AM) + 4/bit	2–4									

Figure A.11: Table A.8

Mnemonic	Description	Clock Cycles	Number of bytes	Flags O	D	I	T	S	Z	A	P	C
CMPS/ CMPSB CMPSW	Compare string/compare byte string/compare word string		1	m	–	–	–	m		m	m	m	m
	Not repeated	22											
	Repeated	9 + 22/rep											
MOVS/ MOVSB/ MOVSW	Move string/move byte string/move Word string		1	–	–	–	–	–	–	–	–	–	
	Not repeated	18											

Figure A.12: Table A.9 String Instructions

Mnemonic	Description	Clock Cycles	Number of bytes	Flags O	D	I	T	S	Z	A	P	C
	Repeated	9 + 17/rep										
LODS/ LODSB/ LODSW	Load string/Load String byte/Load String word		1	–	–	–	–	–	–	–	–	–	
	Not repeated	12											
	Repeated	9 + 13/rep											
SCAS/ SCASB/ SCASW	Scan string/ Scan byte string/ Scan word string		1										
	Not repeated		15	m	–	–	–	m		m	m	m	m
	Repeated	9 + 15/rep											
STOS/ STOSB/ STOSW	Store string/ Store byte string/ Store word string		1	–	–	–	–	–	–	–	–	–	
	Not repeated		11										
	Repeated	9 + 10/rep											

Figure A.13: Table A.9

Appendix B :Instruction set of 8051 microcontroller

Appendix A lists two arrangements of mnemonics for the 8051: by function, and alpha betically. The mnemonic definitions differ from that of the original manufacturer (Int Corporation) by the names used for addresses or data; for example, "add" is used to repre sent an address in internal RAM, while Intel uses the name "direct." The author believe that the names used are clearer than those used by Intel. Appendix A also includes a alphabetical listing of the mnemonics using Intel names. There is no difference betwee the mnemonics when real numbers replace the names. For example; MOV add,#n an MOV direct,#data become MOV 10h,#40h when the number 10h replaces the interna RAM address (add/direct), and 40h replaces the number (#n/# data).

Mnemonics, Arranged by Function

Arithmetic

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ADD A,Rr	A+Rr → A	1	1	C OV AC
ADD A,add	A+(add) → A	2	1	C OV AC
ADD A,@Rp	A+(Rp) → A	1	1	C OV AC
ADD A,#n	A+n → A	2	1	C OV AC
ADDC A,Rr	A+Rr+C → A	1	1	C OV AC
ADDC A,add	A+(add)+C → A	2	1	C OV AC
ADDC A,@Rp	A+(Rp)+C → A	1	1	C OV AC
ADDC A,#n	A+n+C → A	2	1	C OV AC
DA A	Abin → Adec	1	1	C
DEC A	A−1 → A	1	1	
DEC Rr	Rr−1 → Rr	1	1	
DEC add	(add)−1 → (add)	2	1	
DEC @Rp	(Rp)−1 → (Rp)	1	1	

Figure B.1: Arithmetic instructions

Arithmetic

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
				Continue
DIV AB	A/B → AB	1	4	0 0V
INC A	A+1 → A	1	1	
INC Rr	Rr+1 → Rr	1	1	
INC add	(add)+1 → (add)	2	1	
INC @Rp	(Rp)+1 → (Rp)	1	1	
INC DPTR	DPTR+1 → DPTR	1	2	
MUL AB	A×B → AB	1	4	0 0V
SUBB A,Rr	A−Rr−C → A	1	1	C OV AC
SUBB A,add	A−(add)−C → A	2	1	C OV AC
SUBB A,@Rp	A−(Rp)−C → A	1	1	C OV AC
SUBB A,#n	A−n−C → A	2	1	C OV AC

Logic

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ANL A,Rr	A AND Rr → A	1	1	
ANL A,add	A AND (add) → A	2	1	
ANL A,@Rp	A AND (Rp) → A	1	1	
ANL A,#n	A AND n → A	2	1	
ANL add,A	(add) AND A → (add)	2	1	
ANL add,#n	(add) AND n → (add)	3	2	
ORL A,Rr	A OR Rr → A	1	1	
ORL A,add	A OR (add) → A	2	1	
ORL A,@Rp	A OR (Rp) → A	1	1	
ORL A,#n	A OR n → A	2	1	
ORL add,A	(add) OR A → (add)	2	1	
ORL add,#n	(add) OR n → (add)	3	2	
XRL A,Rr	A XOR Rr → A	1	1	
XRL A,add	A XOR (add) → A	2	1	
XRL A,@Rp	A XOR (Rp) → A	1	1	
XRL A,#n	A XOR n → A	2	1	
XRL add,A	(add) XOR A → (add)	2	1	
XRL add,#n	(add) XOR n → (add)	3	2	
CLR A	00 → A	1	1	
CPL A	$\overline{A}$ → A	1	1	
NOP	PC+1 → PC	1	1	
RL A	A0←A7←A6. .←A1←A0	1	1	
RLC A	C←A7←A6. .←A0←C	1	1	C
RR A	A0→A →A6. .→A1→A0	1	1	
RRC A	C→A7→A6. .→A0→C	1	1	C
SWAP A	Alsn ↔ Amsn	1	1	

Data Moves

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
MOV A,Rr	Rr → A	1	1	
MOV A,add	(add) → A	2	1	
MOV A,@Rp	(Rp) → A	1	1	

Figure B.2: Arithmetic instructions.

