University of Global Village (UGV), Barishal

Dept. of Electrical and Electronic Engineering (EEE)

Lab Manual

Microprocessor & Interfacing Lab
EEE-0714-3110 / EEE 314

Noor Md Shahriar
BSc in EEE, RUET
Senior Lecturer
Co-chairman, Dept. of EEE
University of Global Village (UGV)
874/322, C&B Road, Barishal, Bangladesh.
Contact: +8801743500587
Facebook | LinkedIn | Twitter

Course Title:	Microprocessor Interfacing Lab	Total Class Hour	37
Course Code:	EEE 0714-3110	Total Practice Hour	37
Supervised by	Noor Md Shahriar	Total Hour	85

Course Rationale

The Microprocessor Interfacing Lab is designed to provide students with a hands-on learning experience in microprocessor-based system design and interfacing. By working with the MDA-8086 kit, assembly language programming, and various interfacing techniques, students gain practical skills crucial for understanding and implementing real-world embedded systems. This course bridges theoretical knowledge of microprocessors with practical applications, preparing students for challenges in hardware design, programming, and system integration. It fosters critical thinking, problem-solving, and technical proficiency required in modern engineering and technological advancements.

Course Objectives

- To familiarize students with the architecture and operation of the 8086 microprocessor.
- To develop proficiency in assembly language programming for logic, arithmetic, and control operations.
- To enable students to interface the 8086 microprocessor with peripheral devices such as LEDs, 7-segment displays, and dot-matrix displays.
- To introduce students to microcontrollers and their applications in interfacing with I/O and analog devices.
- To build practical skills in designing and debugging microprocessor-based systems for real-world applications.

Course Learning Outcomes (CLO)

CLOs	Learning Outcome
CLO1	Demonstrate an understanding of the architecture and functionalities of the 8086 microprocessor.
CLO2	Write and execute assembly language programs for performing arithmetic, logic, and control operations.
CLO3	Interface the 8086 microprocessor with various peripherals such as LEDs, 7-segment, and dot-matrix displays.

CLO4	Explain the fundamental concepts of microcontrollers and their applications in interfacing with external devices.
CLO5	Design and troubleshoot microprocessor-based systems for solving real-world interfacing and control problems.

Course Outline

Sl. No.	Topic & Details	Class Hours
1	Familiarization with the MDA-8086 microprocessor kit and introduction to machine code loading.	2
2	Loading machine codes of a sample program to MDA-8086, executing in single-step mode, and verifying results.	2
3	Exploring the "Serial Monitor" mode operation and verification of arithmetic operations.	2
4	Writing assembly language programs for logic operations on MDA-8086.	2
5	Programming control instructions in assembly language.	2
6	Interfacing LEDs and 7-segment displays with the 8086 microprocessor.	2
7	Interfacing a dot-matrix LED display with the 8086 microprocessor.	2
8	Introduction to microcontrollers and their role in embedded systems.	2
9	Interfacing a 7-segment display with microcontrollers.	2
10	Interfacing with the analog world, including sensors and actuators.	2

Course Schedule

Week	Topic & Details	Class Hours	Teaching-Learning Strategy	Assessment Strategy	CLO Mapping
Week 1	Familiarization with MDA-8086 microprocessor kit and loading machine codes of a sample program to MDA-8086.	2	Lecture, hands-on session	Observation, Q&A, lab report	CLO1
Week 2	Loading machine codes of a sample program to MDA-8086, executing instructions in single-step	2	Guided practice, hands-on session	Evaluation of program execution, lab report	CLO1

	mode, and verifying results.				
Week 3	Familiarization with the "Serial Monitor" mode operation of MDA-8086 and verification of arithmetic operations.	2	Demonstration, hands-on session	Performance evaluation, Q&A	CLO1, CLO2
Week 4	Logic operations in assembly language in MDA-8086 kit.	2	Lecture, practical implementation	Review of program outputs, lab report	CLO1, CLO2
Week 5	Program control instructions in assembly language.	2	Demonstration, guided implementation	Execution and debugging review, short quiz	CLO2
Week 6	Interfacing of LED and 7-segment display with 8086 microprocessor.	2	Explanation, hands-on implementation	Evaluation of interfacing, documentation	CLO3
Week 7	Interfacing of Dot-Matrix LED display with 8086 microprocessor.	2	Demonstration, practice session	Performance evaluation, lab report	CLO3
Week 8	Introduction to Microcontroller.	2	Lecture, discussion, demonstration	Observation, Q&A	CLO4
Week 9	Interfacing with I/O devices (7-segment display).	2	Practical implementation, guided exercises	Functionality verification, lab report	CLO4
Week 10	Interfacing with Analog World.	2	Demonstration, hands-on practice	Evaluation of interfacing, review of observations	CLO4
Week 11	Midterm Lab Assessment	2	Practical test	Evaluation of programming and interfacing skills	CLO1, CLO2, CLO3, CLO4
Week 12	Advanced Interfacing: Combining multiple devices	2	Problem-solving, guided implementation	Evaluation of integration and debugging	CLO3, CLO4
Week 13	Project Work Discussion and Planning	2	Group discussions, project brainstorming	Submission of project plan, group presentation	CLO4

Week 14	Project Implementation: Phase 1	2	Guided project work	Review of progress, troubleshooting support	CLO4
Week 15	Project Implementation: Phase 2	2	Group practice, troubleshooting	Final project testing, peer evaluation	CLO4
Week 16	Final Project Presentation and Evaluation	2	Presentation, Q&A	Evaluation of project functionality and demonstration	CLO4
Week 17	Lab Final Exam	2	Practical assessment	Comprehensive evaluation of lab skills	CLO1, CLO2, CLO3, CLO4

Assessment Pattern

- Continuous Assessment

Bloom's Category	**Tests**
Imitation	12
Manipulation	8
Precision	6
Articulation	2
Naturalization	2

- Semester End Examination: (SEE):

Bloom's Category Marks (out of 30)	**Tests (20)**	**Quiz (10)**	**External Participation in Curricular/Co-Curricular Activities (20)**
Imitation	**06**	**06**	Bloom's Affective Domain: (Attitude or will) • **Attendance: 10** • **Viva-Voca: 5** • **Report Submission: 5**
Manipulation	**04**	**04**	
Precision	**06**		
Articulation	**02**		
Naturalization	**02**		

References

1. Mazidi, M. A., & Mazidi, J. G. (2006). *The 8051 Microcontroller and Embedded Systems: Using Assembly and C*. Pearson Education.

2. Hall, D. V. (2000). *Microprocessors and Interfacing: Programming and Hardware*. Tata McGraw-Hill.

3. Leventhal, L. A. (1997). *Introduction to Microprocessors: Software, Hardware, Programming*. Prentice-Hall.

4. MDA-8086 User Manual: Official manual for MDA-8086 Microprocessor Kit.

5. MATLAB Documentation: Official resources for microprocessor simulation tools.

Experiment List

1.	Familiarization with MDA-8086 microprocessor kit and loading machine codes of a sample program to MDA-8086.
2.	To load the machine codes of a sample program to MDA-8086, execute instructions in single-step mode, and verify results.
3.	Familiarization with the "Serial Monitor" mode operation of MDA-8086 and verification of arithmetic operations.
4.	Logic operations in assembly language in MDA8086 kit.
5.	Program control instructions in assembly language.
6.	Interfacing of LED and 7-segment display with 8086 microprocessor.
7.	Interfacing of Dot-Matrix LED display with 8086 microprocessor.
8.	Introduction to Microcontroller.
9.	Interfacing with I/O devices (7-segment display).
10.	Interfacing with Analog World.

University of Global Village (UGV),Barishal

Department of Electrical and Electronic Engineering

EEE-314: Microprocessor and Interfacing Sessional

Experiment No. 01: Familiarization with MDA-8086 microprocessor kit and loading machine codes of a sample program to MDA-8086.

1.1 Objectives: The objectives of this experiment are-

a) To familiarize with MDA-8086 system configuration.
b) To operate MDA-8086 in "Machine Code" mode.
c) To know about different registers inside 8086 microprocessors.

1.2 MDA-8086 System Configuration:

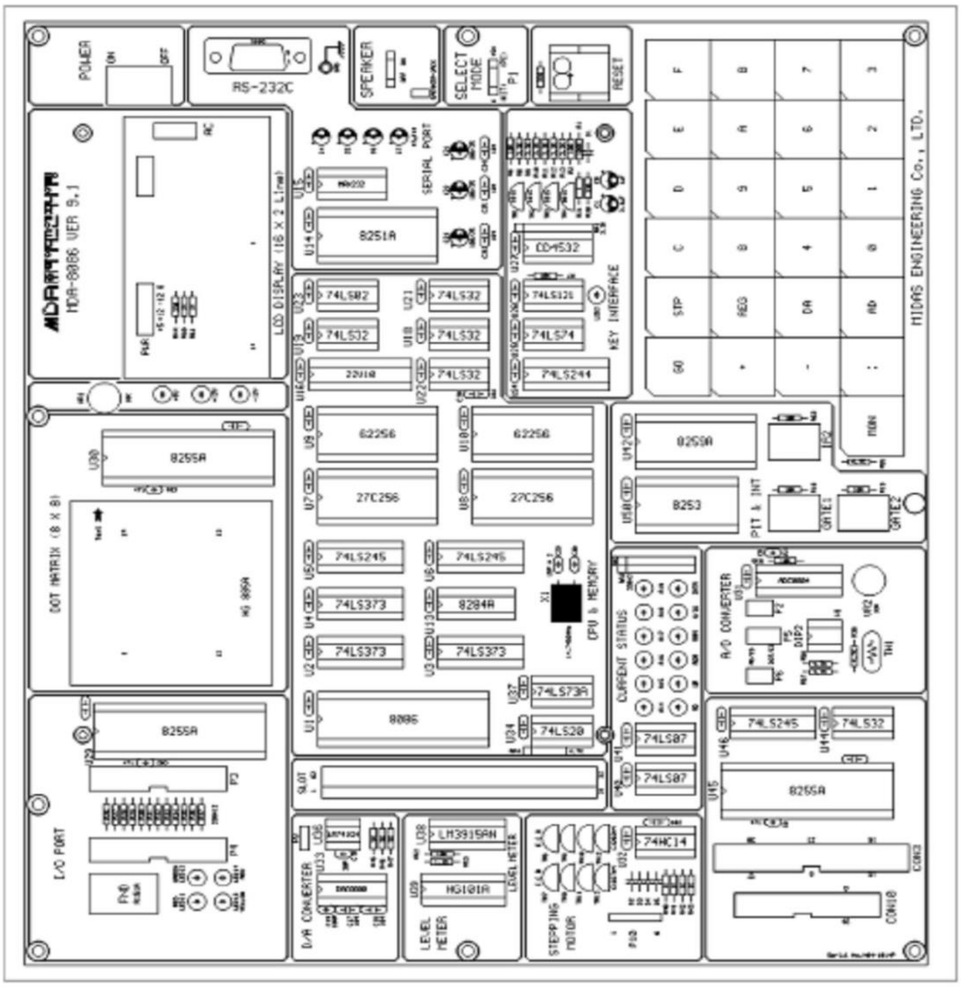


Figure 1.1: MDA-8086 System Configuration.

The functions of ICs at Figure 1 are:

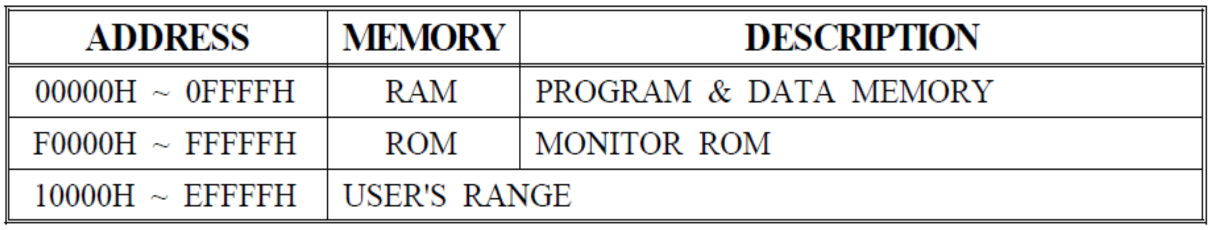
a) CPU (Central processing unit): Using Intel 8086, Using 4.9152Mhz.

b) ROM (Read Only Memory): It has program to control user's key input, LCD display, user's program. 64K Byte, it has data communication program. Range of ROM Address is F0000~FFFFFH.

c) SRAM (Static Random Access Memory): Input user's program & data. Address of

memory is
00000H~0FFFFH, totally 64 Kbyte.
d) DISPLAY: It is LCD, 16(Character) × 2(Line).
e) KEY BOARD: It is used to input machine language and has 16 of hexa-decimal keys and 8 of function keys.
f) SPEAKER: Able to test sound using with speaker and further more able to test synthesizer.
g) RS-232C: It is ready to do data communication with IBM compatible personal computer.
h) DOT MATRIX LED: To understand & test of dot matrix structure and principle of display it is interfaced to 8255A (PPI).
i) A /D CONVERTER: Convert analog signal to digital signal using with ADC0804.
j) D /A CONVERTER: Convert digital signal to analog signal using with DAC0800 and it is interfaced so as to more Level meter.
k) STEPPING MOTOR INTER FACE: So as to control stepping motor driver circuit of stepping motor is interfaced.
l) POWER: AC 110~220V, DC +5V 3A, +12V 1A, -12V 0.5A SMPS.

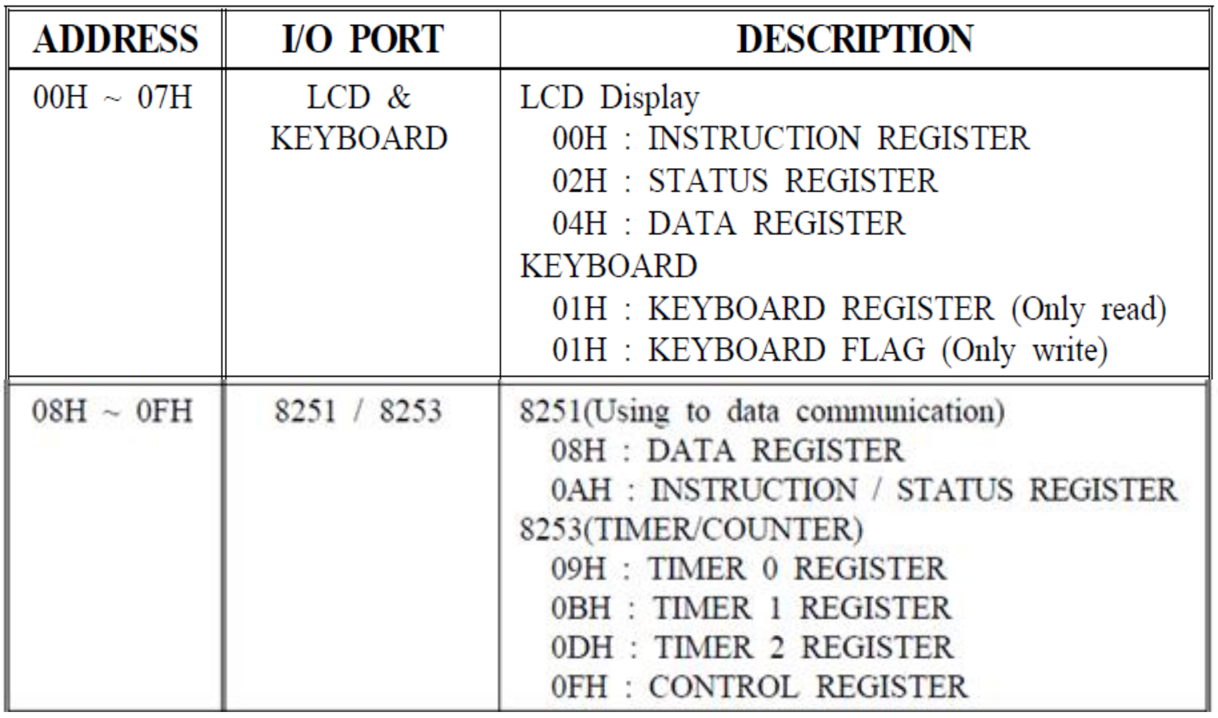
1.3 Memory map:



ADDRESS	MEMORY	DESCRIPTION
00000H ~ 0FFFFH	RAM	PROGRAM & DATA MEMORY
F0000H ~ FFFFFH	ROM	MONITOR ROM
10000H ~ EFFFFH	USER'S RANGE	

Figure 1.2: Memory map.

1.4 I/O address map:



ADDRESS	I/O PORT	DESCRIPTION
00H ~ 07H	LCD & KEYBOARD	LCD Display 00H : INSTRUCTION REGISTER 02H : STATUS REGISTER 04H : DATA REGISTER KEYBOARD 01H : KEYBOARD REGISTER (Only read) 01H : KEYBOARD FLAG (Only write)
08H ~ 0FH	8251 / 8253	8251(Using to data communication) 08H : DATA REGISTER 0AH : INSTRUCTION / STATUS REGISTER 8253(TIMER/COUNTER) 09H : TIMER 0 REGISTER 0BH : TIMER 1 REGISTER 0DH : TIMER 2 REGISTER 0FH : CONTROL REGISTER

10H ~ 17H	8259/SPEAKER	8259(Interrupt controller) 10H : COMMAND REGISTER 12H : DATA REGISTER SPEAKER → 11H : SPEAKER
18H ~ 1FH	8255A-CS1/ 8255A-CS2	8255A-CS1(DOT & ADC INTERFACE) 18H : A PORT DATA REGISTER 1AH : B PORT DATA REGISTER 1CH : C PORT CONTROL REGISTER 8255-CS2(LED & STEPPING MOTOR) 19H : A PORT DATA REGISTER 1BH : B PORT DATA REGISTER 1DH : C PORT CONTROL REGISTER 1FH : CONTROL REGISTER
20H ~ 2FH	I/O EXTEND CONNECTOR	
30H ~ FFH	USER'S RANGE	

Figure 1.3: I/O address map.

1.5 Kind and Function of Key:

MDA-8086 has high performance 64K-byte monitor program. It is designed for easy function. After power is on, the monitor begins to work. In addition to all the key function the monitor has a memory checking routine.

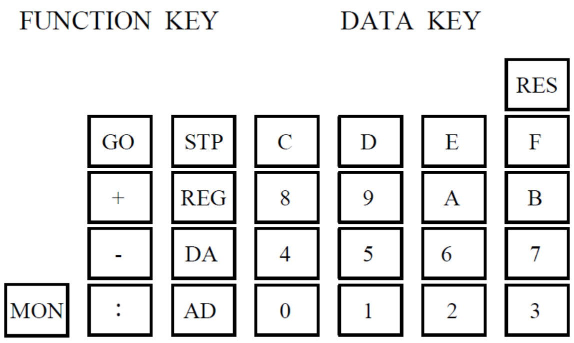


Figure 1.4: Keypad

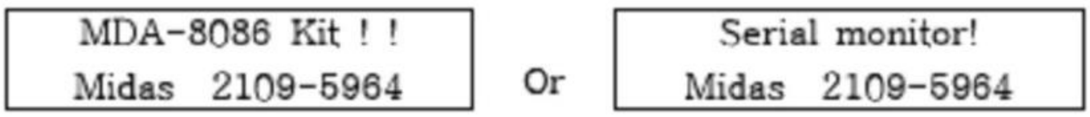
The functions of the keys are given below:

RES	system reset	STP	execute user's program, a single step
AD	set memory address	GO	go to user's program or execute monitor functions
DA	Update segment & Offset. and input data to memory	MON	Immediately break user's program and Non makable interrupt.
:	Offset.	REG	Register Display.
+	Segment & Offset +1 increment. Register display increment.		
–	Segment & Offset -1 increment. Register display decrement.		

1.6 Basic Operation

MDA-8086 can operate in two modes. (a) Machine Code Mode (b) Serial Monitor Mode
In machine code mode, user can load instructions/ program directly by keypad and can observe the contents of different registers in LCD. On the other hand, user can load instruction/program from computer via serial port in Serial Monitor Mode.

On a power-up, following message will be displayed on a LCD.



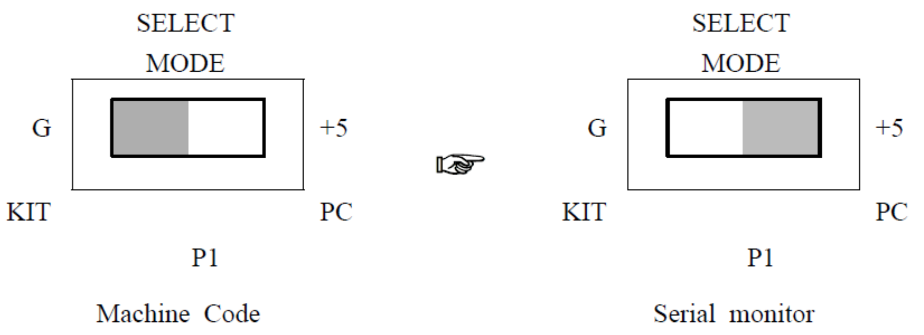
MDA-8086 Kit ! !
Midas 2109-5964

Or

Serial monitor!
Midas 2109-5964

Figure 1.5: Power on monitor.

To use “Machine Code” mode, move jumper P1 which located on the PCB like this.

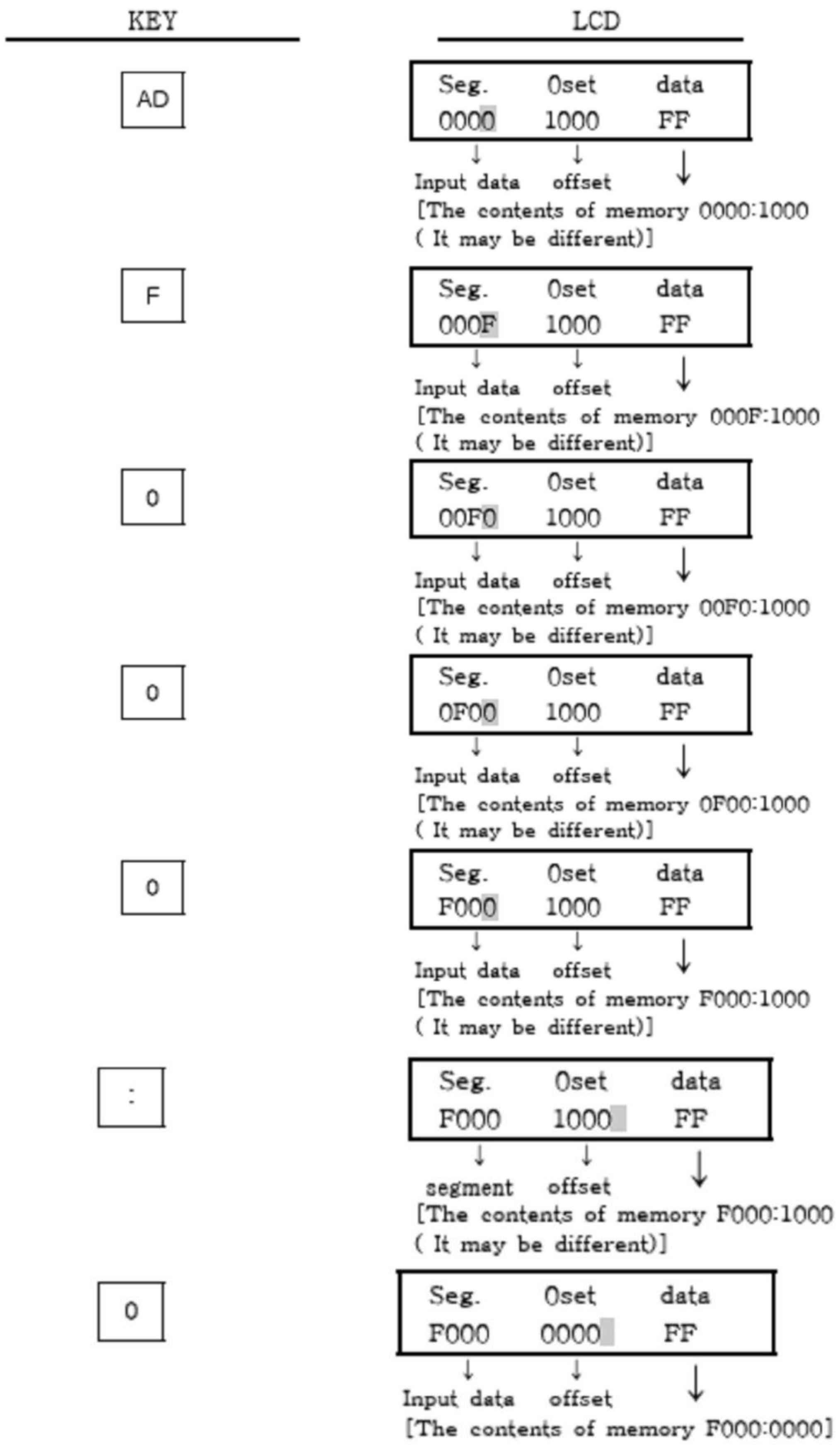


Whenever RES is pressed, the display becomes Figure 5 and user can operate keyboard only in this situation.

1.7 Finding the content of a memory address

$\boxed{\text{AD}}$, $\boxed{:}$ HEXA-DIGIT KEY: Substitute to segment & offset address.

Example 1: Check the contents in memory.

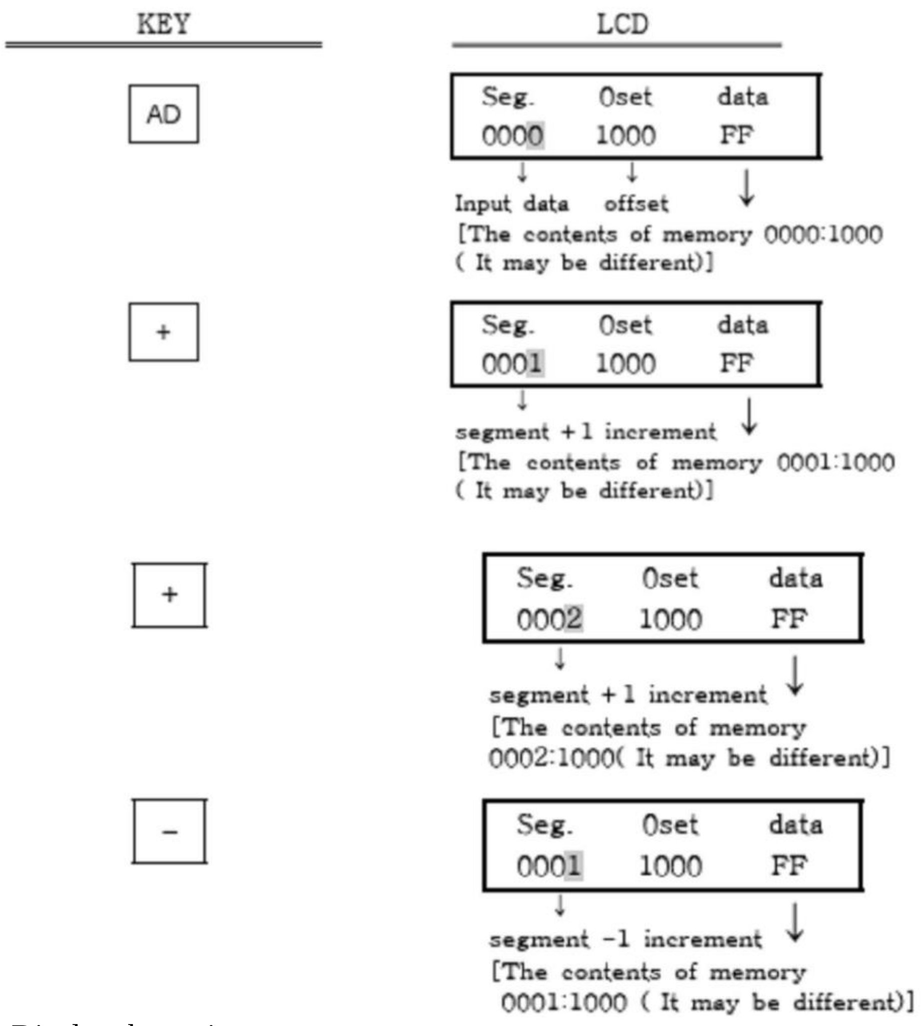


KEY	LCD
AD	Seg. Oset data 0000 1000 FF Input data / offset [The contents of memory 0000:1000 (It may be different)]
F	Seg. Oset data 000F 1000 FF Input data / offset [The contents of memory 000F:1000 (It may be different)]
0	Seg. Oset data 00F0 1000 FF Input data / offset [The contents of memory 00F0:1000 (It may be different)]
0	Seg. Oset data 0F00 1000 FF Input data / offset [The contents of memory 0F00:1000 (It may be different)]
0	Seg. Oset data F000 1000 FF Input data / offset [The contents of memory F000:1000 (It may be different)]
:	Seg. Oset data F000 1000 FF segment / offset [The contents of memory F000:1000 (It may be different)]
0	Seg. Oset data F000 0000 FF Input data / offset [The contents of memory F000:0000]

Example-2: Increment and decrement to segment & offset address.

KEY: Increment and decrement to segment & offset address.

AD , + , –



KEY | LCD

AD

Seg. Oset data
0000 1000 FF

Input data offset
[The contents of memory 0000:1000
(It may be different)]

+

Seg. Oset data
0001 1000 FF

segment +1 increment
[The contents of memory 0001:1000
(It may be different)]

+

Seg. Oset data
0002 1000 FF

segment +1 increment
[The contents of memory
0002:1000(It may be different)]

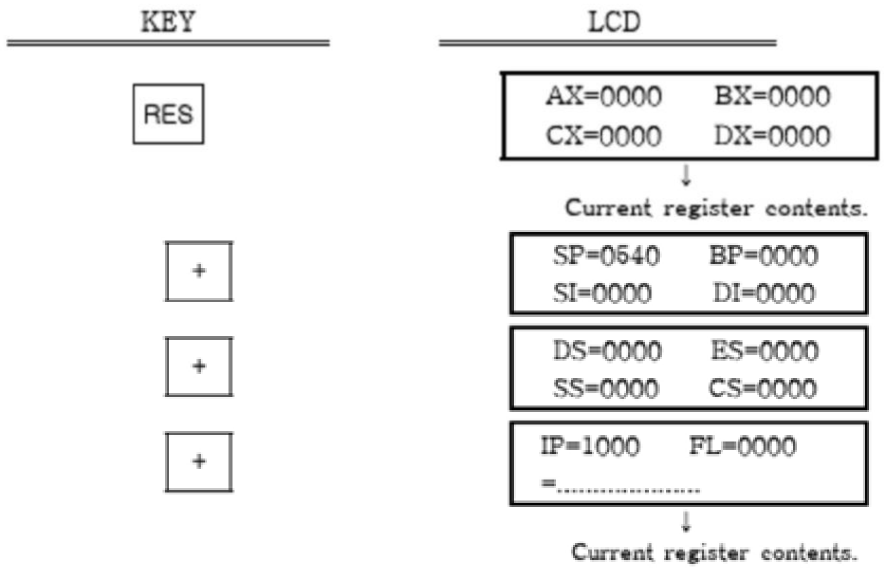
–

Seg. Oset data
0001 1000 FF

segment -1 increment
[The contents of memory
0001:1000 (It may be different)]

Example 3: Display the register contents.

REG , + , –   KEY: Display to register contents



KEY | LCD

RES

AX=0000 BX=0000
CX=0000 DX=0000

Current register contents.

+

SP=0640 BP=0000
SI=0000 DI=0000

+

DS=0000 ES=0000
SS=0000 CS=0000

+

IP=1000 FL=0000
=.....................

Current register contents.

1.8 Architecture of 8086 microprocessors:

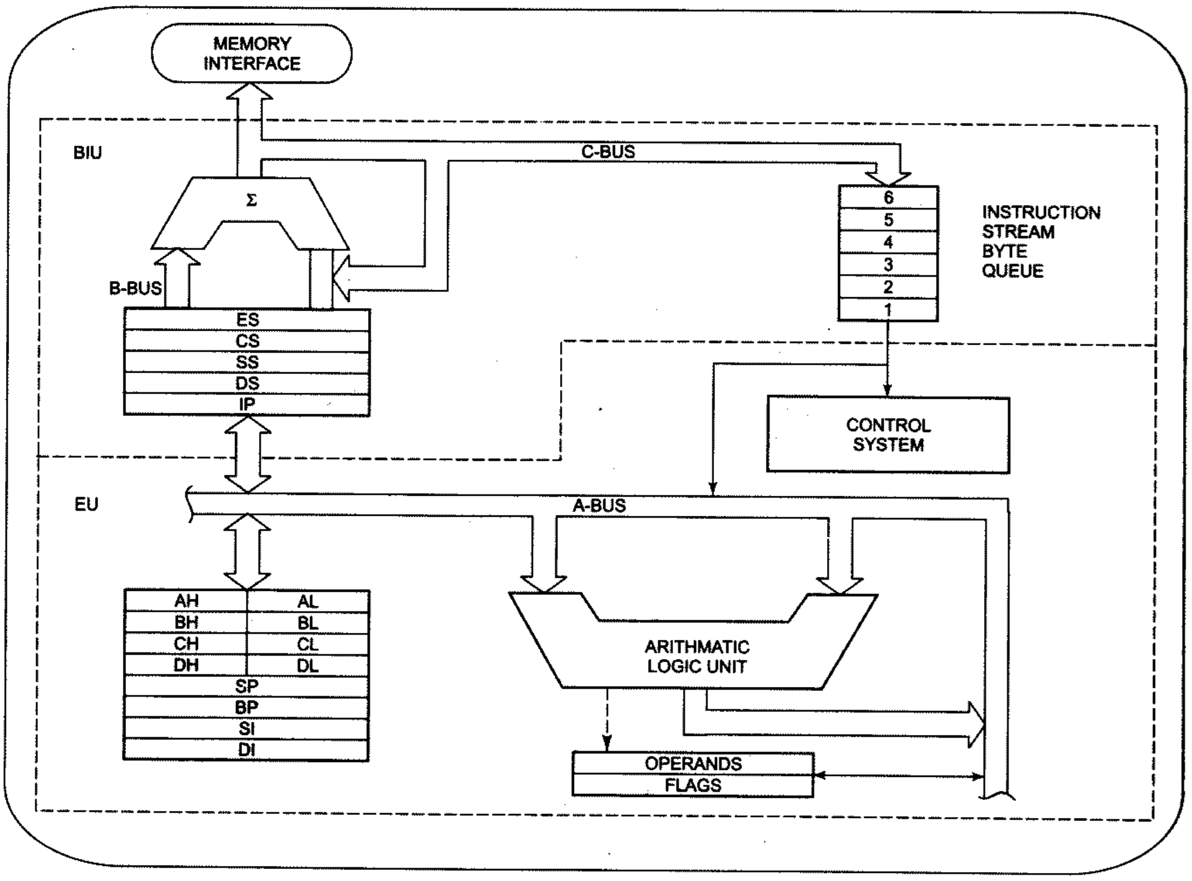


Figure 1.6: Architecture of 8086 microprocessors.

