King Faisal University
ENGINEERING
KFU
Uniqueness - Partnership - Success

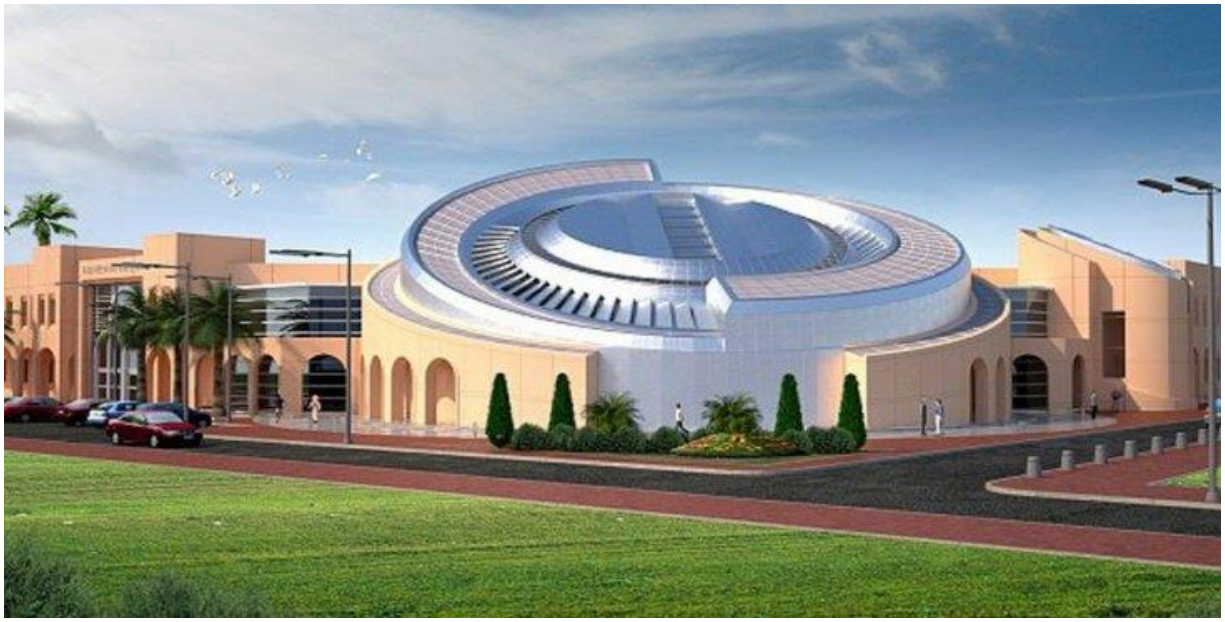
# KING FAISAL UNIVERSITY

## College Of Engineering

## DEPARTMENT OF ELECTRICAL ENGINEERING

# EE234: Microprocessors Lab

## "Lab Manual"



**Prepared By: Dr Slim Chtourou**

## Major Topics covered and schedule in weeks:

| Topic | Week # | Courses Covered |
|---|---|---|
| Introduction and Lab safety. | 1 | EE233 |
| Introduction to Microprocessor Lab Requirements and Policy. | 2 | |
| 8086 Microprocessor, MDA-Win8086 Hardware Kit and Emu8086 Software. | 3,4,5 | |
| 8086 Programming Based Experiments using assembly language | 6-8 | |
| Memory & Stack Access Operations for Intel 8086 Microprocessor | 9 | |
| Assembly optimization techniques | 10 | |
| Understanding Parallel Input Output Intel 8255A (PIO) Interface. | 11,13 | |
| Interruption Techniques for Intel 8086 Microprocessor Programming. | 14 | |
| Final Exam | 15 | |

**Specific Outcomes of Instruction (Lab Learning Outcomes):**

1. . An Ability to use MDA-Win8086 hardware kit and EMU8086 software emulator to utilize and emulate Intel 8086 Microprocessor.
2. An Ability to write assembly language programs for Intel 8086.
3. An Ability to evaluate addressing modes for 8086 Intel Microprocessor.
4. An Ability to perform memory & stack access operations for Intel 8086.
5. An Ability to operate Intel 8255A chip to interface 8086 processor with I/O devices such as LEDs, 7SD, DAC, LCD and others.
6. An Ability to recognize the interruption techniques for Intel 8086 Microprocessor.
7. An Ability to prepare a small Microprocessors based system such Traffic Light System.

**Student Outcomes (SO) Addressed by the Lab:**

| z | Outcome Description<br>General Engineering Student Outcomes | Contribution |
|---|---|---|
| 1. | an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics | M |
| 2. | an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors | |
| 3. | an ability to communicate effectively with a range of audiences | M |
| 4. | an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts | |
| 5. | an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives | M |
| 6. | an ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions | M |
| 7. | an ability to acquire and apply new knowledge as needed, using appropriate learning strategies | |

# Table of Content

# Experiment 1: Knowing and using MDA-win8086 microprocessor kit: Familiarization with the kit components, Modes of operations and MDA8086 IDE software

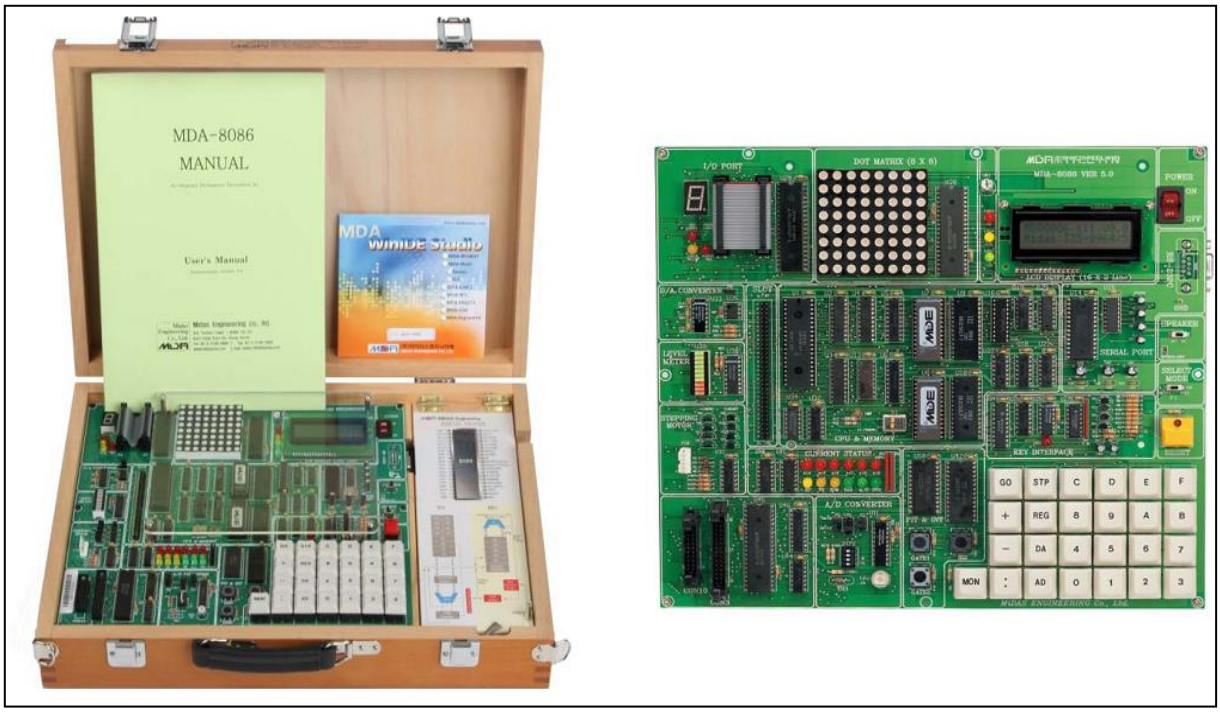
## I. Objective:

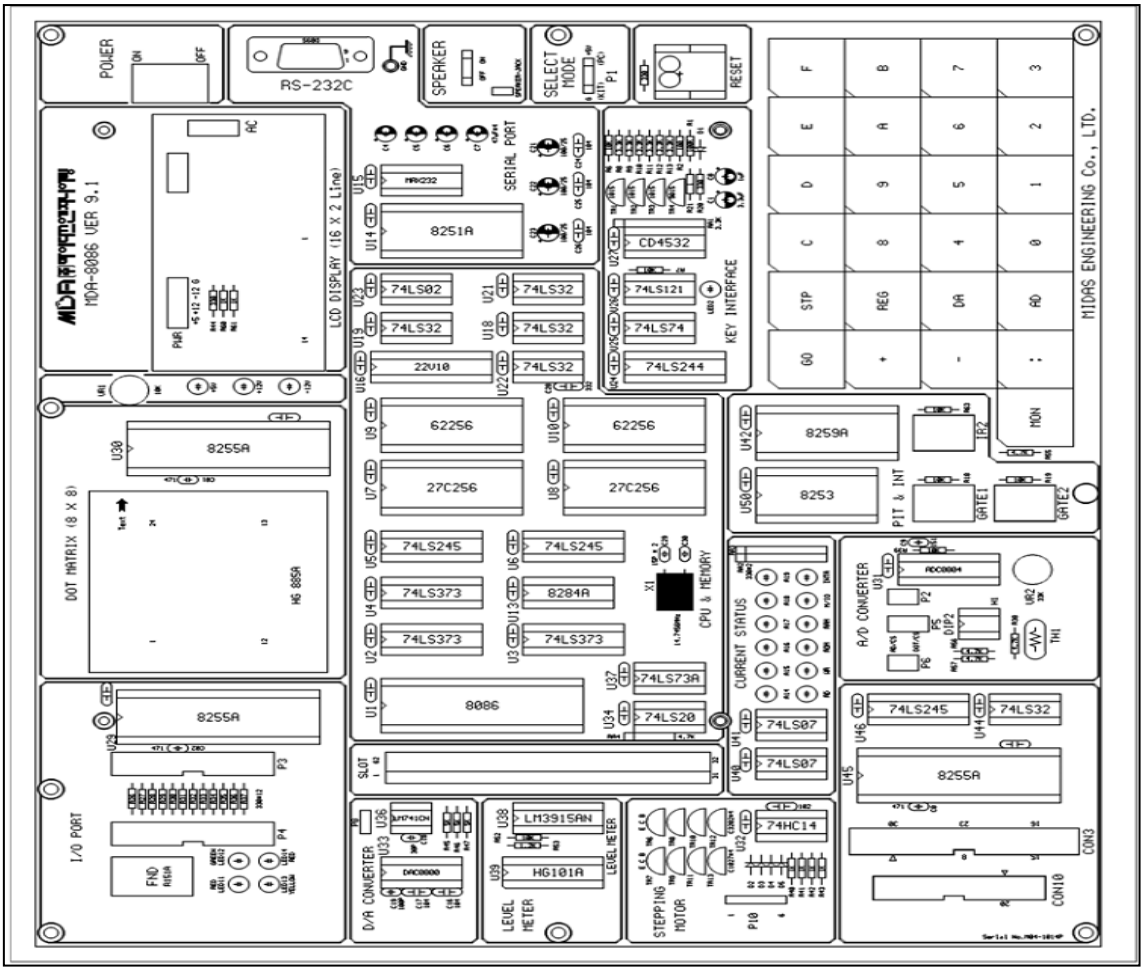
The purpose of this experiment is to learn how to use the MDA-Win8086 training kit and to familiarize with its various operations that can be helpful in using and programming the Microprocessor.