Arithmetic

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
				Continue
DIV AB	A/B → AB	1	4	0 0V
INC A	A+1 → A	1	1	
INC Rr	Rr+1 → Rr	1	1	
INC add	(add)+1 → (add)	2	1	
INC @Rp	(Rp)+1 → (Rp)	1	1	
INC DPTR	DPTR+1 → DPTR	1	2	
MUL AB	A×B → AB	1	4	0 0V
SUBB A,Rr	A−Rr−C → A	1	1	C OV AC
SUBB A,add	A−(add)−C → A	2	1	C OV AC
SUBB A,@Rp	A−(Rp)−C → A	1	1	C OV AC
SUBB A,#n	A−n−C → A	2	1	C OV AC

Logic

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ANL A,Rr	A AND Rr → A	1	1	
ANL A,add	A AND (add) → A	2	1	
ANL A,@Rp	A AND (Rp) → A	1	1	
ANL A,#n	A AND n → A	2	1	
ANL add,A	(add) AND A → (add)	2	1	
ANL add,#n	(add) AND n → (add)	3	2	
ORL A,Rr	A OR Rr → A	1	1	
ORL A,add	A OR (add) → A	2	1	
ORL A,@Rp	A OR (Rp) → A	1	1	
ORL A,#n	A OR n → A	2	1	
ORL add,A	(add) OR A → (add)	2	1	
ORL add,#n	(add) OR n → (add)	3	2	
XRL A,Rr	A XOR Rr → A	1	1	
XRL A,add	A XOR (add) → A	2	1	
XRL A,@Rp	A XOR (Rp) → A	1	1	
XRL A,#n	A XOR n → A	2	1	
XRL add,A	(add) XOR A → (add)	2	1	
XRL add,#n	(add) XOR n → (add)	3	2	
CLR A	00 → A	1	1	
CPL A	$\overline{A}$ → A	1	1	
NOP	PC+1 → PC	1	1	
RL A	A0←A7←A6..←A1←A0	1	1	
RLC A	C←A7←A6..←A0←C	1	1	C
RR A	A0→A →A6..→A1→A0	1	1	
RRC A	C→A7→A6..→A0→C	1	1	C
SWAP A	Alsn ↔ Amsn	1	1	

Data Moves

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
MOV A,Rr	Rr → A	1	1	
MOV A,add	(add) → A	2	1	
MOV A,@Rp	(Rp) → A	1	1	

Figure B.3: Arithmetic instructions.

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
MOV A,#n	n → A	2	1	
MOV Rr,A	A → Rr	1	1	
MOV Rr,add	(add) → Rr	2	2	
MOV Rr,#n	n → Rr	2	1	
MOV add,A	A → (add)	2	1	
MOV add,Rr	Rr → (add)	2	2	
MOV add1,add2	(add2) → (add1)	3	2	
MOV add,@Rp	(Rp) → (add)	2	2	
MOV add,#n	n → (add)	3	2	
MOV @Rp,A	A → (Rp)	1	1	
MOV @Rp,add	(add) → (Rp)	2	2	
MOV @Rp,#n	n → (Rp)	2	1	
MOV DPTR,#nn	nn → DPTR	3	2	
MOVC A,@A+DPTR	(A+DPTR) → A	1	2	
MOVC A,@A+PC	(A+PC) → A	1	2	
MOVX A,@DPTR	(DPTR)^ → A	1	2	
MOVX A,@Rp	(Rp)^ → A	1	2	
MOVX @Rp,A	A → (Rp)^	1	2	
MOVX @DPTR,A	A → (DPTR)^	1	2	
POP add	(SP) → (add)	2	2	
PUSH add	(add) → (SP)	2	2	
XCH A,Rr	A ↔ Rr	1	1	
XCH A,add	A ↔ (add)	2	1	
XCH A,@Rp	A ↔ (Rp)	1	1	
XCHD A,@Rp	Alsn ↔ (Rp)lsn	1	1	

Calls and Jumps

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ACALL sadd	PC+2 → (SP); sadd → PC	2	2	
CJNE A,add,radd	[A<>(add)]: PC+3+radd → PC	3	2	C
CJNE A,#n,radd	[A<>n]: PC+3+radd → PC	3	2	C
CJNE Rr,#n,radd	[Rr<>n]: PC+3+radd → PC	3	2	C
CJNE @Rp,#n,radd	[(Rp)<>n]: PC+3+radd → PC	3	2	C
DJNZ Rr,radd	[Rr−1<>00]: PC+2+radd → PC	2	2	
DJNZ add,radd	[(add)−1<>00]: PC+3+radd → PC	3	2	
LCALL ladd	PC+3 → (SP); ladd → PC	3	2	
AJMP sadd	sadd → PC	2	2	
LJMP ladd	ladd → PC	3	2	
SJMP radd	PC+2+radd → PC	2	2	
JMP @A+DPTR	DPTR+A → PC	1	2	
JC radd	[C=1]: PC+2+radd → PC	2	2	
JNC radd	[C=0]: PC+2+radd → PC	2	2	
JB b,radd	[b=1]: PC+3+radd → PC	3	2	
JNB b,radd	[b=0]: PC+3+radd → PC	3	2	
JBC b,radd	[b=1]: PC+3+radd → PC; 0 → b	3	2	
JZ radd	[A=00]: PC+2+radd → PC	2	2	
JNZ radd	[A>00]: PC+2+radd → PC	2	2	
RET	(SP) → PC	1	2	
RETI	(SP) → PC; EI	1	2	