The architecture of 8086 microprocessors is divided into two independent functional parts:

(a) The Bus interface unit (BIU)
(b) Execution Unit (EU)

Functions of EU

- It contains control circuitry which directs internal operations.
- A decoder in EU decodes instructions
- The EU has a 16 bit ALU which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift binary numbers.

Functions of BIU

- To fetch instructions
- To read data from memory and ports
- To write data to memory and I/O ports.
- To interface the 8086 to the outside world
- To provide all external bus operations

8086 Register Set

EU Registers:

The EU has nine 16 bit registers AX, BX, CX, DX, SP, BP, SI, DI and Flag Register.

The 16 bit general registers AX, BX, CX, DX can be considered as 8 bit registers (AH, AL; BH, BL; CH, CL; DH, DL)

BIU Registers:
Segment Registers: ES, CS, SS, DS and Instruction Pointer: IP
Segment Registers and respective offset registers

Segment	*Offset*	*Special Purpose*
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8-bit number or a 16-bit number	Data address
ES	DI for string instructions	String destination address

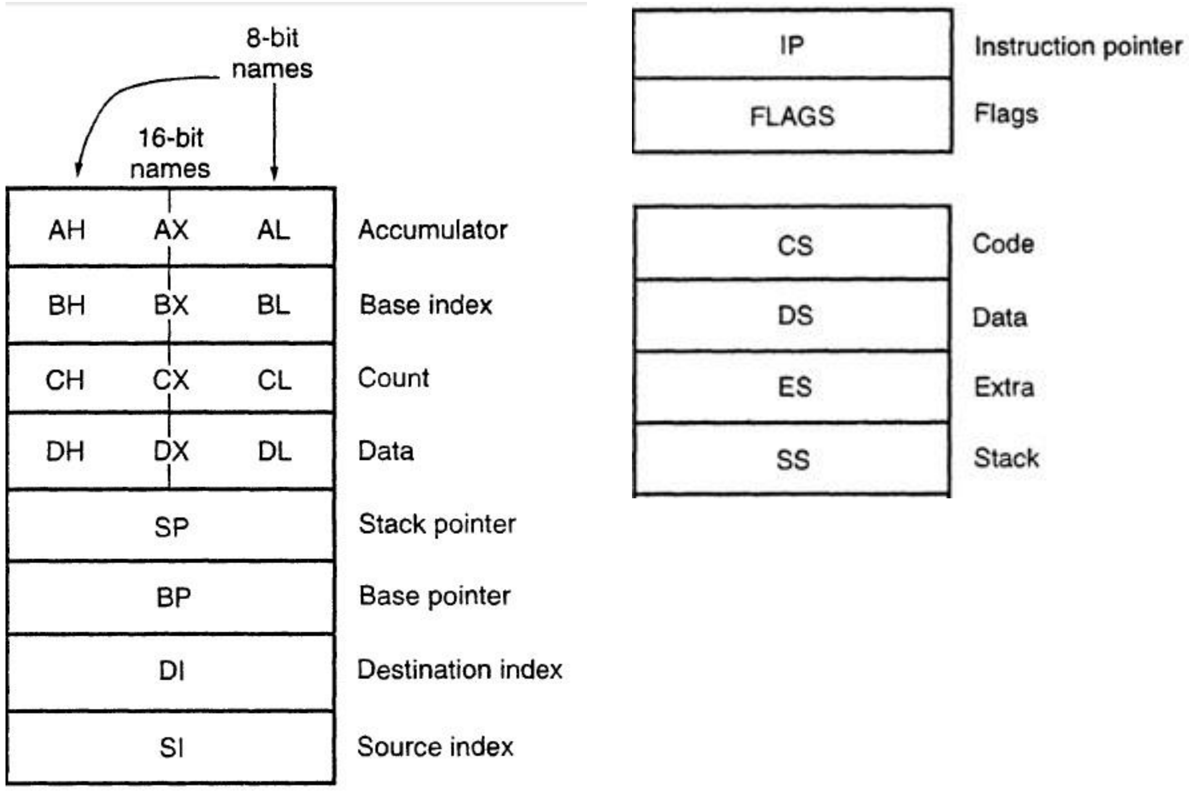


Figure 1.7: All registers in 8086 microprocessors

Questions:

a) What is microprocessor? What is the purpose of microprocessor in digital system?
b) Define the terms: microprocessor, microcomputer and microcontroller.
c) Is MDA-8086 a microprocessor or microcontroller? Justify your answer by definition.
d) Which memory of MDA-8086 stores user program and data?
e) Write down the features of MDA-8086.
f) What is register? What is the purpose of registers inside microprocessor?

Quiz:

Question 1.
What is the purpose of RS-232C in MDA-8086 trainer kit?

(a) to do data communication with A/D converter
(b) to do data communication with IBM compatible personal computer
(c) to do data communication with Matrix Display
(d) to do data communication with D/A converter

Question 2.
The key board used in MDA-8086 kit has 16 hexadecimal keys and 8 functional keys.

(a) True
(b) False

Question 3.
What is the function of key shown in figure below?

$\boxed{\textbf{STP}}$

(a) Set memory address
(b) Offset set
(c) Execute user's program in single step
(d) Set parity flag

Question 4.
Match the following keys with their functions:

AD
DA
MON
:

Update segment & offset and input data to memory
set memory address
Non-mask able interrupt
Offset Set

Question 5.
MDA-8086 can operate in three modes: machine code, serial monitor and auto capture mode**.**

(a) True
(b) False

Question 6.
Which of the following formulas is used to obtained Physical address?

(a) $Segment \times 10 + Offset$

(b) $Segment + Offset + 10$

(c) $Offset \times 10 + Segment$

(d) $(Segment + Offset) \times 10$

Question 7.
CPU of MDA-8086 has a clock frequency of -----------------------------

(a) 4.9152 GHz
(b) 2.5191 GHz
(c) 4.9152 MHz
(d) 2.5191GHz

Question 8.
ROM has the program to control-

(a) User's key input
(b) LCD display
(c) Dot matrix display
(d) Data communication program
(e) 7-segment display
(f) User's program

Question 9.
SRAM means Systems Random Access Memory.

(a) True
(b) False

Question 10.
User program in MDA- 8086 is stored in ------------------------------

(a) ROM
(b) SRAM
(c) DRAM
(d) Hard Disk

Question 11.
Which one is the size of SRAM?

(a) 64 Kbyte
(b) 32 Kbyte
(c) 64 Kbit
(d) 32 Kbit

Question 12.
Which option indicates the range of SRAM?

(a) F0000~FFFFFH
(b) 0FFFF~F0000H
(c) 00000~0FFFFH
(d) None of this

Question 13.
Which is not exist in MDA-8086 trainer board?

(a) Speaker
(b) A/D Converter
(c) Graphic Display
(d) Stepper Motor Interface

Question 14.
Match the following memory with their respective range of address.

00000H~0FFFFH
10000H~EFFFFH
F0000H~FFFFFH

SRAM
ROM
User's Range

Question 15.
Which are not the internal part of 8086 microprocessors?

(a) ALU (b) Flag Resister
(c) RAM (d) ROM
(e) Instruction Queue

Question 16.
Which of following are 8- bit register?

(a) DS (b) AH
(c) Flag Register (d) DL

Question 17.
Which register hold the offset address of code segment?

(a) IP (b) AX
(c) SP (d) BP
(e) DI

University of Global Village(UGV),Barishal
Department of Electrical and Electronic Engineering
EEE-314: Microprocessor and Interfacing Sessional

Experiment No. 02: To load the machine codes of a sample program to MDA-8086, execution of instructions in single step mode and verification of results.

2.1 Objectives: The objectives of this experiment are-

a) To learn the procedure of loading program in RAM of MDA-8086 in “Machine Code” mode.
b) To load the program to MDA-8086, execute the program in single step mode and verify the result.

2.2 Instructions/Program:

```
CODE SEGMENT
ASSUME CS: CODE, DS: CODE
        ORG 1000H
        MOV AX, 805EH
        MOV DX, 0540H
        MOV CL, 02H
        MOV CH, 91H
        ADD AX, DX
        MOV BX, 0050H
        SUB AX, BX
        INC DL
        DEC BX
        XCHG CX, BX
        HLT
CODE ENDS
END
```

2.3 Experiment Procedures:

1. Write the above program in notepad and save the file as “filename.asm”. Place this file in the folder where “masm.exe” exists.

2. Go to command prompt and execute “masm.exe”. You will see the following message

```
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Misrosoft Corp 1981, 1988. All right reserved. Source filename [.ASM]:
```

3. Follow the procedure given below to prepare machine code for your program:

```
Source filename [.ASM]: filename Press ENTER

Object filename [C:filename.OBJ]: Press ENTER
```

Source listing [NUL.LST]: filename Press ENTER

Cross reference [NUL.CRF]: Press ENTER

4. Open the file "filename.lst" using notepad. Here you will find respective machine code for each instruction.

5. Turn on the 8086 microprocessor kit

6. Set the address 0000H: 1000H in MDA-8086 kit.

7. Write the machine codes in the appropriate memory address according the following table:

Seg. Address	Offset Address	Machine Code	Instruction / Mnemonics
0000H	1000	B8 805E	MOV AX, 805EH
0000H	1003	BA 0540	MOV DX, 0540H
0000H	1006	B1 02	MOV CL, 02H
0000H	1008	B5 91	MOV CH, 91H
0000H	100A	03 C2	ADD AX, DX
0000H	100C	BB 0050	MOV BX, 0050H
0000H	100F	2B C3	SUB AX, BX
0000H	1011	FE C2	INC DL
0000H	1013	4B	DEC BX
0000H	1014	87 CB	XCHG CX, BX
0000H	1016	F4	HLT

8. Execute this program in single step using "STP". Observe the register contents after execution of each instruction and note down in the data table. Perform theoretical calculations and verify results.

2.4 Data Table:

Instruction	AX	BX	CX	DX	Set Flag
MOV AX, 805EH					
MOV DX, 0540H					
MOV CL, 02H					
MOV CH, 91H					
ADD AX, DX					
MOV BX, 0050H					
SUB AX, BX					
INC DL					
DEC BX					
XCHG CX, BX					
HLT					

Question
Write down the flags of 8086 microprocessors. Describe the flags of 8086 microprocessors.

Quiz:

Question 1.
Which flag bits may change after addition or subtraction according to result.

(a) Carry Flag	(b) Parity Flag
(c) Auxiliary Carry	(d) Trap Flag
(e) All of them	

Question 2.
How many conditional flags are here in 8086 microprocessors?

(a) 4	(b) 6
(c) 3	(d) 9

Question 3.
The size of flag resister is ----------------- bit among which ------------------ bits are active.

Question 4.
Find the content of AX after following instructions -----------

```
MOV BX, F230H
MOV DX, F009H
MOV AL, 1284H
MOV AH, DH
```

Question 5.
Find the status of different conditional flag bits after executing following instructions.

```
MOV CX, 101FH
MOV DX, F009H
XCHG CH, DL
MOV DX, CX
```

Question 6.
Suppose 805EH is a sign numbers. The decimal equivalent of this number is -----------.

Question 7.
Suppose 805EH is an unsigned number. The decimal equivalent of this number is ------------.

Question 8.
ORG 1000H indicates ------------.

(a) writing of machine code starts from
(b) initial IP is 1000H
(c) segment address is set to 1000H
(d) turn on 8086 microprocessors bit

Question 9.
Suppose set address to write machine code is 0000H: 1000H. Find the content of CS and IP register after writing the following machine codes.

B8 805E
BA 0540

University of Global Village(UGV),Barishal
Department of Electrical and Electronic Engineering
EEE-314: Microprocessor and Interfacing Sessional

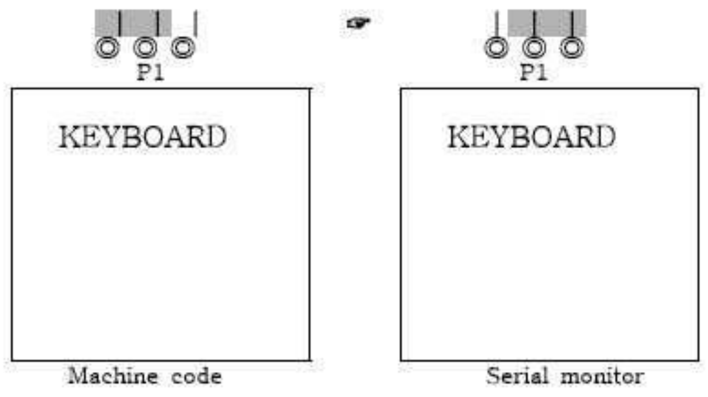
Experiment No. 03: Familiarization with "Serial Monitor" mode operation of MDA-8086 and verification of arithmetic operations.

3.1 Objective: The objectives of this experiment are-

a) To familiarize with the operation of MDA-8086 in "Serial Monitor" mode.
b) To learn the procedure of loading program in RAM of MDA-8086 in "Serial Monitor"
mode.
c) To load a program containing arithmetic operations to MDA-8086, execute the program in single step mode and verify the results.

3.2 Serial Monitor:

Serial monitor is the basic monitor program to perform data communication between MDA-8086 and a computer. So as to use serial monitor, we have to move jumper P1 which located on the PCB like this.



3.3 Connection between computer and MDA-8086

The connector of computer RS-232C is 25 pin and RS-232C of MDA-8086 is 9 pin, must be connect like figure1.

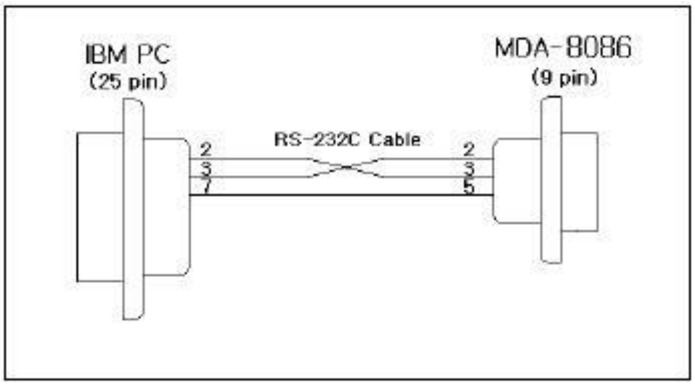


Figure1: PC 25 PIN - MDA-8086 9 PIN connection

When the connector of computer RS-232C is 9 pin and RS-232C of MDA-8086 is 9 pin, must be connected like figure 2

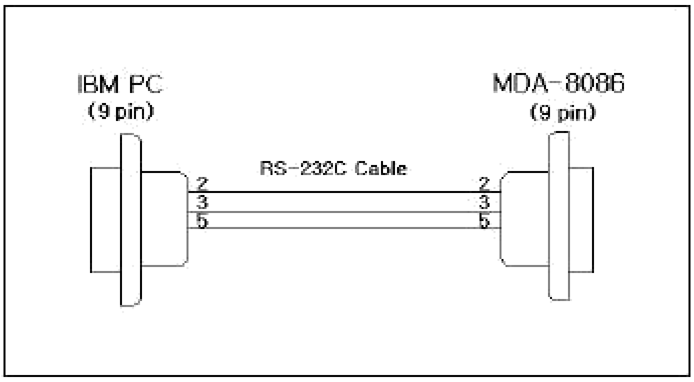
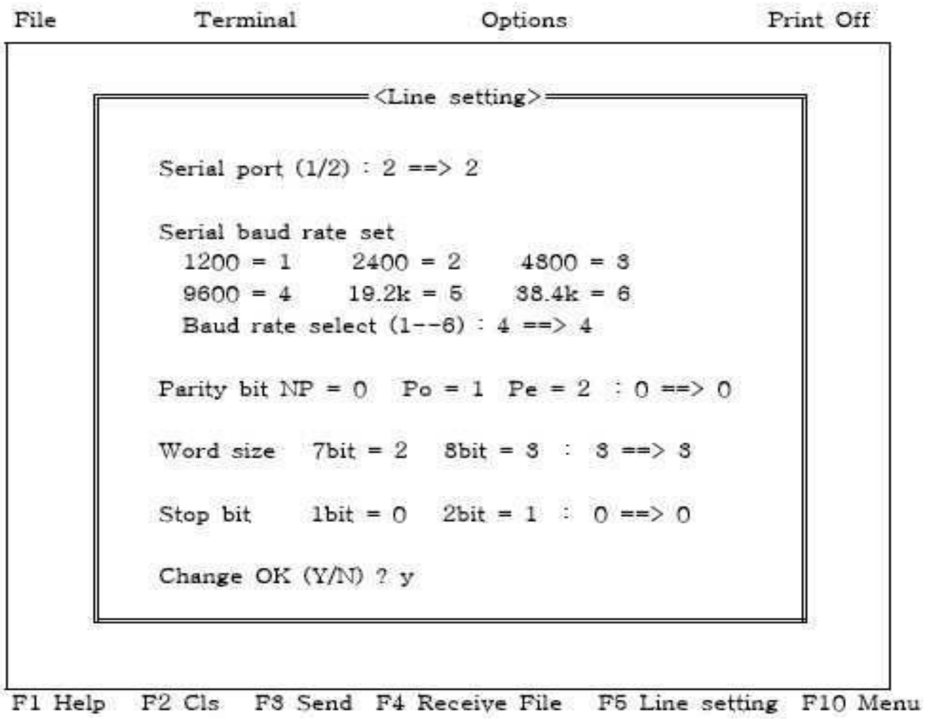


Figure 2: PC 9 PIN - MDA-8086 9 PIN connection

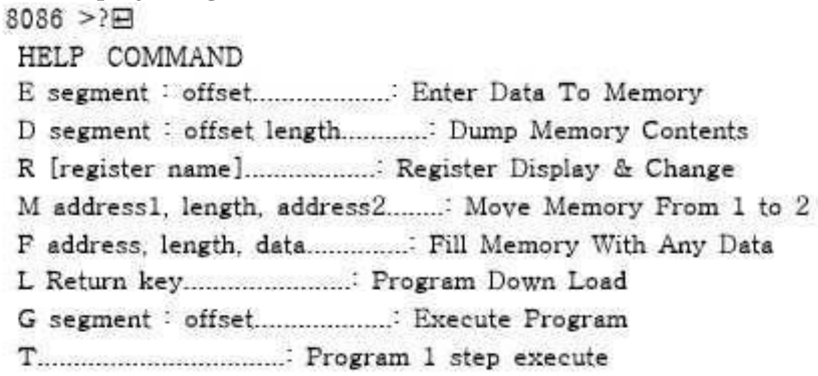
Data communication between MDA-8086 and a computer need fixing initial of WinComm software. When we press F5 key, following is displayed and the step of fixing initial is like as follows.



```
File              Terminal                Options               Print Off

                  <Line setting>

      Serial port (1/2) : 2 ==> 2

      Serial baud rate set
        1200 = 1      2400 = 2      4800 = 3
        9600 = 4      19.2k = 5     38.4k = 6
        Baud rate select (1--6) : 4 ==> 4

      Parity bit NP = 0    Po = 1   Pe = 2   : 0 ==> 0

      Word size    7bit = 2    8bit = 3   :   3 ==> 3

      Stop bit     1bit = 0    2bit = 1   :   0 ==> 0

      Change OK (Y/N) ? y

F1 Help   F2 Cls   F3 Send   F4 Receive File   F5 Line setting   F10 Menu
```

3.4 Operation of serial monitor command

User can only use command which stored at serial monitor. Serial monitor can execute to command when user type command and then CR (carriage return) key. If there is no any command at serial monitor, error message will be displayed with bell sound and serial monitor prompt will be displayed again.



```
8086 >?⊟
 HELP  COMMAND
 E segment : offset...................: Enter Data To Memory
 D segment : offset length............: Dump Memory Contents
 R [register name]..................: Register Display & Change
 M address1, length, address2........: Move Memory From 1 to 2
 F address, length, data..............: Fill Memory With Any Data
 L Return key.......................: Program Down Load
 G segment : offset...................: Execute Program
 T..................................: Program 1 step execute
```

1 Memory modify command.

```
         Segment  Offset
            ↓       ↓
8086 >E 0000:1000⏎
0000:1000 FF ? 11⏎
0000:1001 FF ? 22⏎
0000:1002 FF ? 33⏎
0000:1003 FF ? 44⏎
0000:1004 FF ? 55⏎
0000:1005 FF ? /  ⏎ ← (Offset decrement)
0000:1004 55 ? /  ⏎
0000:1003 44 ? .  ⏎  ← (Escaping command)
```

2 Memory display command.

```
         Segment  Offset
            ↓       ↓
8086 >D 0000:1000⏎
0000:1000 11 22 33 44 55 FF FF FF - FF FF FF FF FF FF FF FF    ."3DU...........
0000:1010 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF     ................
0000:1020 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF     ................
0000:1030 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF     ................
0000:1040 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF     ................
0000:1050 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF     ................
0000:1060 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF     ................
0000:1070 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF     ................
8086 >                                              Display the ASCII code to data
```

3 Memory fill command.

```
       Segment  Length Data
            ↓     ↓    ↓
8086 >F 1000 FF 1234⏎
```

☛ Verifying ?

```
8086 >D 0000:1000⏎
0000:1000 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:1010 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:1020 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:1030 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:1040 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:1050 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:1060 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:1070 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
8086 >D⏎
0000:1080 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:1090 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:10A0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:10B0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:10C0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:10D0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:10E0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:10F0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
```

4 Block move command.

The M command is used to move blocks of memory from one area to another.

```
          Segment  Length  Data
                ↓       ↓      ↓
8086 >M 1000 100 2000↵
```

☛ Resulting ?

```
8086 >D 2000↵
0000:2000 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:2010 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4

0000:2020 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:2030 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:2040 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:2050 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:2060 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:2070 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4


8086 >D↵
0000:2080 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:2090 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:20A0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:20B0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:20C0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:20D0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:20E0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
0000:20F0 12 34 12 34 12 34 12 34 - 12 34 12 34 12 34 12 34     .4.4.4.4.4.4.4.4
```

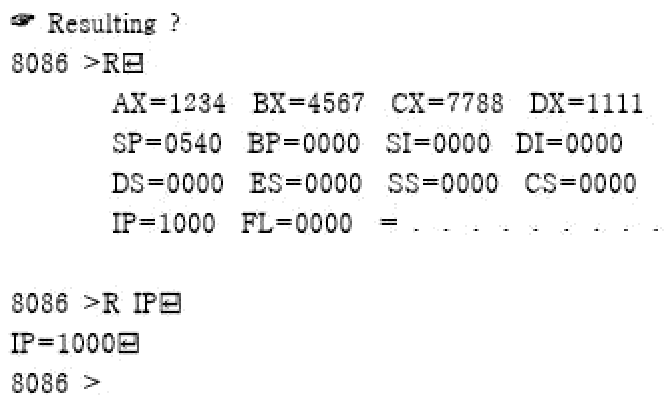
5 Display Registers command.

The R command is used to display the i8086 processor registers.

```
8086 >R↵
        AX=0000  BX=0000  CX=0000  DX=0000
        SP=0540  BP=0000  SI=0000  DI=0000
        DS=0000  ES=0000  SS=0000  CS=0000
        IP=1000  FL=0000  = . . . . . . . . .
```

☛ Individual register change

```
8086 >R AX↵
AX=0000  1234↵
BX=0000  4567↵
CX=0000  7788↵
DX=0000  1111↵
SP=0540↵
```



```
☞ Resulting ?
8086 >R↵
        AX=1234  BX=4567  CX=7788  DX=1111
        SP=0540  BP=0000  SI=0000  DI=0000
        DS=0000  ES=0000  SS=0000  CS=0000
        IP=1000  FL=0000  = . . . . . . . . . .

8086 >R IP↵
IP=1000↵
8086 >
```

3.5 Program:

```
CODE SEGMENT
        ASSUME CS: CODE, DS: CODE
                AX,
        MOV     0001H
                AX,
        ADD     6789H
        STC
                AX,
        ADC     0488H
        ;
                AX,
        SUB     156FH
        STC
                AX,
        SBB     080FH
        ;
        MOV AX, 00FEH
        INC AL
        DEC AL
        CBW
        NEG AL
        ;
        MOV
        AL,
        F0H
        MOV
        BL,
        11H
        MUL BL
        ;
        MOV AX,
        F000H
        MOV BX,
        1234H
        IMUL BX
        ;
        MOV AX,
        00F0H
        MOV BL,
        10H
        DIV BL
        ;
        MOV AX, −205
        MOV BL, 4
        IDIV   BL
        ;
        HLT
CODE ENDS
END
```

3.6 Experiment Requirements:

1. 8086 microprocessor kit.
2. Assembler “MASM” and loader “LOD186”.
3. WinComm.

3.7 Experiment Procedures:

1. Write the above program in notepad and save the file as “filename.asm”. Place this file in the folder where “masm.exe” exists.

2. Go to command prompt and execute “masm.exe”. You will see the following message Microsoft (R) Macro Assembler Version 5.10

```
Copyright (C) Microsoft Corp 1981, 1988. All right reserved.

Source filename [.ASM]:
```

3. Follow the procedure given below to prepare machine code for your program:

```
Source filename [.ASM]: filename Press ENTER

Object filename [C: file name.OBJ]: Press ENTER

Source listing [NUL.LST]: filename Press ENTER

Cross reference [NUL.CRF]: Press ENTER
```

4. Execute “LOD186.exe”. You will see the following message Paragon LOD186 Loader-Version 4.0h

```
Copyright (C) 1983 - 1986 Microtec Research Inc.
ALL RIGHT RESERVED.

Object/Command File [.OBJ]:
```

5. Follow the procedure given below to prepare HEX (ABS) file for your program:

```
                          [.OBJ]:
Object/Command File       filename          Press ENTER

Output Object File        [C:filename.ABS]: Press ENTER
                          [C:NUL.MAP]
Map Filename              :                 Press ENTER

**LOAD COMPLETE
```

6. Turn on the 8086 microprocessor kit

7. Open the “Wincomm” window. Press “L” then “Enter”. You will see the following message:

```
**      Serial Monitor 1.0 **
**      Midas 335-0964/5 **

8086 >L Press ENTER

Down load start!!
```

8. Strike PgUp or F3 key of your keyboard. A new window will appear. Locate the “filename.ABS” file and open it.

9. You will observe that file download has started. A message like the following one will be shown:

```
:14100000B800008ED88EC0BB00208B078A6F028A4F038BEBB6
:101014003E8B5604268B76068B7E088B1E0A20CCCC
:0E20000012345678ABCDF0146853B1C41020E2
:00000001FF
OK completed!!
```

10. After loading the program, execute it in single step mode. Fill up the data table and verify the results.

3.8 Data Table:

Offset Address	**Instruction/ Mnemonics**	**AX**	**BX**	**DX**	**Set Flag Bits**	**IP Address**
	INITIAL STATUS					
	MOV AX, 0001H					
	ADD AX, 6789H					
	STC					
	ADC AX, 0488H					
	SUB Ax, 156FH					
	STC					
	SBB AX, 080FH					
	MOV AX, 00FEH					
	INC AL					
	DEC AL					
	CBW					
	NEG AL					
	MOV AL, F0H					
	MOV BL, 11H					
	MUL BL					
	MOV AX, F000H					
	MOV BX, 1234H					
	IMUL BX					
	MOV AX, 00F0H					
	MOV BL, 10H					

	DIV BL					
	MOV AX, −205					
	MOV BL, 4					
	IDIV BL					
	HLT					

Questions:

a) What is the function of INC BYTYE PTR [BX] instruction?
b) What will occur after CMP CL, BL instruction?
c) Write instructions to multiply 3 with 4.

Quiz

Question 1.

INC BYTE PTR[BX] instruction adds 1 to the byte content of stack segment memory location addressed by BX.

(a) True

(b) False

Question 2.

Which flag bits may change after addition or subtraction according to result.

(a) Carry Flag

(b) Trap Flag

(c) Auxiliary Carry

(d) Parity Flag

(e) All of them

Question 3.

Match the items on the left with the items on the right for the following instruction
CMP CL, BL

CF=0 ZF=1 SF=0	CL>BL
CF=0 ZF=0 SF=0	CL=BL
CF=1 ZF=0 SF=1	CL<BL

Question 4.

In 8-bit multiplication, multiplicand is always stays in .... register.

Question 5.

Which will be the content of AH after executing following instructions?
MOV AL, F0H
CBW

(a) 0b11111111

(b) 0b10101010

(c) 0b11110000

(d) 0b00000000

(e) 0b00001111

Question 6.

CWD copies the sign of a word in to all the bits of the ……….......... register.

Question 7.

Which one of the followings is a subtraction instruction?

(a) TEST

(b) CBW

(c) NEG

(d) CMP

(e) DAA

Question 8.

For a 16-bit division, match the items on the left with the items on the right.

Remainder appears in
Dividend stored in
Divider stored in
Quotient appears in

DX register
AX register
any 16-bit register or memory.
DX-AX register

Question 9.

Which of the flag bits are changes in CMP instructions?

(a) Overflow Flag

(b) Carry Flag

(c) Sign Flag

(d) Zero Flag

(e) Interrupt Flag

Question 10.

Suppose, 1001 is a 4-bit sign number. Which one of the following will be the decimal equivalent of this number?

(a) -1

(b) -3

(c) -6

(d) -7

Question 11.

Which instruction set is correct to multiply 3 with 4?

(a)
```
MOV CL,3
MOV BL, 2
MUL  CL ,BL
```

(b)
```
MOV AL,3
MOV BL, 2
MUL  AL ,BL
```

(c)
```
MOV AL,3
MOV BL, 2
MUL BL
```

(d)
```
MOV AL,3
MOV BL, 2
MUL AL
```

Question 12.

Which one is not an arithmetic Instructions?

(a) CMP

(b) DEC

(c) CBW

(d) SBB

(e) TEST

University of Global Village(UGV),Barishal

Department of Electrical and Electronic Engineering (EEE)

EEE-314: Microprocessor and Interfacing Sessional

Experiment No. 04: Logic operations in assembly language.

Objective:

To load programs containing logic instructions to MDA-8086, execute the program in single step mode and verify the results.

Logical instructions:

Logical instructions include NOT, AND, OR, XOR, TEST etc. instructions. There job is to compare the data values and make results according to logic specified. For example,

```
MOV BX, 30H ; In binary 110000
NOT BX ; In binary 001111
```

This code takes BX value and then complements all the bits and stores the new value to BX. So it stores 0F value in BX after executing NOT operation. For another example,

```
MOV BX, 70H ; In binary 1110000
MOV CX, 40H ; In binary 1000000
AND CX, BX ; In binary 1000000
```

AND operation performs bit by bit AND operation and then stores the value in first operand. In upper code CX holds the final result.

```
MOV BX, 70H ; In binary 1110000
MOV CX, 40H ; In binary 1000000
OR CX, BX ; In binary 1110000
```

OR operation performs bit by bit OR operation and then stores the value in first operand. In upper code CX holds the final result. Similar case happens for XOR and it is given below,

```
MOV BX, 70H ; In binary 1110000
MOV CX, 40H ; In binary 1000000
XOR CX, BX ; In binary 0110000
```

Test operation is a little different from AND operation. It performs bit by bit AND operation but it does not change any operands value.

```
MOV BX, 70H ; In binary 1110000
MOV CX, 40H ; In binary 1000000
TEST CX, BX ; In binary CX value is 1000000
```

All the logical instructions stated above upgrades all the flag register values except
AF register. NOT command does not effect any flags. How flags are affected is stated below.

```
MOV BX, 70H ; In binary 1110000
MOV CX, 40H ; In binary 1000000
AND CX, BX ; In binary 1110000
```

After this operation Zero Flag is 0 (ZF = 0; as the value of CX is not 0), Carry Flag is
0 (CF = 0; as there is no carry), Parity Flag is 0 (PF = 0; as there are odd number of
1's), Sign Flag is 0 (SF = 1), Overflow Flag is 0 (OF = 0; as there is no overflow). In this all the flags can be determined.

Do not confuse yourself with semicolon given after each line in assembly codes above. Comments are written after semi colon ‘;’ in assembly language.

Program:

```
CODE SEGMENT
        ASSUME CS:CODE, DS:CODE
        ORG 1000H
        MOV AX, 1027H
        MOV BX, 5A27H
        MOV CX, 54A5H
        OR AX, BX
        XOR AX, CX
        NOT AX
        TEST CX, BX
        AND CX, AX
        HLT
CODE ENDS
END
```

Experiment Requirements:

1. 8086 microprocessor kit.
2. Assembler “MASM” and loader “LOD186”.
3. WinComm.

Experiment Procedures:

1. Write the program in notepad and save the file as “filename.asm”. Place this file in the folder where “masm.exe” exists.

2. Go to command prompt and execute “masm.exe”. You will see the following message

```
        Microsoft (R) Macro Assembler Version 5.10
        Copyright (C) Misrosoft Corp 1981, 1988. All right reserved.

        Source filename [.ASM]:
```

3. Follow the procedure given below to prepare machine code for your program:

```
        Source filename [.ASM]: filename Press ENTER

        Object filename [C:filename.OBJ]: Press ENTER

        Source listing [NUL.LST]: filename Press ENTER

        Cross reference [NUL.CRF]:  Press ENTER
```

4. Execute “LOD186.exe”. You will see the following message

```
        Paragon LOD186 Loader-Version 4.0h
        Copyright (C) 1983 - 1986 Microtec Research Inc.
        ALL RIGHT RESERVED.

        Object/Command File [.OBJ]:
```

5. Follow the procedure given below to prepare HEX (ABS) file for your program:

```
        Object/Command File         [.OBJ]: filename Press ENTER
```

```
Output Object File        [C:filename.ABS]:  Press ENTER

Map Filename              [C:NUL.MAP]:  Press ENTER

**LOAD COMPLETE
```

6. Turn on the 8086 microprocessor kit

7. Open the “Wincomm” window. Press “L” then “Enter”. You will see the following message:

```
** Serial Monitor 1.0 **
** Midas 335-0964/5 **

8086 >L Press ENTER

Down load start !!
```

8. Strike PgUp or F3 key of your keyboard. A new window will appear. Locate the “filename.ABS” file and open it.

9. You will observe that file download has started. A message like the following one will be shown:

```
:14100000B800008ED88EC0BB00208B078A6F028A4F038BEBB6
:101014003E8B5604268B76068B7E088B1E0A20CCCC
:0E20000012345678ABCDF0146853B1C41020E2
:00000001FF

OK completed !!
```

10. After loading the program, execute it in single step mode. Fill up the data table and verify the results.

Data Table:

Offset Address	Instruction / Mnemonics	AX	BX	CX	DX	Set Flag Bit(s)	IP
	Initial Status						
	MOV AX, 1027H						
	MOV BX, 5A27H						
	MOV CX, 54A5H						
	OR AX, BX						
	XOR AX, CX						
	NOT AX						
	TEST CX, BX						
	AND CX, AX						

Report:

1. Discuss the effect of each instruction/ mnemonics that is used in this program.

References:

1. User’s manual of MDA-8086 microprocessor kit, Midas Engineering, www.midaseng.com.

2. “Assembly Language Programming and Organization of the IBM PC”, Ytha Yu and Charles Marut, Mitchell McGraw-Hill.