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:






## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

The working procedure (methodology) throughout this experiment consists of the following steps:

1. Start by reading the experiment environment, main objective and procedure.
2. Connect PC and MDA-Win 8086 Microprocessor Kit using the Data cable provided.
3. Switch 'ON' the MDA-Win 8086 Microprocessor Kit and the Computer, and go to Local Disk C: of the PC and Open the MDA folder and double click the ' MDA_Win IDE8086 ' Icon and start the MDA- Win 8086 Software Tool.
4. 'Reset' MDA-Win 8086 Microprocessor Kit by pressing the RESET button provided on the kit.

5. Follow up with the Experimental work step by step until you finish the required tasks. Also, solve the required exercises and make your final conclusions about the experiment.
6. Finally, you should submit your technical lab report for this experiment and the reports will be submitted by each group before the deadline specified by the instructor (In this lab report, you are only requested to write a conclusions section, in addition to solving the experimental work tasks as well as the exercises).

## VI. Experimental Work:

### 1. Discovering the Kit

Using the knowledge learned about I8086 MP, MDA-8086 Kit & User manual, answer the following questions:

1- On a power-up, what is the message displayed on a LCD? And what If you press the RESET key for the KIT?

……..……..
……..……..

2- What are the modes of operation for the KIT? How to switch between modes? How to verify that the required mode is ON?

……..……..
……..……..

3- What type of data cable is used to connect MDA-8086 to the PC?

……..……..

4- How many LEDs are there in the Kit? What are their purposes?

……..……..

5- List 4 output devices that are provided in this KIT.

……..……..

6- List 2 Input methods that can be used to instruct the 8086 Microprocessor.

……..……..

7- List 3 applications that are provided with the MDA-8086 KIT.

……..……..

8- What type of interfaces are used to communicate between the DOT MATRIX LED and INTEL 8086 MP?

……..……..

9- How many pins does INTEL 8086 have?

……..……..

10- How many address pins are in the address bus of Intel 8086? What are the pins' numbers in Intel 8086?

…….....

11- How many data pins are in the data bus for Intel 8086. What are the pins' numbers in Intel 8086?

…….....

12- Why does we say that the data and address bus for Intel 8086 are multiplexed?

…….....

13- List 5 control pins (signals) for Intel 8086.

…….....

14- How many pins in 8086 are ground pins? Vcc Pins? What are the pins' numbers in 8086 MP.

…….....

15- How many hexadecimal digits are used to locate an address in the Memory? An address for I/O Port?

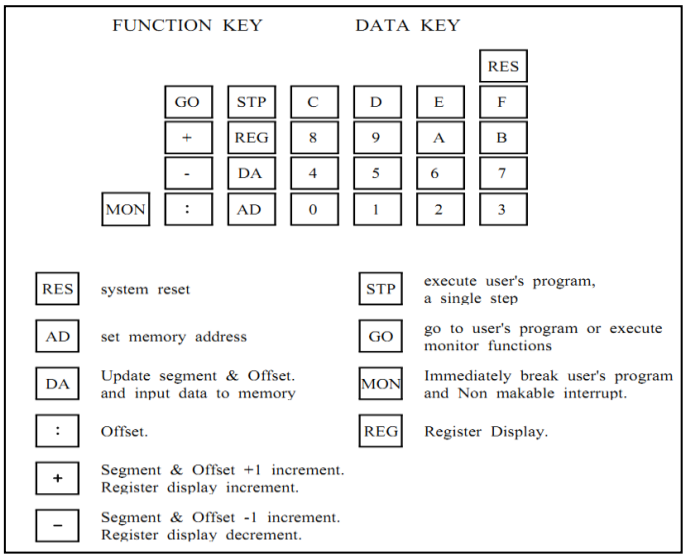
…….....

16- What are the roles of RAM and ROM?

…….....
…….....

## 2. Using the kit:

Using MDA-8086 Kit, Keypad & User manual, answer the following tasks:




1- Select the KIT mode from SELECT MODE Switch, Press RESET key. What happened to the LCD?

…….....

2- Check the contents of memory locations 01000, 01001, 01002, 01003. List the keys used to accomplish this task in order. What is the data stored in these locations?

……
……

3- Store the following data into corresponding memory locations. List the keys used to accomplish this task in order.

| Address | Data |
|---|---|
| 01003 | 34 |
| 01002 | EF |
| 01001 | CD |
| 01000 | 12 |

……
……
……
……

4- Press RESET key, then re-check the contents of memory locations 01000, 01001, 01002, 01003. Explain.

……

5- Check the contents of the registers for Intel 8086. List the keys used to accomplish this task in order. What is the data stored in this location?

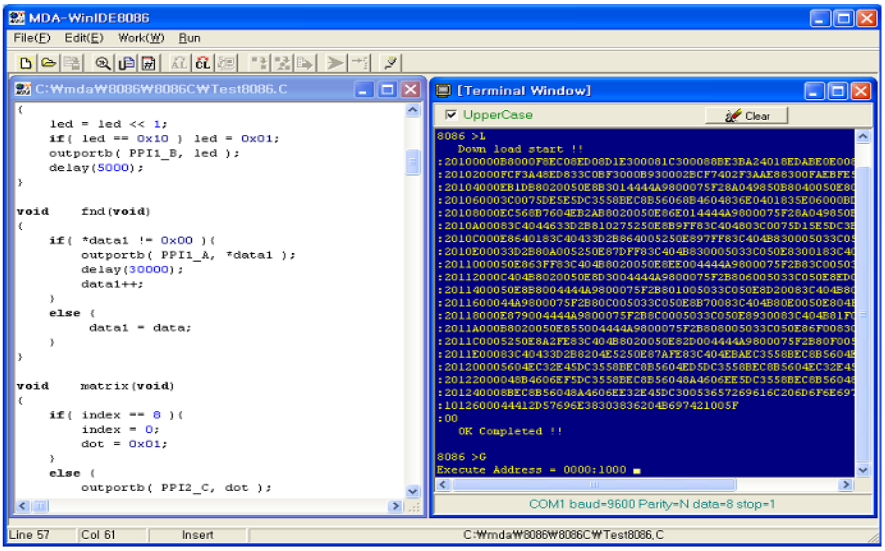
……
……

6- Explain the purpose for the first four registers you checked in the previous part.

……
……

## 3. WinIDE 8086 Software

Using MDA-WinIDE8086 software and MDA-8086 user manual, answer the following questions:



1- Now, In order to use MDA-Win IDE8086 program correctly with the KIT, we should select the PC mode from SELECT MODE Switch. Why?

……..…….....

2- After launching the program, there are 2 windows: editor window and Terminal window. What is the purpose for each window?

……..…….....
……..…….....

3- What is the difference between the buttons: ? 


……..…….....
……..…….....

4- After connecting MDA-Win 8086 Kit to the PC through the provided data cable. How to check that it is connected correctly? How can we change the correct configurations for the connections?

……..…….....

5- What is the difference between: Run and Trace buttons?

……..…….....

6- What is the procedure used to execute any program in MDA-Win IDE 8086? (list in order)

……..…….....

7- Now apply the help command by typing **?** to the terminal window. Explain the purpose of this command.

……..…….....

8- Use the Memory dump window to show the contents of 10 consecutive memory locations starting from 01000. How to perform this? What are the contents?

……..…….....
……..…….....

9- Use Data fill window to store the value 1234 into of 10 consecutive memory locations starting from 01000 with the data. How to perform this? What is the new contents?

……..…….....
……..…….....

10- In the terminal window; use the command **R** ( 8086 >R↵ ) to display Intel 8086 MP registers. What if we want to display the contents of IP register only?

……..…….....

11- Change the contents of AX, BX, CX, DX, and SP to: 1234, 4567, 7788, 1111, and 0540 respectively. Then reuse the R command to show the new contents.

…….....

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 2: Working and understanding emu8086 emulator software: running/debugging ALPs and tracing/analyzing 8086 registers, memory, ALU and stack

## I. Objective:

The purpose of this experiment is to :

- Learn how to use Emu-8086 Emulation software
- Familiarize with its various options and tools that can be helpful in using and programming the Microprocessor 8086.

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

You will be using an emulation software (Emu-8086) which is installed on a (PC). All needed application software are stored locally on drive c: which is the system drive. All personals files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use Emu_8086' software icon as a tool to write and debug assembly programs for the 8086 Microprocessor.

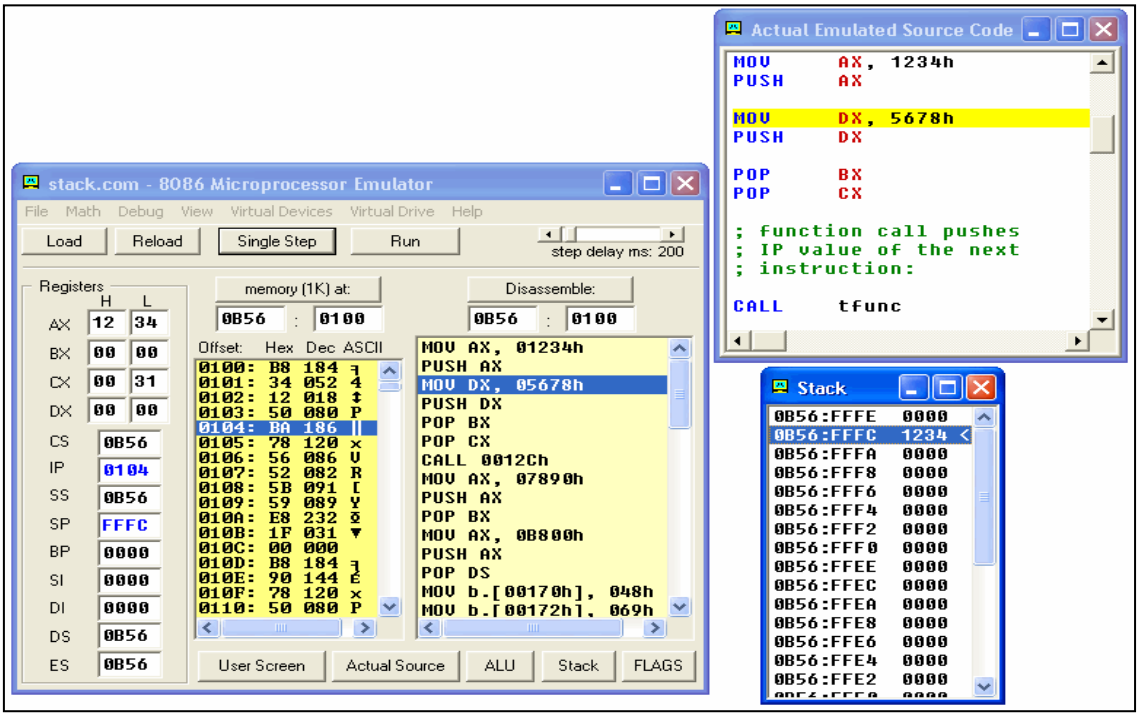



*Figure1: Major Windows for EMU8086*

## IV. Apparatus:

PC
Emu8086 software

## V. Procedure:

The working procedure throughout this experiment consists of the following steps:

1. Start by reading the experiment environment, main objective and procedure.

2. Turn On your PC, then Click on Emu_8086 Icon in order to start working in the emulated environment for Intel 8086 Microprocessor.

3. Click code examples and select **"Hello, world"**. A code example with many comments should open. All comments are green and they take up about 90% of all text, so don't be scared by this tiny "Hello Word" code. The compiled executable is only about 100 bytes long, because it uses no interrupts and has only one loop for color highlighting the text. All other codes are straight-forward and write directly to video memory.

4. To run this example in the emulator, click **emulate** (or press F5). The program then attempts to assemble and save the executable to **"c:\emu8086\MyBuild"**. If the assembler succeeds in creating the file, the emulator will also automatically load it into memory. To run the program, click on **run** or press F9 and run the program.

5. You can then click **single step** (or press F8) to run the code, one instruction at a time, observing changes in registers and the emulator screen. You can also click **step back** (or press F6) to see what happens when reversing those changes.