Figure B.4: call and jump instructions

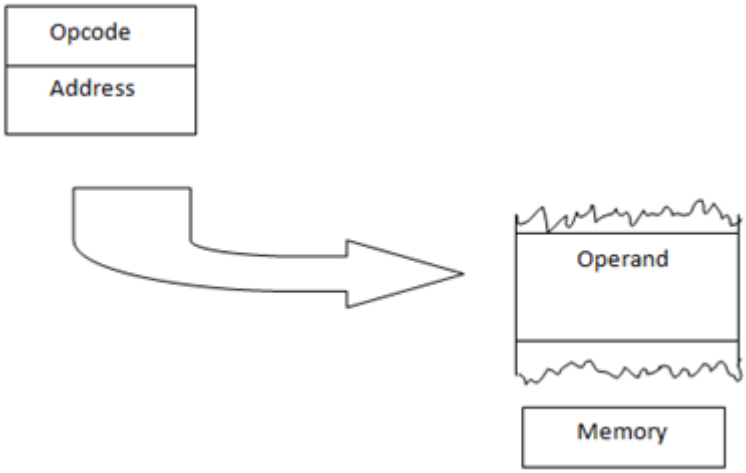


Figure B.5: Boolean instructions

Appendix C :Addressing Modes of 8086 microprocessor

Introduction::
Addressing mode tells us what is the type of the operand and the way they are accessed from the memory for execution of an instruction and how to fetch particular instruction from the memory. There are mainly 8 addressing modes of an 8086 microprocessor. **Immediate Addressing Mode:**
In this immediate data is the part of the instruction itself.
Example: Mov AX, 0005H
Absolute or Direct Addressing Mode:
In it, a 16-bit memory address (offset) or an input/ output address is directly specified in the instruction as a part of it.

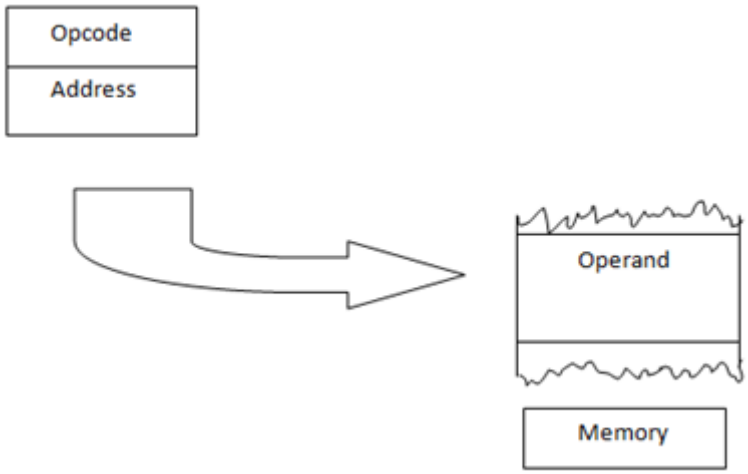


Figure C.1: Absolute or Direct Addressing Mode

Register Addressing Mode:
Here data is stored in a registered and referred using the particular register.

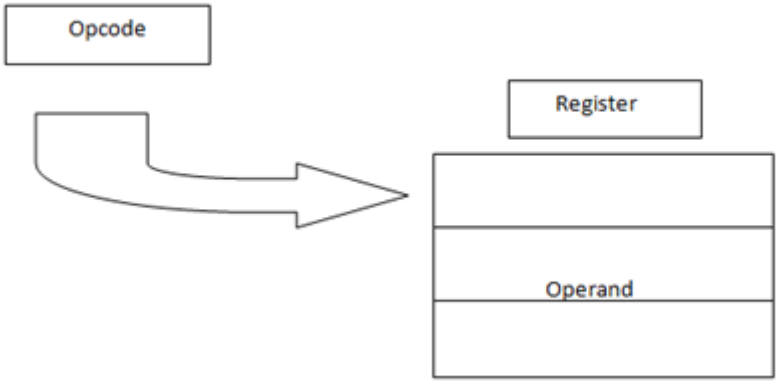


Figure C.2: Register Addressing Mode

Register Indirect Addressing Mode:
In this offset address of data is in either Bx, SI, DI, (Base register, source index or Destination index) default segment is either DS or ES.
Data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