Prepared by---------
Md. Rifat Shahriar
Lecturer/ Dept. of EEE/ IIUC

University of Global Village(UGV),Barishal Department of Electrical and Electronic Engineering (EEE)

EEE-314: Microprocessor and Interfacing Sessional

Experiment No. 05: Program control instructions in assembly language.

Objectives:

To load programs containing program control instructions to MDA-8086, execute the program in single step mode and verify the results.

Some program control instructions are discussed below.

JUMP Commands:

Sometimes it is necessary to go from one line of program to another line without executing some intermediate lines. For this Jump commands are used. We can explain this with a simple example.

```
        MOV AX, 3254H
        MOV BX, 1F4BH
        MOV CX, 412AH
        ADD AX, CX
        JMP S7
        SUB AX, BX
S7:     AND AX, BX
        HLT
```

In this example S5 is a level. As we can see in fifth line JMP command is used. It makes the program to go from fifth line to S7 level that is seventh line. So sixth line will not be executed. There are two types of Jump commands. These are (i) Conditional jump and (ii) Unconditional Jump. Previous example is an unconditional jump. Conditional Jumps are like if statements. If some flags are affected only then these jump instructions executed. We can look at the following example,

```
        MOV AX, 125BH
        MOV BX, 125BH
        MOV CX, 412AH
        SUB AX, BX
        JZ S5
        DIV BX
S5:     AND AX,CX
        HLT
```

Clearly observe the code. In fourth line subtraction operation is performed. As both AX and BX have same value. Their subtracted value is 0. So ZF is set to 1. In fifth line JZ S5 is written. It means if ZF = 1 then go to S5, otherwise continue. As ZF = 1, program moves to seventh line. This is a conditional Jump. Some other conditional jumps are listed below.

MNEMONIC	CONDITION TESTED	"JUMP IF ... "
JA/JNBE	(CF or ZF)=O	above/not below nor equal
JAE/JNB	CF=O	above or equal/not below
JB/JNAE	CF=l	below/not above nor equal
JBE/JNA	(CF or ZF)= 1	below or equal/not above
JC	CF=l	carry
JE/JZ	ZF=l	equal/zero
JG/JNLE	((SF xor OF) or ZF) = 0	greater/not less nor equal
JGE/JNL	(SF xor OF)=O	greater or equal/not less
JL/JNGE	(SF xor OF) = 1	less/not greater nor equal
JLE/JNG	((SF xor OF) or ZF) = 1	less or equal/not greater
JNC	CF=O	not carry
JNE/JNZ	ZF=O	not equal/not zero
JNO	OF=O	not overflow
JNP/JPO	PF=O	not parity/parity odd
JNS	SF=O	not sign
JO	OF=l	overflow
JP/JPE	PF= 1	parity/parity equal
JS	SF= 1	sign
JCXZ	CX=0	Register cx = 0

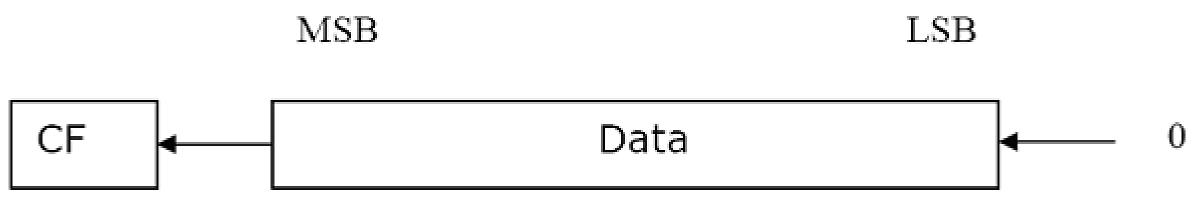
Note: "above" and "below" refer to the relationship of two unsigned values; "greater" and "less" refer to the relationship of two signed values.

Program 1:

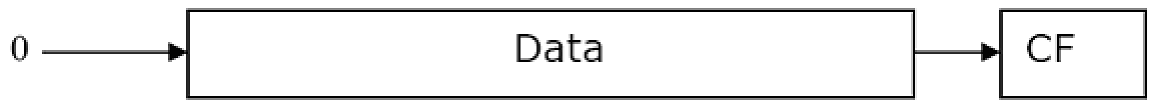
```
CODE SEGMENT
            ASSUME CS:CODE, DS:CODE
            ORG 1000H
            MOV AX, 7A24H
            MOV BX, 95A3H
            ADD AX, BX
            JC IIUC
EEE:        OR AX, 23H
            JNZ LAST
IIUC:       MOV CX, 0FC7H
            SUB AX,CX
            JZ EEE
LAST:       HLT
CODE ENDS
END
```

Shift and Rotate command:

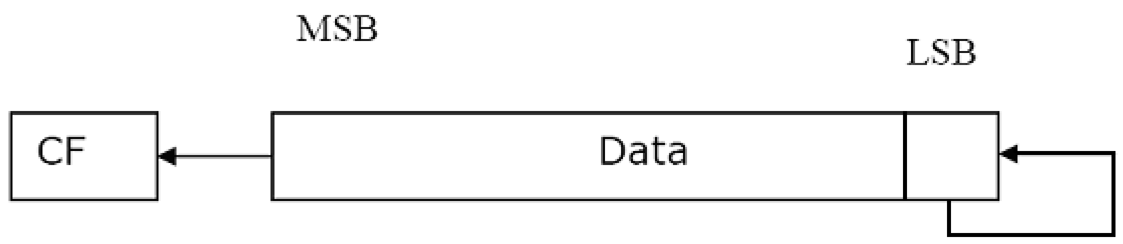
Shift and Rotate commands are used to convert a number to another form where some bits are shifted or rotated. Basic difference between shift and rotate is shift command makes “fall of” bits at the end of register. Where rotate command makes “Wrap around” at the end of the register. There are both arithmetic (SAL and SAR) and logical (SHL and SHR) shift instructions. Graphical operations for these commands are shown below.



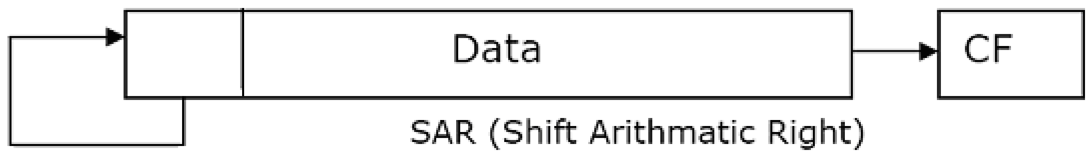
SHL (Shift Logical Left)



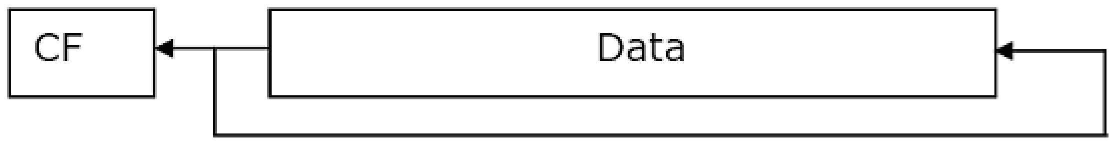
SHR (Shift Logical Right)



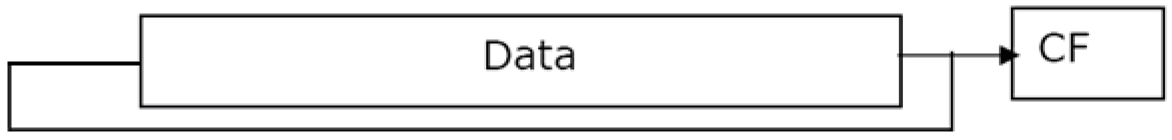
SAL (Shift Arithmatic Left)



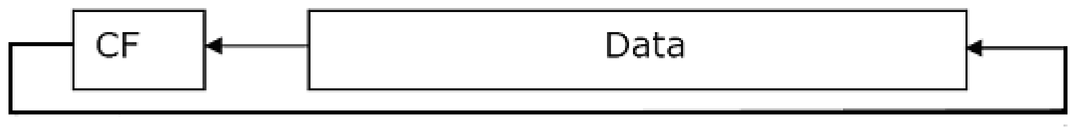
SAR (Shift Arithmatic Right)



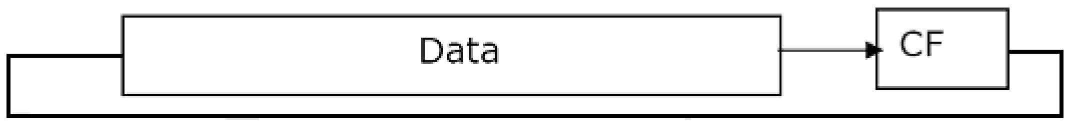
ROL (Rotate Left)



ROR (Rotate Right)



RCL (Rotate Through Carry Left)



RCR (Rotate Through Carry Right)

Some simple codes can be given to clarify the idea.

```
MOV CL,03H ;
MOV AX,02F3H ; In binary 0000 0010 1111 0011
SHR AX,CL ; In binary 0000 0000 0101 1110
```

In this procedure, SHR commands inserts 0's from left side. Each time a 0 is inserted right most bit is vanished from register content.

```
MOV CL,03H ;
MOV AX,82F3H ; In binary 1000 0010 1111 0011
SAR AX,CL ; In binary 1111 0000 0101 1110
```

In this procedure, SAR command inserts MSB content from left side. Each time it is inserted right most bit is vanished from register content.

```
MOV CL,03H ;
MOV AX,82F3H ; In binary 1000 0010 1111 0011
ROR AX,CL ; In binary 0111 0000 0101 1110
```

In this case, ROR instruction picks up the LSB and inserts it as MSB and so on.

Program 2:

```
CODE SEGMENT
        ASSUME CS:CODE, DS:CODE
        ORG 1000H
        MOV AX, 0055H
        MOV DX, 0505H
        MOV CL, 3
        SAL AX, CL
        SAR DX, CL
        MOV CL, 2
        ROR AX, CL
        ROL DX, CL
        STC
        RCL AL, CL
        RCR DX, CL
        HLT
CODE ENDS
END
```

Experiment Requirements:

1. 8086 microprocessor kit.
2. Assembler “MASM” and loader “LOD186”.
3. WinComm.

Experiment Procedures:

1. Write the program 1 in notepad and save the file as “filename.asm”. Place this file in the folder where “masm.exe” exists.

2. Go to command prompt and execute “masm.exe”. You will see the following message

```
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Misrosoft Corp 1981, 1988. All right reserved.

Source filename [.ASM]:
```

3. Follow the procedure given below to prepare machine code for your program:

```
Source filename [.ASM]: filename Press ENTER

Object filename [C:filename.OBJ]: Press ENTER

Source listing [NUL.LST]: filename Press ENTER

Cross reference [NUL.CRF]:  Press ENTER
```

4. Execute “LOD186.exe”. You will see the following message

```
Paragon LOD186 Loader-Version 4.0h
Copyright (C) 1983 - 1986 Microtec Research Inc.
ALL RIGHT RESERVED.

Object/Command File [.OBJ]:
```

5. Follow the procedure given below to prepare HEX (ABS) file for your program:

```
Object/Command File        [.OBJ]: filename Press ENTER

Output Object File         [C:filename.ABS]:  Press ENTER

Map Filename               [C:NUL.MAP]:  Press ENTER

**LOAD COMPLETE
```

6. Turn on the 8086 microprocessor kit

7. Open the “Wincomm” window. Press “L” then “Enter”. You will see the following message:

```
** Serial Monitor 1.0 **
** Midas 335-0964/5 **

8086 >L Press ENTER

Down load start !!
```

8. Strike PgUp or F3 key of your keyboard. A new window will appear. Locate the “filename.ABS” file and open it.

9. You will observe that file download has started. A message like the following one will be shown:

```
:14100000B800008ED88EC0BB00208B078A6F028A4F038BEBB6
:101014003E8B5604268B76068B7E088B1E0A20CCCC
:0E20000012345678ABCDF0146853B1C41020E2
:00000001FF
```

OK completed !!
 10. After loading the program, execute it in single step mode. Fill up the data table and verify the results.
11. Follow procedure 1 to 10 for program 2.

Data Table:

Program 1:

Offset Address	Instruction / Mnemonics	AX	BX	CX	DX	Set Flag Bit(s)	IP
	Initial Status						
	MOV AX, 7A24H						
	MOV BX, 95A3H						
	ADD AX, BX						
	JC IIUC						
	EEE: OR AX, 23H						
	JNZ LAST						
	IIUC: MOV CX, 0FC7H						
	SUB AX,CX						
	JZ EEE						
	LAST: HLT						

Program 2:

Offset Address	Instruction / Mnemonics	AX	BX	CX	DX	Set Flag Bit(s)	IP
	Initial Status						
	MOV AX, 0055H						
	MOV DX, 0505H						
	MOV CL, 3						
	SAL AX, CL						
	SAR DX, CL						
	MOV CL, 2						
	ROR AX, CL						
	ROL DX, CL						
	STC						
	RCL AL, CL						
	RCR DX, CL						

Report:

1. Discuss the effects of each instruction/ mnemonics that are used in this program.

References:

1. User's manual of MDA-8086 microprocessor kit, Midas Engineering, www.midaseng.com.

2. "Assembly Language Programming and Organization of the IBM PC", Ytha Yu and Charles Marut, Mitchell McGraw-Hill.

Prepared by---------
Md. Rifat Shahriar
Lecturer/ Dept. of EEE/ IIUC

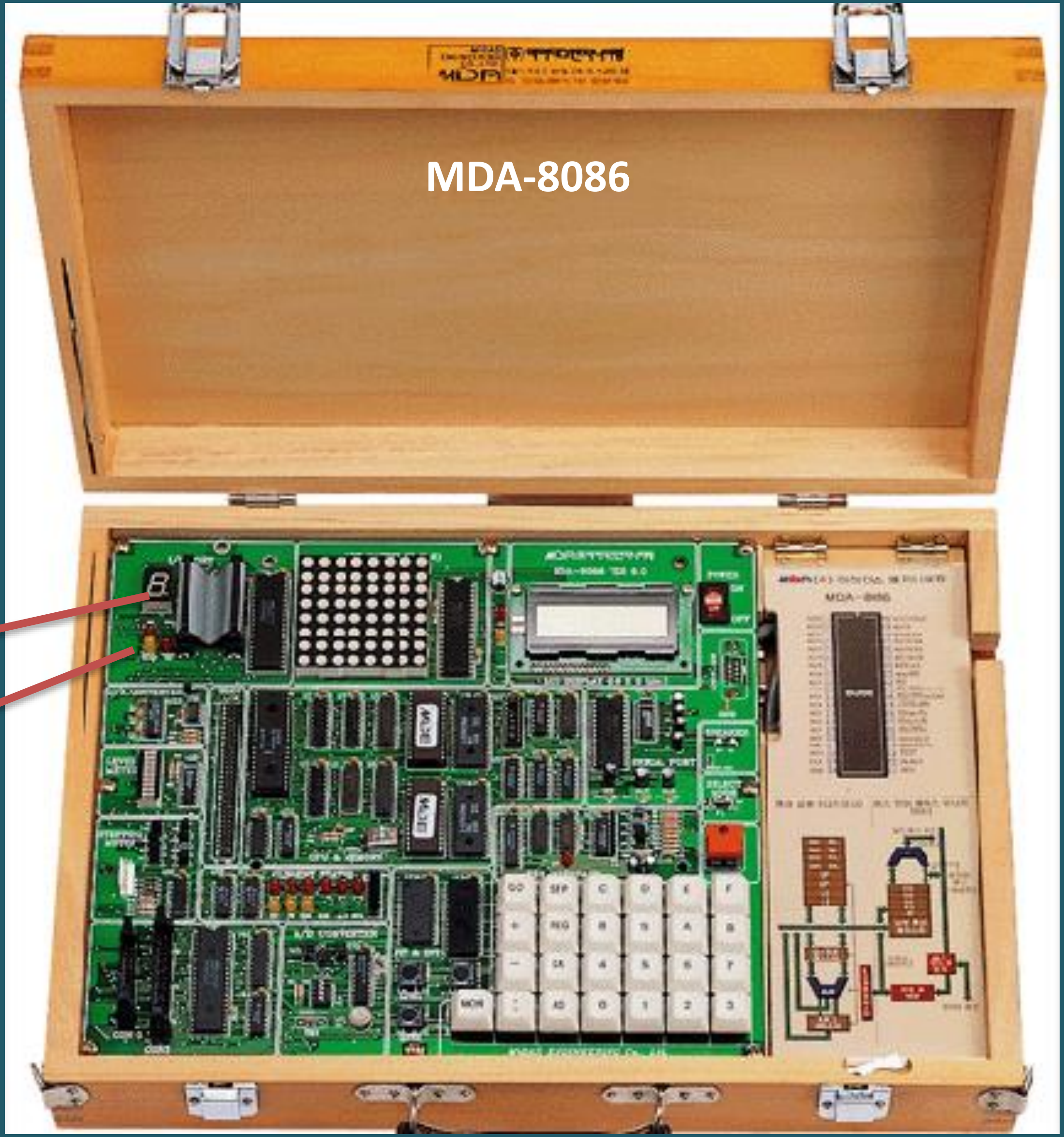
EEE 314
Microprocessor and Interfacing Sessional

Experiment-6

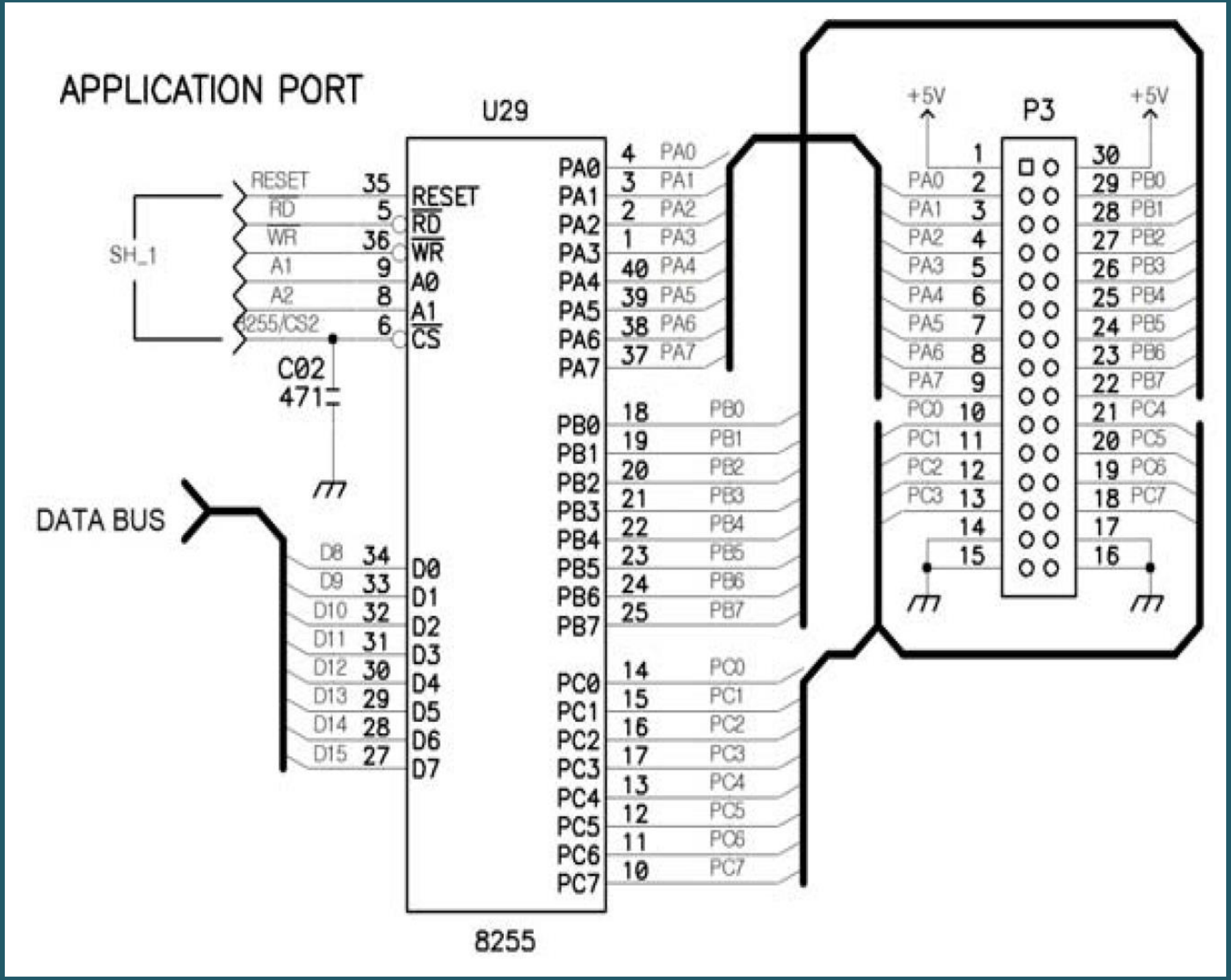
Interfacing of LED and 7-segment display with 8086 microprocessor.

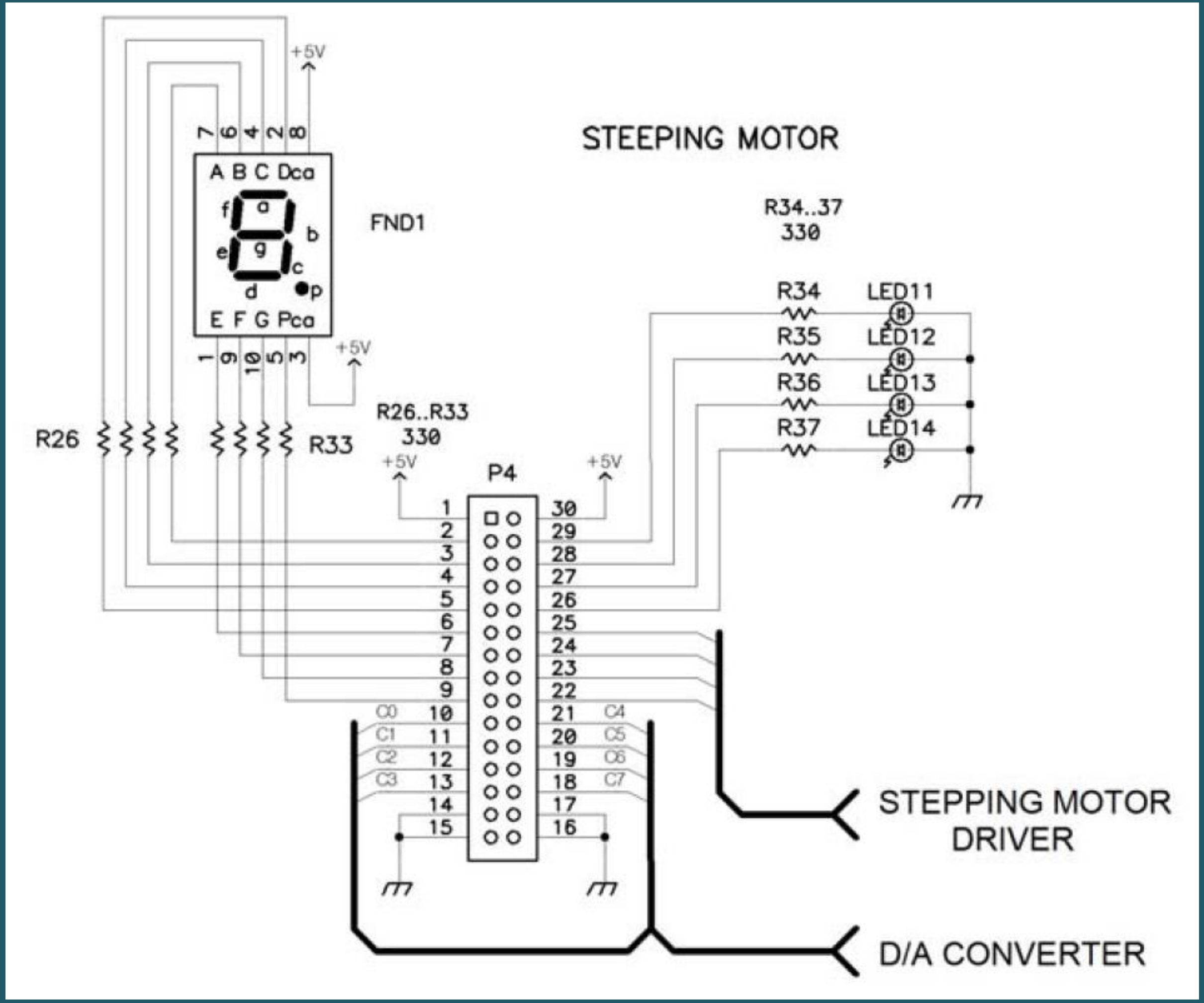
Objectives

1) To interface LED with 8086 microprocessor by 8255 PPI.
2) To interface 7-segment display with 8086 microprocessor by 8255 PPI.

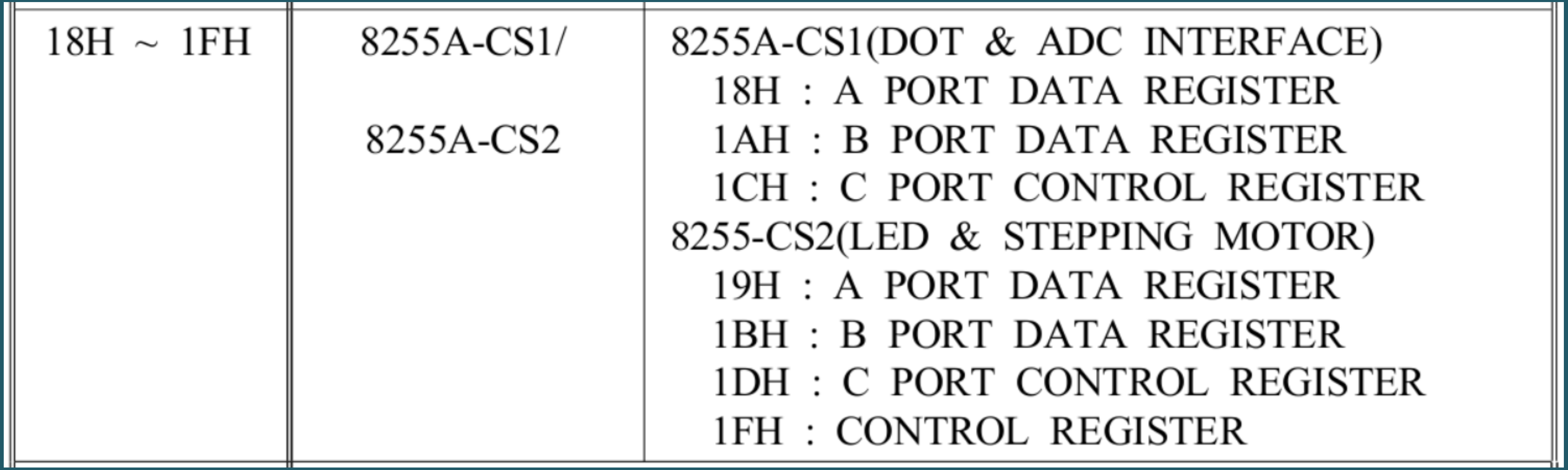


Schematic of LED and 7-segment display interface with 8086



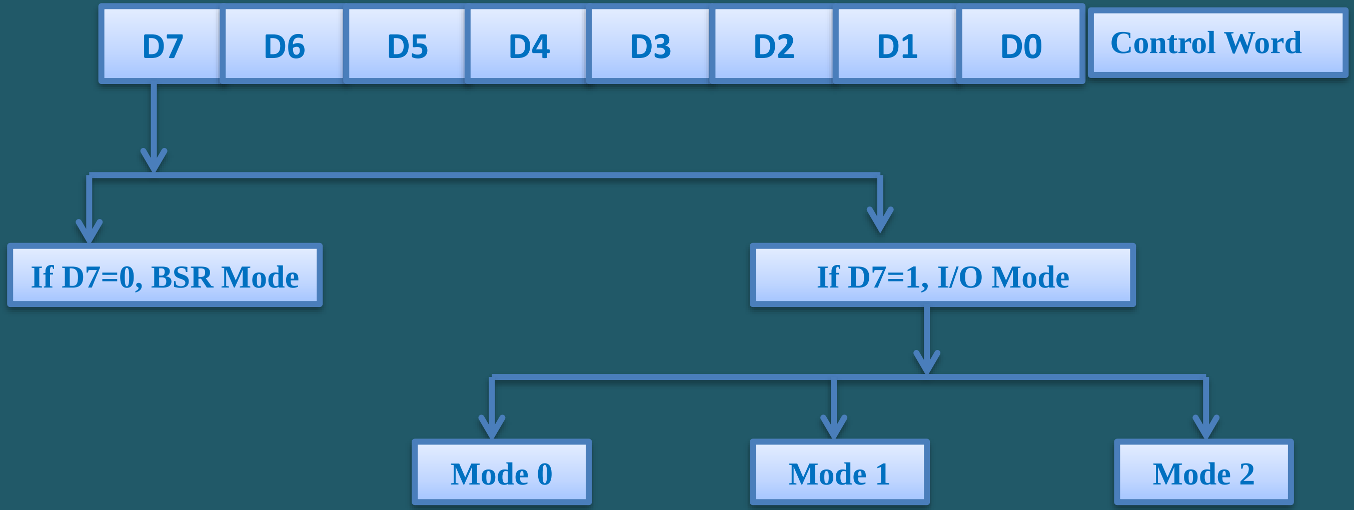


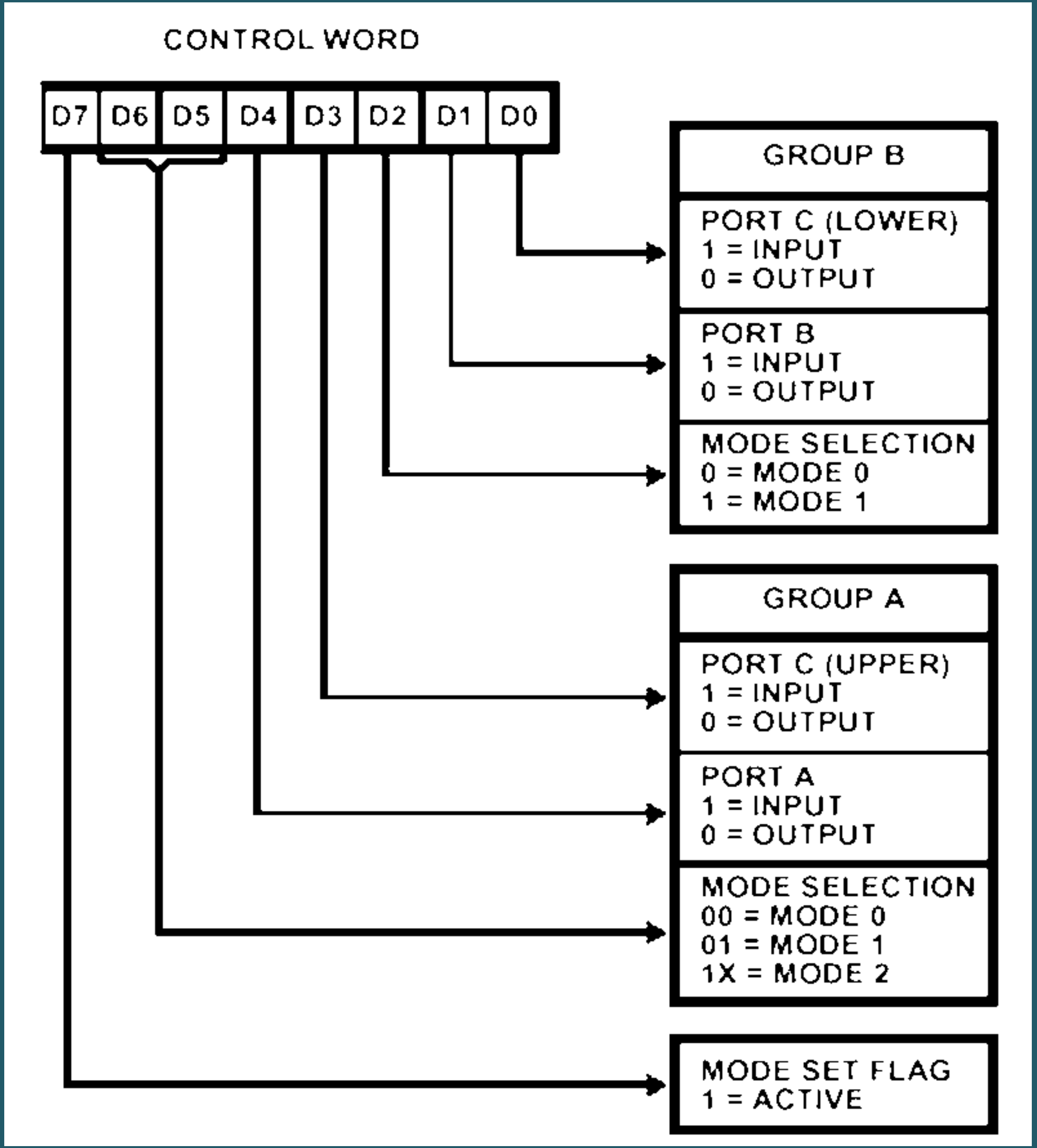
Address of the internal registers of 8255 (In MDA-8086)



18H ~ 1FH	8255A-CS1/ 8255A-CS2	8255A-CS1(DOT & ADC INTERFACE) 18H : A PORT DATA REGISTER 1AH : B PORT DATA REGISTER 1CH : C PORT CONTROL REGISTER 8255-CS2(LED & STEPPING MOTOR) 19H : A PORT DATA REGISTER 1BH : B PORT DATA REGISTER 1DH : C PORT CONTROL REGISTER 1FH : CONTROL REGISTER

Control Word for LED Interfacing





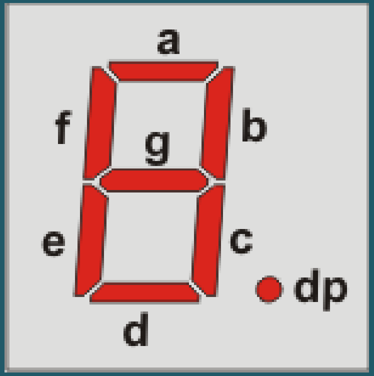
LED Blinking

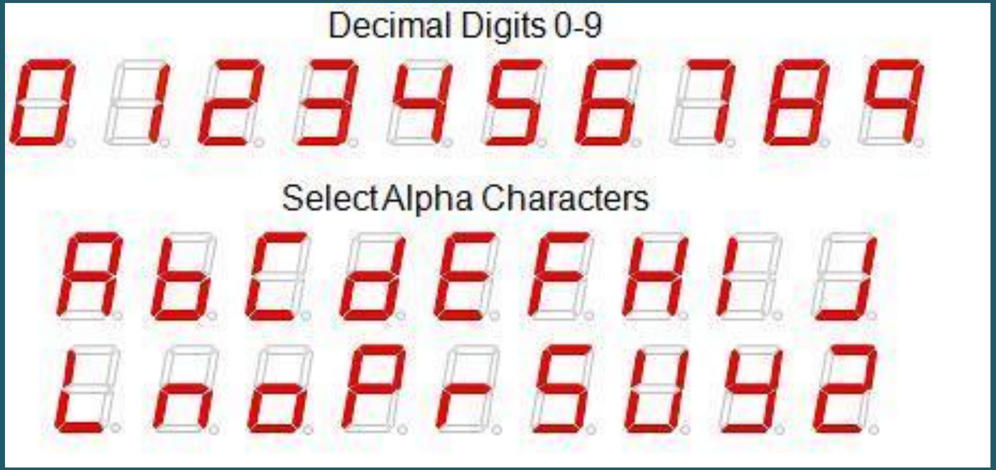
```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
        ;
PPIC_C  EQU     1FH
PPIC    EQU     1DH
PPIB    EQU     1BH
PPIA    EQU     19H
        ;
        ORG     1000H
        MOV     AL,10000000B
        OUT     PPIC_C,AL
        ;
L1:     MOV     AL,00000001B
        OUT     PPIB,AL
        CALL    DELAY
        MOV     AL,00000000B
        OUT     PPIB,AL
        CALL    DELAY
        JMP     L1
```

```
DELAY:  MOV     CX,1111111111111111B
TIMER1: NOP
        NOP
        NOP
        NOP
        LOOP    TIMER1
        RET
        ;
CODE    ENDS
        END
```

7-segment Display

The 7-segment display, also written as "seven segment display", consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a **numerical digit (both Decimal and Hex) to be displayed**. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal point, (DP) when two or more 7-segment displays are connected together to display numbers greater than ten.