6. There are many ways to print "Hello, World" in assembly language and this certainly isn't the shortest way. If you click examples and browse c:\emu8086\examples, you will find **HelloWorld.asm** which assembles into only a 30-byte executable. Unlike the previous example which carries out each step by itself, this one is much smaller because it uses a built-in interrupt function of the operating system to write to the display.

*** The integrated 8086 assembler can generate console programs that can be executed on any computer that runs x86 machine code (Intel/AMD architecture). The architecture of the 8086 Intel microprocessor is called "Von Neumann architecture" after the mathematician who conceived of the design. NOTE: A CPU can interpret the contents of memory as either instructions or data; there's no difference in the individual bytes of memory, only the way in which they're arranged. Because of this, it's even possible for programs to re-write their own instructions, then execute the instructions they've changed.

7. Start working with the Experimental work, step by step, until you finish the required tasks. Also, solve the required exercises and make your final conclusions about the experiment.

8. Finally, you should write your technical lab report for this experiment and the reports will be submitted by each group before the deadline specified by the instructor.

## VI. Experimental Work:

Using the acquired knowledge about INTEL 8086, Emu8086 software and the tutorial given by the instructor, perform the following tasks:

1. Once the program is open, choose **emulate** from the main window of EMU8086. Then describe the new displayed windows.

……..…….....
……..…….....
……..…….....
……..…….....

2. After this, record the values stored in the **registers** and the **starting memory address**. Explain the results for each register and for the memory address also.

……..…….....
……..…….....
……..…….....
……..…….....
……..…….....
……..…….....

3. Now, check the **logical address** for each **instruction** in the program. Are all instructions following each other in address sequence? Explain.

……..…….....
……..…….....

4. Now, for each instruction, check the **number of bytes** needed to store it in the memory. Do all instructions need the same number of bytes? Explain.

……..…….....
……..…….....
……..…….....

5. From the emulator window, check the functions of **SCREEN, SOURCE AND RESET** options.

……..…….....
……..…….....
……..…….....

6. Now, check the contents of memory locations: **07012, 07122, 07173**. How did you got these results? (Hint: use the aux option from the emulator window).

……..…….....
……..…….....
……..…….....

7. Check the values of **variables** used (if any) in the program. Explain.

……..…….....
……..…….....

8. Check the contents of the **Stack** segment used in this program. At what segment does the **SS register** point? What is the first logical and physical address of the first location in this stack segment?

……..…….....
……..…….....
……..…….....

9. Check and Analyze the **Flag bits** in the status register.

……..…….....
……..…….....

10. Now, **Run** the program from the emulator window then describe the output you got.

……..…….....
……..…….....

11. After this, record the values stored in the **registers** and the **ending memory address**. Explain how these results were obtained.

……..…….....
……..…….....
……..…….....
……..…….....

12. Now, check the contents of memory locations: **07012, 07122, 07173**. Explain the results?

……..…….....
……..…….....
……..…….....

13. Check the contents of the **Stack** segment used in this program.

……..…….....
……..…….....

14. Check and Analyze the **Flag bits** in the status register.

……..…….....
……..…….....
……..…….....

15. **Stop** the program, then **reload** it again from the emulator window. Explain the behavior of the emulator window.

……..…….....
……..…….....

16. Now, close the emulator window and get back to the main window (the source code window), Explain the behavior of output if we change the value in the following instruction: **MOV [di], 11101100b ➔ 00111111b** ? Explain the role of this instruction and what are the possible values that can be used with it.

……..…….....
……..…….....
……..…….....

17. Now, Use the source code and modify the program to **print the message "I ♥ microprocessors"** on the screen instead of Hello World!. Explain how did you do that?

……
……

18. **Save** the new program by the name "Exp_2" in your folder on the desktop. What is the **file extension** that you used to save this program?

……
……

19. Now, after you **run** the program, **Press Any Key** to exit. What happens? Then, stop the program and go back to the source code screen, **disable** the last instruction in the program and re-run it. Now Press Any Key again to Exit. What happens? Explain your answer.
(Hint: You can disable any instruction in your program by simply adding **;** before instruction.)

……
……
……
……

20. Go back to source code window: in the **second line** of this program, what will happen if we remove the **semicolon (;)** at the beginning of the line and then emulate the program? Explain.

……
……
……

……

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 3: Understanding bit & arithmetic manipulation using 8086 Intel MP: Programming-based experiment for MDA-win8086 microprocessor kit and emu8086 emulator

## I. Objective:

The purpose of this experiment is to learn you how to assemble, link, download, run, debug and write an ALP that utilize the Intel 8086 MP to perform arithmetic and logic operations using the training kit MDA-Win8086 and using the emulator software EMU8086.

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

**Introduction to 8086 Programming:** Learning any programming language involves mastering a number of common concepts:

- **Variables**: declaration/definition
- **Assignment**: assigning values to variables
- **Input/output**: Displaying messages, Displaying variable values
- **Control flow**: if-then , Loops
- **Subprograms**: Definition and Usage

Programming in assembly language involves mastering these concepts and a few other issues.
**Variables:** For the moment we will skip details of variable declaration and simply use the 8086 registers as the variables in our programs. Registers have predefined names and do not need to be declared. We have seen that the 8086 has 14 registers. Initially, we will use four of them – the so called the general purpose registers: **AX, BX, CX, DX.** These 16-bit registers can also be treated as eight 8-bit registers: **AH, AL, BH, BL, CH, CL, DH, DL.**
**Assignment**: In C++, assignment takes the form:

```
x = 42 ;
y = 24;
z = x + y;
```

In assembly language we carry out the same operation but we use an instruction to denote the assignment operator ("=" in C++).

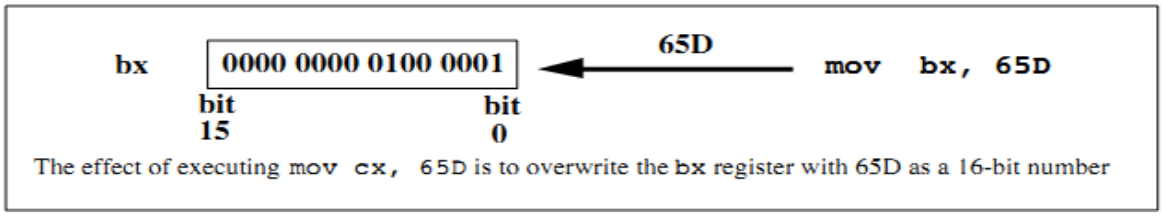
```
MOV X, 42
MOV Y, 24
ADD Z, X
ADD Z, Y
```

The **MOV** instruction carries out assignment in 8086 assembly language. It allows us place a number in a register or in a memory location (a variable) i.e. it assigns a value to a register or variable. For Example: Store the **ASCII** code for the letter **A** in register **BX**.

**A has ASCII code 65d (01000001B, 41h)**

The following MOV instruction carries out the task: **MOV BX, 65d**. We could also write it as: **MOV BX, 41h**, or **MOV BX, 01000001B**, or **MOV BX, 'A'**

All of the above are equivalent. They each carry out exactly the same task, namely the binary number representing the **ASCII** code of A is copied into the **BX** register.




The value is copied into the right-hand side (low-order byte) of the register. The left-hand side will contain all 0's. Thus, we could also have written it as:

**MOV BL, 65d**
**MOV BL, 'A'**

Since the register **BL** represents the low-order byte of register **BX.**
Note: The 8086 Assembler converts a character constant i.e. a character in single quotes (e.g. 'A') to its ASCII code automatically. This is a very useful feature and it means that you can specify many characters without having to look up their ASCII code. You simply enclose the character in single quotes. You will have to use the ASCII code for control characters such as carriage return and line feed.

**Notation**: MOV is one of the many 8086 instructions that we will be using. Most assembly language books use uppercase letters to refer to an instruction e.g. MOV. However, the assembler will also recognize the instruction if it is written in lowercase or in mixed case e.g. MOV. (In fact, the assembler converts all instructions to uppercase).

The MOV instruction also allows you to copy the contents of one register into another register. For Example: **MOV BX, 2**
**MOV CX, BX**

The first instruction loads the value 2 into BX where it is stored as a binary number. [a number such as 2 is called an integer constant].

The **MOV** instruction takes two operands, representing the destination where data is to be placed and the source of that data.

General Form of **MOV** Instruction: **MOV destination, source**.

Where destination must be either a register or memory location and source may be a constant, another register or a memory location. In 8086 assembly language, the source and destination cannot both be memory locations in the same instruction.

Note: The comma is essential. It is used to separate the two operands. A missing comma is a common syntax error.

More Examples: The following instructions result in registers AX, BX, CX all having the value 4:

**MOV BX, 4** ; copy number 4 into register BX
**MOV AX, BX** ; copy contents of BX into register AX
**MOV CX, AX** ; copy contents of AX into register CX

**Comments:** Anything that follows semi-colon **(;)** is ignored by the assembler. It is called a comment. Comments are used to make your programs readable. You use them to explain what you are doing in English. It is recommended that you use comments frequently in your programs, not only so that others can understand them, but also for yourself, when you look back at programs you have previously written. Every programming language has a facility for defining comments.

**More 8086 Instructions:** ADD, INC, DEC and SUB instructions. The 8086 provides a variety of arithmetic instructions. For the moment, we only consider a few of them. To carry out arithmetic such as addition or subtraction, you use the appropriate instruction. In assembly language you can only carry out a single arithmetic operation at a time. This means that if you wish to evaluate an expression such as:

**z = x + y + w – v**

You will have to use 3 assembly language instructions – one for each arithmetic operation. These instructions combine assignment with the arithmetic operation. For Example:

```
MOV AX, 5   ; load 5 into AX
ADD AX, 3   ; add 3 to the contents of AX,
            ;AX now contains 8
INC AX      ; add 1 to AX
            ; AX now contains 9
DEC AX      ; subtract 1 from AX
            ; AX now contains 8
SUB AX, 6   ; subtract 4 from AX
            ; AX now contains 2
```

The **ADD** instruction adds the source operand to the destination operand, leaving the result in the destination operand. The destination operand is always the first operand in 8086 assembly language. (In M68000 assembly language, it is the other way round i.e. the source operand is always the first operand e.g. move #10, x).

The **INC** instruction takes one operand and adds 1 to it. It is provided because of the frequency of adding 1 to an operand in programming.

The **DEC** instruction like **INC** takes one operand and subtracts 1 from it. This is also a frequent operation in programming.

The **SUB** instruction subtracts the source operand from the destination operand leaving the result in the destination operand.

Some microprocessors do not provide instructions for multiplication or division (e.g. the M6800). With such microprocessors, multiplication and division have to be programmed using repeated additions and subtractions and shift operations. The 8086 provides **MUL** and **DIV** (and others) for multiplication and division.

**Ambiguity**: Suppose you wish to load the hexadecimal value A (decimal 10) written as AH in the register BL. You might be tempted to write: **MOV BL, AH.**

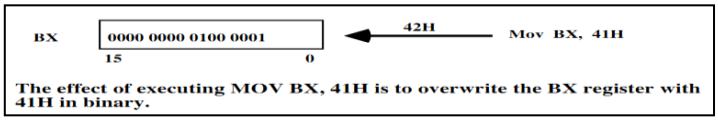
But we have already seen that there is a register called **AH** (the high-order byte of **AX**) and so the above does not do what we intend. Instead it copies the contents of register **AH** into **BL**. In order to avoid ambiguity when writing hexadecimal numbers that begin with a letter we can write like this: **MOV BL, 0Ah** or **MOV BL, 0xA** ; copy hex number ah into **BX**.

It is common practice to write decimal numbers with the letter D appended so as to distinguish them from hexadecimal. The 8086 assembler take all numbers to be decimal numbers unless there is a B (binary), H (hex) or O (octal) appended to them.