Instruction	Opcode	Bytes	Cycles
MOV A, @ R0	E6H	1	1

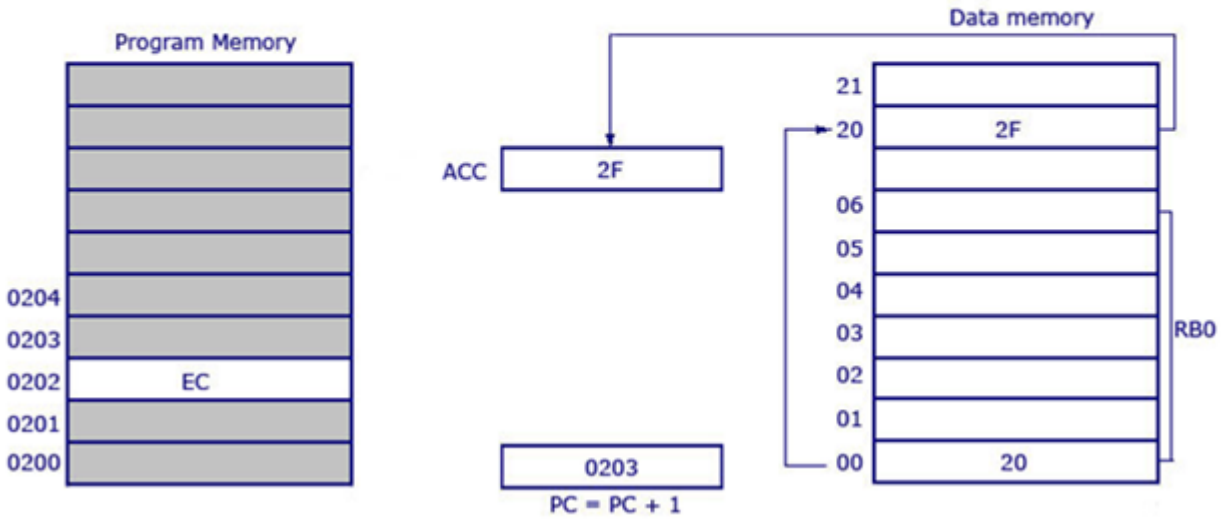


Figure C.3: Register Indirect Addressing Mode

Indexed Addressing Mode:
Here offset of the operand is stored in one of the index registers. DS is the default segment for SI and DI in string instruction DS and ES default segment for register SI and DI.

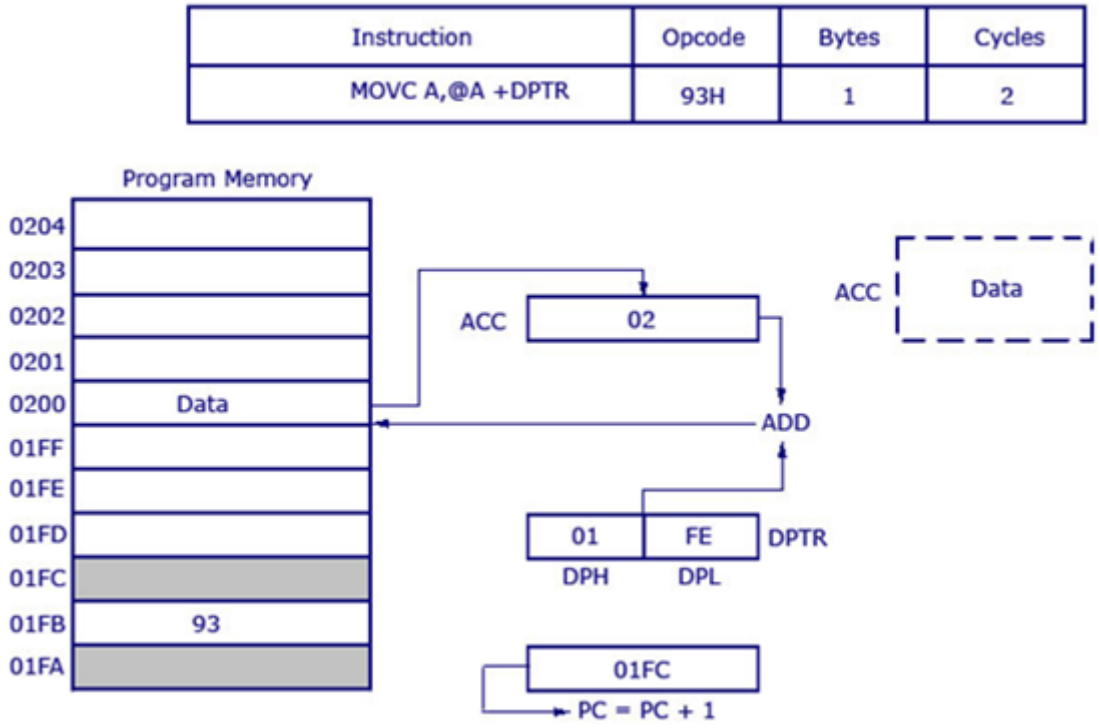


Figure C.4: Indexed Addressing Mode

Register Relative Addressing Mode:
In it, data is available at an effective address formed by adding an 8 bit or 16-bit displacement with content, any one of the registers Bx, Bp, SI, DI in the default (DS or ES) segment.

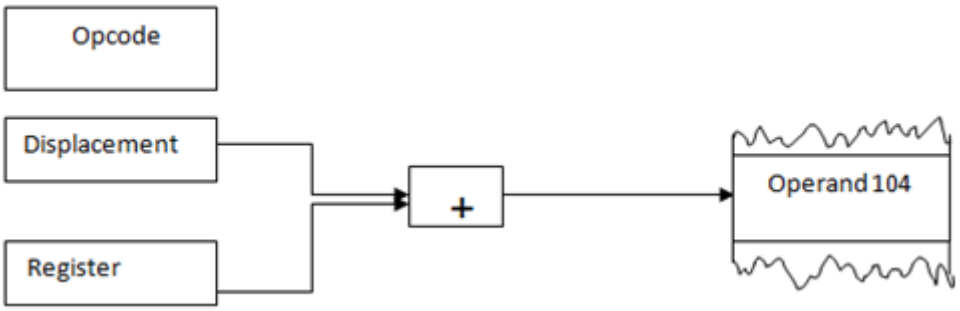


Figure C.5: Register Relative Addressing Mode

Based Indexed Addressing Mode:
The effective address of data is formed by adding content of base register Bx or Bp to the content of index register.