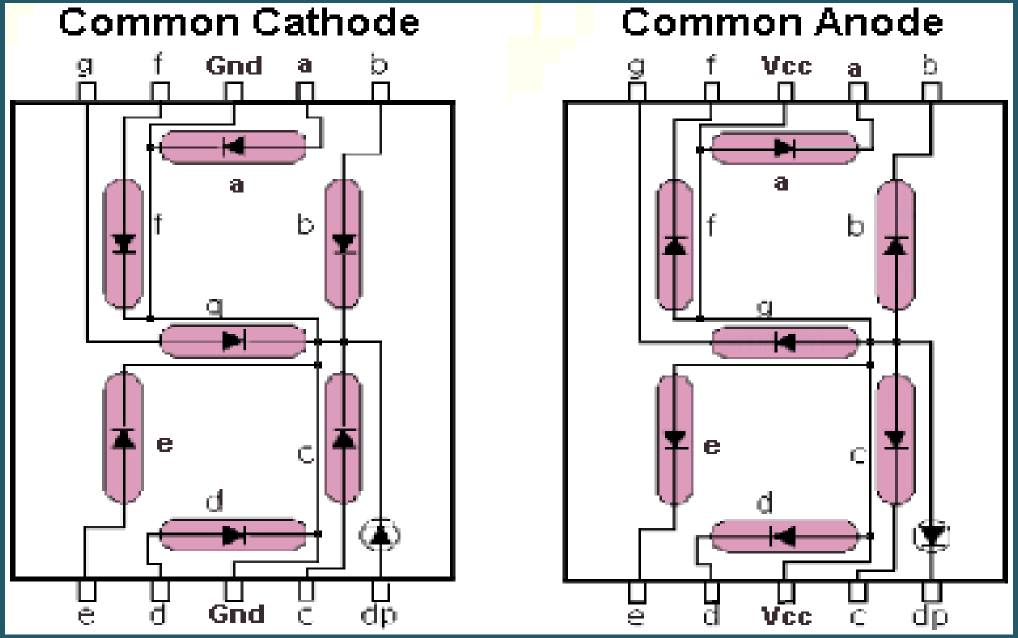




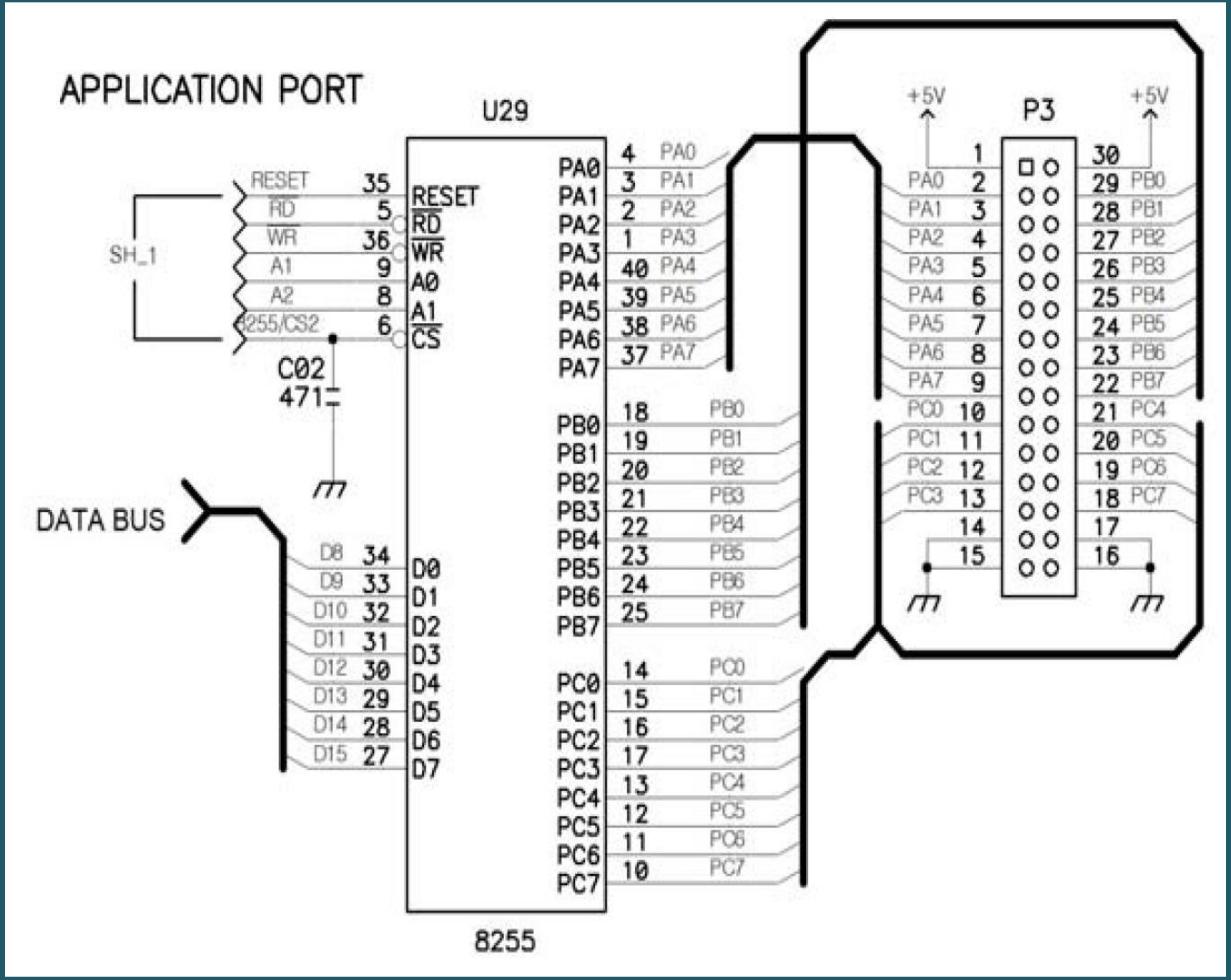
Common Anode and Common Cathode 7-segment Display

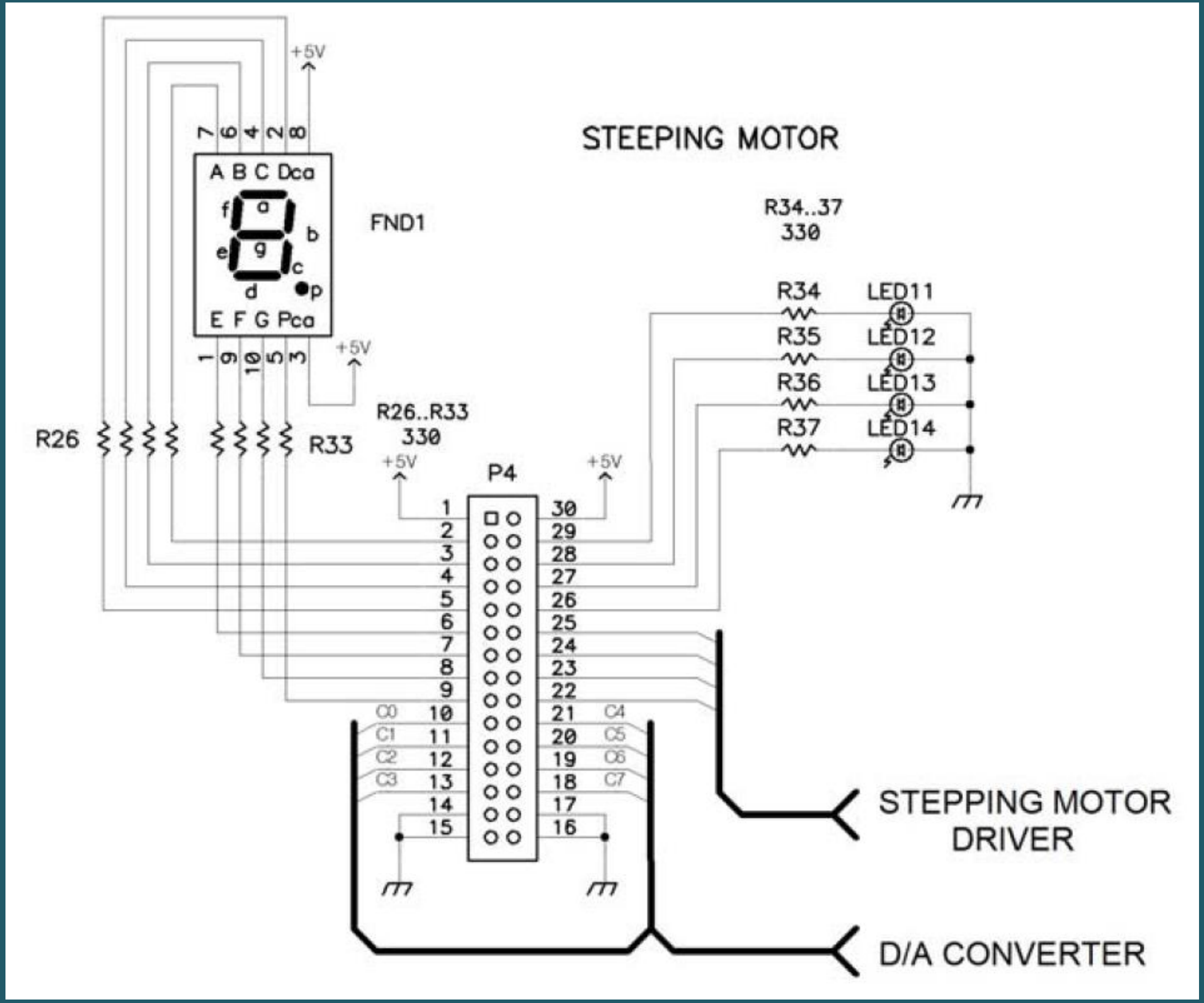
The Common Cathode (CC) – In the common cathode display, all the cathode connections of the LED segments are joined together to logic “0” or ground. The individual segments are illuminated by application of a “HIGH”, or logic “1”

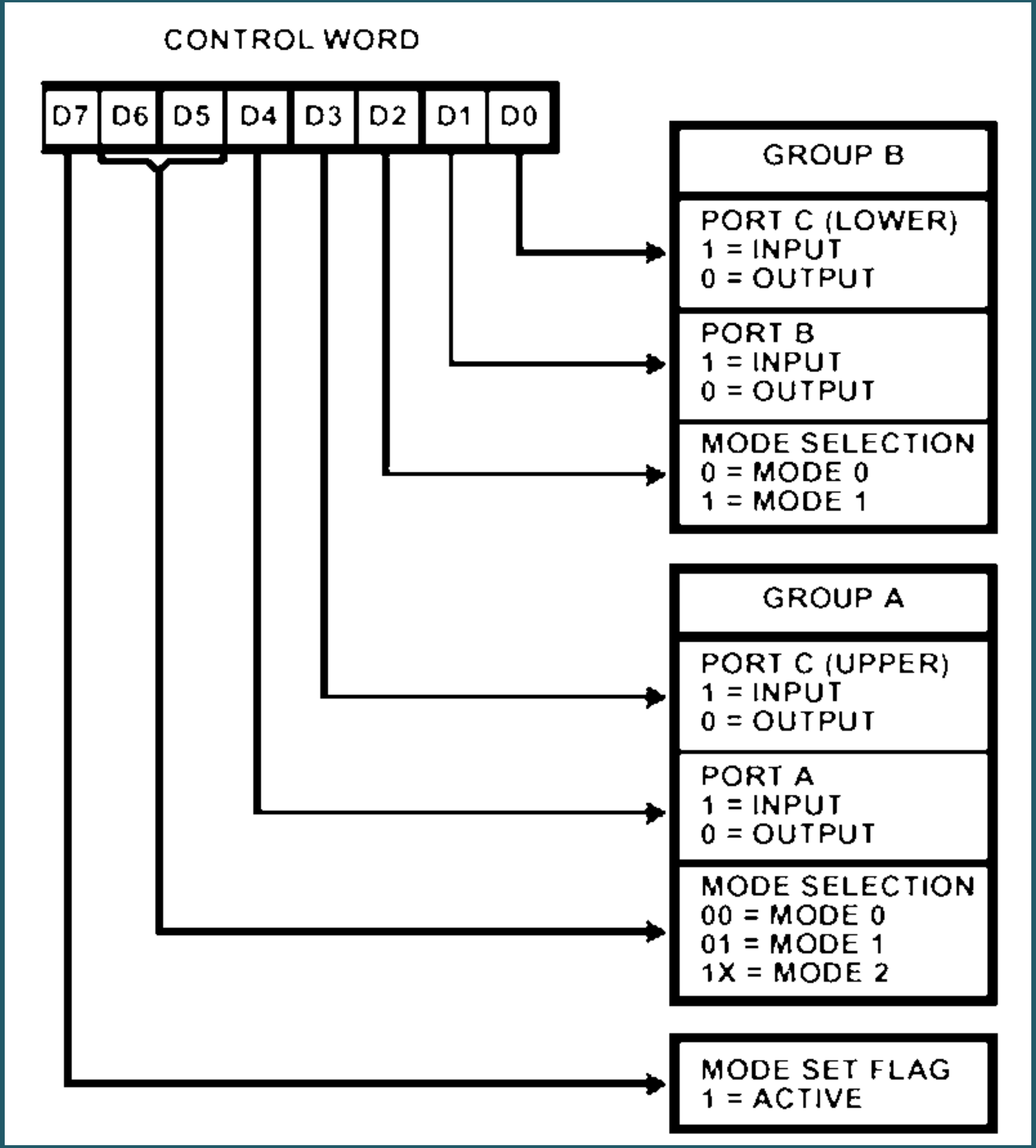
The Common Anode (CA) – In the common anode display, all the anode connections of the LED segments are joined together to logic “1”. The individual segments are illuminated by applying a ground, logic “0” or “LOW”



Schematic of LED and 7-segment display interface with 8086






CONTROL WORD
D7
D6
D5
D4
D3
D2
D1
D0
GROUP B
PORT C (LOWER)
1 = INPUT
0 = OUTPUT
PORT B
1 = INPUT
0 = OUTPUT
MODE SELECTION
0 = MODE 0
1 = MODE 1
GROUP A
PORT C (UPPER)
1 = INPUT
0 = OUTPUT
PORT A
1 = INPUT
0 = OUTPUT
MODE SELECTION
00 = MODE 0
01 = MODE 1
1X = MODE 2
MODE SET FLAG
1 = ACTIVE

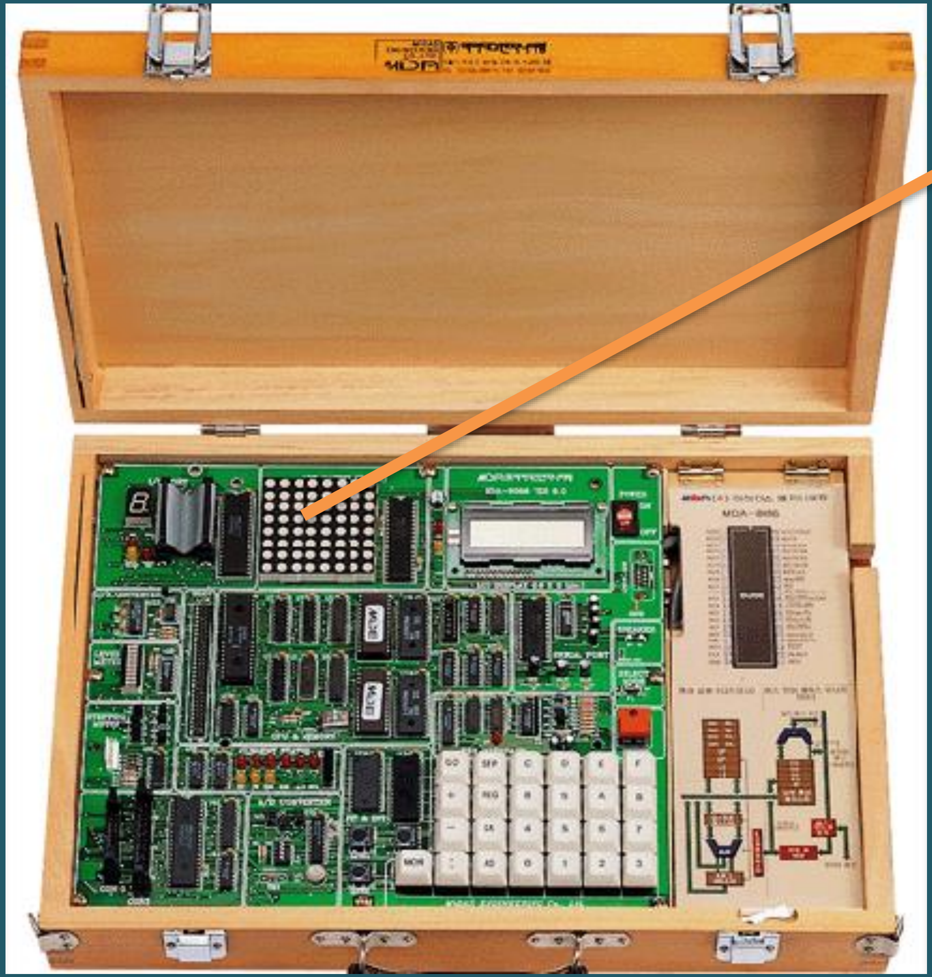
EEE 314
Microprocessor and Interfacing Sessional

Experiment-7

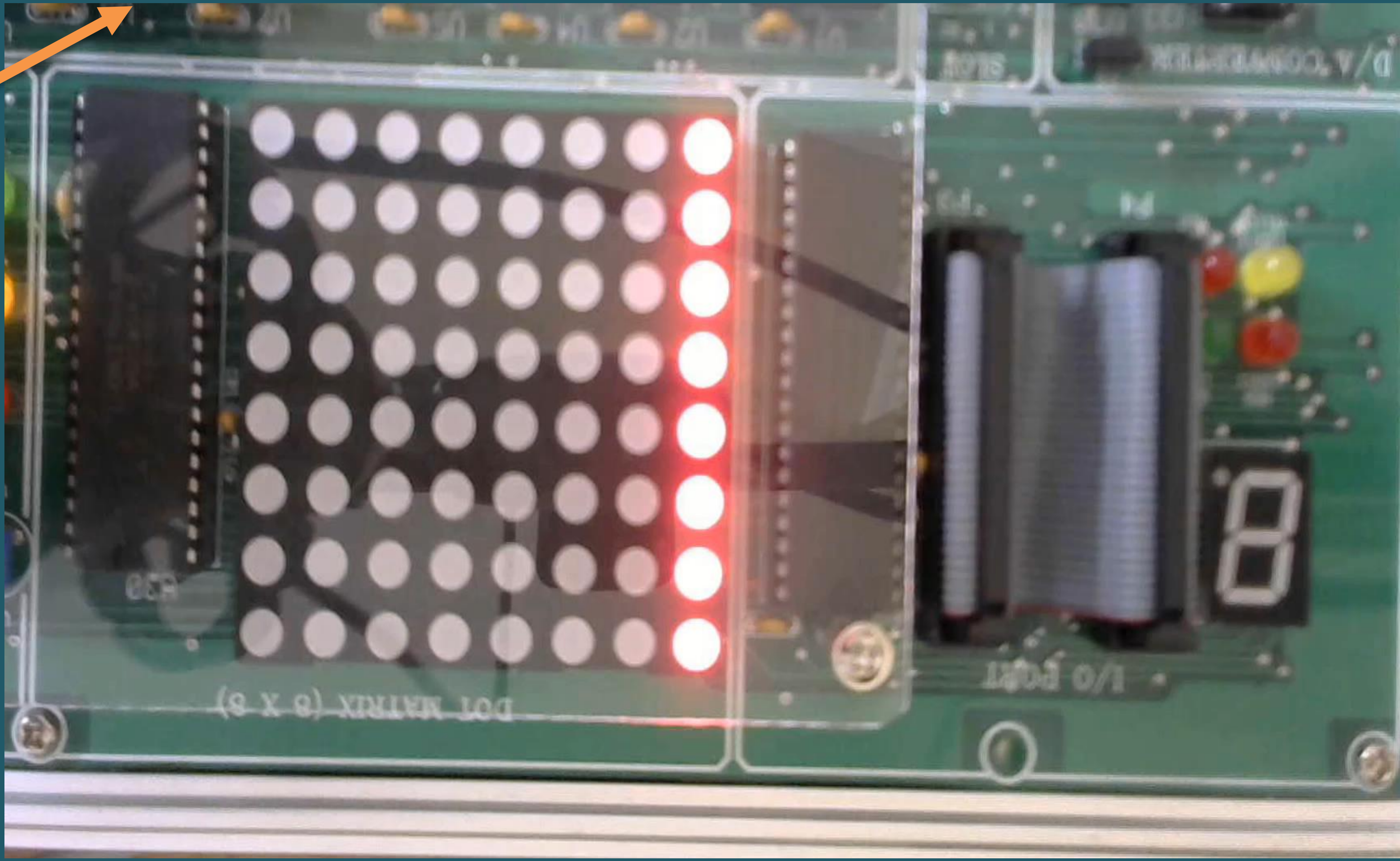
Interfacing of Dot-Matrix LED display with 8086 microprocessor.

Objective-1

To interface Dot-Matrix LED display with 8086 microprocessor by 8255 PPI (in MDA-8086).

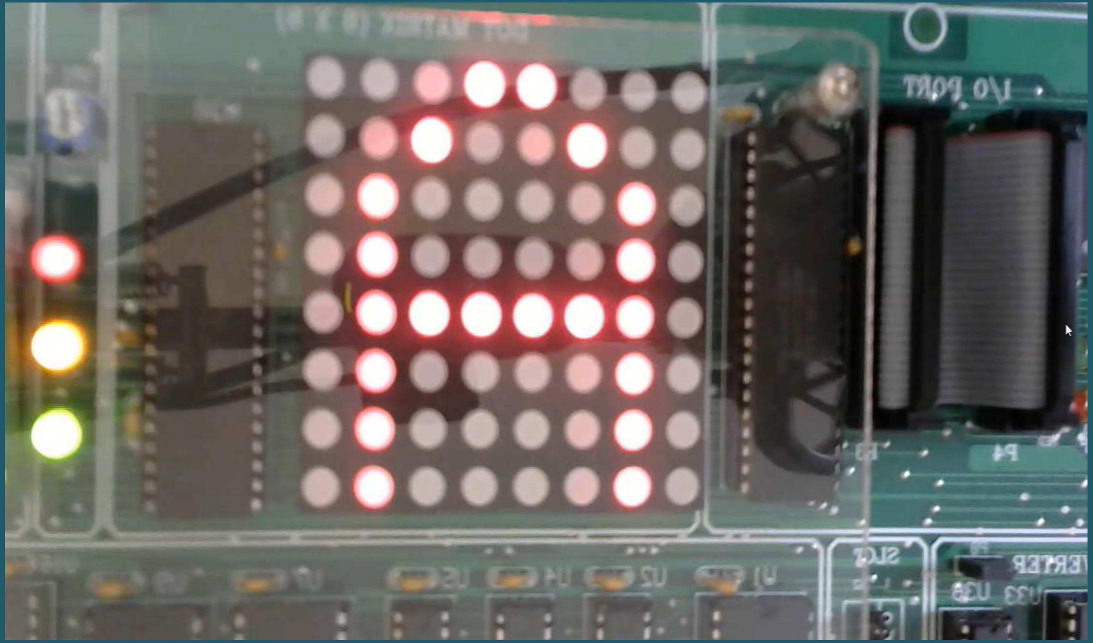


8X8 Dot Matrix Display



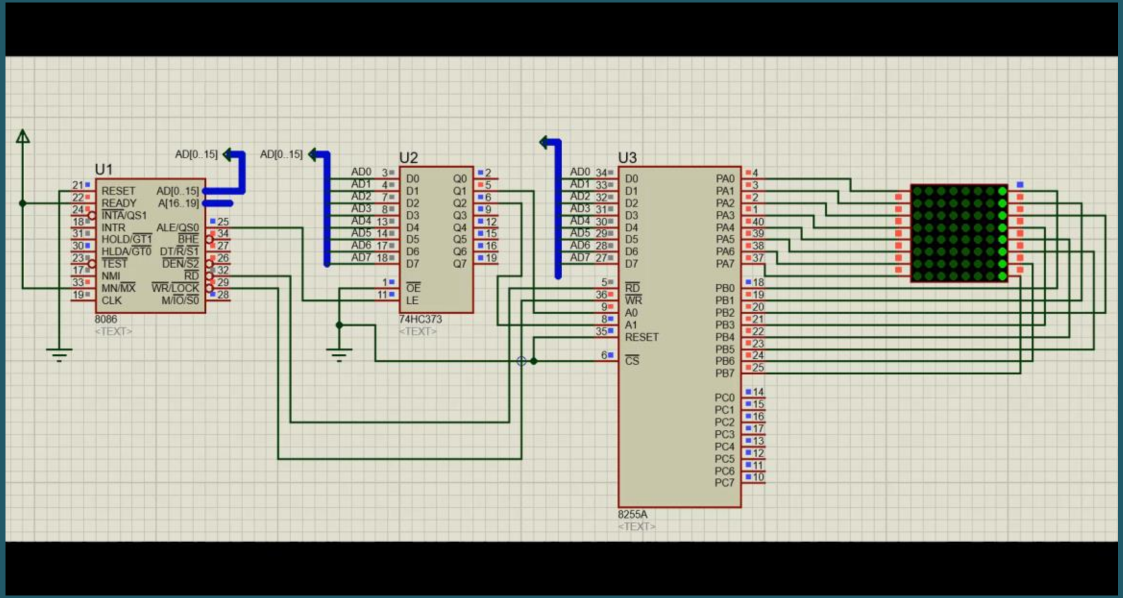
Objective-2

To display a character in LED Dot-Matrix Display.

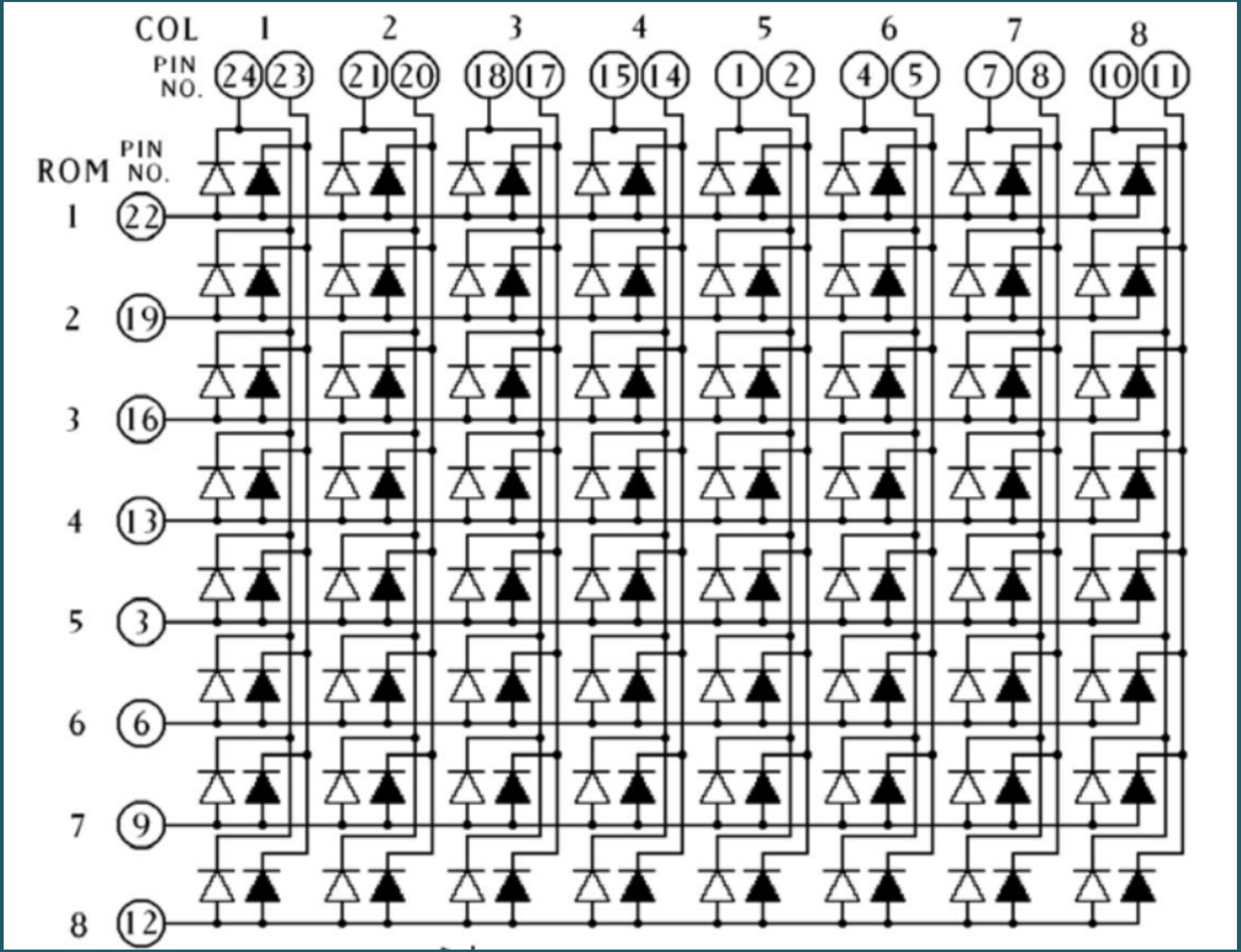


Objective-3

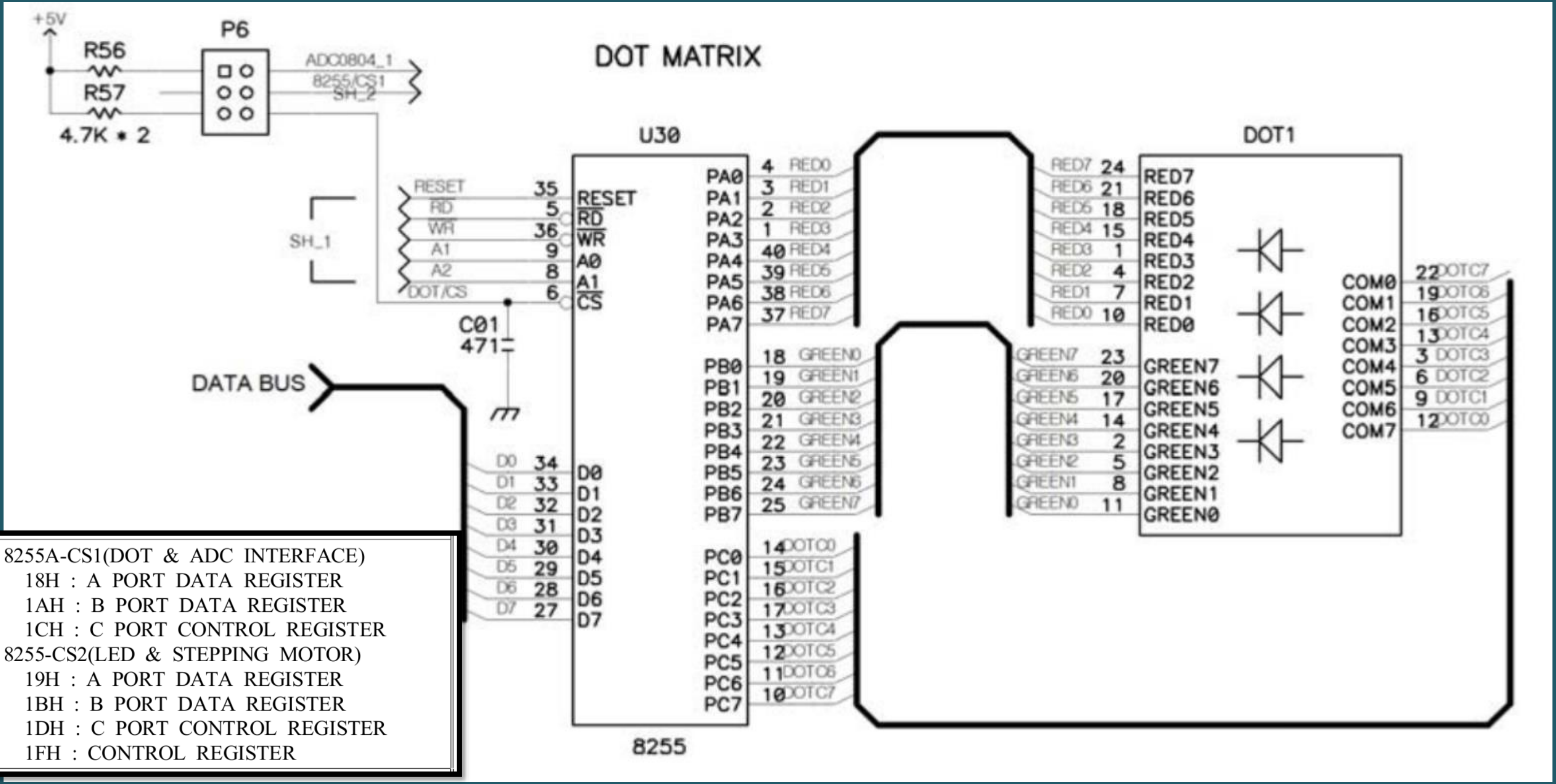
To simulate the interfacing of Dot-Matrix LED display with 8086 microprocessor in PROTEUS.