**Note**: When data is moved to a register, all 16 bits (or 8 bits) are given a value. The assembler will automatically fill in 0's on the left-hand side. For Example:

**MOV BX, 42H ; COPY 42 HEX INTO BX**

42H is 100 0001 in binary. This padded out with nine 0-bits on the left-hand side to fill all 16-bits of the register.



The effect of executing MOV BX, 41H is to overwrite the BX register with 41H in binary.

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

Working procedure (methodology) throughout this experiment consists of the following steps:

- Start by reading the experiment environment, main objective and procedure.
- Connect PC and MDA-Win 8086 Microprocessor Kit using the Data cable provided.
- Switch 'ON' the MDA-Win 8086 Microprocessor Kit and the Computer, and go to Local Disk C: of the PC and Open the MDA folder and double click the ' MDA_Win IDE8086 ' Icon and start the MDA- Win 8086 Software Tool.
- 'Reset' MDA-Win 8086 Microprocessor Kit by pressing RESET button provided on the kit.

- Start working with the Experimental work step by step until you finish the required tasks. Also, Solve the required exercises and make your final conclusions about the experiment.
- Finally, you should write your technical lab report for this experiment and the reports will be submitted by each group before the deadline specified by the instructor.

## VI. Experimental Work:

1. Use MDA8086-IDE software to locate, open the ALP in the source file "BIT.ASM". Now assemble/link and run the program. Record the contents of AX, BX, FL registers.

2. Run the program in trace mode and record the values of the registers AX, BX, FL. Describe the results you get. For any new instruction, review the data sheet or consult emu8086's help for the instruction to understand its functionality.

3. Provide detailed description of the functionality of each instruction.

4. Now, reset the KIT and again use MDA8086-IDE software to locate, open the source file "BIT_2.ASM". Assemble/link and run the program and record the contents of AX, BX, FL registers.

5. Run the program in trace mode and record the values of the registers AX, BX, CX, FL, SI, Memory locations [02000], …, [02020]. Describe the results you get. For any new instruction, review the data sheet or emu8086 help for the instruction to understand its functionality.

6. Provide detailed description of the functionality of each instruction.

7. Now, reset the KIT and again use MDA8086-IDE software to locate, open and run ALP in the source file "ARITH.ASM". Assemble/link the program and record the contents of AX, BX, DX, FL registers.

8. Run the program again then record the contents of AX, BX, DX, FL registers and describe the results you get.

9. Provide detailed description of the functionality of each instruction.

10. Now, reset the KIT and again use MDA8086-IDE software to locate, open the ALP in the source file "ARITH_2.ASM".Assemble/link and run the program, then record the contents of AX, BX, DX, FL registers.

11. Run the program again then record the contents of AX, BX, DX, FL registers and describe the results you get.

12. Provide the detailed description of the functionality of each instruction.

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 4: Using and verifying addressing modes for 8086 Intel microprocessor

## I. Objective:

The purpose of this experiment is to learn how to assemble, link, download, debug, trace, emulate, run and write an ALP that uses the addressing modes for Intel 8086 MP using the training kit MDA-Win8086 and using the emulator software EMU8086. Also, you should be able to classify and describe the addressing modes used in Intel 8086 MP and select the appropriate addressing mode to accomplish simple data transfer operation.

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

### 1. Addressing Modes:

Addressing modes are an aspect of the instruction set architecture in most central processing unit (CPU) designs. The various addressing modes that are defined in a given instruction set architecture define how machine language instructions in that architecture identify the operand (or operands) of each instruction. An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere. In computer programming, addressing modes are primarily of interest to compiler writers and to those who write code directly in assembly language.

The 8086 provides various addressing modes to access instruction operands. Operands may be contained in registers, within the instruction op-code, in memory, or in I/O ports.

The 8086 has 12 addressing modes, which can be classified into five groups:

- Register and immediate modes (2 modes): Register Mode, Immediate Mode.
- Memory addressing modes (6 modes) : Memory Direct Addressing, Register Indirect Addressing, Based Addressing, Indexed Addressing, Based Indexed Addressing, String Addressing.
- Port addressing mode (2 modes): Direct port mode, Indirect port mode.
- Relative addressing mode (1 mode).
- Implied addressing mode (1 mode).

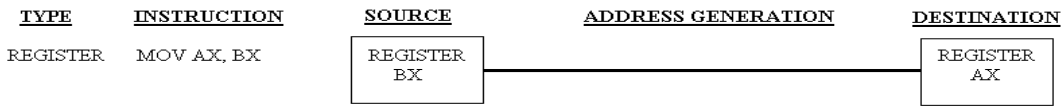



Figure 1: Example of 8086 Register Addressing Mode

## 2. Writing ALP for MDA-Win8086:

To write an Assembly Language Program (ALP) and run it over MDA-Win8086 kit, you should choose **New** from the **file** menu. You can always use the following template: (you should also make your program well-commented)

```
;*****************************************
;    MDA-Win8086 EXPERIMENT PROGRAM*
;    FILENAME  : ADDRESSING_MODES.ASM
;    PROCESSOR : I8086
;    By Student : Your Name Here
;*****************************************
CODE  SEGMENT
ASSUME        CS:CODE,DS:CODE,ES:CODE,SS:CODE
          ;
ORG     1000H
          ;
****Add Your Code Here ****
          ;
CODE  ENDS
END
```

## 3. Writing ALP for EMU8086:

To write an Assembly Language Program (ALP) and run it over EMU8086 Emulator Software, choose **New** from the **file** menu, and then choose the **COM** template. You can always use the same template we described above for MDA-Win8086 or you can use the following template: (make sure that your program well-commented)

```
;*****************************************
;    EMU8086 EXPERIMENT PROGRAM*
;    FILENAME  : ADDRESSING_MODES.ASM
;    PROCESSOR : I8086
;    By Student : Your Name Here
;*****************************************
ORG 100H
;
****Add Your Code Here ****
;
RET
```

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use Emu_8086' software icon as a tool to write and debug assembly programs for the 8086 Microprocessor.

## V. Procedure:

Working procedure (methodology) throughout this experiment consists of the following steps:

- Start by reading the experiment environment, main objective and procedure.
- Locate EMU8086 Icon on the PC. Open a new file.
- Start working with the Experimental work step by step until you finish the required tasks. Also, solve the required exercises and make your final conclusions about the experiment.
- Finally, you should write your technical lab report for this experiment and the reports will be submitted by each group before the deadline specified by the instructor.

## VI. Experimental Work:

1. Open EMU8086 software, make a new file, name it as "**AD_MODES.ASM**" and prepare the file to write an ALP (use the template).

2. Now, store the values: **1234H, 5678H, 1000H, 2000H, 3000H** into registers: **AX, BX, CX, DX**, and **BP** respectively. What type of addressing mode is used to accomplish this? What is the size of the used data: 8- or 16 bit? What type of instruction formats is used here?

3. Copy the contents of **DX**, **AX**, **BX** to the registers **DS**, **SI**, **DI**. What type of addressing mode is used to accomplish this?

4. Check the memory location **1234H** and then copy its contents to the register **AX**. What type of addressing mode is used to accomplish this?

5. Use Register Indirect Memory Addressing to load the CX by the contents of memory located at BX. What is the physical address to access this memory location and how did you get it?

6. Now, follow your program by writing an appropriate instruction to do each of the following:

- **XOR the contents of CX register with word at DS:E873.**
- **Store the word at DS:BX onto stack.**
- **Store the word at SS:(BP-4) into AX**
- **Subtract the contents of register BX from word at DS:(SI+2)**
- **Invert bits of byte at SS:(BP+DI)**
- **Add the contents of register DX to word at DS:(BX+SI+2)**
- **Increment the 8-bit contents of Mem Loc. in DS with offset START by 1.**
- **Move the contents of Port 02 to AL.**
- **Move the contents of Port number taken from DX to AL.**

7. In a tabulated form; provide a detailed description of the functionality of each instruction used in part 6 and determine the type of addressing mode for each step.

8. Run your program then record the registers and memory location used in your program.

9. Now, what if we use the command "NOP" after each instruction of the above program. Re-run your program. Did you notice any difference?

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 5: memory & stack access operations for 8086 Intel microprocessor: memory, stack operations & use several programming components such as subroutines

## I. Objective:

The purpose of this experiment is to learn how to assemble, link, download, debug, trace, emulate, run and write an ALP that uses the memory access operations for Intel 8086 MP using the training kit MDA-Win8086 and using the emulator software EMU8086. Also, you should be able to understand and convert the pseudo code program to ALP that can be used in Intel 8086 Microprocessor. Moreover, you learn how to use 8086 stack to read data or write data.

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

### 1. 8086 Memory:

8086 has 20 bit address bus which means, it can address up to $2^{20}$ = 1MB memory locations. These memory locations can be accessed by MP through Word/Byte read or write operations.

### 2. Word Read (or Write):

Each of 1 MB memory address of 8086 represents a byte wide location where a 16bit words will be stored in two consecutive Memory location.

If the first byte of the data is stored at an even address, 8086 can read the entire word in one operation. For example, if the 16 bit data is stored at the address 00520H is 2607:

MOV BX, [00520]

8086 reads the first byte and stores the data in BL and reads the $2^{nd}$ byte and stores the data in BH

BL ← (00520)
BH ← (00521)

If the first byte of the data is stored at an ODD address, 8086 needs two operation to read the 16 bit data. For example if the 16 bit data is stored at the address 00521H is F520:

MOV BX, [00521]

In the first operation, 8086 reads the 16 bit data from the 00520 location and stores the data of 00521 location in register BL and discards the data of 00520 location.

In 2nd operation, 8086 reads the 16 bit data from the 00522 location and stores the data of 00522 location in register BH and discards the data of 00523 location

BL← (00521)
BH← (00522)

### 3. Byte Read (or Write):

For example: MOV BH, [Addr], there are also 2 cases here:

- For Even Address: such as MOV BH, [00520]

  8086 reads the first byte from 00520 location and stores the data in BH and reads the 2nd byte from the 00521 location and ignores it BH ← [00520]

- For Odd Address: such as MOV BH, [00521]

  8086 reads the first byte from 00520 location and ignores it and reads the 2nd byte from the 00521 location and stores the data in BH: BH ←[00521]

### 4. Data transfer between the CPU and the memory:

- Memory Write:
  - Byte Transfer: move BYTEPTR DS : [SI], 37H
  - Word Transfer: move WORDPTR DS : [SI], 1237H
- Memory Read:
  - Byte Transfer: move al, BYTEPTR DS : [SI]
  - Transfers data from the physical memory address calculated using DS and [SI] to register AL ( Lower byte of AX Register)
- Word Transfer: move ax, WORDPTR DS : [SI]
  - Transfers data from the physical memory address calculated using DS and [SI] to register AL ( Lower byte of AX Register) and the next byte from the next memory location calculated as DS:[SI +1] is transferred to AH ( Higher byte of AX Register)

### 5. Memory operation through AX Register:

- Write:
  - MOV AX , 1234H
  - MOV WORDPTR DS: [SI], AX
  - Ds: 0000H & SI: 0500H
  - Physical Address: 00000+0500= 00500 H
  - The instruction transfers: 34 ← 00500H & 12 ←00501H
- Read:
  - MOV AX, WORDPTR DS: [SI]
  - Ds: 0000H & SI: 0500H
  - Physical Address: 00000+0500= 00500 H
  - The instruction transfers: AL _ (00500) & AH _ (00501)

### 6. 8086 Stack:

Each 8086 stack segment is 64K bytes long & is organized as 32K 16-bit words. The lowest byte (valid data) of the stack is pointed to by the 20-bit physical address computed from the current SP and SS. This is the lowest memory location in the stack (Top of Stack) where data is pushed. The 8086 PUSH & POP instructions always utilize 16-bit words. Therefore, stack locations should be configured at even addresses in order to minimize the number of memory cycles for efficient stack operations. 8086 can have several stack segments; however, only one stack segment is active at a time.