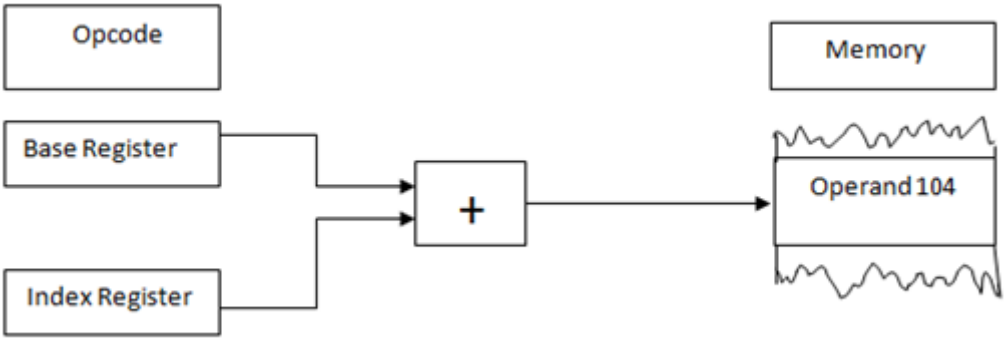


Figure C.6: Based Indexed Addressing Mode

Relative Based Indexed Addressing Mode:
Here the effective address is formed by adding an 8 bit or 16-bit displacement with the sum of the content of any one of the index registers in the default segment.

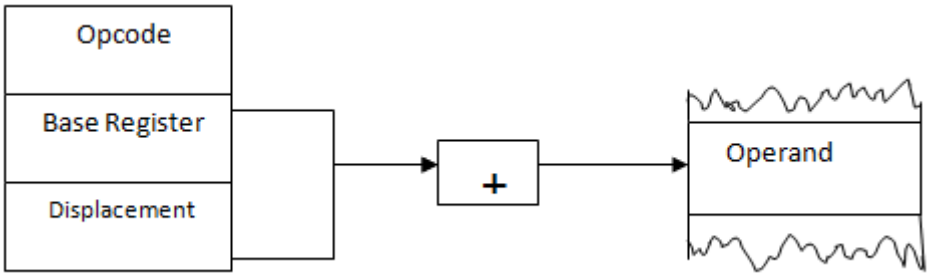


Figure C.7: Relative Based Indexed Addressing Mode

Appendix D :Addressing modes of 8051 Microcontroller:

Introduction::
There are 5 different ways to execute this instruction and hence we say, we have got 5 addressing modes for 8051. They are

1. Immediate addressing mode
2. Direct addressing mode
3. Register direct addressing mode
4. Register indirect addressing mode
5. Indexed addressing mode.

Immediate Addressing Mode:
MOV A, #6AH
In general we can write MOV A, #data This addressing mode is named as "immediate" because it transfers an 8-bit data immediately to the accumulator (destination operand).

Instruction	Opcode	Bytes	Cycles
MOV A, #6AH	74H	2	1

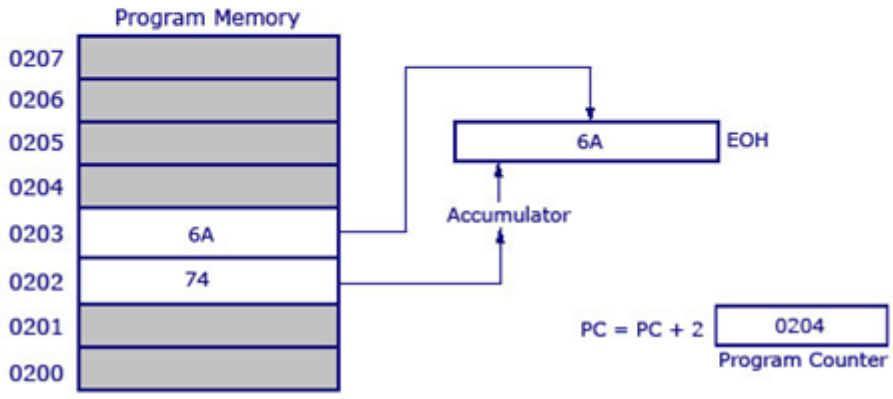


Figure D.1: Immediate Addressing Mode

The picture above describes the above instruction and its execution. The opcode for MOV A, # data is 74H. The opcode is saved in program memory at 0202 address.
The data 6AH is saved in program memory 0203. (See, any part of the program memory can

be used, this is just an example) When the opcode 74H is read, the next step taken would be to transfer whatever data at the next program memory address (here at 0203) to accumulator A (E0H is the address of accumulator). This instruction is of two bytes and is executed in one cycle. So after the execution of this instruction, program counter will add 2 and move to o204 of program memory.
Note: The # symbol before 6AH indicates that operand is a data (8 bit). If # is not present then the hexadecimal number would be taken as address.

Direct Addressing Mode:
This is another way of addressing an operand. Here the address of the data (source data) is given as operand. Lets take an example.