8X8 Bi-color Dot-Matrix Display

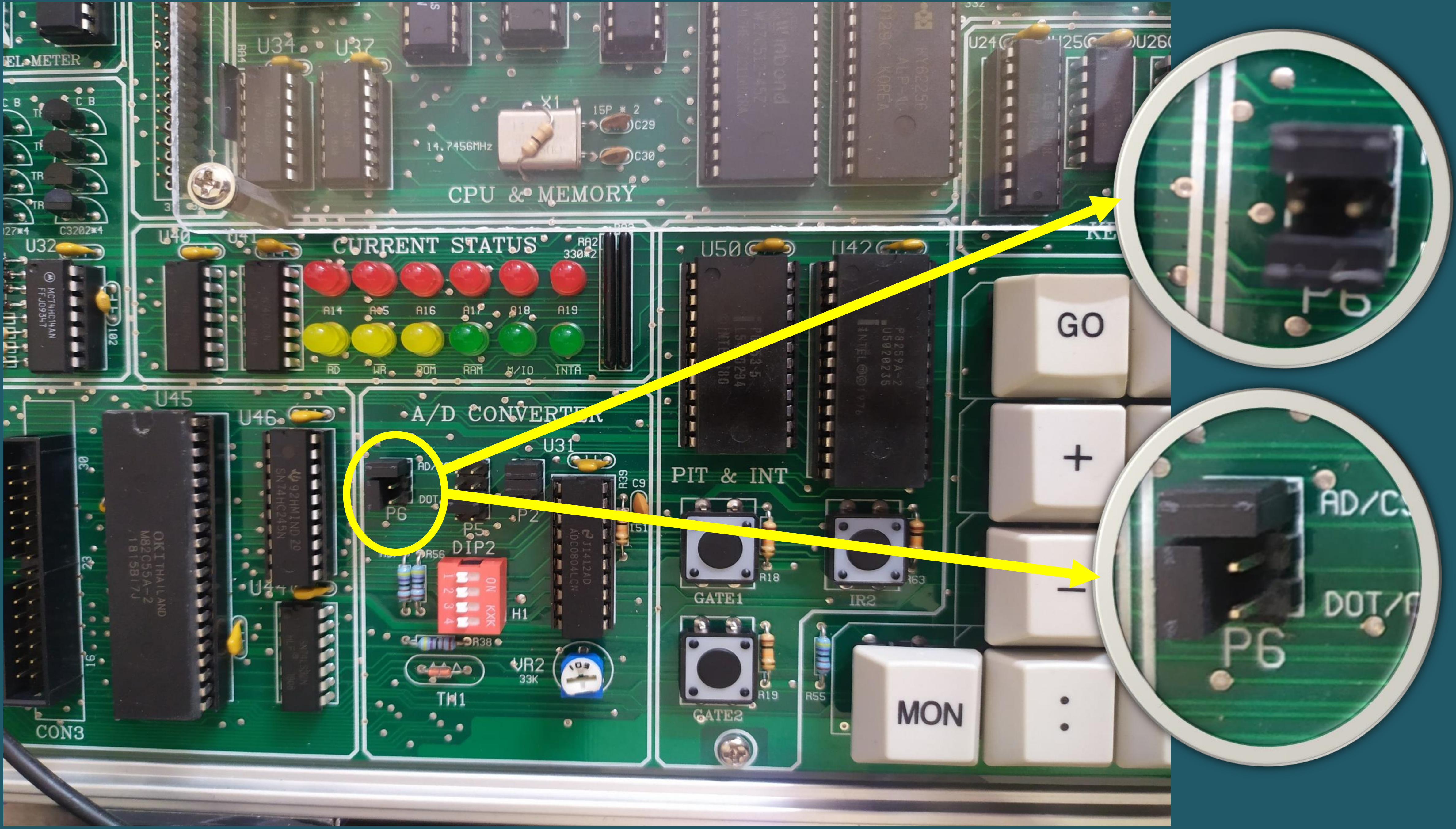


Schematic of Dot-Matrix display interface with 8086

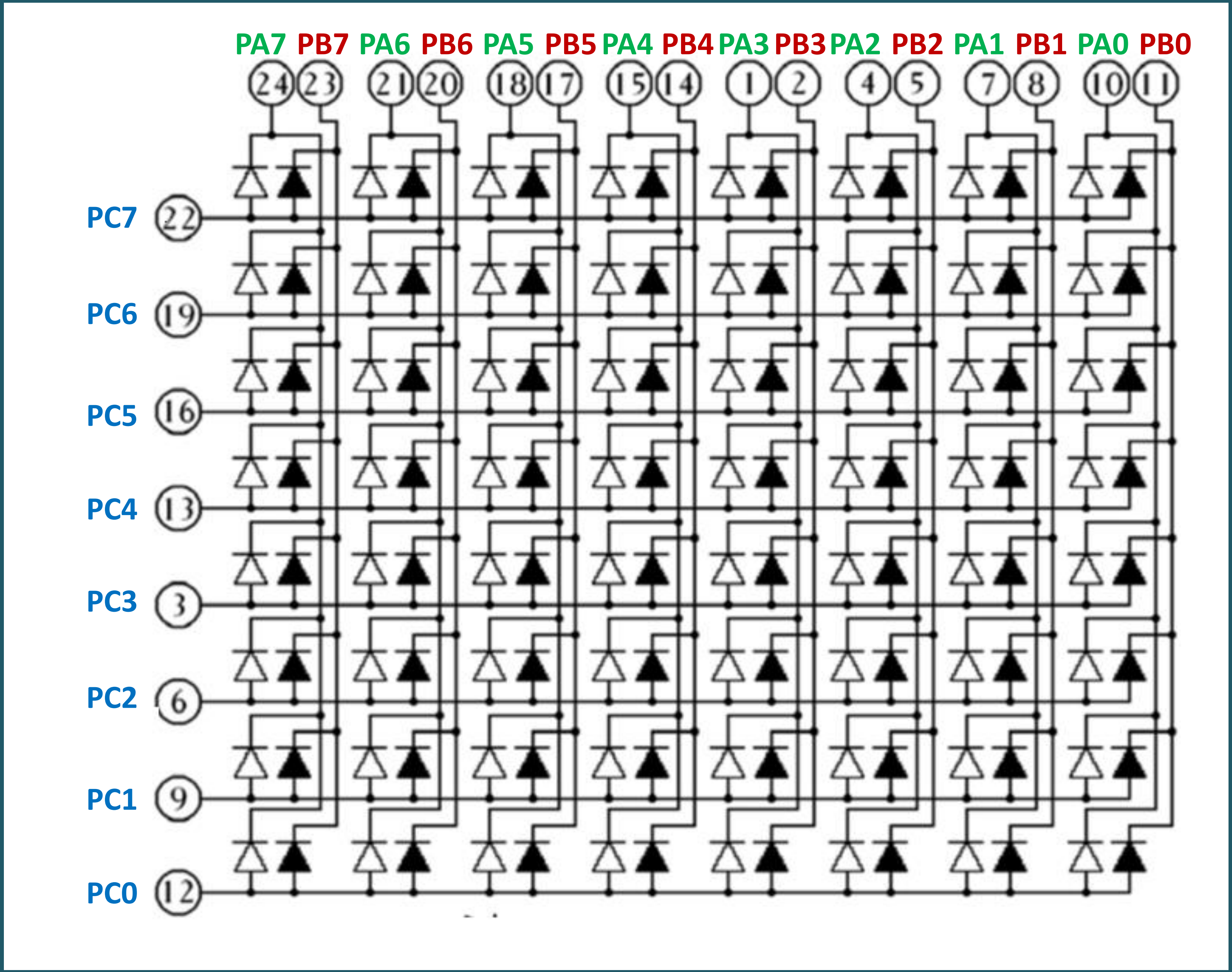

+5V
R56
R57
4.7K * 2
P6
ADC0804_1
8255/CS1
SH_2
DOT MATRIX
U30
DOT1
SH_1
RESET
RD
WR
A1
A2
DOT/CS
C01
471
DATA BUS
8255
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7
PC0 PC1 PC2 PC3 PC4 PC5 PC6 PC7
D0 D1 D2 D3 D4 D5 D6 D7
RED0 RED1 RED2 RED3 RED4 RED5 RED6 RED7
GREEN0 GREEN1 GREEN2 GREEN3 GREEN4 GREEN5 GREEN6 GREEN7
DOTC0 DOTC1 DOTC2 DOTC3 DOTC4 DOTC5 DOTC6 DOTC7
COM0 COM1 COM2 COM3 COM4 COM5 COM6 COM7

8255A-CS1(DOT & ADC INTERFACE)
- 18H : A PORT DATA REGISTER
- 1AH : B PORT DATA REGISTER
- 1CH : C PORT CONTROL REGISTER

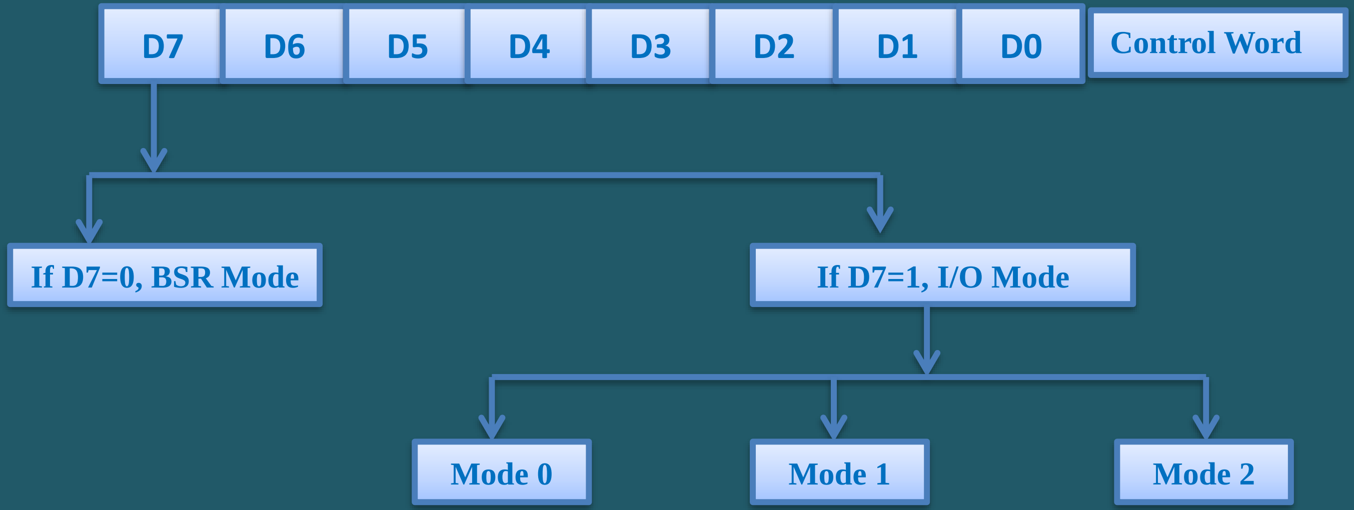
8255-CS2(LED & STEPPING MOTOR)
- 19H : A PORT DATA REGISTER
- 1BH : B PORT DATA REGISTER
- 1DH : C PORT CONTROL REGISTER
- 1FH : CONTROL REGISTER

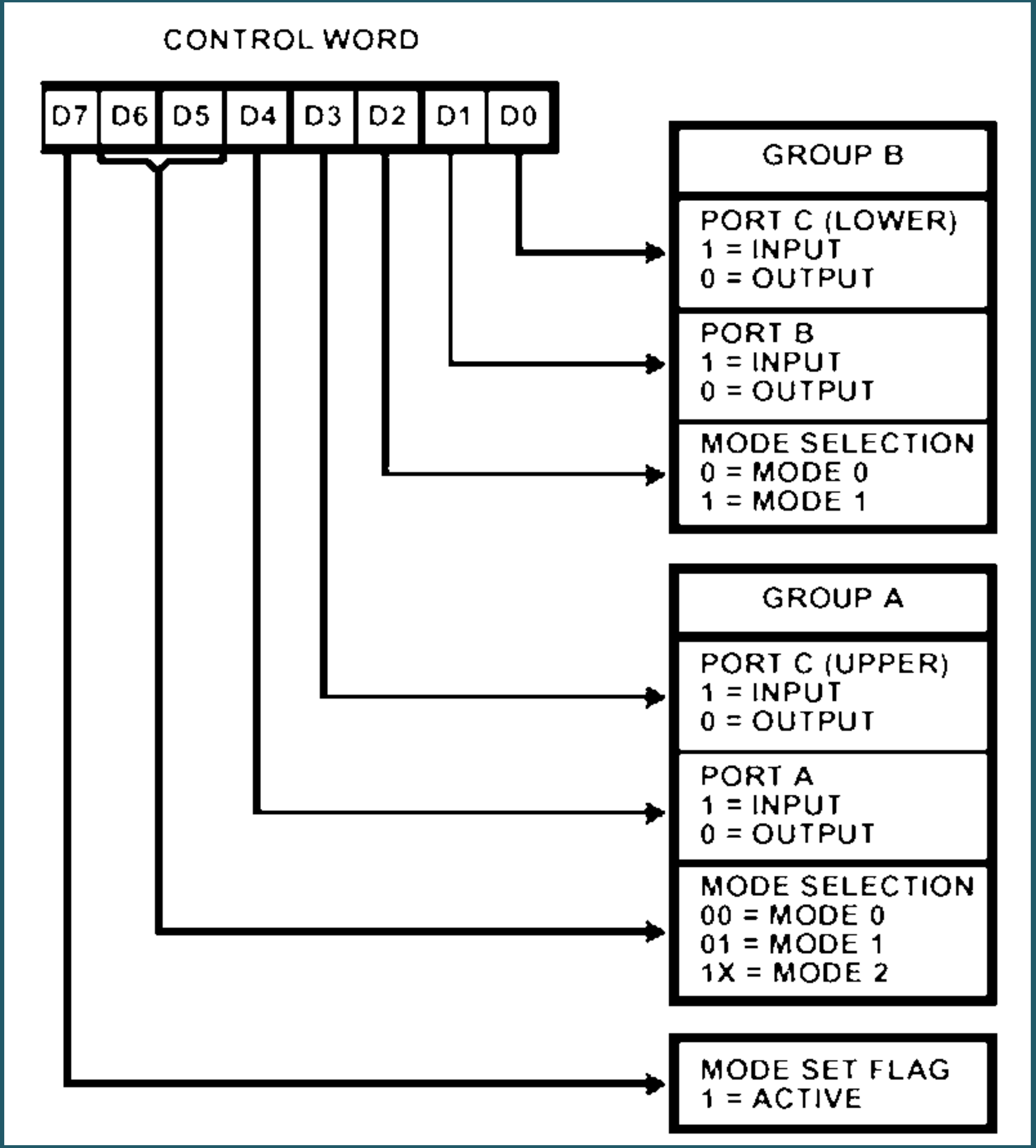

CPU & MEMORY
CURRENT STATUS
A14
A15
A16
A17
A18
A19
RD
WR
ROM
RAM
M/IO
INTA
A/D CONVERTER
AD/CS
DOT/C
P6
P5
P2
DIP2
U31
PIT & INT
GATE1
GATE2
IR2
TH1
VR2
CON3
GO
+
−
MON
:

PORTA- GREEN
PORTB- RED

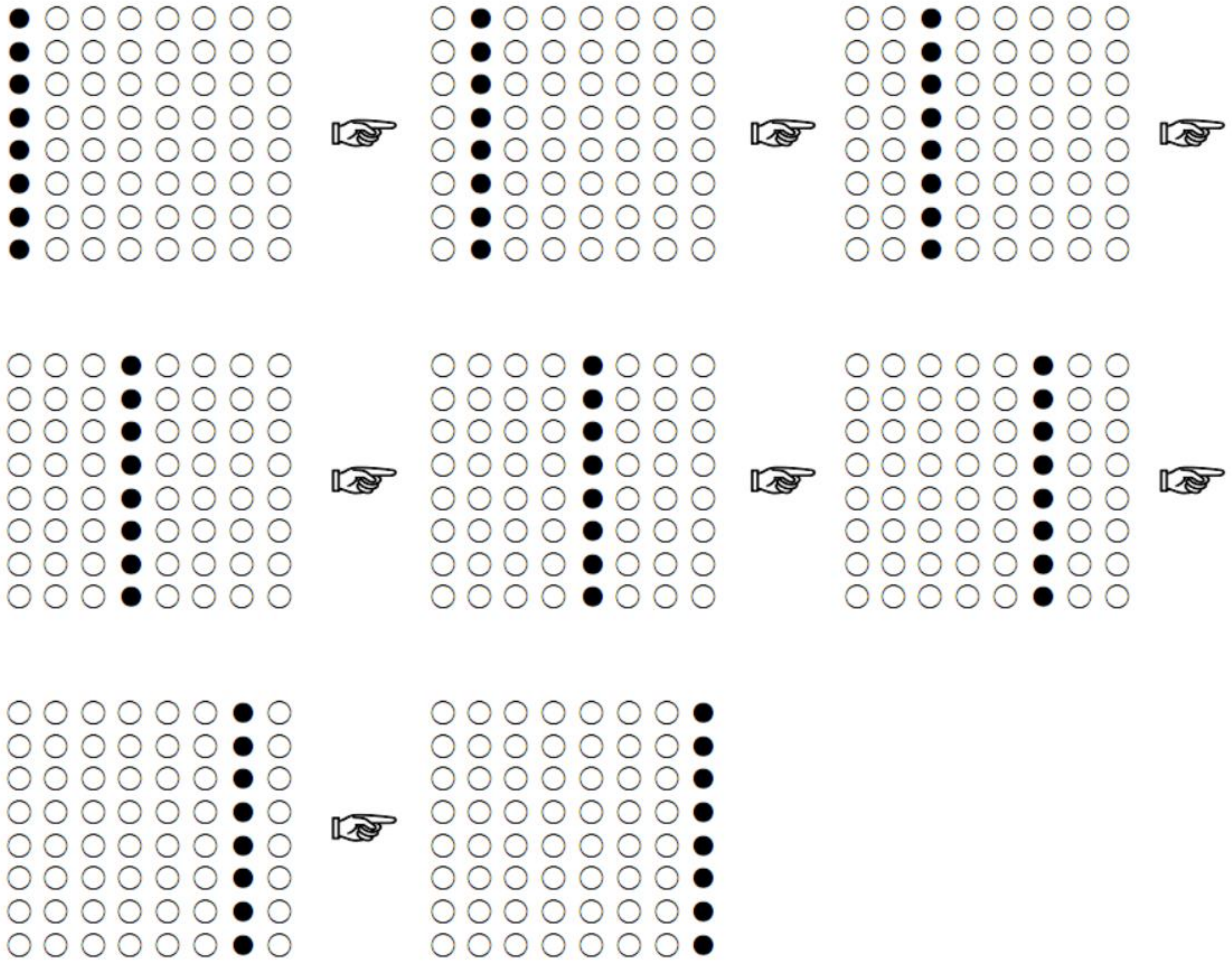


Control Word for Dot-Matrix Display Interfacing





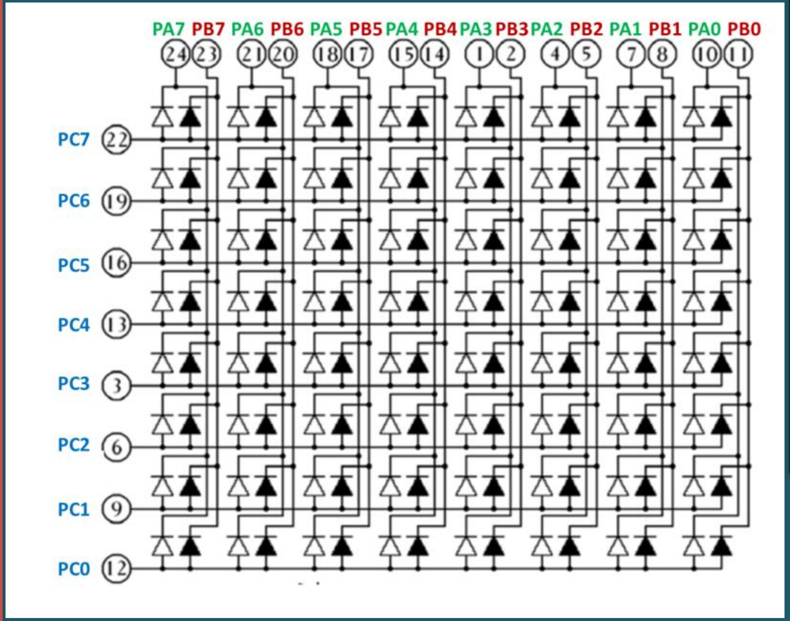
Purpose

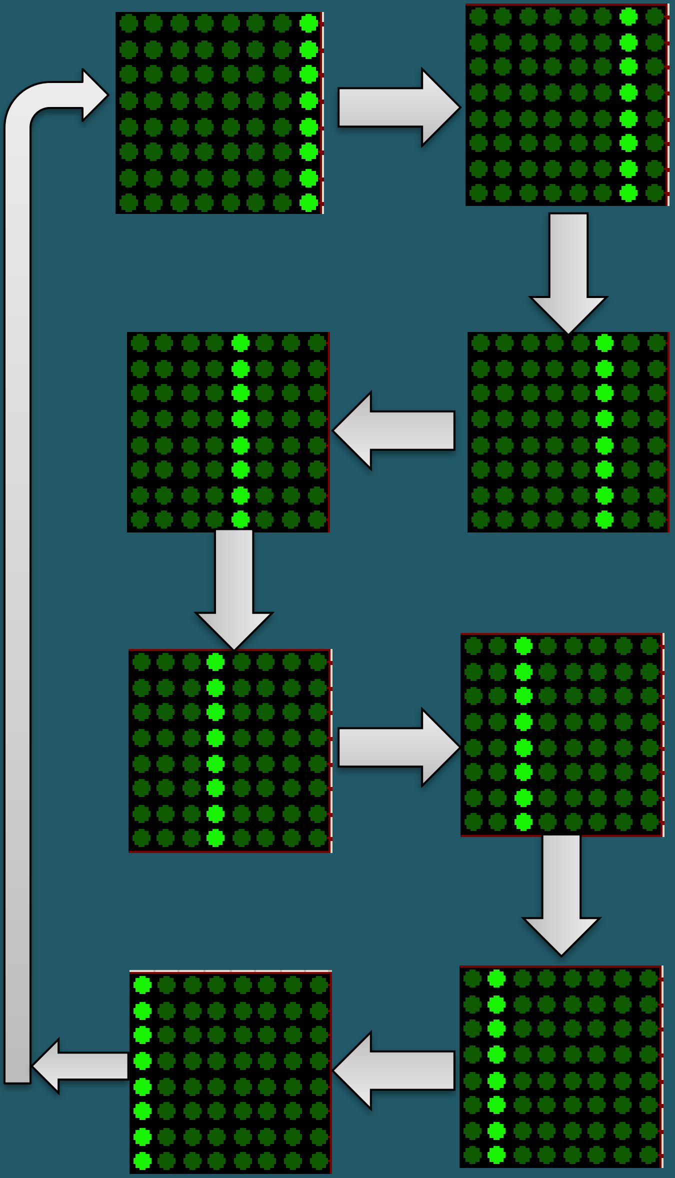


Animation in Dot Matrix Display

```
CODE      SEGMENT
ASSUME    CS:CODE,DS:CODE,ES:CODE,SS:CODE
          ;
PPIC_C    EQU       1EH ; control register
PPIC      EQU       1CH
PPIB      EQU       1AH
PPIA      EQU       18H
          ORG       1000H
          MOV       AL,10000000B
          OUT       PPIC_C,AL
          MOV       AL,11111111B
          OUT       PPIC,AL
          MOV       AL,11111111B
          OUT       PPIB,AL

L1:       MOV       AL,11111110B
L2:       OUT       PPIA,AL
          CALL      TIMER
          ROL       AL,1
          JC        L2
          JMP       L1
```

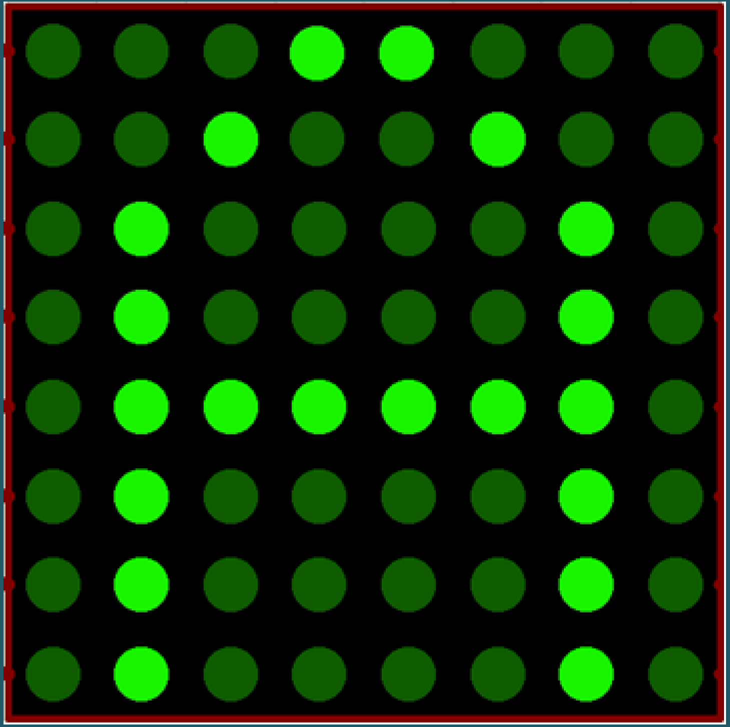
```
TIMER:    MOV       CX,0FFFFH
TIMER1:   NOP
          NOP
          NOP
          NOP
          LOOP      TIMER1
          RET
CODE      ENDS
          END
```

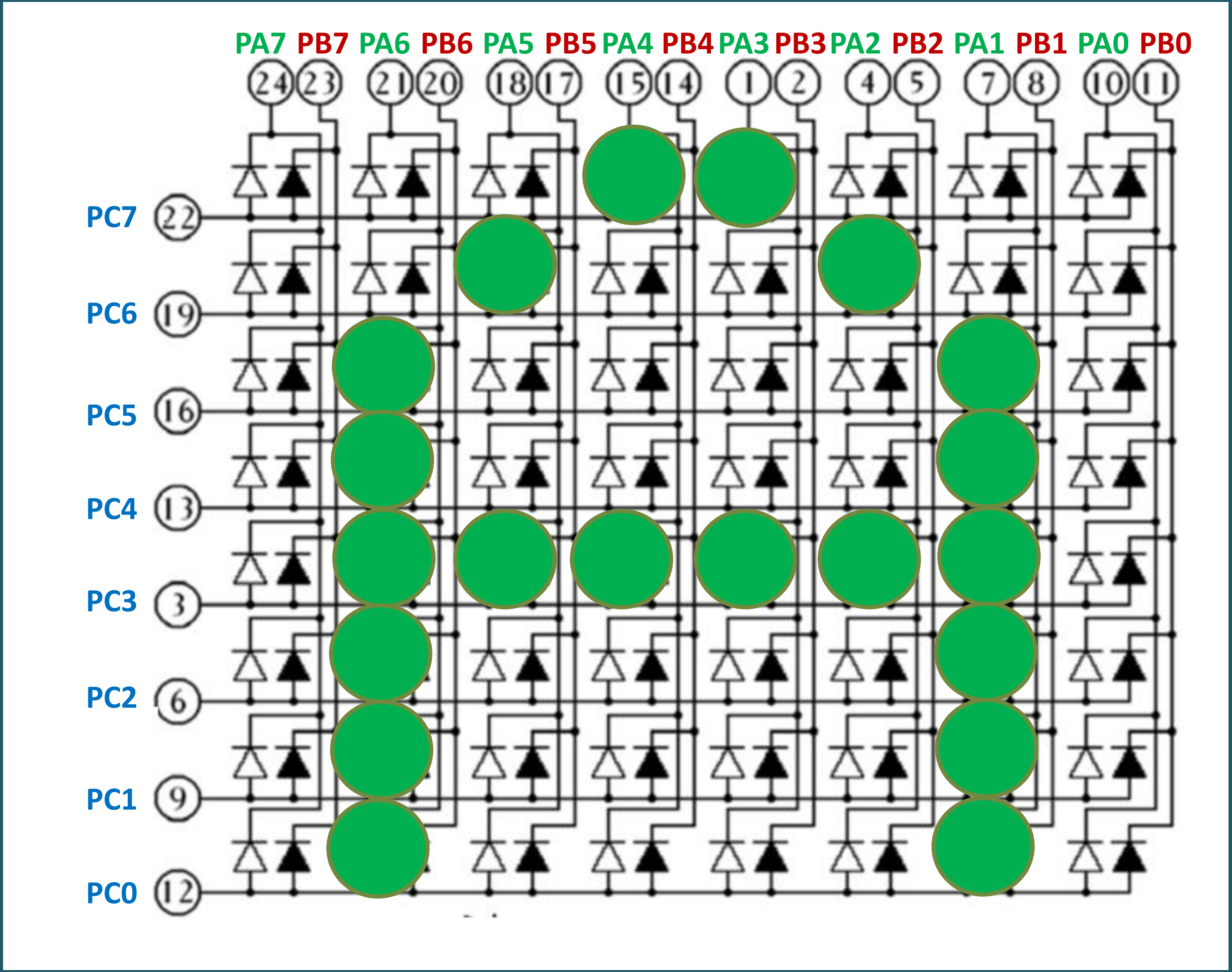


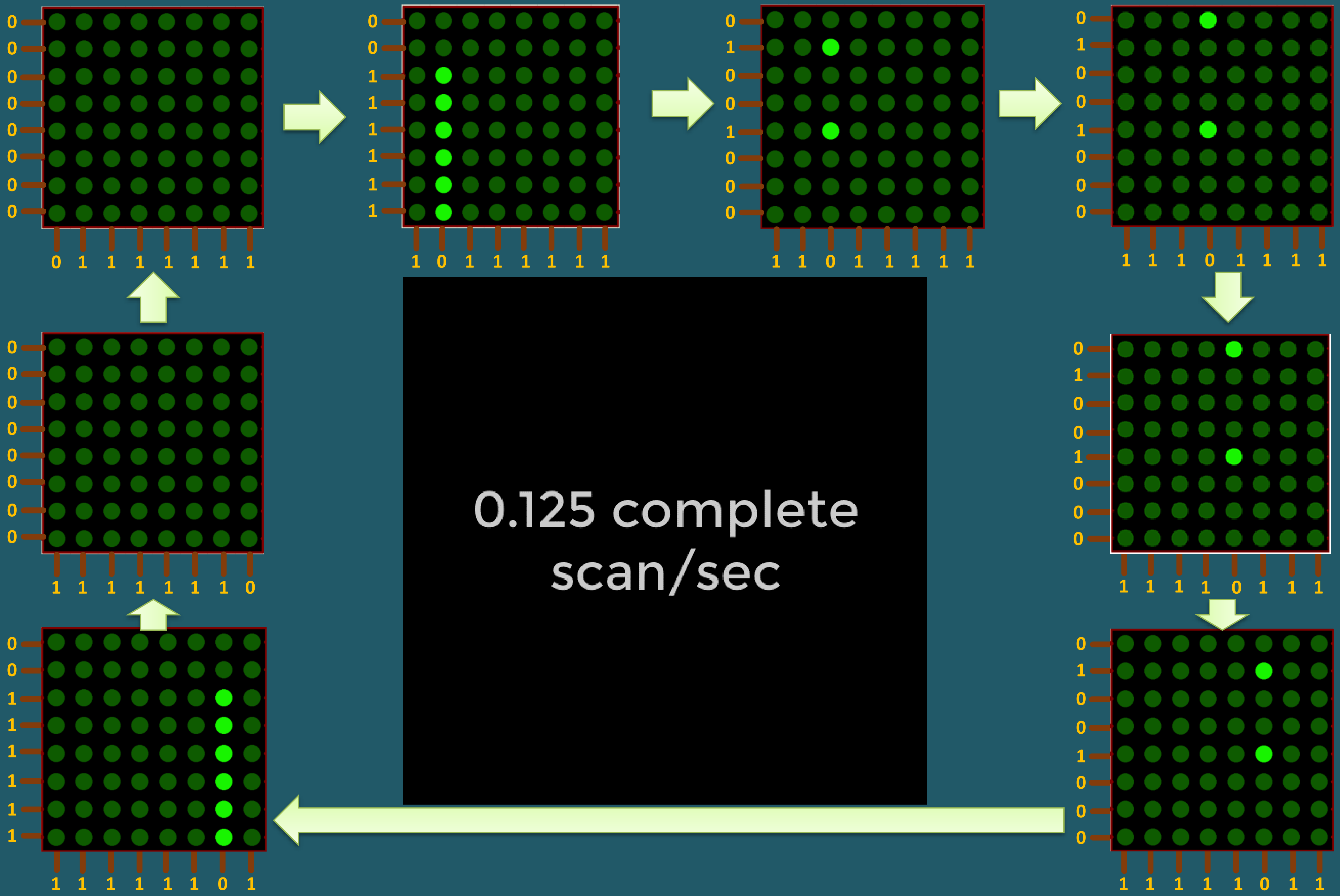


Showing alphabet in Dot Matrix Display (Concept of scanning)

- Individual control of LED in dot matrix display is possible for the LED's of same row or same column.