Since 8086 uses 16-bit data for PUSH & POP operations from the top of stack, 8086 PUSH instruction first decrements SP by 2 then the 16-bit data is written onto the stack. Therefore, 8086 stack grows from high to low memory addresses of the stack. Also, when a 16-bit data is popped from the top of stack using 8086 POP instruction, 8086 reads 16-bit data from the stack into the specified register or memory, 8086 then increments SP by 2. Note that the 20-bit physical address computed from SP and SS always points to the last data pushed onto the stack. One can save and restore flags in the 8086 using PUSHF and POPF instructions. Memory locations can also be saved and restored using PUSH and POP instructions without using any 8086 registers. Finally, One must POP registers in the reverse order in which they are PUSHed.

| For example, if registers BX, DX, & SI are PUSHed using: | Then the registers must be POPped using: |
|---|---|
| PUSH BX<br>PUSH DX<br>PUSH SI | POP SI<br>POP DX<br>POP BX |

### 7. How to define the new subroutine (Procedure):

```
PROCEDURE_NAME      PROC
                    //YOUR CODE HERE
RET
```

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

Working procedure (methodology) throughout this experiment consists of the following steps:

- Start by reading the experiment environment, main objective and procedure.
- Connect PC and MDA-Win 8086 Microprocessor Kit using the Data cable provided.

- Switch 'ON' MDA-Win 8086 Kit and the Computer, and go to Local Disk C: of the PC and Open the MDA folder and double click the ' MDA_Win IDE8086 ' Icon and start the MDA- Win 8086 Software Tool.
- 'Reset' MDA-Win 8086 Microprocessor Kit by pressing the RESET button provided on the kit.
- Start working with the Experimental work step by step until you finish the required tasks. Also, Solve the required exercises and make your final conclusions about the experiment.
- Finally, you should write your technical lab report for this experiment and the reports will be submitted by each group at the deadline specified by the instructor.

## VI. Experimental Work:

1. Open emu8086 software, create a new file, name it as "MEMSTACK" and prepare the file to write an ALP (use the template provided from the previous labs).

……
……
……
……

2. Now, clear the stack segment register and let stack pointer to point to the offset value 0540H.

……...
……

3. Now, load registers AX, BX, CX, DX with the values: 1234H, 5678H, 3ABCH, 0A0BH respectively.

……...
……...
……...
……...

4. Now, save these registers into the stack segment pointed to in your program (use the same order).

……...
……...
……...
……...

5. Then, exchange the registers contents in the following order: AX:BX, BX:CX, CX:DX, and DX: AX.

……...
……...
……...
……...

6. Now, save the new contents of the registers to memory locations starting at 30000H.

……...

……...

……...

7. Read the contents of the registers from the stack where you saved them in part 4.

……...

……...

……...

……

8. Run your program to check the contents of all registers, stack locations (used here) and memory locations 30000H….30007H for each instruction used in your program. Describe the results you get.

**Intel 8086 Register File Trace Table**

| Trace | AX | BX | CX | DX | CS | IP | SS | SP | BP | SI | DI | DS | ES | FL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T_1$ | | | | | | | | | | | | | | |
| $T_2$ | | | | | | | | | | | | | | |
| $T_3$ | | | | | | | | | | | | | | |
| $T_4$ | | | | | | | | | | | | | | |
| $T_5$ | | | | | | | | | | | | | | |
| $T_6$ | | | | | | | | | | | | | | |
| $T_7$ | | | | | | | | | | | | | | |
| $T_8$ | | | | | | | | | | | | | | |
| $T_9$ | | | | | | | | | | | | | | |
| $T_{10}$ | | | | | | | | | | | | | | |

**Memory Locations Trace Table**

| Trace | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T_1$ | | | | | | | | | | | | | | | | | | | | |
| $T_2$ | | | | | | | | | | | | | | | | | | | | |
| $T_3$ | | | | | | | | | | | | | | | | | | | | |
| $T_4$ | | | | | | | | | | | | | | | | | | | | |
| $T_5$ | | | | | | | | | | | | | | | | | | | | |
| $T_6$ | | | | | | | | | | | | | | | | | | | | |
| $T_7$ | | | | | | | | | | | | | | | | | | | | |
| $T_8$ | | | | | | | | | | | | | | | | | | | | |
| $T_9$ | | | | | | | | | | | | | | | | | | | | |
| $T_{10}$ | | | | | | | | | | | | | | | | | | | | |

9. Now, go back to your program and after the end of your program, define a procedure "P1", to XOR the contents of BX with itself and the contents of CX with itself.

……

……

……

……

……

10. Now, call this subroutine after you have read the stack in part 7. Describe the results of this subroutine and how IP changes its value.

……

……

……

……

……

11. Run your program to check the contents of the registers, stack and the memory locations. Describe the results you get.

……

……

……

……

……

12. Now, write another subroutine to add the memory location 30000H with the contents of register AX. Store the results into memory location 30008H and into stack. Call this subroutine after part 3.

……

……

……

……

……

……

……

13. Now, in the MDA software, Click on **Open File** option and select the STACK_1.ASM source file and open the program file.

……

14. Run your program in trace mode to check the contents of all registers for each instruction you used in your program. Describe the results you get.

**Intel 8086 Register File Trace Table**

| Trace | AX | BX | CX | DX | CS | IP | SS | SP | BP | SI | DI | DS | ES | FL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T_1$ | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **T 2** | | | | | | | | | | | | | | |
| **T 3** | | | | | | | | | | | | | | |
| **T 4** | | | | | | | | | | | | | | |
| **T 5** | | | | | | | | | | | | | | |
| **T 6** | | | | | | | | | | | | | | |
| **T 7** | | | | | | | | | | | | | | |
| **T 8** | | | | | | | | | | | | | | |
| **T 9** | | | | | | | | | | | | | | |
| **T 10** | | | | | | | | | | | | | | |

**Memory Locations Trace Table**

| **Trac** | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **T 1** | | | | | | | | | | | | | | | | | | | |
| **T 2** | | | | | | | | | | | | | | | | | | | |
| **T 3** | | | | | | | | | | | | | | | | | | | |
| **T 4** | | | | | | | | | | | | | | | | | | | |
| **T 5** | | | | | | | | | | | | | | | | | | | |
| **T 6** | | | | | | | | | | | | | | | | | | | |
| **T 7** | | | | | | | | | | | | | | | | | | | |
| **T 8** | | | | | | | | | | | | | | | | | | | |
| **T 9** | | | | | | | | | | | | | | | | | | | |
| **T 10** | | | | | | | | | | | | | | | | | | | |

15. Provide detailed description of the functionality of each instruction you used in the program also determine the type of addressing mode used for each step.

……
……
……
……
……
……
……

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 6: Interruption Techniques for Intel 8086 Microprocessor Programming : Understanding software Interrupt techniques for Intel 8086 MP

## I. Objective:

The purpose of this experiment is to familiarize with interrupt concepts and its major types for Intel 8086.

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

Interrupts provide a mechanism of transferring control from a foreground process (the current executing program) to an Interrupt Service Routine. When such a transfer is initiated by the hardware in response to special internal or external conditions, a hardware interrupt is said to have occurred. External hardware interrupts are generated by peripheral devices and are the main mechanism used by these devices to get the attention of the processor. Certain external hardware interrupts are maskable in that they may be disabled by clearing the Interrupt enable flag (IF) in the flags register. External hardware interrupts that cannot be disabled by clearing IF are called non-maskable interrupts. Typically, non-maskable interrupts are hardware events that must be responded to immediately by the CPU. An example of such an event is the occurrence of a memory or I/O parity error.

In this lab, we will focus on software interrupts, mainly int21h that is used to perform different actions on a computer, the nature of this action depends on the value stored inside the AH registry. Therefore, int21h can be translated to the following algorithm

```
Call int21h
AH=read_registry_contents(AH)
if AH==0 {do action 1}
if AH==1 {do action 2}
….

if AH==99 {do action 99}
```

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows

environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

Working procedure (methodology) throughout this experiment consists of the following steps:

- Start by reading the experiment environment, main objective and procedure.
- Turn On your PC, then Click on Emu_8086 Icon in order to start working in the emulated environment for Intel 8086 Microprocessor.
- Use the emu8086 help to explore different available interruptions
- Start working with the Experimental work step by step until you finish the required tasks. Also, Solve the required exercises and make your final conclusions about the experiment.
- Finally, you should write your technical lab report for this experiment and the reports will be submitted by each group before the deadline specified by the instructor.

## VI. Experimental Work:

1. Find and interruption that displays a character, give an example
2. Which registries are involved with this interruption? What are their roles?
3. Find and interruption that prompts the user to type a character, but without displaying it, give an example
4. Which registries are involved with this interruption? What are their roles?
5. Find an interruption that combines the features of the previous two interrupts, give an example
6. Which registries are involved with this interruption? What are their roles?
7. Find and interruption that displays a string, give an example that displays “online learning”
8. Which registries are involved with this interruption? What are their roles? How to recognize the end of the string?
9. Find and interruption that can assume the role of the RET instruction

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 7: Assembly optimization techniques: display the system time application

## I. Objective:

The purpose of this experiment is to develop an application to display the system time and refresh it every second. The program will be built using various interrupts. The developed ALP is a non-optimized version. The second objective consists of optimizing the program to achieve the same result using fewer lines of codes and leads to a smaller program

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

**Various useful interrupts (int 21h):**
01h: read a character and display it, the data will be stored in the AL registry
02h: display the character which ASCII codes correspond to the DL register contents
09h: display a string, the start is stored in the DX register and the end is marked by the **$** symbol

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

Working procedure (methodology) throughout this experiment consists of the following steps:

- Start by reading the experiment environment, main objective and procedure.
- Open the emu8086 software.
- Start working with the Experimental work along with the instructor and make your final conclusions about the experiment.
- Finally, you should write your technical lab report for this experiment and the reports will be submitted by each **student** before the deadline specified by the instructor.

## VI. Experimental Work:

Build the program little by little based on the instructor's tutorial. Make a note of every step since it will be included to the report.

In the report, write the program at every step and explain the new changes in the program, especially when optimizing the ALP.

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 8: running & validating the operation of led and 7-segment display: parallel input output (PIO) interface (8255a interface) with Intel 8086 µp

## I. Objective:

The purpose of this experiment is to learn how Parallel Peripheral Interface (PPI)-Intel 8255 works and its importance for I/O. Students will also how to run and validate the operation of LED and 7-Segment display. You should be able to understand how LEDs and 7-Segment Display (As an I/O devices) are interfaced to Intel 8086 through the Parallel Peripheral Interface (PPI) Intel 8255.

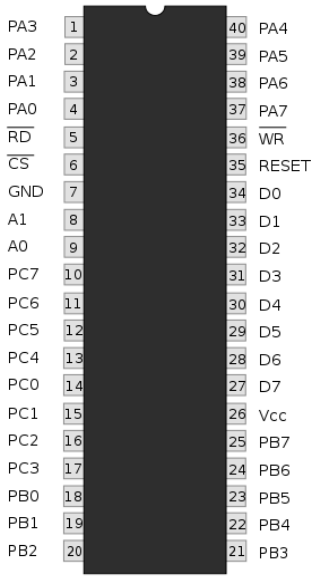
## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

### 1. INTEL 8255 PPI:

The Intel 8255 (or i8255) Programmable Peripheral Interface chip is a peripheral chip originally developed for the Intel 8085 microprocessor, and as such is a member of a large array of such chips, known as the MCS-85 Family. This chip was later also used with the Intel 8086 and its descendants. It was later made (cloned) by many other manufacturers. It is made in DIP 40 and PLCC 44 pins encapsulated versions.