MOV A, 04H

Here 04H is the address of register 4 of register bank 0. When this instruction is executed, what ever data is stored in register 04H is moved to accumulator. In the picture below we can see, register 04H holds the data 1FH. So the data 1FH is moved to accumulator. Note: We have not used # in direct addressing mode, unlike immediate mode. If we had used #, the data value 04H would have been transferred to accumulator instead 0f 1FH.

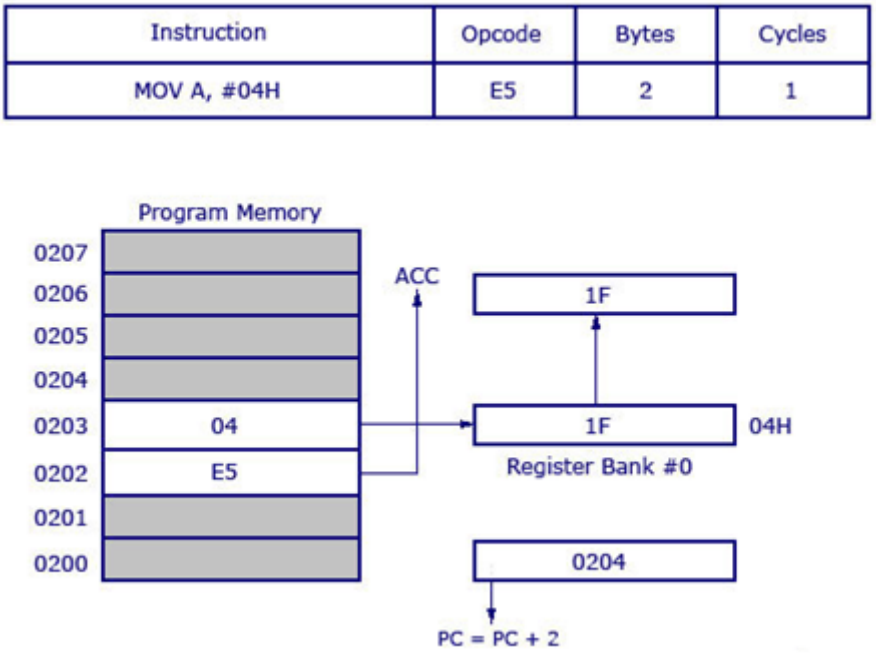


Figure D.2: Direct Addressing Mode

As shown in picture above this is a 2 byte instruction which requires 1 cycle to complete. Program counter will increment by 2 and stand in 0204. The opcode for instruction MOV A, address is E5H. When the instruction at 0202 is executed (E5H), accumulator is made active and ready to receive data. Then program control goes to next address that is 0203 and look up the address of the location (04H) where the source data (to be transferred to accumulator) is located. At 04H the control finds the data 1F and transfers it to accumulator and hence the execution is completed.

Register Direct Addressing Mode:
In this addressing mode we use the register name directly (as source operand). An example is shown below.
MOV A, R4
At a time registers can take value from R0,R1...to R7. You may already know there are 32

such registers. So how you access 32 registers with just 8 variables to address registers? Here comes the use of register banks. There are 4 register banks named 0,1,2 and 3. Each bank has 8 registers named from R0 to R7. At a time only one register bank can be selected. Selection of register bank is made possible through a Special Function Register (SFR) named Processor Status Word (PSW). PSW is an 8 bit SFR where each bit can be programmed. Bits are designated from PSW.0 to PSW.7 Register banks are selected using PSW.3 and PSW.4 These two bits are known as register bank select bits as they are used to select register banks. A picture below shows the PSW register and the Register Bank Select bits with status.