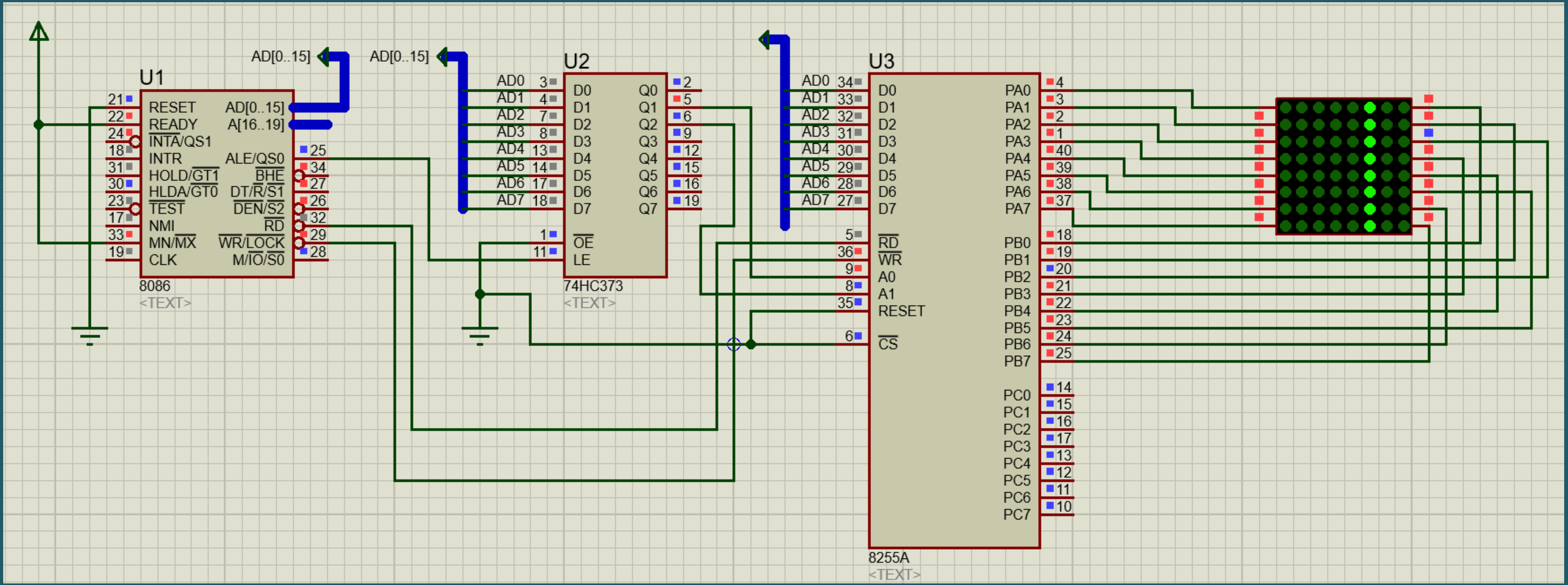


0.125 complete scan/sec

Showing Alphabet ‘A’ in Dot Matrix Display

```
CODE    SEGMENT
ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
        ;
PPIC_C  EQU     1EH ; control register
PPIC    EQU     1CH ; c port
PPIB    EQU     1AH
PPIA    EQU     18H
        ;
        ORG     1000H
        MOV     AL,10000000B
        OUT     PPIC_C,AL
        ;
        MOV     AL,11111111B
        OUT     PPIA,AL
        ;
L1:     MOV     SI,OFFSET FONT
        ;
        MOV     AH,11111110B
        ;
L2:     MOV     AL,BYTE PTR CS:[SI]
        OUT     PPIC,AL
        MOV     AL,AH
        OUT     PPIB,AL
        CALL    TIMER
        INC     SI
        CLC
        ROL     AH,1
        JC      L2
        JMP     L1

TIMER:  MOV     CX,300
TIMER1: NOP
        NOP
        NOP
        NOP
        LOOP    TIMER1
        RET
        ;
FONT:
        DB      00000000B
        DB      11111100B
        DB      00010010B
        DB      00010001B
        DB      00010001B
        DB      00010010B
        DB      11111100B
        DB      00000000B

CODE    ENDS
        END
```

Schematic of dot-matrix display interface with 8086



Thanks for Watching

In the name of Allah, Most Gracious, Most Merciful.

EEE-314 Microprocessor and Interfacing Sessional

Experiment-8

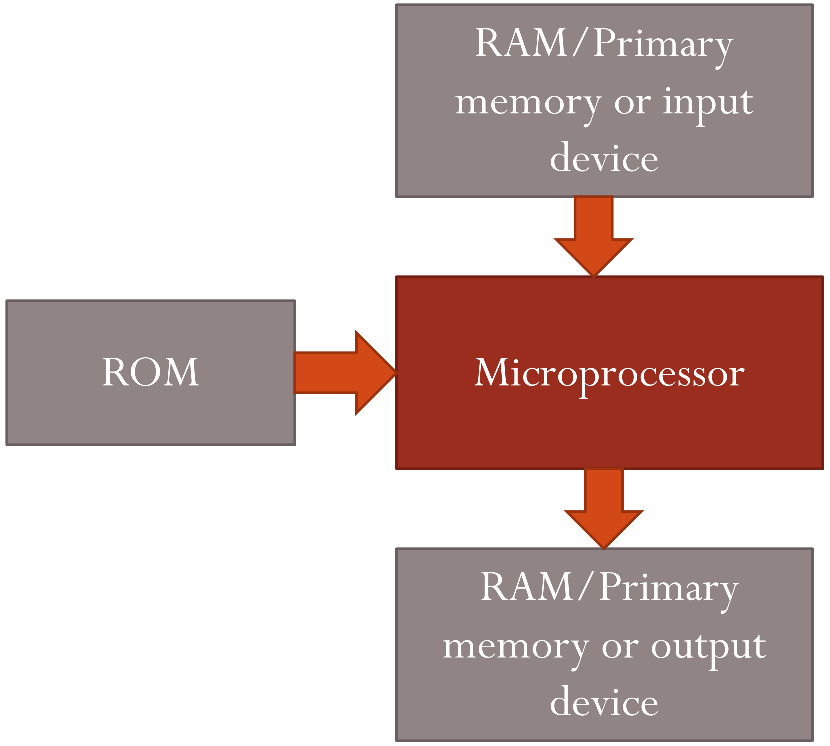
Familiarization with Microcontroller

Objectives

- To become familiar with Microprocessor, Microcomputer and Microcontroller.
- To know about internal architecture of Microcontroller.
- To become familiar with the development board 'ARDUINO UNO'.
- To introduce with the programming platform 'ARDUINO IDE'.
- To become familiar with digital I/O pins and digital write operation (LED Blinking, binary counter, Shift operation).
- To learn the simulation of microcontroller circuit in PROTEUS
- To become familiar with digital Read Operation. (Reading PUSH button)
- To do some basic project having both read and write operation.

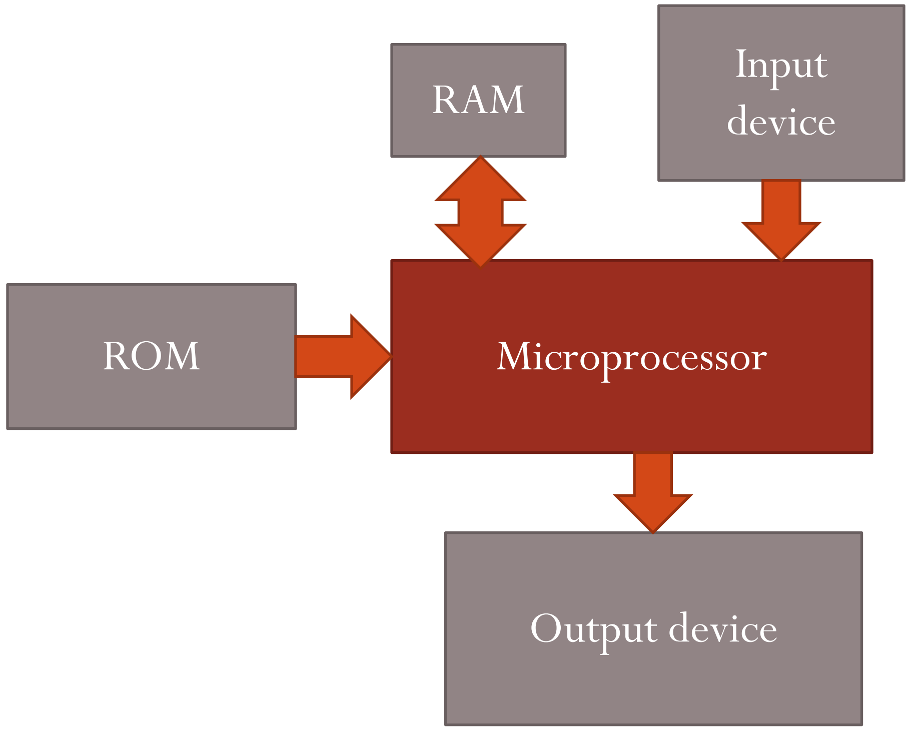
Familiarization with Microprocessor, Microcomputer and Microcontroller

Microprocessor is a multipurpose, programmable register based electronic device which read binary instructions from memory, processes the input data as per instructions and provides output.


RAM/Primary memory or input device
ROM
Microprocessor
RAM/Primary memory or output device

Familiarization with Microprocessor, Microcomputer and Microcontroller

Microcomputer is used to describe a system that includes a minimum of a microprocessor, program memory, data memory, and input/output (I/O). Some microcomputer systems include additional components such as timers, counters, analogue-to-digital converters and so on.

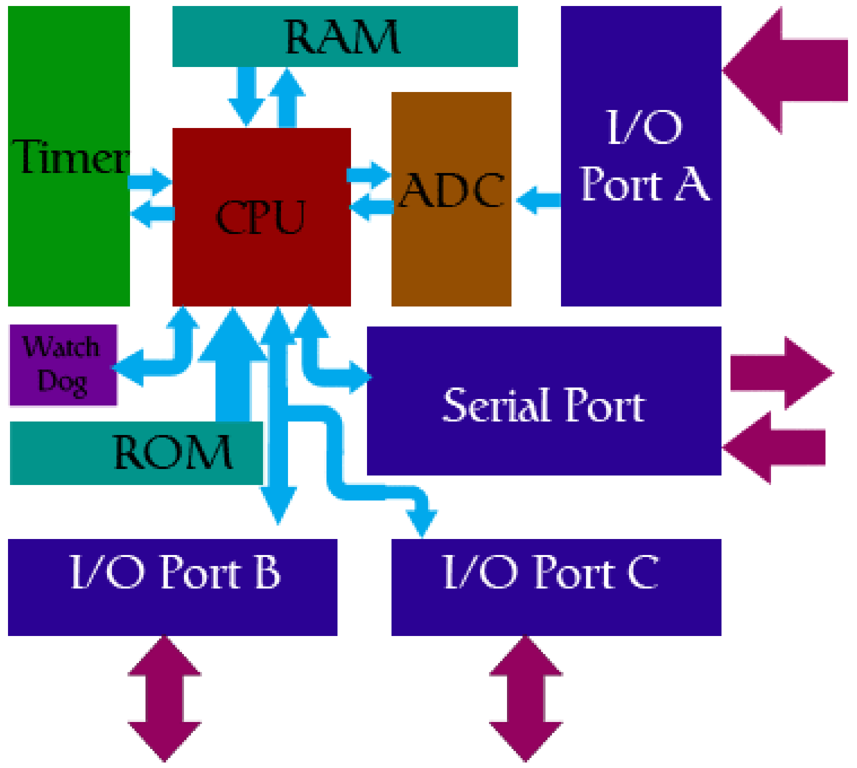


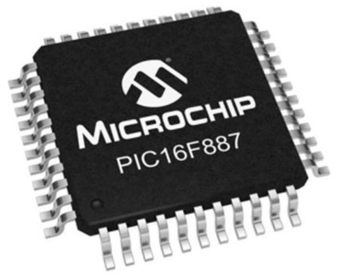
Familiarization with Microprocessor, Microcomputer and Microcontroller

Microcontroller is a device that includes microprocessor, memory and input/output devices on a single chip. Microcontroller has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single chip. At times it is also termed as **a mini computer or a computer on a single chip**.

A microcontroller consist following functional units:

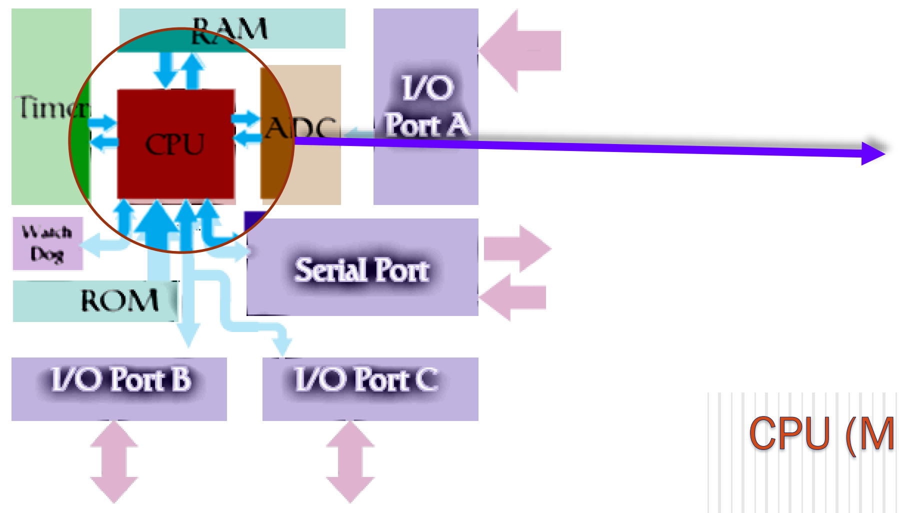
- Central Processing Unit
- Memory Unit
- System Bus
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- Oscillator
- **Program**



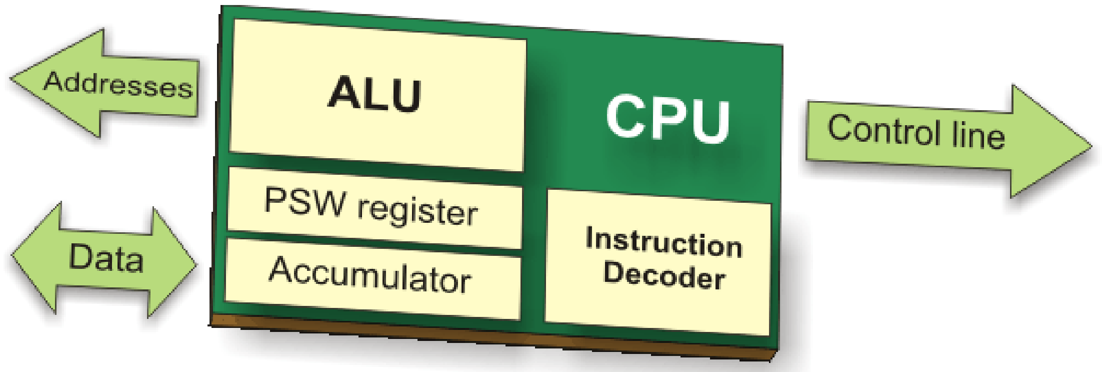


Basic microcontroller architecture

Microprocessor



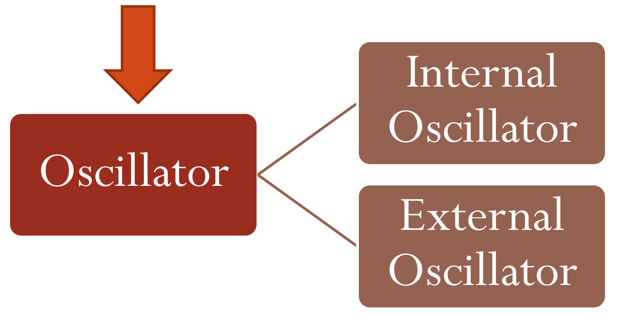
- **Central Processing Unit**
- System Bus
- Memory Unit
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- **Oscillator**
- **Program**



CPU (Microprocessor) is the brain of the microcontroller. It controls all tasks of microcontroller reading the user instructions/program.

Processor Bandwidth
- 8-bit
- 16-bit
- 32-bit

Processor Clock

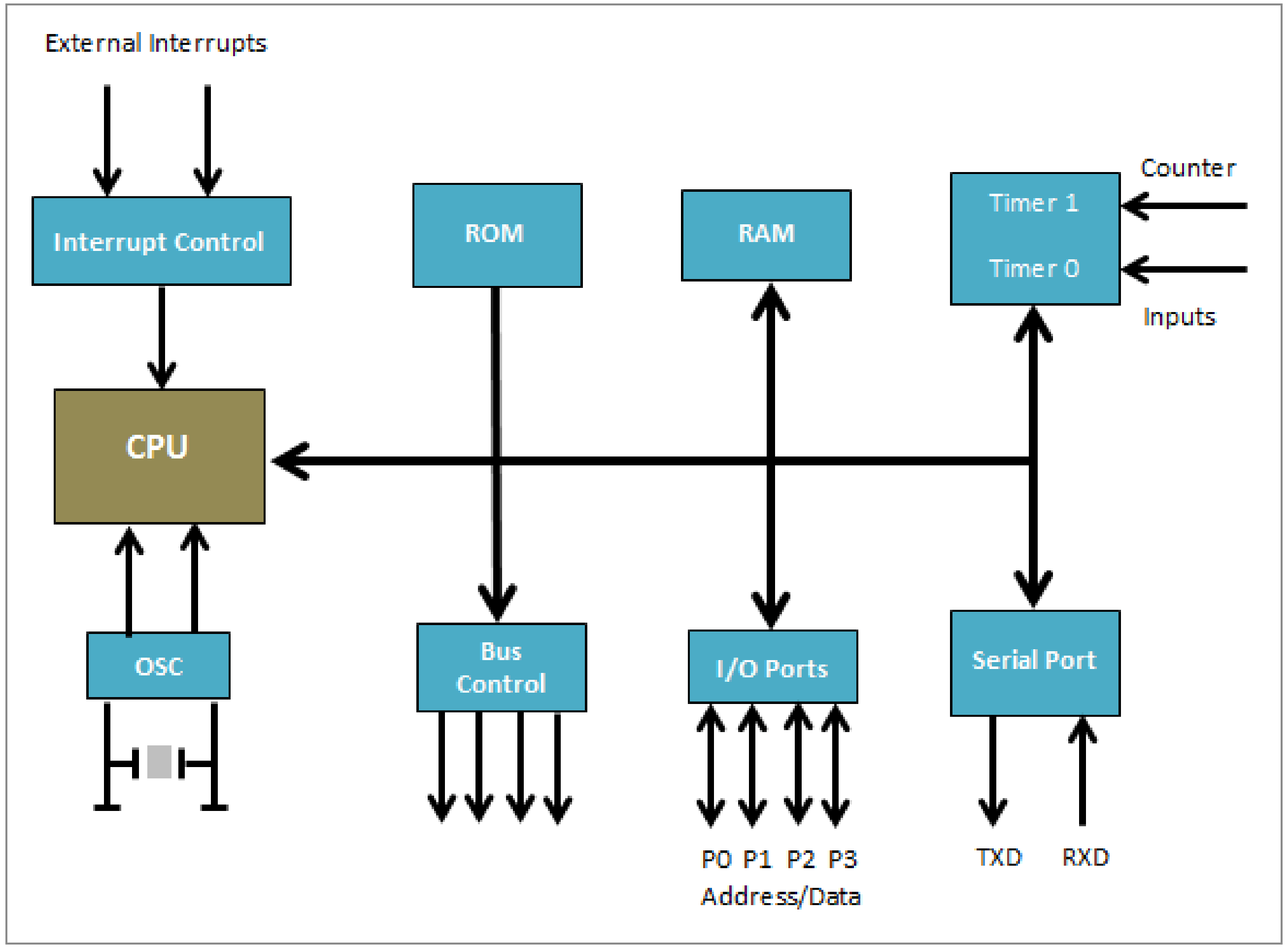


8-bit processor having clock frequency 16 MHz
Processor can execute 4×10^6 instructions/sec where data is 8-bit long (0 to 255 or -128 to +127)

Basic microcontroller architecture (Cont.)

System Bus

- Central Processing Unit
- System Bus
- Memory Unit
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- Oscillator
- Program



It is used for connections between the processor, memory and peripherals, and transferal of data between the various parts. The system bus consists of three different groups of wiring, called the data bus, control bus and address bus.

Basic microcontroller architecture (Cont.)

Memory Unit

- **Central Processing Unit**
- **System Bus**
- **Memory Unit**
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- **Oscillator**
- **Program**

Read Only Memory (ROM)

Read Only Memory (ROM) is a type of memory used to permanently save the program being executed. The size of the program that can be written depends on the size of this memory.

ROM is nonvolatile. It stores the program.

Random Access Memory (RAM)

Random Access Memory (RAM) is a type of memory used for temporary storing data and intermediate results created and used during the operation of the microcontrollers. The content of this memory is cleared once the power supply is off.

RAM is volatile memory. It stores the variables of the program.

Electrically Erasable Programmable ROM (EEPROM)

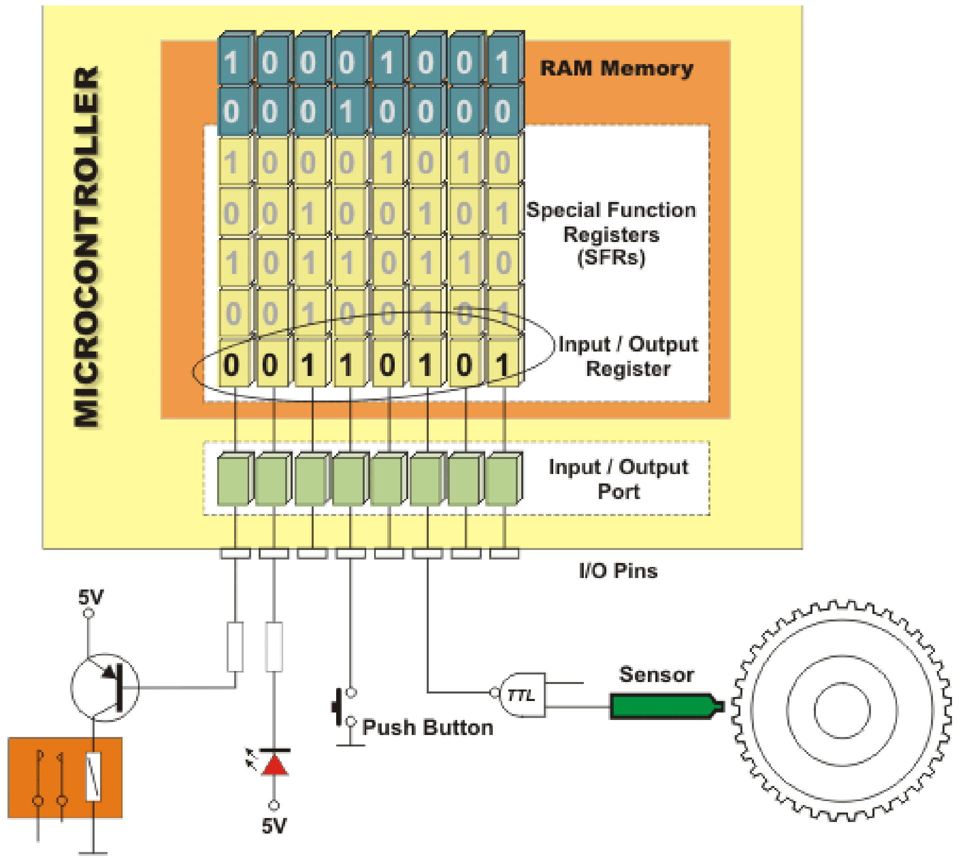
The EEPROM is a special type of memory not contained in all microcontrollers. Its contents may be changed during program execution (similar to RAM), but remains permanently saved even after the loss of power (similar to ROM).

EEPROM is nonvolatile. It stores the variable.

Basic microcontroller architecture (Cont.)

Input/Output Port

- Central Processing Unit
- System Bus
- Memory Unit
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- Oscillator
- Program

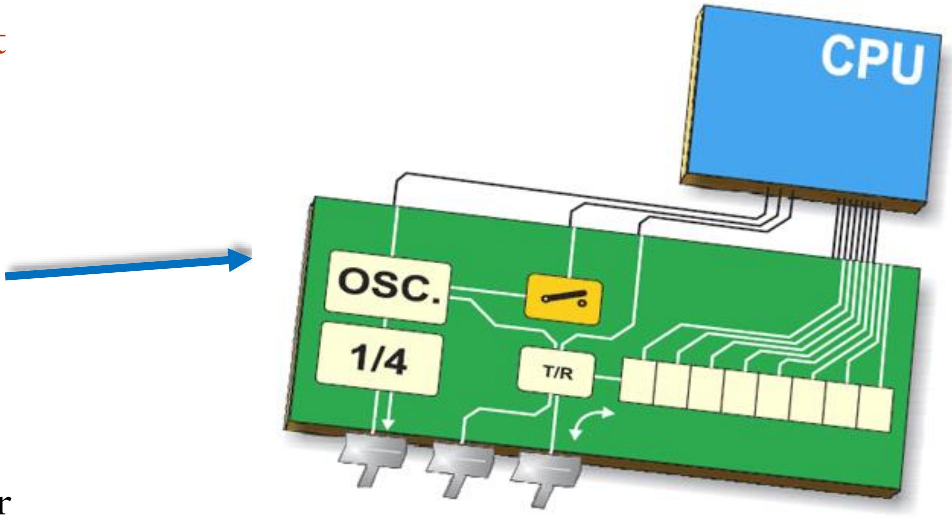


- In order to make the microcontroller useful, it is necessary to connect it to peripheral devices. Each microcontroller has one or more registers (called a port) connected to the microcontroller pins.
- Microcontroller can change the logic state (HIGH/LOW) of these pins or, can read the state if these pins if they changed by other peripherals.

Basic microcontroller architecture (Cont.)

Serial communication

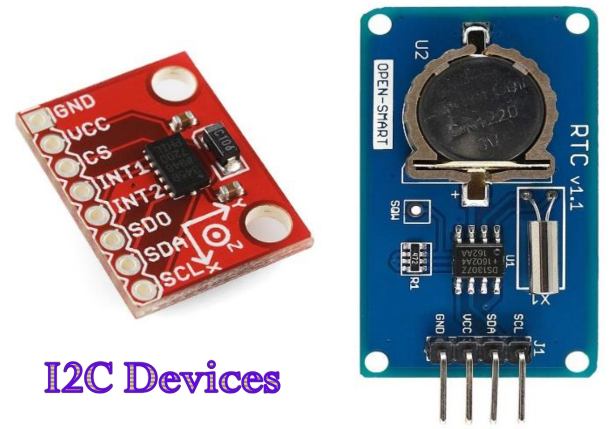
- Central Processing Unit
- System Bus
- Memory Unit
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- Oscillator
- Program



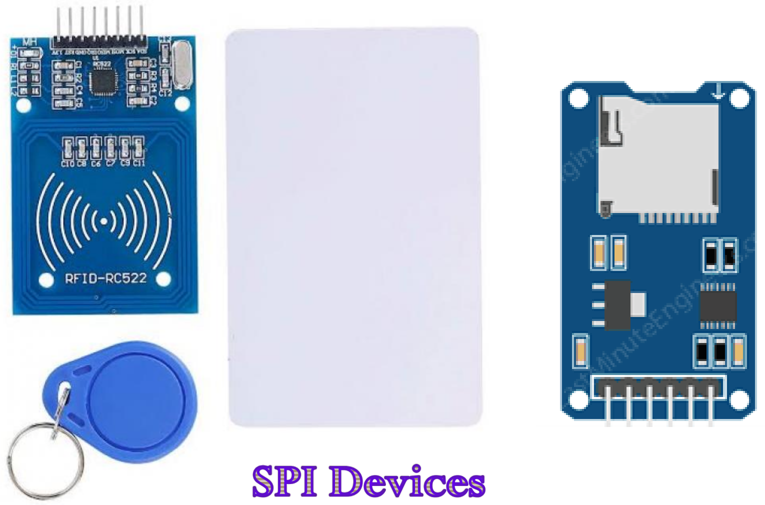
Serial communication is the most widely used approach to transfer information (data) between microcontroller and peripherals (sensors, communication devices etc)

The most commonly used serial communication systems are:

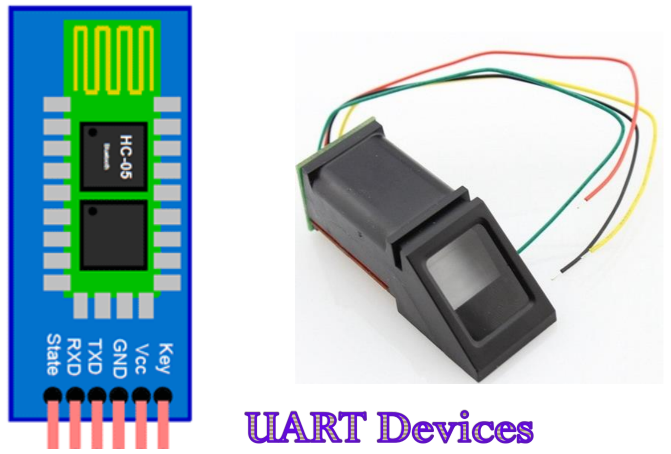
- I^{2}C (Inter Integrated Circuit)
- SPI (Serial Peripheral Interface Bus)
- UART (Universal Asynchronous Receiver/Transmitter)



I2C Devices



SPI Devices



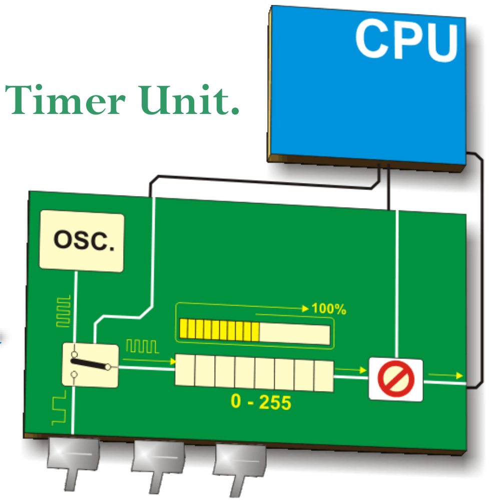
UART Devices

Basic microcontroller architecture (Cont.)

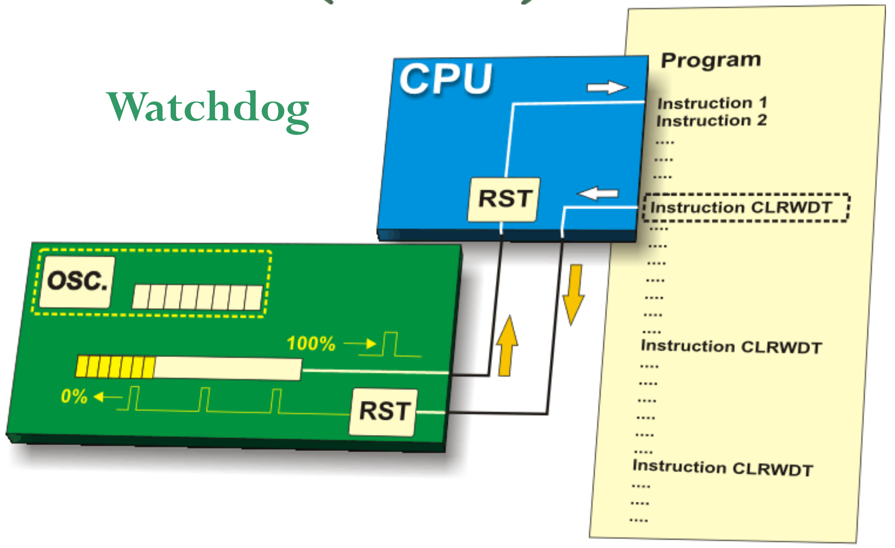
Timer/Counter

- Central Processing Unit
- System Bus
- Memory Unit
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- Oscillator
- Program

Timer Unit.



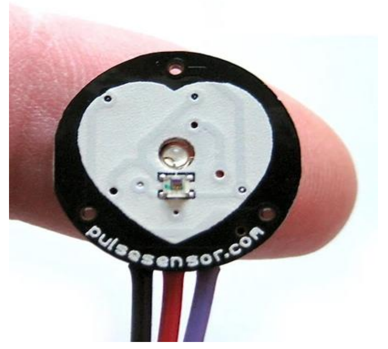
Watchdog



Timer act as miniature electronic "stopwatches“. These are commonly 8- or 16-bit SFRs the contents of which is automatically incremented by each coming pulse. When timer clock comes from peripherals they are called counter.

Flow Rate Measurement Sensor

Pulse Count → Counter
Time Count → Timer
No of rotation per second

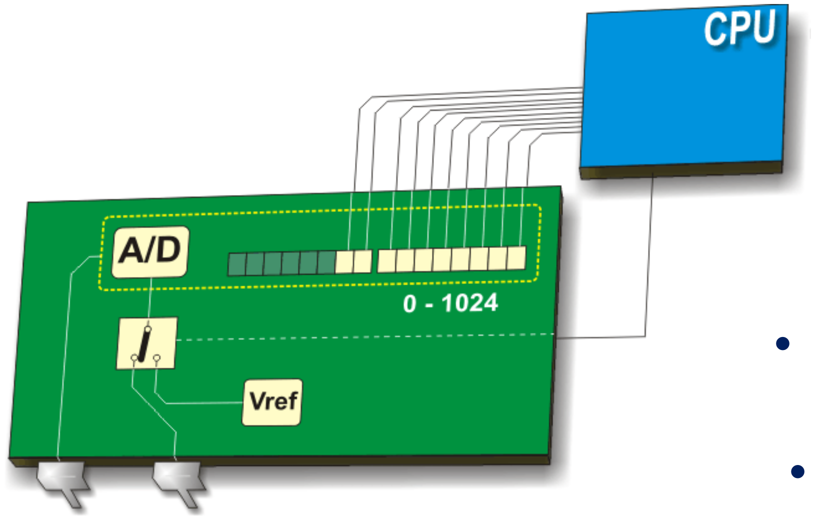


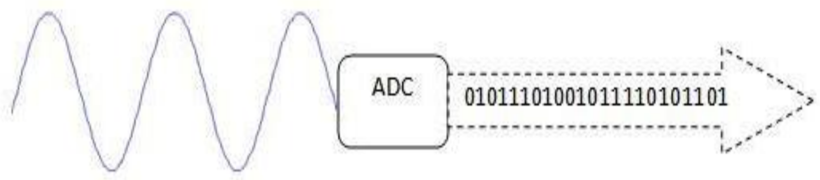
Pulse Rate Sensor

Basic microcontroller architecture (Cont.)