This chip is used to give the CPU access to programmable parallel I/O, and is similar to other such chips like Motorola 6520 PIA (Peripheral Interface Adapter) MOS Technology 6522 (Versatile Interface Adapter) and MOS Technology CIA (Complex Interface Adapter) all developed for the 6502 family. However, most often the functionality 8255 offered is now not implemented with the 8255 chip itself anymore, but is embedded in a larger VLSI chip as a sub function. 8255 chip itself is still made and is sometimes used together with a microcontroller to expand its I/O capabilities.

### 2. FUNCTIONAL BLOCK OF 8255:

The 8255 has 24 input/output pins in all. These are divided into three 8-bit ports. Port A and port B can be used as 8-bit input/output ports. Port C can be used as an 8-bit input/output port or as two 4-bit input/output ports or to produce handshake signals for ports A and B. The three ports are further grouped as follows:

- Group A consisting of port A and upper part of port C.
- Group B consisting of port B and lower part of port C.

Eight data lines (D0 - D7) are available (with an 8-bit data buffer) to read/write data into the ports or control register under the status of the "RD" (pin 5) and "WR" (pin 36), which are active low signals for read and write operations respectively. The address lines A1 and A0 allow to successively access any one of the ports or the control register as listed below:

| A1 | A0 | Function |
|---|---|---|
| 0 | 0 | port A |
| 0 | 1 | port B |
| 1 | 0 | port C |
| 1 | 1 | control register |

The control signal "'¬CS" (pin 6) is used to enable the 8255 chip. It is an active low signal, i.e., when ¬CS = '0' , the 8255 is enabled. The RESET input (pin 35) is connected to a system (like 8085, 8086, etc. ) reset line so that when the system is reset, all the ports are initialized as input lines. This is done to prevent 8255 and/or any peripheral connected to it, from being destroyed due to mismatch of ports. This is explained as follows. Suppose an input device is connected to 8255 at port A. If from the previous operation, port A is initialized as an output port and if 8255 is not reset before using the current configuration, then there is a possibility of damage of either the input device connected or 8255 or both since both 8255 and the device connected will be sending out data. The control register or the control logic or the command word register is an 8-bit register used to select the modes of operation and input/output designation of the ports.

## 3. OPERATIONAL MODES OF 8255:

There are two main operational modes of 8255: Input/output mode (I/O Mode) and Bit set/reset mode (BSR Mode).

### 1.Input/output mode

There are three types of the input/output mode which are as follows:

**Mode 0**

In this mode, the ports can be used for simple input/output operations without handshaking. If both port A and B are initialized in mode 0, the two halves of port C can be either used together as an additional 8-bit port, or they can be used as individual 4-bit ports. Since the two halves of port C are independent, they may be used such that one-half is initialized as an input port while the other half is initialized as an output port. The input/output features in mode 0 are as follows:

- O/p are latched.
- I/p are buffered not latched.
- Port do not have handshake or interrupt capability.

**Mode 1**

When we wish to use port A or port B for handshake (strobed) input or output operation, we initialize that port in mode 1 (port A and port B can be initialized to operate in different modes, i.e., for e.g., port A can operate in mode 0 and port B in mode 1). Some of the pins of port C function as handshake lines.

For port B in this mode (irrespective of whether is acting as an input port or output port), PC0, PC1 and PC2 pins function as handshake lines.

If port A is initialized as mode 1 input port, then, PC3, PC4 and PC5 function as handshake signals. Pins PC6 and PC7 are available for use as input/output lines.

The mode 1 which supports handshaking has following features:

- Two ports i.e. port A and B can be use as 8-bit i/o port.
- Each port uses three lines of port c as handshake signal and remaining two signals can be function as i/o port.
- Interrupt logic is supported.
- Input and Output data are latched.

**Mode 2**

Only group A can be initialized in this mode. Port A can be used for bidirectional handshake data transfer. This means that data can be input or output on the same eight lines (PA0 - PA7). Pins PC3 - PC7 are used as handshake lines for port A. The remaining pins of port C (PC0 - PC2) can be used as input/output lines if group B is initialized in mode 0. In this mode, the 8255 may be used to extend the system bus to a slave microprocessor or to transfer data bytes to and from a floppy disk controller.

### 2. Bit set/reset (BSR) mode

In this mode only port B can be used (as an output port). Each line of port C (PC0 - PC7) can be set/reset by suitably loading the command word register. No effect occurs in input-output mode. The individual bits of port c can be set or reset by sending the signal OUT instruction to control reg.

## 4. CONTROL WORD FORMAT

### 1. Input/output mode format

- The figure shows the control word format in the input/output mode. This mode is selected by making **D7 = '1'** .
- **D0, D1, D3, D4** are for lower port C, port B, upper port C and port A respectively. When D0 or D1 or D3 or D4 are *"SET"*, the corresponding ports act as input ports. For e.g., if D0 = D4 = '1', then lower port C and port A act as input ports. If these bits are *"RESET"*, then the corresponding ports act as output ports. For e.g., if D1 = D3 = '0', then port B and upper port C act as output ports.

| D6 | D5 | mode |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | x | 2 |

- **D2** is used for mode selection for group B (Port B and Lower Port C). When D2 = '0', mode 0 is selected and when D2 = '1', mode 1 is selected.
- **D5, D6** are used for mode selection for group A (Upper Port C and Port A). The format is as follows:

**Example:** If port B and upper port C have to be initialized as input ports and lower port C and port A as output ports (all in mode 0), what is the control word?

1. Since it is an input/output mode, **D7 = '1'**.
2. Mode selection bits, **D2, D5, D6 are all '0'** for mode 0 operation.
3. Port B should operate as input port, hence, **D1 = '1'**.
4. Upper port C should also be an input port, hence, **D3 = '1'**.
5. Port A has to operate output port, hence, **D4 = '0'**.
6. Lower port C should also operate as output port, hence, **D0 = '0'**.

Applying the corresponding values to the format in input/output mode, we get the control word **"8A"**.

**2. BSR mode format**

The figure shows the control word format in BSR mode. This mode is selected by making **D7='0'**.

- **D0** is used for bit set/reset. When D0= '1', the port C bit selected (*selection of a port C bit is shown in the next point*) is **SET**, when D0 = '0', the port C bit is **RESET**.
- **D1, D2, D3** are used to select a particular port C bit whose value may be altered using
- D0 bit as mentioned above. The selection of the port C bits are done as follows (table):
- *D4, D5, D6 are not used.*

| D3 | D2 | D1 | Bit/pin of port C selected |
|---|---|---|---|
| 0 | 0 | 0 | PC0 |
| 0 | 0 | 1 | PC1 |
| 0 | 1 | 0 | PC2 |
| 0 | 1 | 1 | PC3 |
| 1 | 0 | 0 | PC4 |
| 1 | 0 | 1 | PC5 |
| 1 | 1 | 0 | PC6 |
| 1 | 1 | 1 | PC7 |

**Example:** If the 5th bit (PC5) of port C has to be "SET", then what is the control word?

1. Since it is BSR mode, D7 = '0'.
2. Since D4, D5, D6 are not used, assume them to be '0'.
3. PC5 has to be selected, hence, D3 = '1', D2 = '0', D1 = '1'.
4. PC5 has to be set, hence, D0 = '1'.

Applying the above values to the format for BSR mode, we get the control word as "**0B (hex)**".

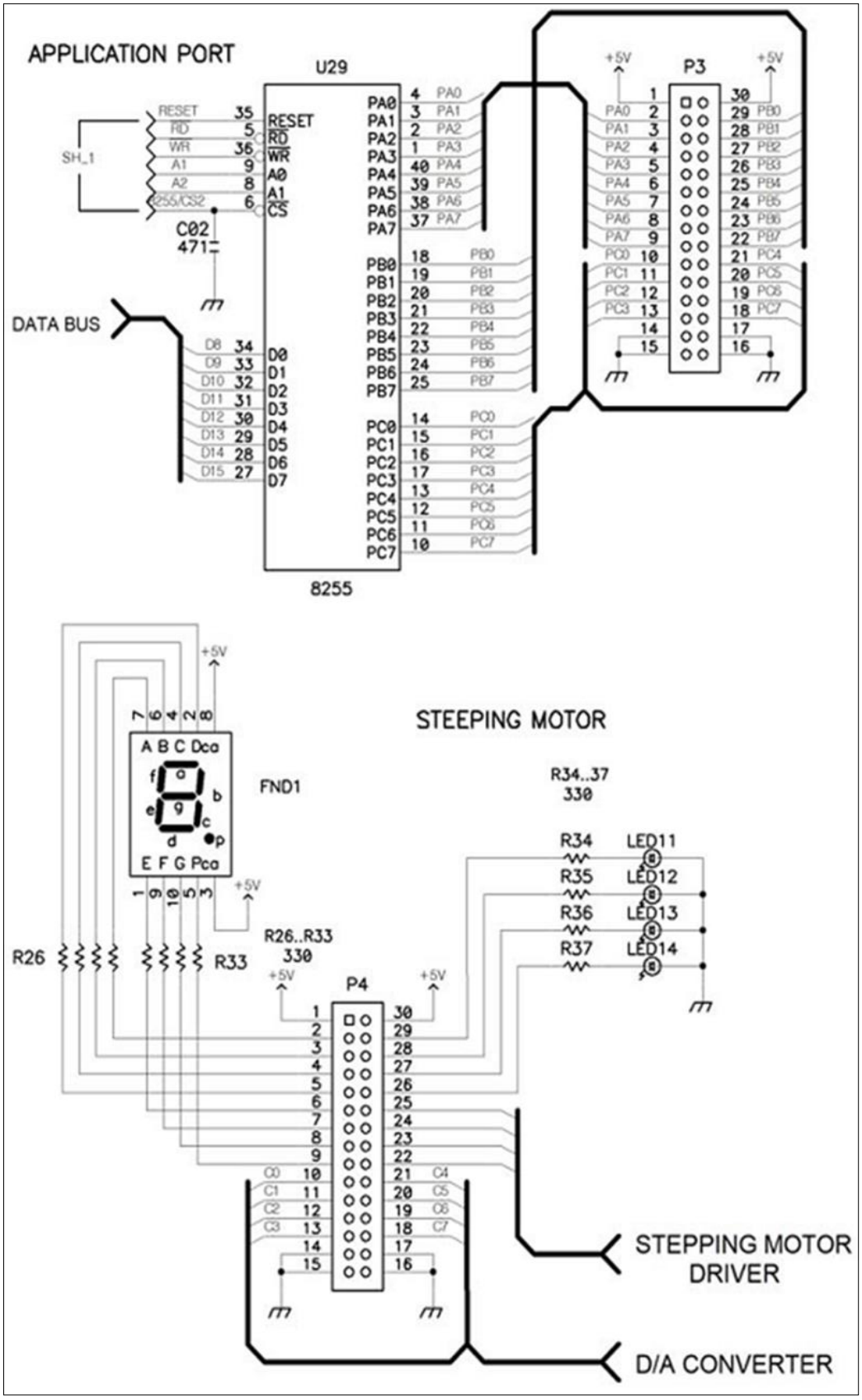



Figure 1: 8255 Interface

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

Working procedure (methodology) throughout this experiment consists of the following steps:

- Start by reading the experiment environment, main objective and procedure.
- Connect PC and MDA-Win 8086 Microprocessor Kit using the Data cable provided.
- Switch 'ON' the MDA-Win 8086 Microprocessor Kit and the Computer, and go to Local Disk C: of the PC and Open the MDA folder and double click the ' MDA_Win IDE8086 ' Icon and start the MDA- Win 8086 Software Tool.
- 'Reset' MDA-Win 8086 Microprocessor Kit by pressing the RESET button from the kit.
- Start working with the Experimental work step by step until you finish the required tasks. Also, Solve the required exercises and make your final conclusions about the experiment.
- Finally, you should write your technical lab report for this experiment and the reports will be submitted by each group before the deadline specified by the instructor.