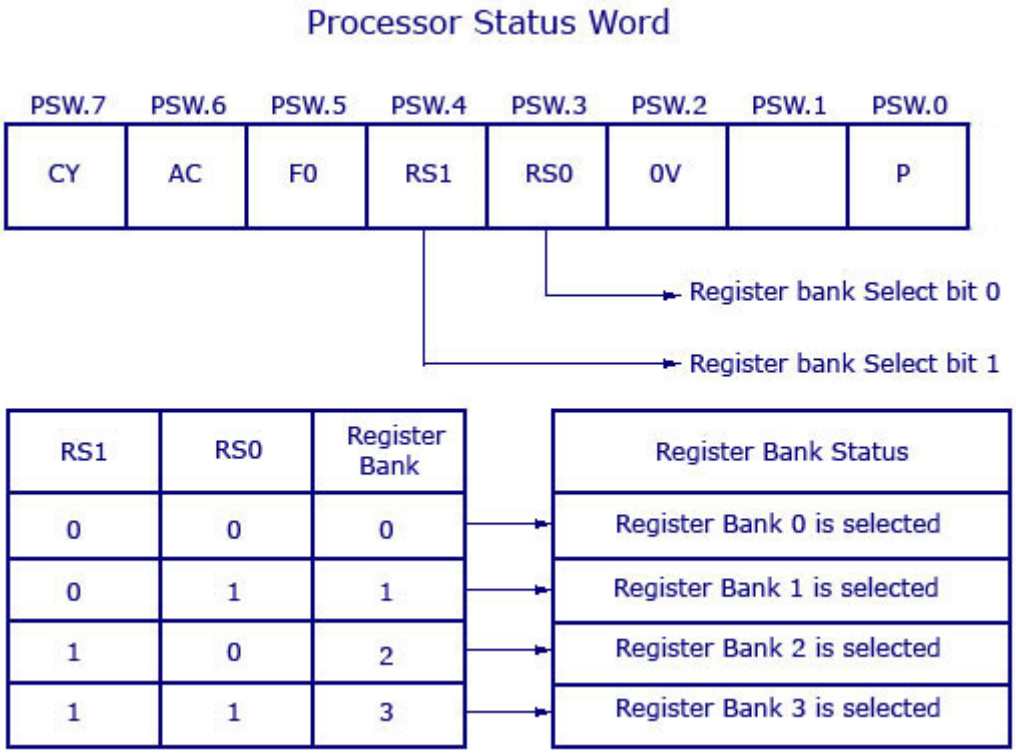


Figure D.3: PSW

So in register direct addressing mode, data is transferred to accumulator from the register (based on which register bank is selected).
Take a look at the picture below.
So we see that opcode for MOV A, R4 is EC. The opcode is stored in program memory address 0202 and when it is executed the control goes directly to R4 of the respected register bank (that is selected in PSW). If register bank #0 is selected then the data from R4 of register bank #0 will be moved to accumulator. (Here it is 2F stored at 04 H). 04 H is the address of R4 of register bank #0. Movement of data (2F) in this case is shown as bold line. Now please take a look at the dotted line. Here 2F is getting transferred to accumulator from data memory location 0C H. Now understand that 0C H is the address location of Register R4 of register bank #1. Programmers usually get confused with register bank selection. Also keep in mind that data at R4 of register bank #0 and register bank #1 (or even other banks) will not be same. So wrong selection of register banks will result in undesired output.
Also note that the instruction above is 1 byte and requires 1 cycle for complete execution. This means using register direct addressing mode can save program memory.

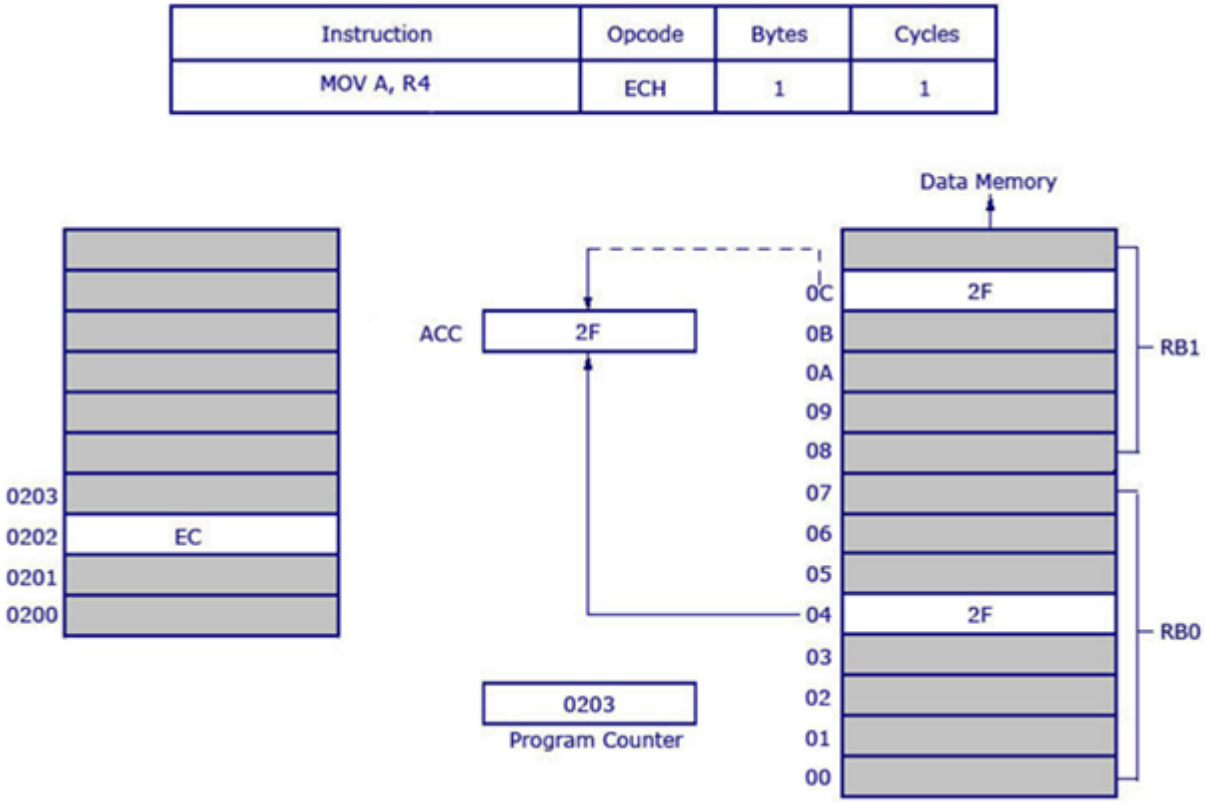


Figure D.4: Register Direct Addressing Mode

Register Indirect Addressing Mode:

So in this addressing mode, address of the data (source data to transfer) is given in the register operand.

MOV A, @R0

Here the value inside R0 is considered as an address, which holds the data to be transferred to accumulator.

Example: If R0 holds the value 20H, and we have a data 2F H stored at the address 20H, then the value 2FH will get transferred to accumulator after executing this instruction. Got it? See the picture below.