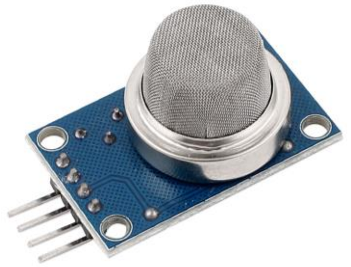
A/D Converter

- Central Processing Unit
- System Bus
- Memory Unit
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- Oscillator
- Program



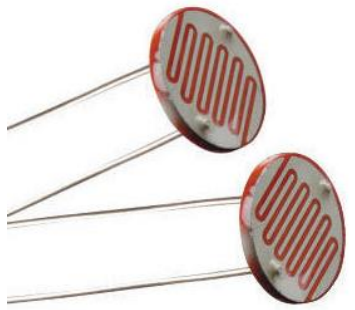


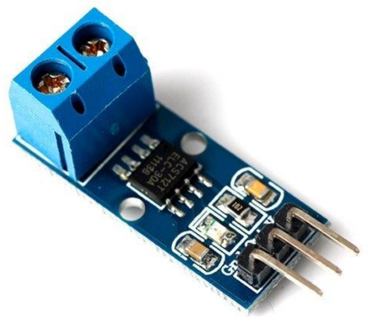
- Converts analog signal into digital signal.
- May have 8-16 bit A/D converter.


Gas Sensor

Temperature Sensor

Rain Sensor

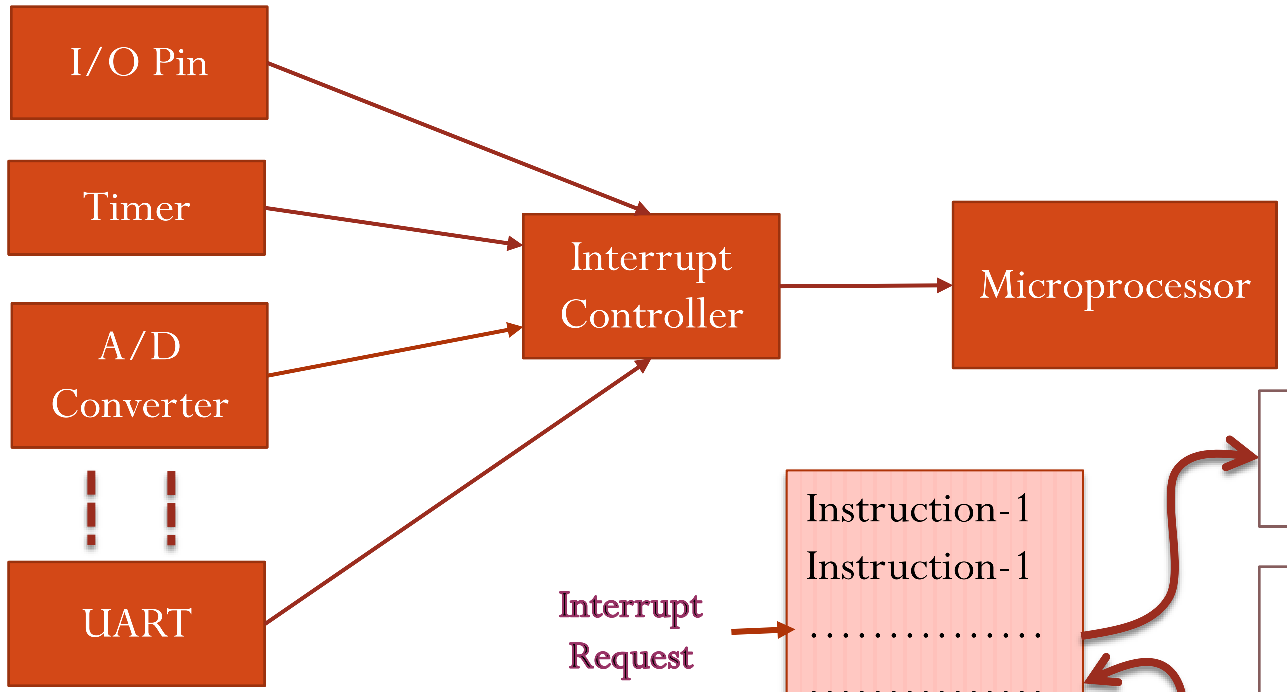

LDR

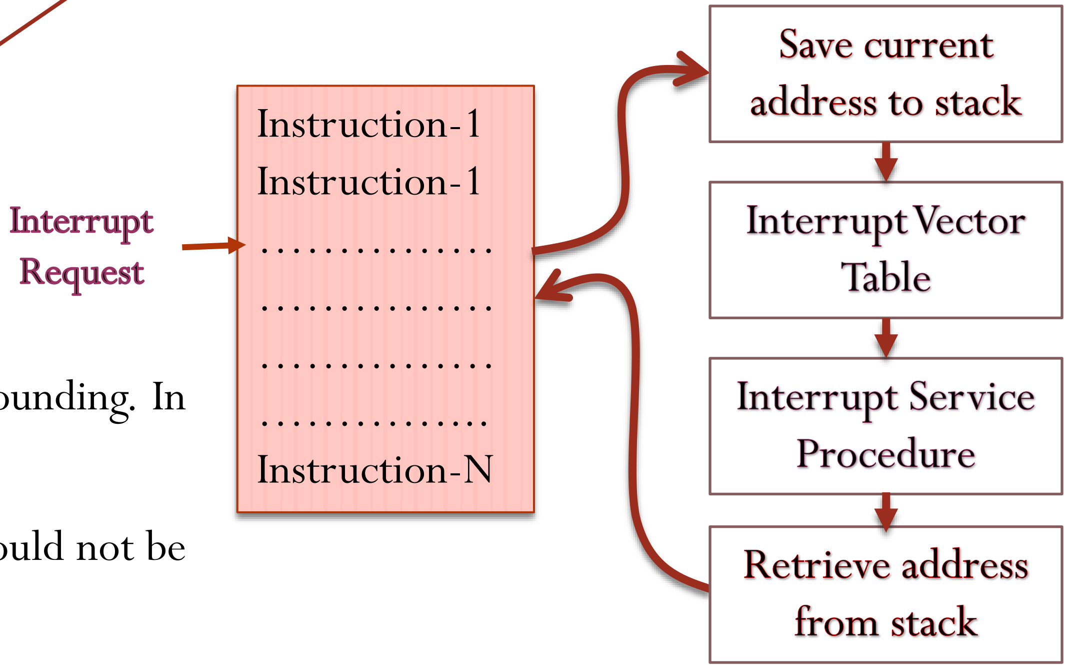

Current Sensor

Basic microcontroller architecture (Cont.)

Interrupt System

- Central Processing Unit
- System Bus
- Memory Unit
- Input/Output Port
- Serial Communication
- Timer Unit.
- Interrupt System.
- Watchdog
- Analog to Digital Converter
- Oscillator
- Program





Why INTERRUPT?

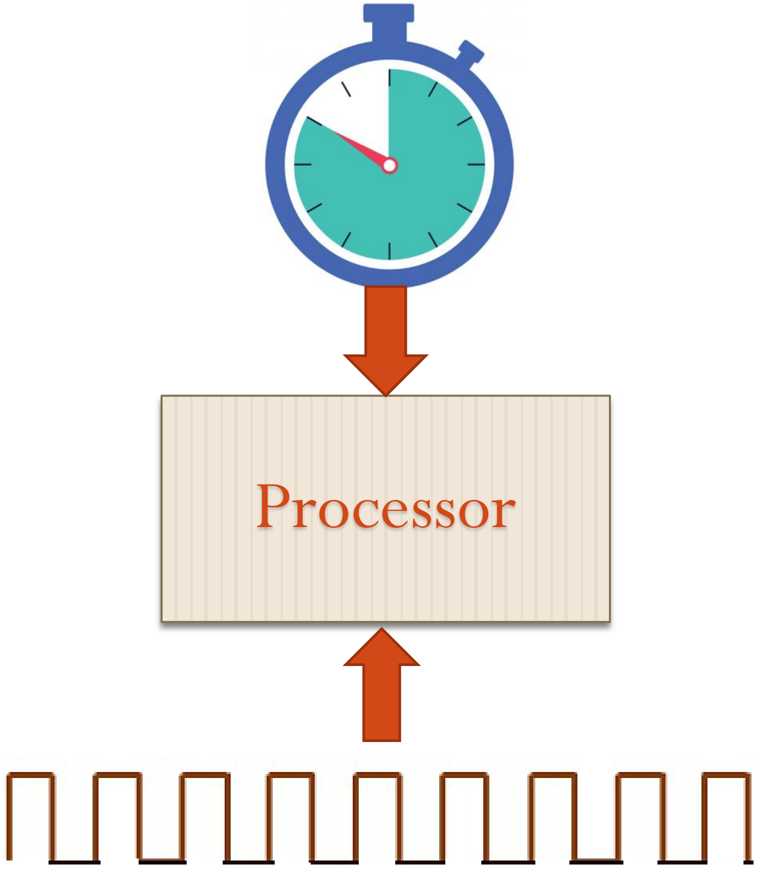
The purpose of the microcontroller is mainly to respond to changes in its surrounding. In other words, when an event takes place, the microcontroller does something.

If the microcontroller spent most of its time endlessly checking peripherals, it would not be practical at all.

This is why the microcontroller has learnt a trick during its evolution. Instead of checking each pin or bit constantly, the microcontroller delegates the 'wait issue' to a 'specialist' which will respond only when something attention worthy happens.

Basic microcontroller architecture (Continued)

RPM Meter



- Read the timer
- Count pulse
- Calculation of RPM
- Displaying Data

Pin Change Interrupt

Go to a special subroutine when any pulse comes at a input pin

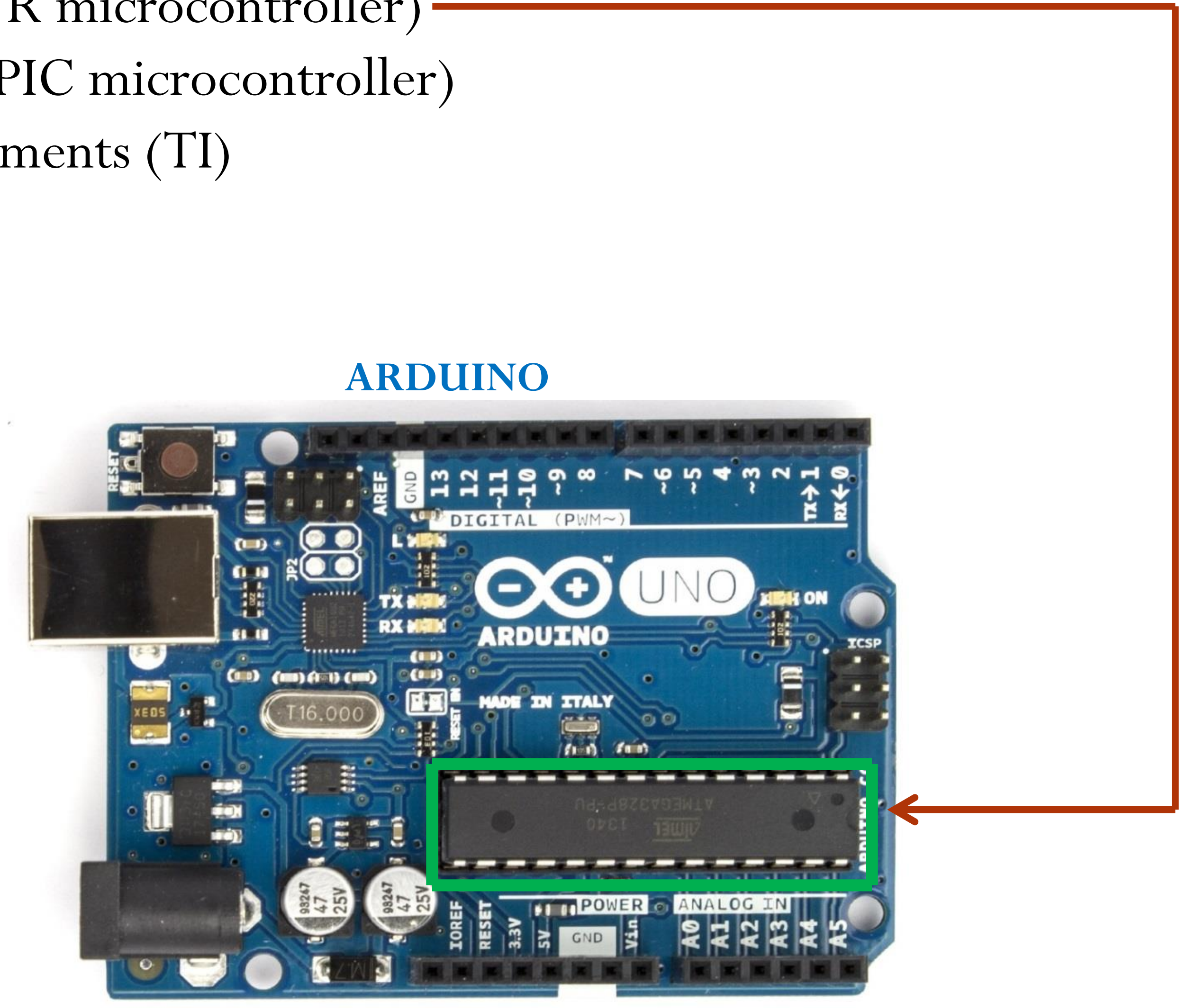
Timer Overflow Interrupt

Go to a special subroutine when set time is elapsed

Name of Some Microcontroller Manufacturer

- ATMEL (AVR microcontroller)
- Microchip (PIC microcontroller)
- Texas Instruments (TI)
- Freescale
- Philips
- Motorola

ARDUINO



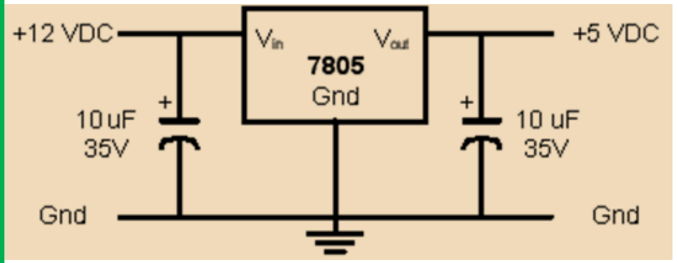
Advantages of Arduino: Reduces hardware complexity

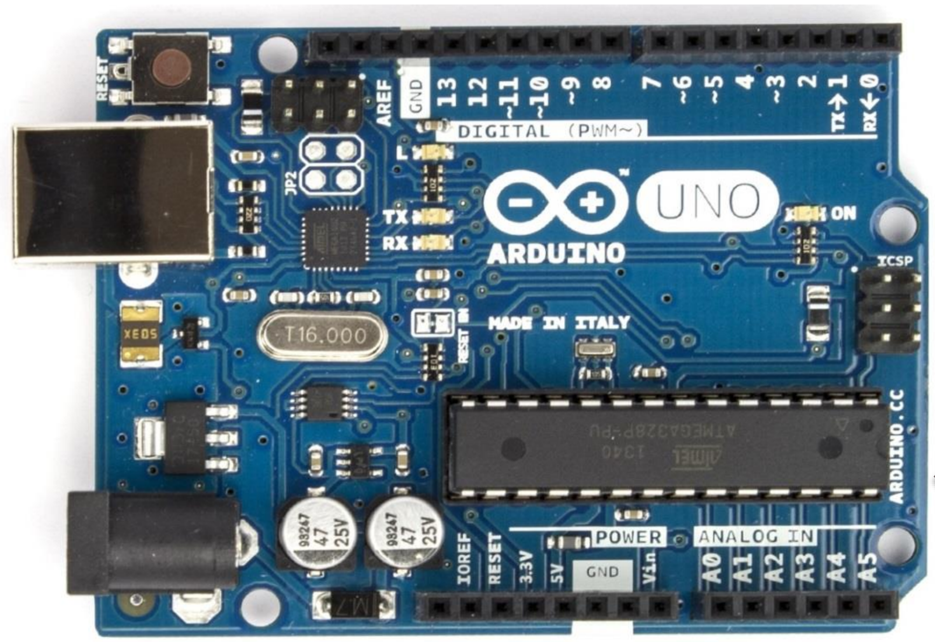
Crystal Oscillator

Program Loader

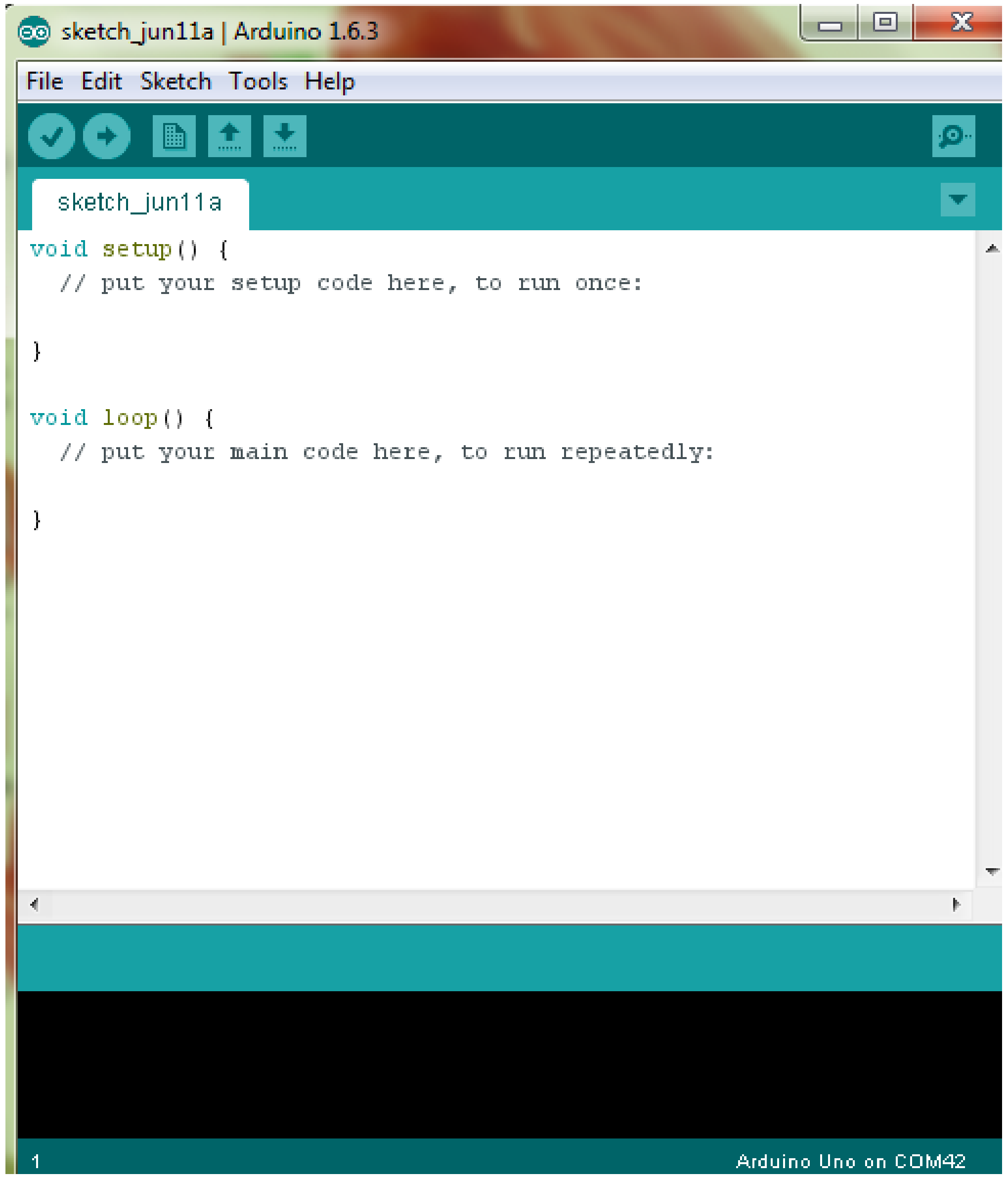
Microcontroller

Linear Voltage Regulator





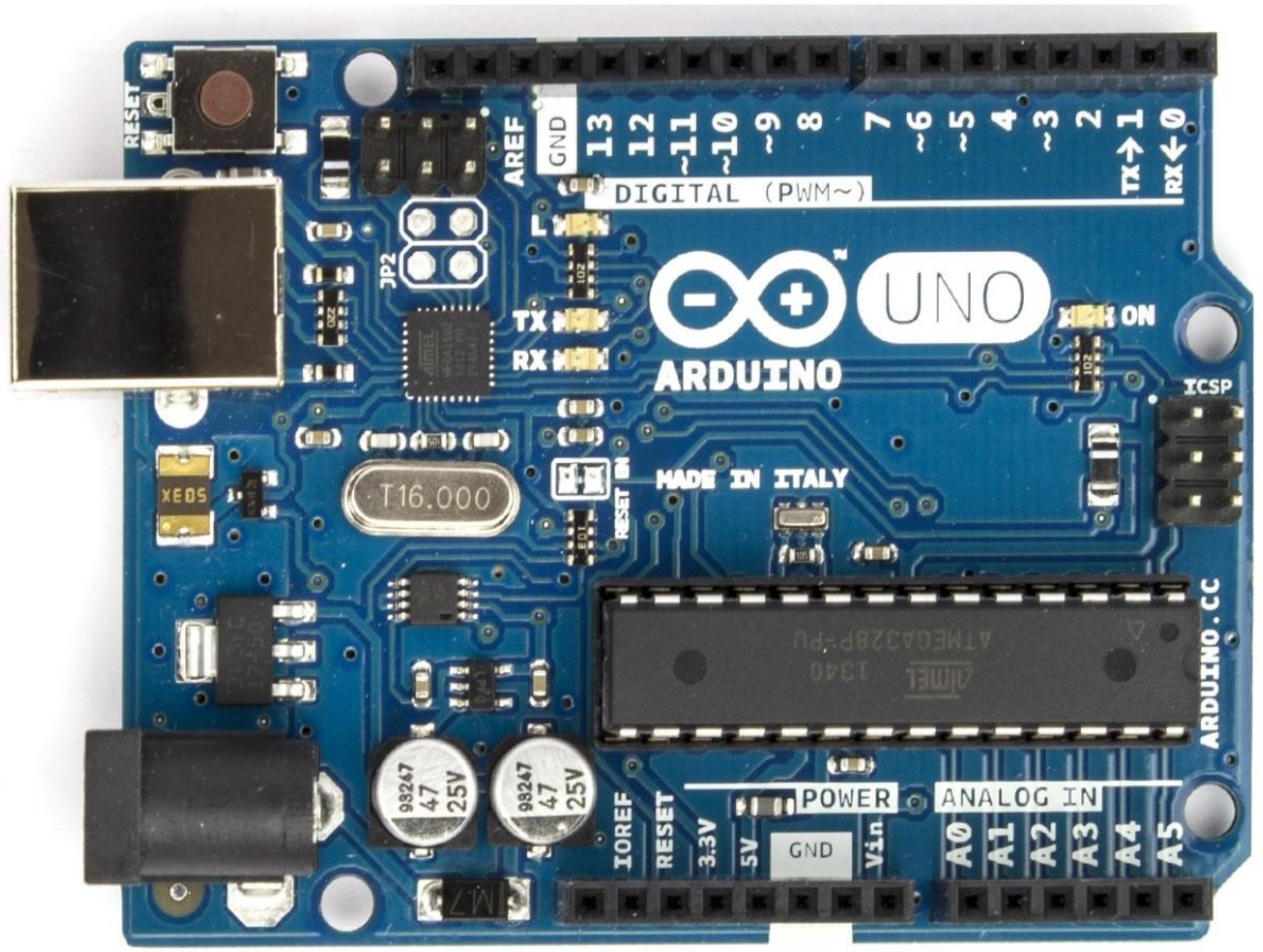
Advantages of Arduino: Programming Easy



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

- Programming in C
- Lot of functions in the library.
- No need to learn internal architecture of microcontroller.

Arduino UNO



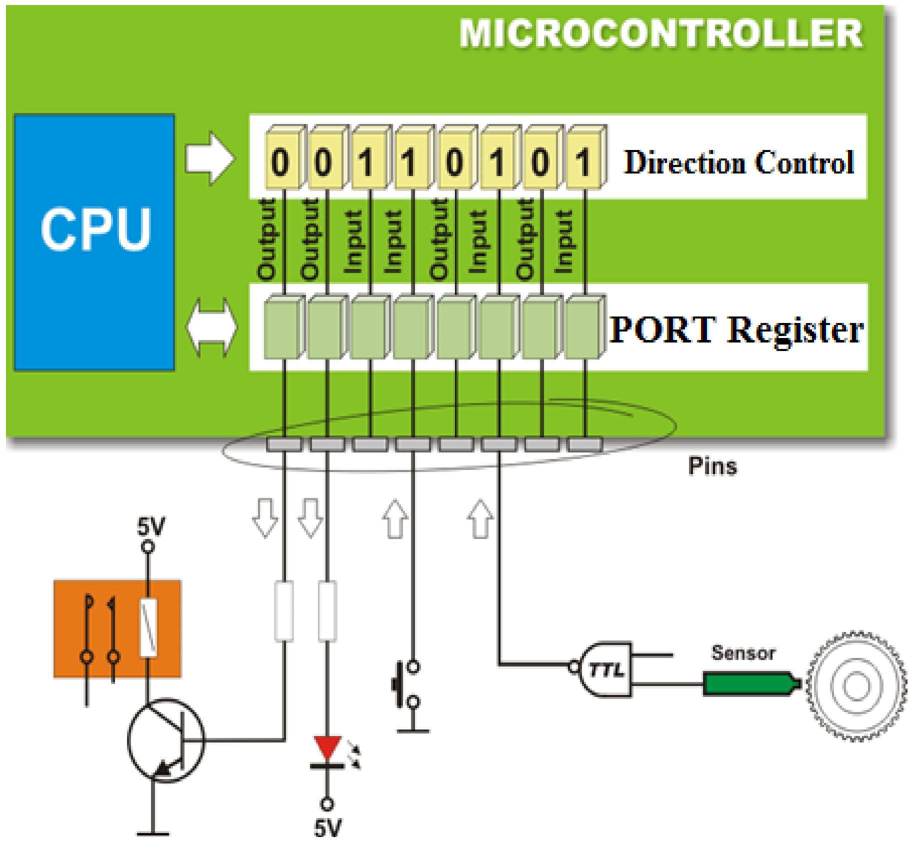
RESET
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
JP2
L
TX
RX
ARDUINO
UNO
ON
ICSP
T16.000
MADE IN ITALY
RESET EN
ATMEGA328P-PU
ARDUINO.CC
IOREF
RESET
3.3V
5V
GND
Vin
POWER
ANALOG IN
A0
A1
A2
A3
A4
A5

Features of Arduino UNO

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Digital I/O Pins

- Most of the important pins of microcontroller are digital input-output pins.
- These pins are used to connect INPUT Device (i.e. Push Button, keypad, digital sensors etc) and OUTPUT Device (i.e. LED, Display, Relay, Motor etc.) with microcontroller.
- These pins can act as INPUT or OUTPUT.
- Digital Output pin means microcontroller can make this pin HIGH or LOW state.
- Digital Input pin means microcontroller can read HIGH or LOW state from other devices

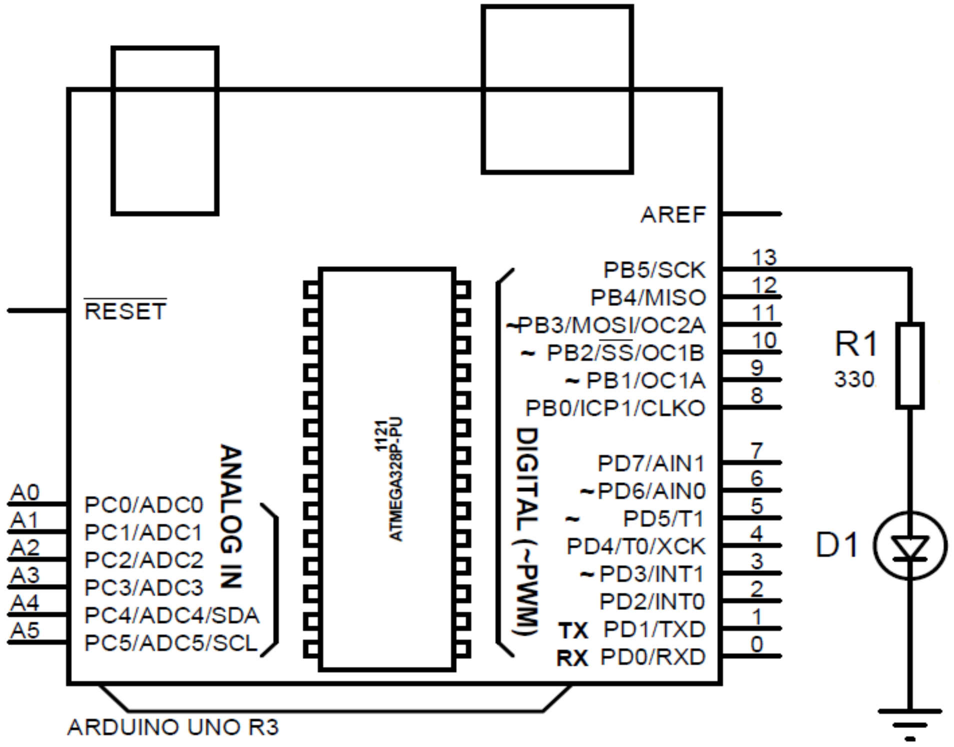


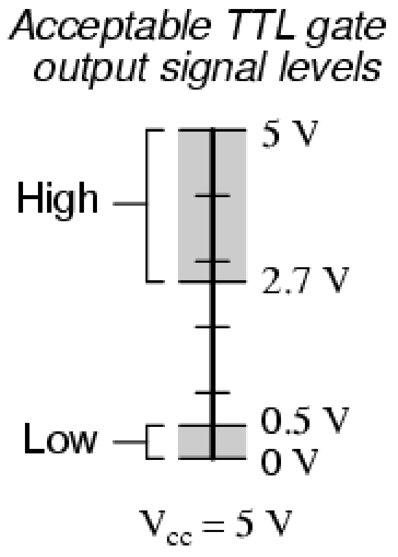
Configuring Digital I/O Pins

Configuring as OUTPUT

- An LED is connected with pin 13.
- The pin should be an OUTPUT pin.
- We can configure a pin as OUTPUT by "pinMode" function.

```
pinMode(pin Number, OUTPUT)
pinMode(13, OUTPUT)
```



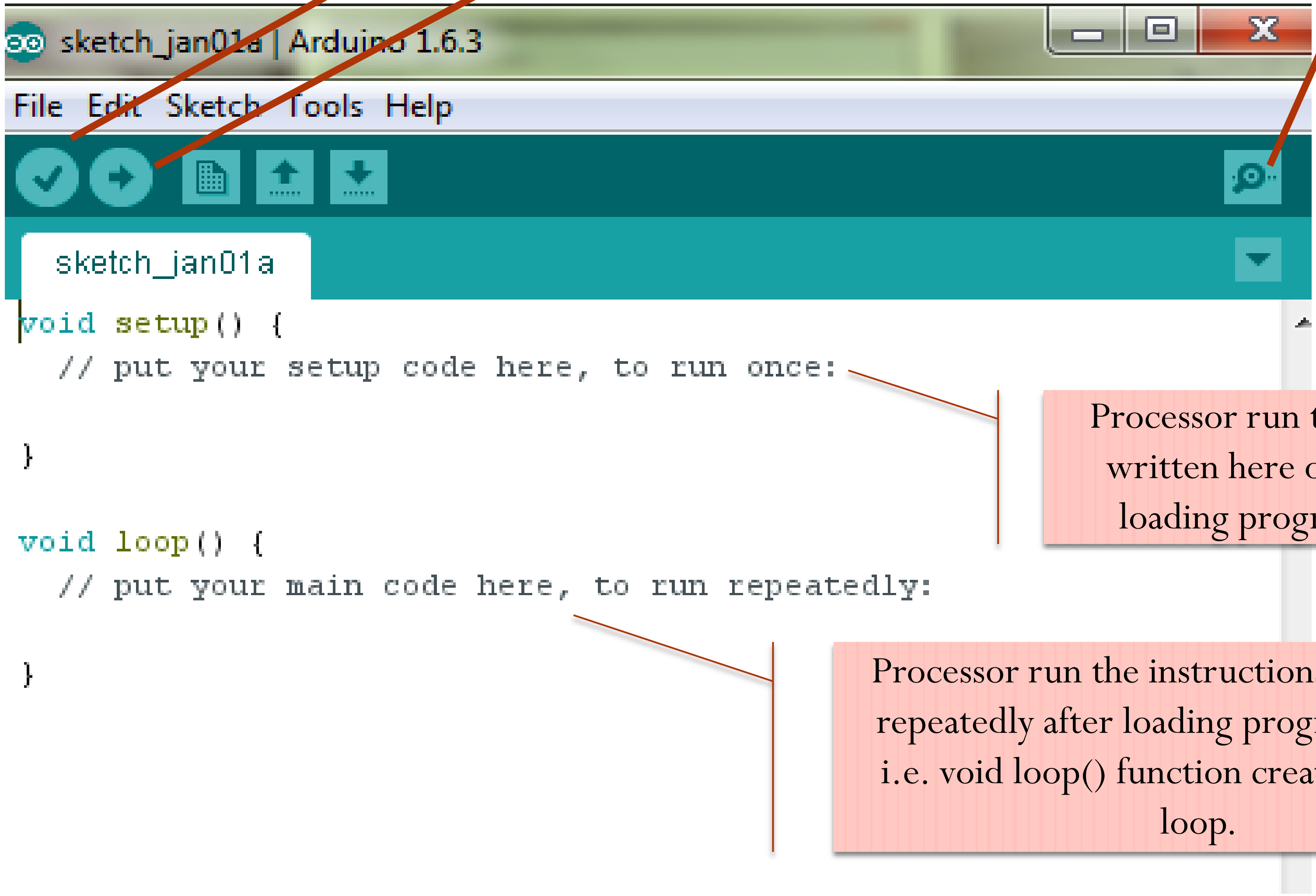


Making a Pin HIGH or LOW

- Microcontroller can make a digital pin HIGH or LOW by digitalWrite function.

```
digitalWrite(pin number, HIGH/LOW)
digitalWrite(13,HIGH)
digitalWrite(13,LOW)
```

Program Structure in ARDUINO



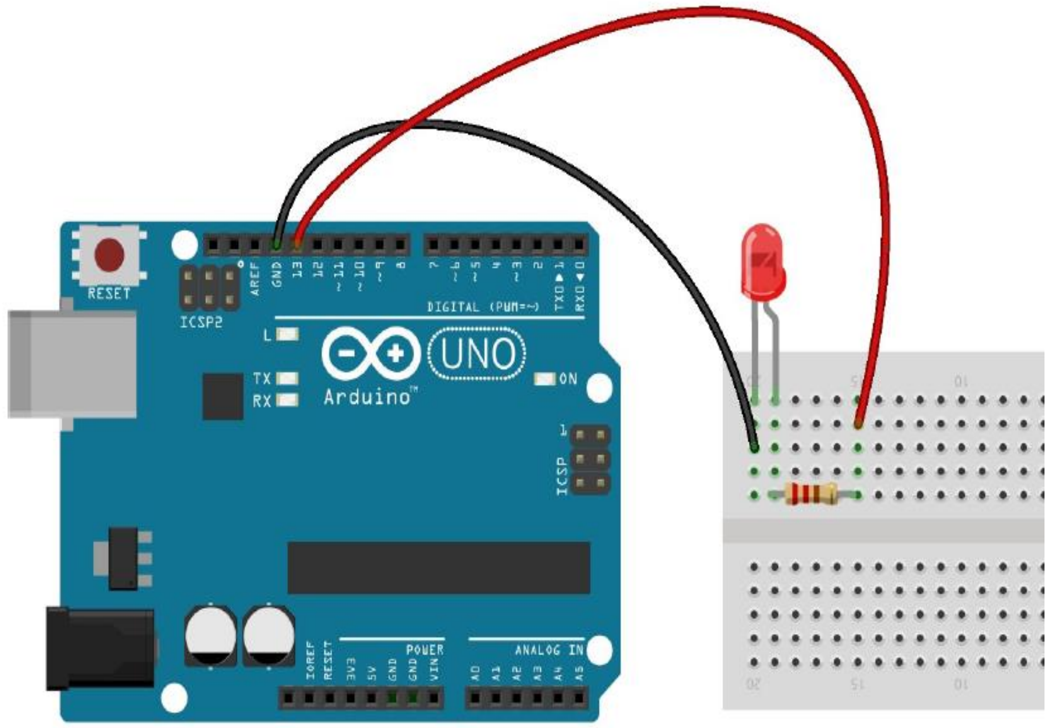
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Processor run the instructions written here only once after loading program or reset .