## VI. Experimental Work:

1. 'Reset' the MDA-Win 8086 Microprocessor Kit by pressing the RESET button provided on the kit. Then, Record the status of LED11, LED12, LED13, LED14 ® and the status of 7-Segment Display ®

……

2. Now, Create an ALP according to the default template and Save it the appropriate location. Use the name: "LED_1".

3. Define four variables in order to use them as a ports for the 8255:

```
PPIC_C    EQU   1FH
PPIC      EQU   1DH
PPIB      EQU   1BH
PPIA      EQU   19H
```

4. After this, you should initiate (output) the 8255 ports by the following values:

PPIC_C ← *(80)H* ; *value (80) H to the Control Register of 8255A chipset.*

PPIA ← *(FF)H* ; *value (FF) H to Port A of the 8255A chipset* (this port is for 7-SD interfacing)

PPIB ←(F1)H ; *value (F1)H to Port B of the 8255A chipset* (this port is for LEDs interfacing)

5. Now, turn on the 7-segment by outputting the value **(A4)H** to the appropriate port that interfaced with 7-segment display.

6. Save your program, assemble/Link it and run it. Record the status of LED11, LED12, LED13, LED14 and the status of 7-Segment Display .

……
……

7. Now, turn on LED12 by outputting **11110010B** to the appropriate port that interfaced with the LEDs.

8. Now, Modify the program to display value 6 on the 7-Segment Display (Hint: Apply the value 10000010B to the appropriate port that interfaced with 7-segment display).

9. Save your program, assemble/Link it and run it. Record the status of LED11, LED12, LED13, LED14 and the status of 7-Segment Display .

……
……

10. Now, add a delay "loop" for 4 iterations which perform nothing and name it as "Timer1". What is the total delay added to this program (in terms of number of instructions)?

……
……

11. Then, output the following values for the 7-segment port and for LEDs port respectively: 10000000B and 11110011B.

12. Save your program, assemble/Link and run it. Record the status of LED11, LED12, LED13, LED14 & the status of 7-Segment. Do you observe any difference in output result by running, explain.

……
……

13. Run the program in trace mode. Do you observe any difference in output result by running in Trace mode instead of Run mode in previous Step? explain.

……
……
……

14. Modify the number of iterations in the delay loop to 9FFFH then run your program. Describe the behavior of the program? What is the total delay added to this program (in terms of number of instructions)? What is the benefit of adding delays to the program?

……
……
……
……

15. Now, Label the first instruction of Part 4 by "START". What is the benefit of using labels in ALPs?

……

16. Add another delay loop to the program, name it "Timer2" with number of iterations 9FFFH.

17. Then, use an appropriate instruction to re-call the program from the START Label. Then run your program & Record the status of LED11, LED12, LED13, LED14 and the status of 7-Segment Display.

……
……

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 9: designing an experiment to simulate the operation of a traffic light: design and debug a small microprocessor-based system with memory and I/O

## I. Objective:

"Design and Conduct an Experiment to simulate the operation of Traffic light using LEDS and Seven-Segments- Display"

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

A traffic light has three colors: red, green and yellow, they operate in a sequential way

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

Apply the knowledge acquired from previous experiments to build the traffic light program

## VI. Experimental Work:

You are asked to design an experiment that simulates the operation of a Traffic light. The traffic light system will consist of two components:

- A Traffic lights with RED, YELLOW and GREEN colors.
- A 7-Segment Display to indicate to drivers how many seconds are left before light changes its color.

Similar to a real life traffic light, the experiment must consider a 30 seconds duration for RED, 15 seconds duration for GREEN Light and 5 second duration for YELLOW light.

In a real traffic light system, the driver will monitor the two 7-Segment Displays. The displays will continuously decrement its values indicating how many seconds are left before the light changes color. As soon as the color changes, the display starts decrementing its value one more time. The process continues indefinitely. However, since the MDA-Win8086 provides

an interface to only ONE 7-Segment Display, you will have to design your system using a single 7-Segment Display. One way to overcome this limitation is by decrementing the 7-Segment display every 3 seconds instead of every second. This way, decrementing from 9 to 0 will effectively span the duration of 30 seconds.

You need to design the program, implement and test it using the MDA-Win8086 kit available in the lab. Consider the following steps while designing this experiment:

- Design the program in Assembly language.
- The program must be Compiled, Downloaded & Run successfully on the kit.
- Your program should call a subroutine for delays calculations.

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 10: interfacing Digital-to-Analog Converter (DAC0800) to Intel 8086 MP: parallel input output (PIO) interface (8255a interface) with Intel 8086 MP

## I. Objective:

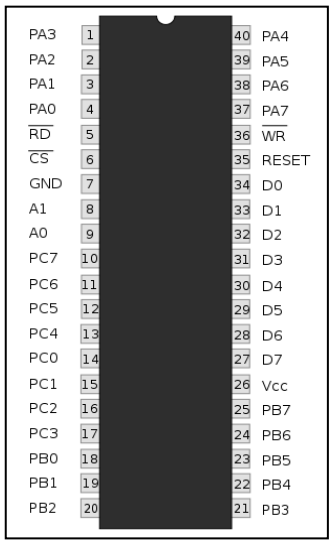
The purpose of this experiment is to learn how Parallel Peripheral Interface (PPI)-Intel 8255 works and its importance for I/O interfacing applications such as DAC. Also, it will also cover how to Interface Digital-to-Analog converter (DAC0800) to INTEL 8086 Microprocessor using 8255Chip and write an Assembly Language Program to generate analog wave form that will be shown on the Level Meter Bar.

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

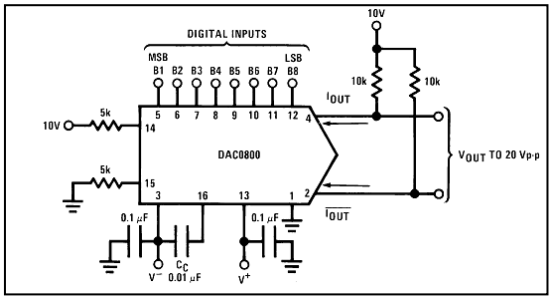
## III. Theory:

### 1. INTEL 8255 PPI:




A peripheral chip originally developed for Intel 8085 microprocessor and as such is a member of a large array of such chips, known as the MCS-85 Family. This chip was later also used with the Intel 8086 and its descendants. It was later made (cloned) by many other manufacturers. It is made in DIP 40 and PLCC 44 pins encapsulated versions. This chip is used to give the CPU access to programmable parallel I/O, and is similar to other such chips like the Motorola 6520 PIA (Peripheral Interface Adapter) the MOS Technology 6522 (Versatile Interface Adapter) and the MOS Technology CIA (Complex Interface Adapter) all developed for the 6502 family. However, most often the functionality 8255 offered is now not implemented with the 8255 chip itself anymore, but is embedded in a larger VLSI chip as a sub function. The 8255 chip itself is still made, and is sometimes used together with a microcontroller to expand its I/O capabilities.

### 2. DAC 0800:




is a monolithic 8 bit high speed current output digital to analog converters featuring setting time of 100nSEC. It also features high compliance complementary current outputs to allow differential output voltage of 20 Vp-p with simple resistor load and it can be operated both in uni-polar and bipolar mode. The figure shows a typical application of ±20 VP-P Output Digital-to-Analog Converter.

*** **HOW DAC 0800 WORKS:** When chip select of DAC is enabled then DAC will convert digital input value given through portliness PC0-PC7 to analog value. The analog output from DAC is a current quantity. This current is converted to voltage using OPAMP based current-to-voltage converter. The voltage outputs (+/- 5V for bipolar 0 to 5V for uni-polar mode) of OPAMP are connected to Level Meter to see the wave form.

**** **LEVEL METER:** is a LED Bar that contains 10 LED's ordered from 1 – 10, so that to drive this meter you should drive these LED's in order.

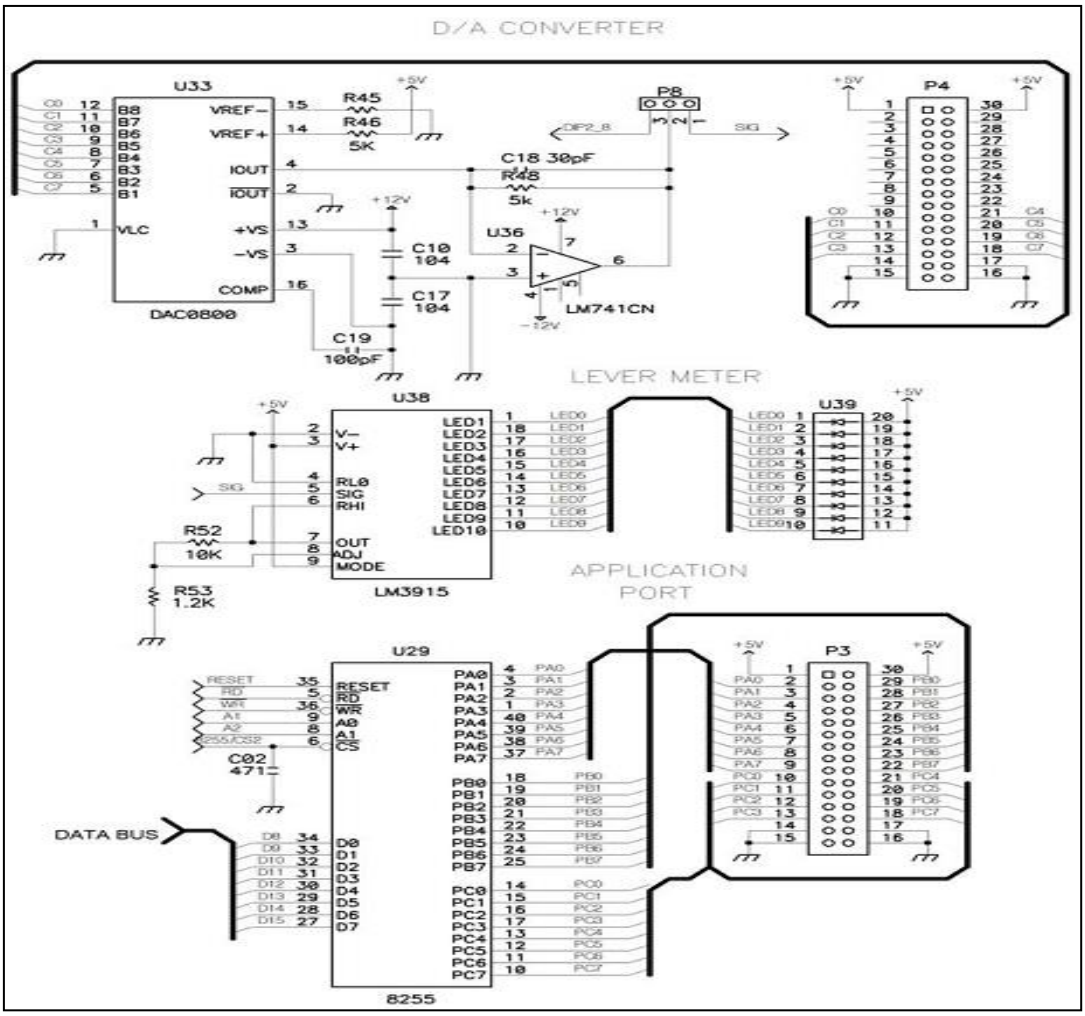



Figure 1: 8255 Interface with DAC 0800

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

Working procedure (methodology) throughout this experiment consists of the following steps:

- Start by reading the experiment environment, main objective and procedure.
- Connect PC and MDA-Win 8086 Microprocessor Kit using the Data cable provided.
- Switch 'ON' the MDA-Win 8086 Microprocessor Kit and the Computer, and go to Local Disk C: of the PC and Open the MDA folder and double click the ' MDA_Win IDE8086 ' Icon and start the MDA- Win 8086 Software Tool.
- 'Reset' MDA-Win 8086 Microprocessor Kit by pressing the RESET button.
- Start working with the Experimental work step by step until you finish the required tasks. Also, Solve the required exercises and make your final conclusions about the experiment.
- Finally, you should write your technical lab report for this experiment and the reports will be submitted by each group before the deadline specified by the instructor.