Instruction	Opcode	Bytes	Cycles
MOV A, @ R0	E6H	1	1

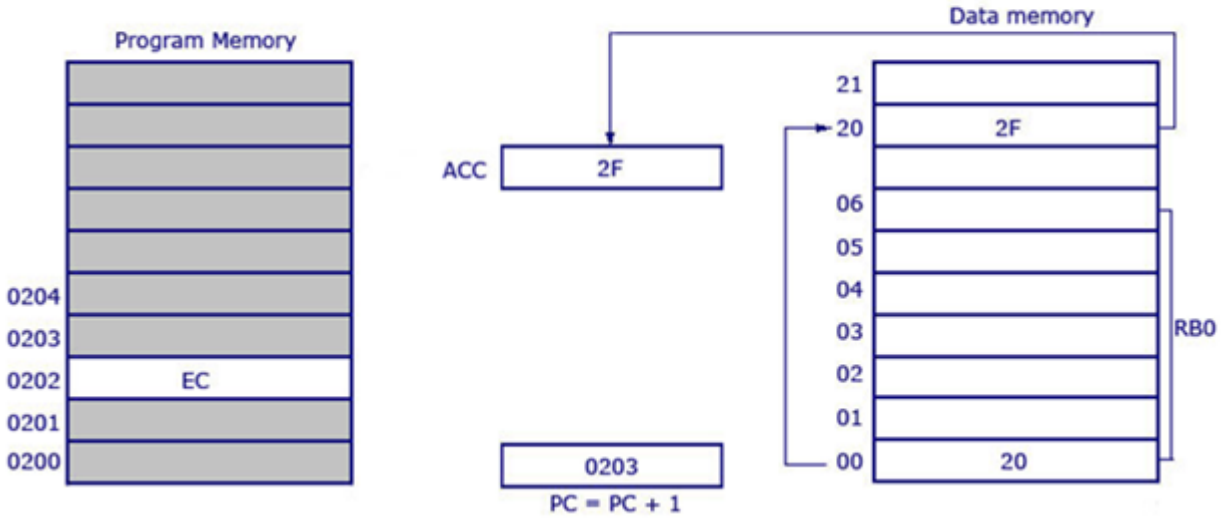


Figure D.5: Register Indirect Addressing Mode

So the opcode for MOV A, R0 is E6H. Assuming that register bank #0 is selected. So the R0 of register bank #0 holds the data 20H. Program control moves to 20H where it locates the data 2FH and it transfers 2FH to accumulator. This is a single byte instruction and the program counter increments 1 and moves to 0203 of program memory.
Note: Only R0 and R1 are allowed to form a register indirect addressing instruction. In other words programmer can must make any instruction either using R0 or R1. All register banks are allowed. **Indexed Addressing Mode:**
Well lets see two examples first.
MOVC A, A+DPTR and MOVC A, A+PC where DPTR is data pointer and PC is program counter (both are 16 bit registers).
Lets take the first example.
MOVC A, A+DPTR The source operand is A+DPTR and we know we will get the source data (to transfer) from this location. It is nothing but adding contents of DPTR with present content of accumulator. This addition will result a new data which is taken as the address of source data (to transfer). The data at this address is then transferred to accumulator. Take a look at the picture below.

Instruction	Opcode	Bytes	Cycles
MOVC A,@A +DPTR	93H	1	2

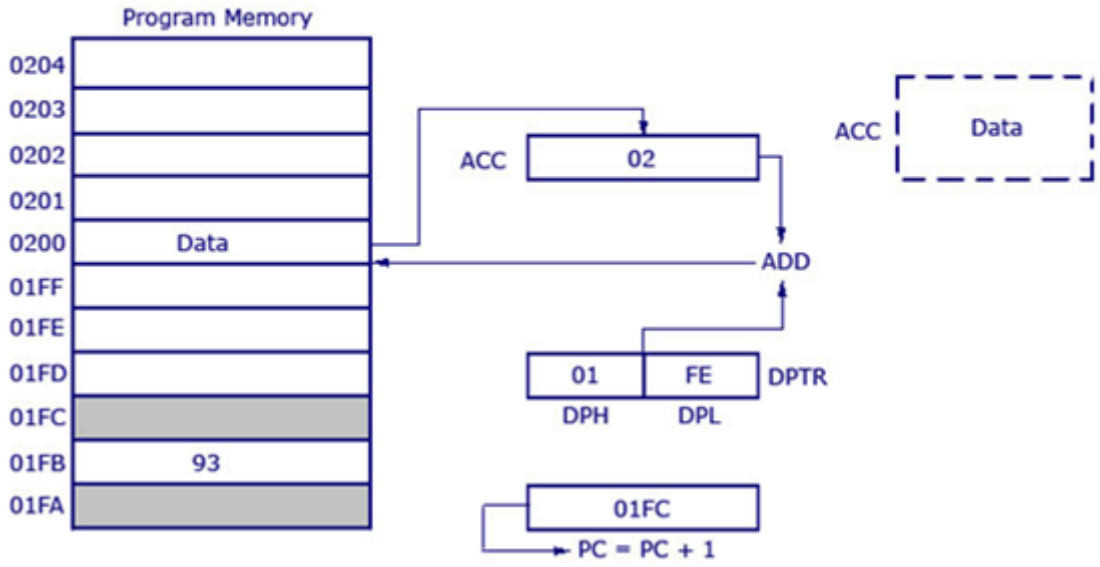


Figure D.6: Indexed Addressing Mode