## VI. Experimental Work:

1. 'Reset' MDA-Win 8086 Kit then create a new ALP, use the name: " DAC9".
2. Declare 8255 ports interfaced to DAC0800, A: 19H, B: 1BH, C: 1DH and CTRL: 1FH.

……
……

3. Configure the three ports to work as output ports.

……

4. Configure ports A & B as they are connected in the kit. Make sure that your configuration for this port will turn off other applications. (you may review previous experiments)
5. Configure Level meter.
    1. Start by clearing LED Bar (level meter) which is connected through Port_C.
    2. Then, you should Call delay subroutine in order to change the level of level meter timely.

*** You can use the following Timer:

```
TIMER:          MOV   CX, 1
TIMER2:    PUSH  CX
           MOV   CX, 0
TIMER1:    NOP
           LOOP  TIMER1
           POP   CX
           LOOP  TIMER2
           RET
```

3. Configure DAC0800 to Uni-polar with an equivalent maximum output value of +5V (50H in digital form which is needed to operate the full bar of level meter). Consider the digital input (square waves) can be fed to DAC0800 through port C to convert it to analogue output (current). By incrementing AL and compare it to the value 50H, then if AL register is NOT equal to 50H, you should transfer its value to Port_C otherwise, go back to clear the AL register

……
……
……

6. Save/assemble/link/download run your program. Explain the behavior of the program.
……
……

7. What are the main ports and registers involved in this program? Explain the role of each one.
……
……
……
……

8. What will happen if we exchange PORT_C with PORT_B. Explain your answer?
……

9. Regarding the timer code, is it finite or infinite? What will stop this timer? And how?
……
……
……

10. What will happen if we compare the value of AL in 5.3 with 10, 20, 30 and 40? Explain.
……
……
……
……

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.

# Experiment 11: Understanding the operation of LCD with Intel 8086 MP: Running and validating the operation of LCD

## I. Objective:

The purpose of this experiment is to let you learn how LCD display is Interfaced to INTEL 8086 Microprocessor and to write an Assembly Language Program to run and validate its operations.

## II. Test Standard

IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language

## III. Theory:

**LCD:** LCD is a 16 Character x 2 Line Module. The Module has 14 pin Connections as provided in the table 1 and figure 1. The Microprocessor Address and Data bus controls the LCD operation through a 22V10 chip Interface circuit.

Table 1: Pin Connections

| Pin NO. | Symbol | Level | Function | |
|---|---|---|---|---|
| 1 | Vss | - | 0V | Power supply |
| 2 | Vdd | - | 5V | |
| 3 | VL | - | - | |
| 4 | RS | H/L | H : Data input<br>L : Instruction input | |
| 5 | R/W | H/L | H : Data read<br>L : Data write | |
| 6 | E | H. H→L | Enable signal | |
| 7 | D0 | H/L | Data bus line | |
| 8 | D1 | H/L | | |
| 9 | D2 | H/L | | |
| 10 | D3 | H/L | | |
| 11 | D4 | H/L | | |
| 12 | D5 | H/L | | |
| 13 | D6 | H/L | | |
| 14 | D7 | H/L | | |

The Microprocessor uses the following addresses to control the LCD Display chipset:

| Address | I/O Port | Comment |
|---|---|---|
| 00H | LCD Instruction Register | For address 00H, A2 = 0, A1 = 0, A0 = 0.<br>A2 is connected to LCD Pin. No. 4<br>A1 is connected to LCD Pin No. 5<br>A0 is connected to logic circuit connected to LCD Pin No. 6 |

| | | |
|---|---|---|
| 02H | LCD Status Register | For address 02H, A2 = 0, A1 = 1, A0 = 0 |
| 04H | LCD Data Register | For address 04H, A2 = 1, A1 = 0, A0 = 0 |

The LCD module has the following Pin Connections

| Pin No. | Symbol | Level | Function |
|---|---|---|---|
| 4 | RS | H/L | H: Data Input<br>L: Instruction Input |
| 5 | R/W | H/L | H: Data read<br>L: Data write |
| 6 | R/W | H/L | H: Enable signal |

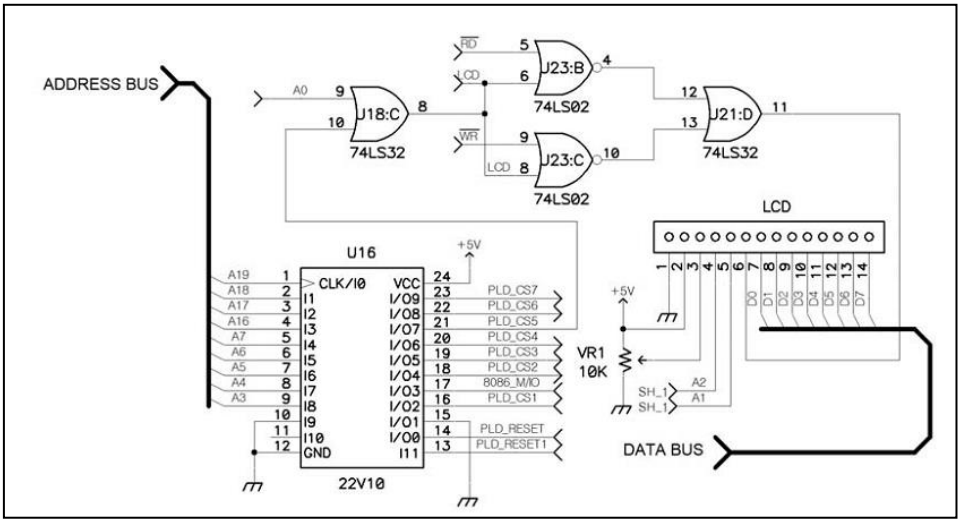



Figure 1: LCD Interface with Intel 8086

## IV. Apparatus:

You will be using a training kit (MDA-8086) connected to a PC. All the needed application software are stored locally on drive c: which is the system drive. All personal files should be stored on the desktop inside a folder holding your name. You will be working entirely in the Microsoft Windows environment. You will use 'MDA_Win IDE8086' software icon as a tool to connect with MDA-8086 kit and start working with the 8086 Microprocessor.

## V. Procedure:

Working procedure (methodology) throughout this experiment consists of the following steps:

- Start by reading the experiment environment, main objective and procedure.
- Connect PC and MDA-Win 8086 Microprocessor Kit using the Data cable provided.
- Switch 'ON' the MDA-Win 8086 Microprocessor Kit and the Computer, and go to Local Disk C: of the PC and Open the MDA folder and double click the 'MDA_Win IDE8086 ' Icon and start the MDA- Win 8086 Software Tool.

- 'Reset' MDA-Win 8086 Microprocessor Kit by pressing the RESET button provided on the kit.
- Start working with the Experimental work step by step until you finish the required tasks. Also, Solve the required exercises and make your final conclusions about the experiment.
- Finally, you should write your technical lab report for this experiment and the reports will be submitted by each group before the deadline specified by the instructor.

## VI. Experimental Work:

1. Use MDA8086-IDE software to locate, open and run ALP in the source file "LCD_A.ASM". Record the status of LCD. Describe the behavior of the output of program.

……
……
……

2. Review the hardware schematic of Microprocessor Address Bus interface to LCD interface and explain the setting of LCDC, LCDC_S, LCDD values.

……
……
……

3. In the LCD_A.asm, can we set the LCDC to control the LCD Instruction register using the following assembly line: **LCDC EQU 04H** ? Please explain your answer:

……
……
……

4. Review LCD instruction register and explain why did we set the AH value to 01H to clear the LCD display?

*** Hint: From the terminal window, Type "R" to display the values of all registers and record the value of the FL Register. Check the content of the FL6 bit i.e., ZF bit.

*** In the LCD_A.asm, we first clear the LCD display by setting the following

```
MOV     AH, 01H
CALL    LNXX
```

……

5. In which port (LCDC, LCDC_S, LCDD) did we output the value of AH to and why?

……

6. Describe the flow of processing for the CALL of LNXX (for now ignore the details of BUSY function).

……

7. In line 40, why did we move the content of AH register to AL before we use OUT instruction?

*** In MP_04A.asm, we are using two new assembly instructions that are useful when interfacing to I/O device:

- IN instruction, will read the contents of a port in to AL register.
- JNZ instruction that checks the content of ZF in the FL register and jumps if not equal to zero.

……

8. In the BUSY subroutine, why did we AND the content of AL to 80H?

……

9. Under what condition will the program RET from the BUSY subroutine?

……

10. Use MDA8086-IDE software to locate, open and run ALP in the source file "LCD_E.ASM". Record the status of the LCD. Describe the behavior of the output of the program.

……

*** The program in LCD_E.asm allows for displaying ASCII Text lines at the LCD Display. To do so, the program first stores the ASCII Text in memory and then starts Output of every character to the LCD display. To store the ASCII Text, the following assembly instruction is used:

FIRSTLINE: **DB 'WELCOME TO KFU !',00H,00H**

*The above instruction will allow the ASCII text to be stored in consecutive Bytes in memory.*

11. Open LCD_E.lst file and find the addresses of the memory locations where the two-sentences displayed on LCD Display.

……

*** For the program to output all characters of the ASCII text, the program will first determine the memory address of the first character. It then, will access the memory location and transfer the contents of the memory location to display it to the LCD Display. The program then increment the memory address for the next character and display it to the output and continue until all characters are displayed. To determine the memory address of the first character, the following instruction is used: **MOV SI, OFFSET FIRSTLINE**

The above instruction will store the memory location of FIRSTLINE in the SI register. The SI register will be used as a Source Index register for all characters of the first line.

12. Use Trace Mode to determine the memory address stored in the SI register after the processing of the above instruction. Hint: Place INT 3 instruction before the above instruction to ease the debugging in Trace mode.

*** The function CALL STRING shall loop through all characters of each line and read them from memory and send them to the LCD display.

```
STRING:  MOV   AH, BYTE PTR CS:[SI]
         CMP   AH,00H
         JE    STRING1
         CALL  BUSY؛
         CALL  CHAROUT
         INC   SI
         JMP   STRING
STRING1: RET
```

In this function, each character is transferred to Register AH by first calculating the memory location of the character by using the offset determined by CS and SI register.

MOV AH,BYTE PTR CS:[SI] ;Transfer contents of memory location addressed by 20 bit address of CS,SI registers i.e., memory location (CS<<4 + SI )

13. Use Trace mode to determine the content of the AH register after the execution of the above instruction. Hint: Place INT 3 instruction before the above instruction to ease the debugging in Trace mode.

14. Modify the program LCD_E.asm to allow you to display two two-sentences circulating one after the other for as long as the Power of the kit is ON.

## References:

Microprocessors and Interfacing: 8086, 8051, and Advanced processors, N. Senthil Kumar, M. Saravanan, S. Jeevanathan, and S.K. Shah, OXFORD University Press, 2012, ISBN 0-19-807906-0.