Processor run the instructions written here repeatedly after loading program or reset . i.e. void loop() function creates an infinite loop.

Task-1: LED Blinking

```
void setup()
{
 pinMode(13,OUTPUT);
}
void loop()
{
 digitalWrite(13,HIGH);
 delay(300);
 digitalWrite(13,LOW);
 delay(300);
}
```

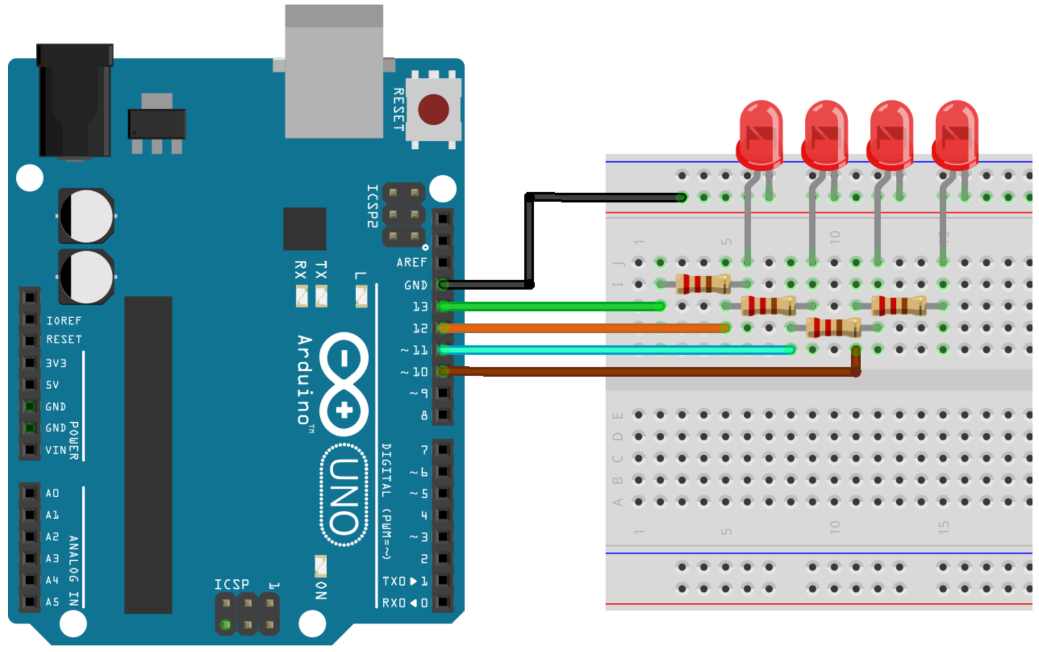


Note: To insert a time delay in the program use following function-
delay(time_in_ms);

Task-2: 4 LED's are connected with pin 13,12,11 and 10 respectively. Write program to show binary 0000 to 1111 in the LEDs at every 1 sec interval. Repeat the sequence.

Or, Design a 4-bit binary counter which automatically increases it's value by 1 at every 1 s and shows the count value to the LED's connected with I/O pins 13,12,11 and 10.

```
void setup() {
pinMode(13,OUTPUT);
pinMode(12,OUTPUT);
pinMode(11,OUTPUT);
pinMode(10,OUTPUT);
}
void loop() {
digitalWrite(13,0);
digitalWrite(12,0);
digitalWrite(11,0);
digitalWrite(10,0);
delay(1000);

digitalWrite(13,1);
digitalWrite(12,0);
digitalWrite(11,0);
digitalWrite(10,0);
delay(1000);

digitalWrite(13,0);
digitalWrite(12,0);
digitalWrite(11,1);
digitalWrite(10,0);
delay(1000);

digitalWrite(13,0);
digitalWrite(12,0);
digitalWrite(11,1);
digitalWrite(10,1);
delay(1000);

digitalWrite(13,0);
digitalWrite(12,1);
digitalWrite(11,0);
digitalWrite(10,0);
delay(1000);
}
```



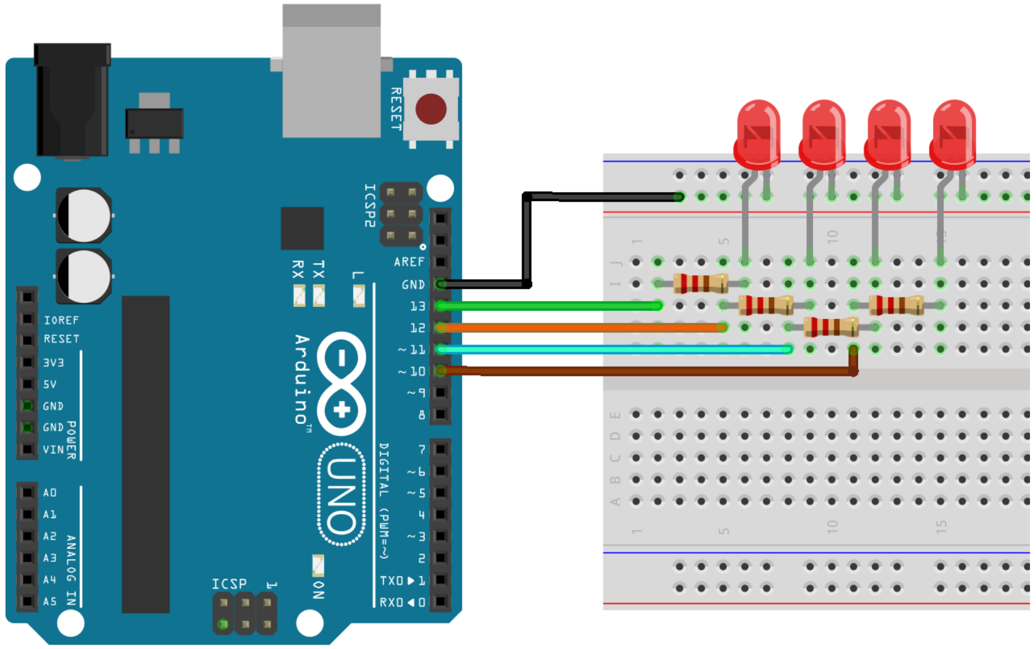
Skill Task-1

Design a 5-bit binary counter which automatically increases it value by 1 at every 1 sec and shows the count value to the 5 LED's connected with I/O pins.

Task-2: 4 LED's are connected with pin 13,12,11 and 10 respectively. Write program to show binary 0000 to 1111 in the LEDs at every 1 sec interval. Repeat the sequence.

Or, Design a 4-bit binary counter which automatically increases it's value by 1 at every 1 s and shows the count value to the LED's connected with I/O pins 13,12,11 and 10.

```
unsigned char j=0;
boolean a,b,c,d;
void setup() {
pinMode(13,OUTPUT);
pinMode(12,OUTPUT);
pinMode(11,OUTPUT);
pinMode(10,OUTPUT);
}
void loop() {
a= j & 1;
b=j & 2;
c= j & 4;
d=j & 8;

digitalWrite(13,d);
digitalWrite(12,c);
digitalWrite(11,b);
digitalWrite(10,a);
delay(1000);
j=j+1;
if(j>15) j=0;
}
```



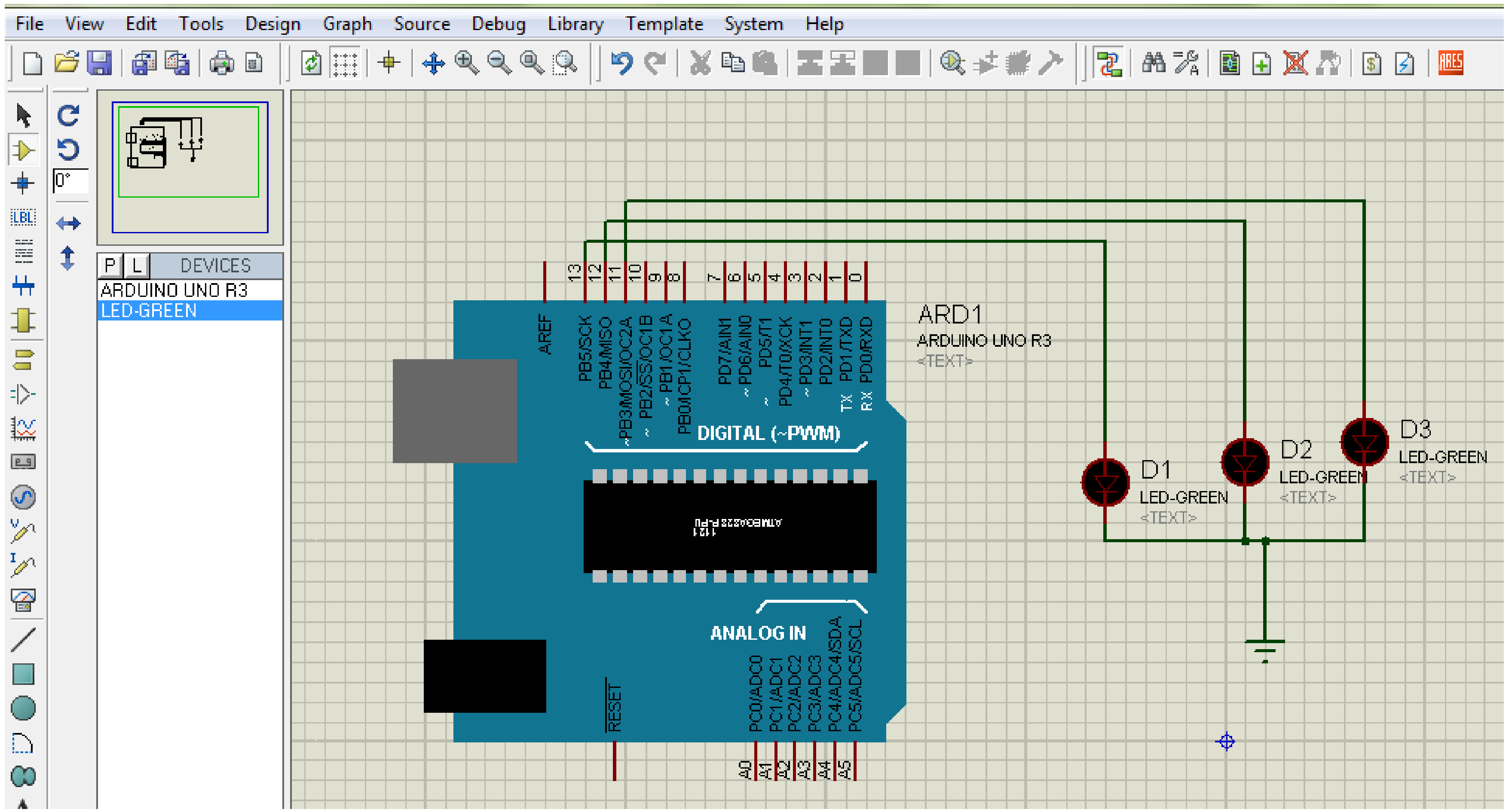
Skill Task-1

Design a 5-bit binary counter which automatically increases it value by 1 at every 1 sec and shows the count value to the 5 LED's connected with I/O pins.

Simulation in Proteus

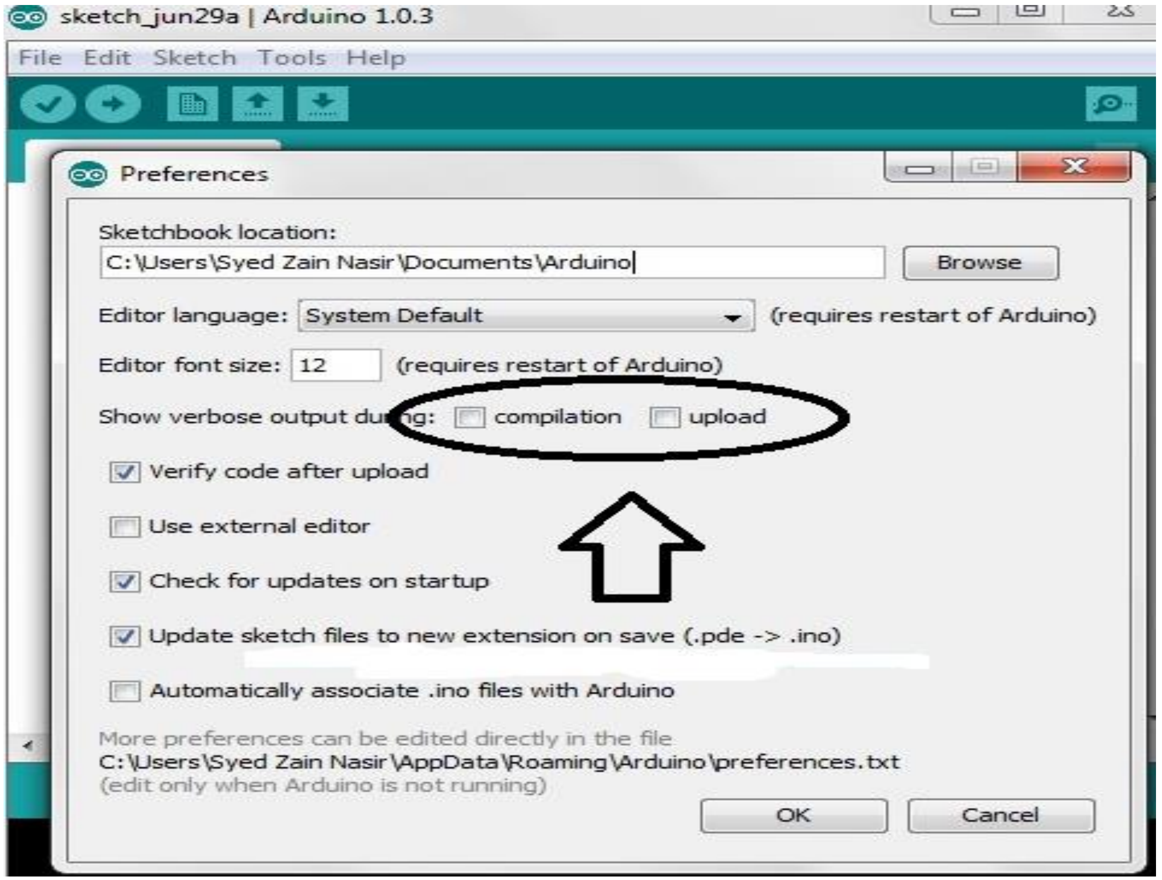
Step-1 [PROTEUS simulator]: Add arduino library in proteus (if the arduino library already exist skip this step)

Step-2 [PROTEUS simulator]: Collect necessary components from component library and connect them as per circuit diagram.



Simulation in Proteus (Cont.)

Step-3 [ARDUINO]: Get HEX file from ARDUINO software.



Sketchbook location:
C:\Users\Syed Zain Nasir\Documents\Arduino

Browse

Editor language: System Default (requires restart of Arduino)

Editor font size: 12 (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

☑ Verify code after upload

☐ Use external editor

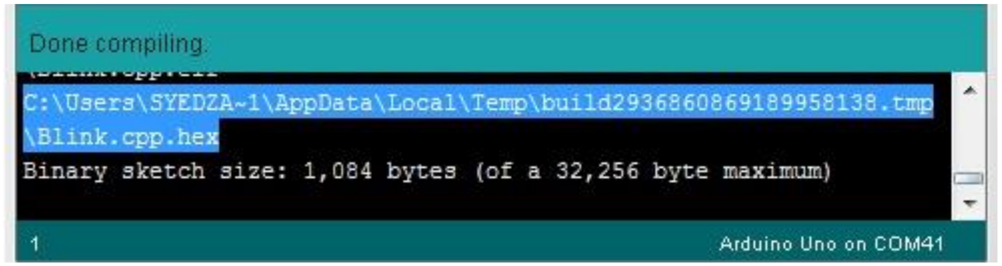
☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☐ Automatically associate .ino files with Arduino

More preferences can be edited directly in the file
C:\Users\Syed Zain Nasir\AppData\Roaming\Arduino\preferences.txt
(edit only when Arduino is not running)

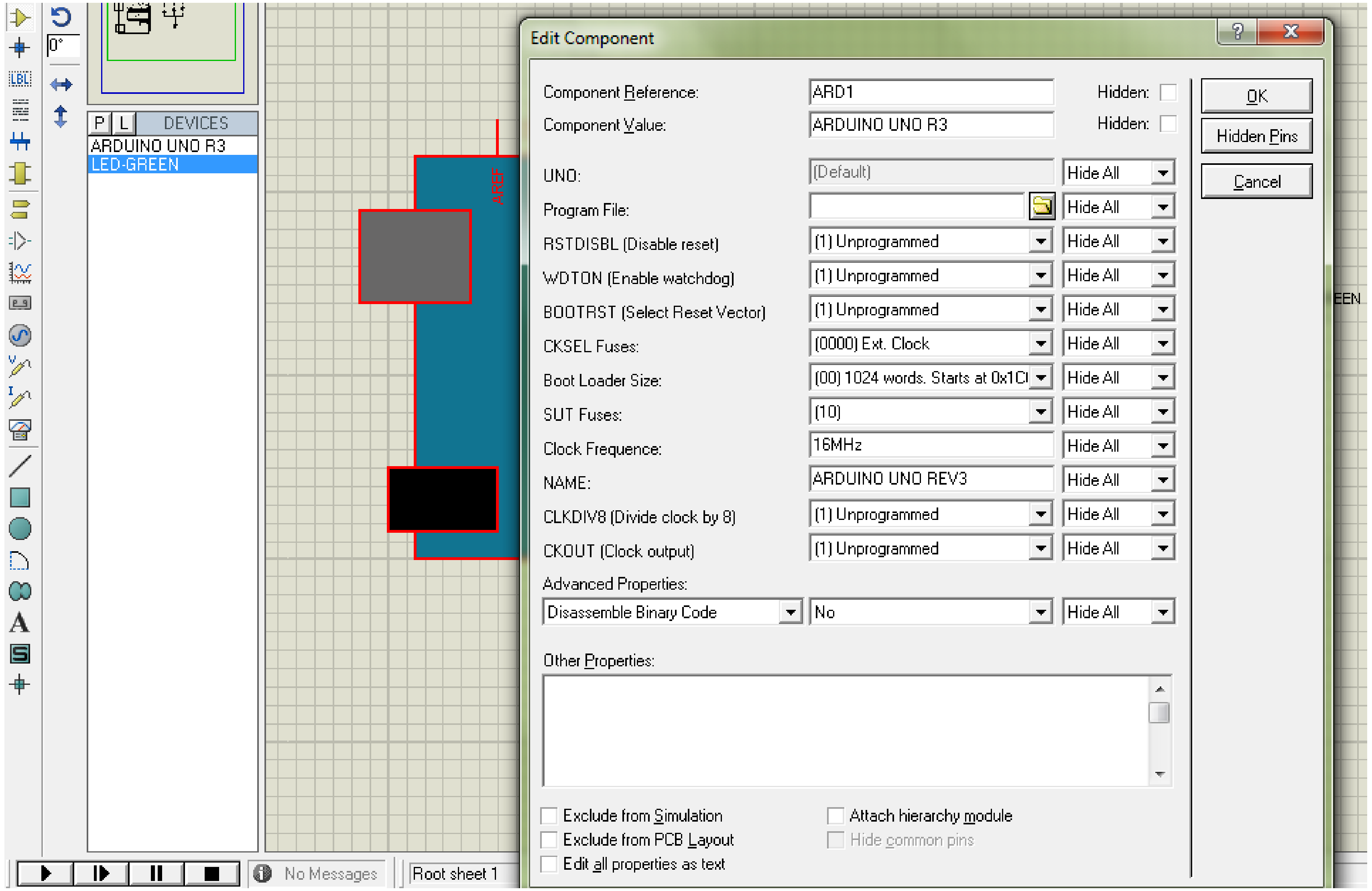
OK Cancel



```
Done compiling.
C:\Users\SYEDZA~1\AppData\Local\Temp\build2936860869189958138.tmp
\Blink.cpp.hex
Binary sketch size: 1,084 bytes (of a 32,256 byte maximum)
```

Simulation in Proteus (Cont.)

Step-4 [PROTEUS] : Load the HEX file and start simulation.



Edit Component

Component Reference:	ARD1	Hidden: ☐	
Component Value:	ARDUINO UNO R3	Hidden: ☐	
UNO:	(Default)	Hide All	
Program File:		Hide All	
RSTDISBL (Disable reset)	(1) Unprogrammed	Hide All	
WDTON (Enable watchdog)	(1) Unprogrammed	Hide All	
BOOTRST (Select Reset Vector)	(1) Unprogrammed	Hide All	
CKSEL Fuses:	(0000) Ext. Clock	Hide All	
Boot Loader Size:	(00) 1024 words. Starts at 0x1C	Hide All	
SUT Fuses:	(10)	Hide All	
Clock Frequence:	16MHz	Hide All	
NAME:	ARDUINO UNO REV3	Hide All	
CLKDIV8 (Divide clock by 8)	(1) Unprogrammed	Hide All	
CKOUT (Clock output)	(1) Unprogrammed	Hide All	

Advanced Properties:

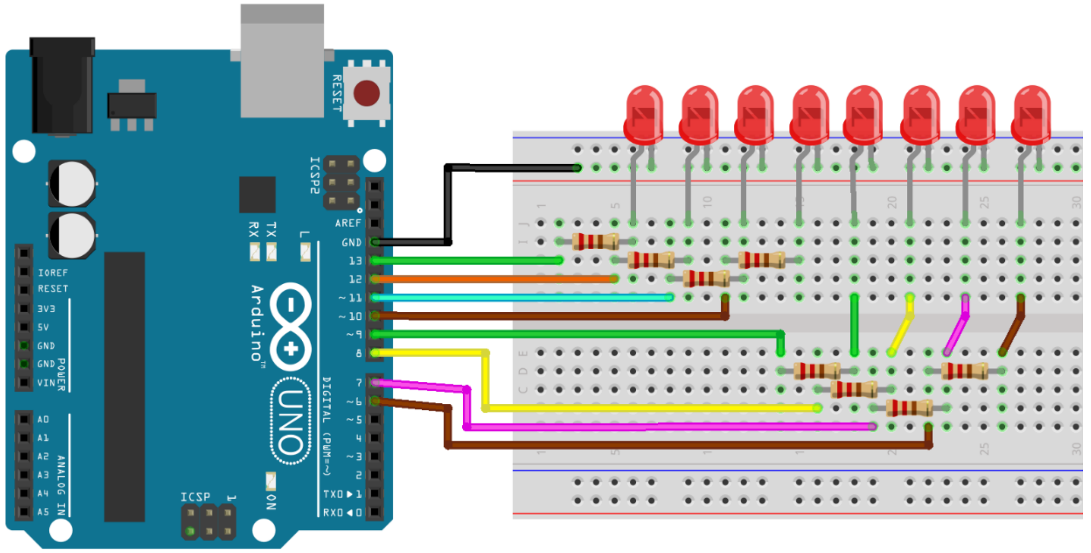
Disassemble Binary Code	No	Hide All

Other Properties:

☐ Exclude from Simulation
☐ Exclude from PCB Layout
☐ Edit all properties as text
☐ Attach hierarchy module
☐ Hide common pins

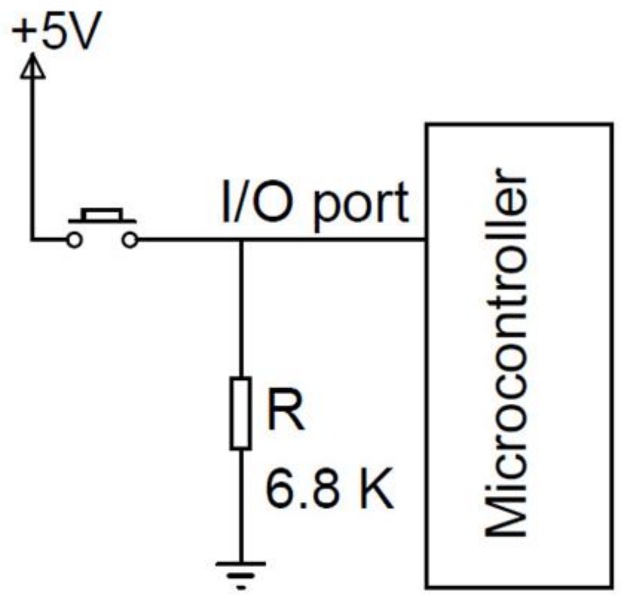
OK
Hidden Pins
Cancel

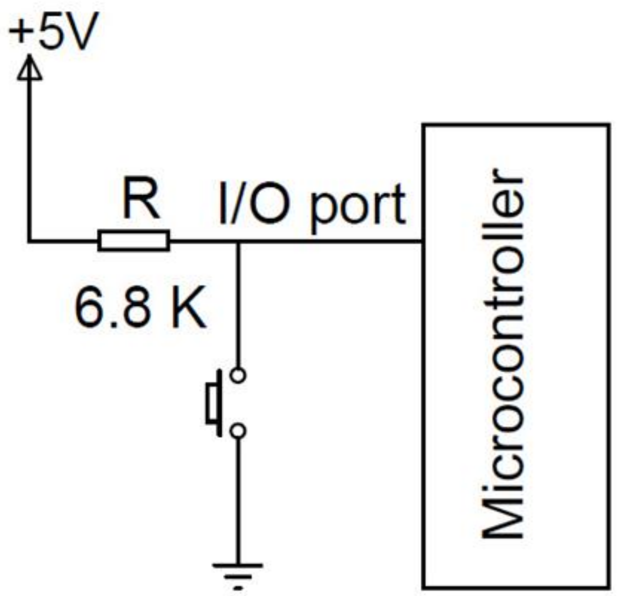
Task-3: Suppose, there is a 8-bit number 00000001. Write a program in ARDUINO to shift the number in the left direction by 1 position in each time and show the value after shift to the I/O pins of microcontroller. Continue the shift operation until the number becomes zero and repeat the sequence. Keep 1 sec time delay between two shift operation.



```
byte number;
boolean a,b,c,d,e,f,g,h;
void setup() {
pinMode(13,OUTPUT);
pinMode(12,OUTPUT);
pinMode(11,OUTPUT);
pinMode(10,OUTPUT);
pinMode(9,OUTPUT);
pinMode(8,OUTPUT);
pinMode(7,OUTPUT);
pinMode(6,OUTPUT);
}
void loop() {
for (number = 00000001;number>0; number <<= 1)
{
a= number & 1;
b=number & 2;
c= number & 4;
d=number & 8;
e=number & 16;
f= number & 32;
g=number & 64;
h=number & 128;
digitalWrite(13,a);
digitalWrite(12,b);
digitalWrite(11,c);
digitalWrite(10,d);
digitalWrite(9,e);
digitalWrite(8,f);
digitalWrite(7,g);
digitalWrite(6,h);
delay(1000);
}}
```

Interfacing switch: Pullup and Pulldown resistor



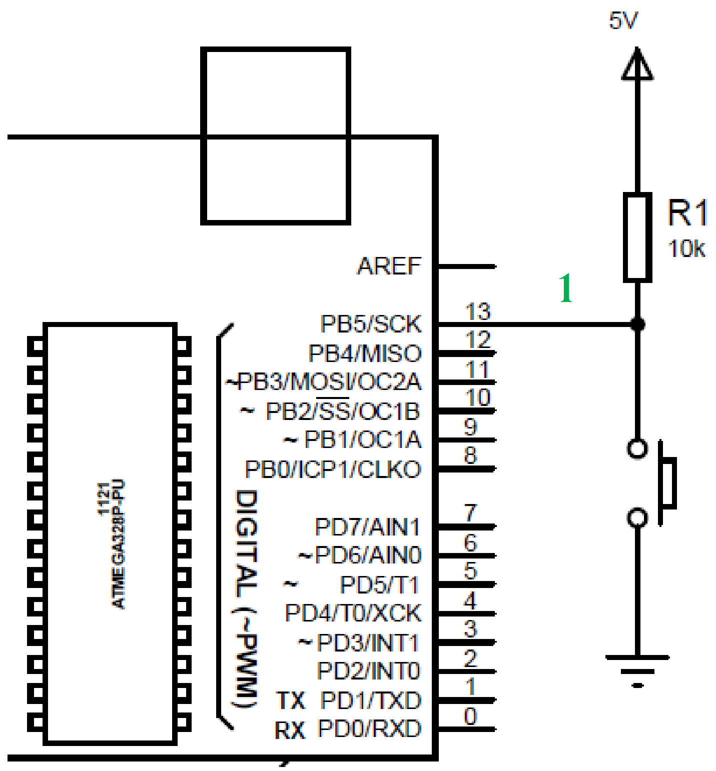


- ✓ The state of a digital input pin must be HIGH or LOW.
- ✓ If the digital input pin is not connected to LOW or HIGH logic then it is called floating state.
- ✓ Pull-up or Pull down resistor remove floating state of a digital input pin.

Reading a Push-Button: Reading a digital Input Pin

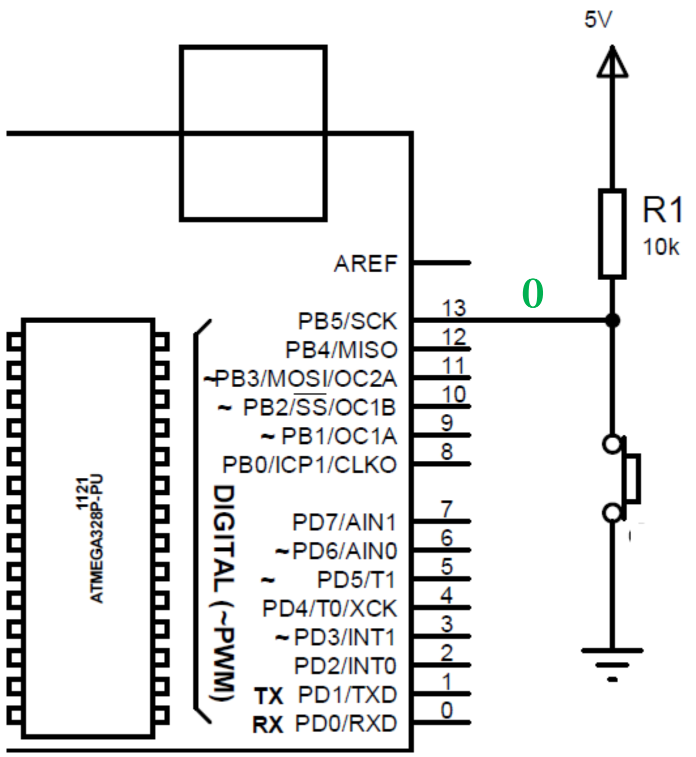
Reading data from an input pin

digitalRead(pin_number)



Button not pressed

digitalRead(13) returns 1

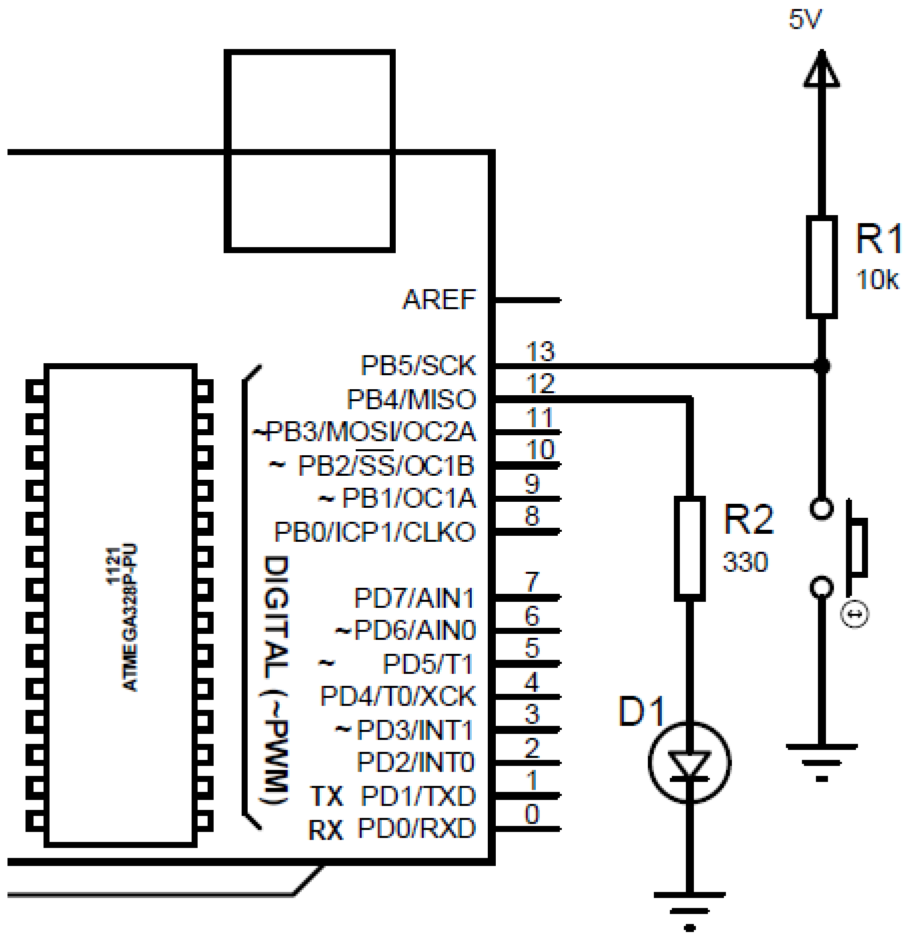


Button is pressed

digitalRead(13) returns 0

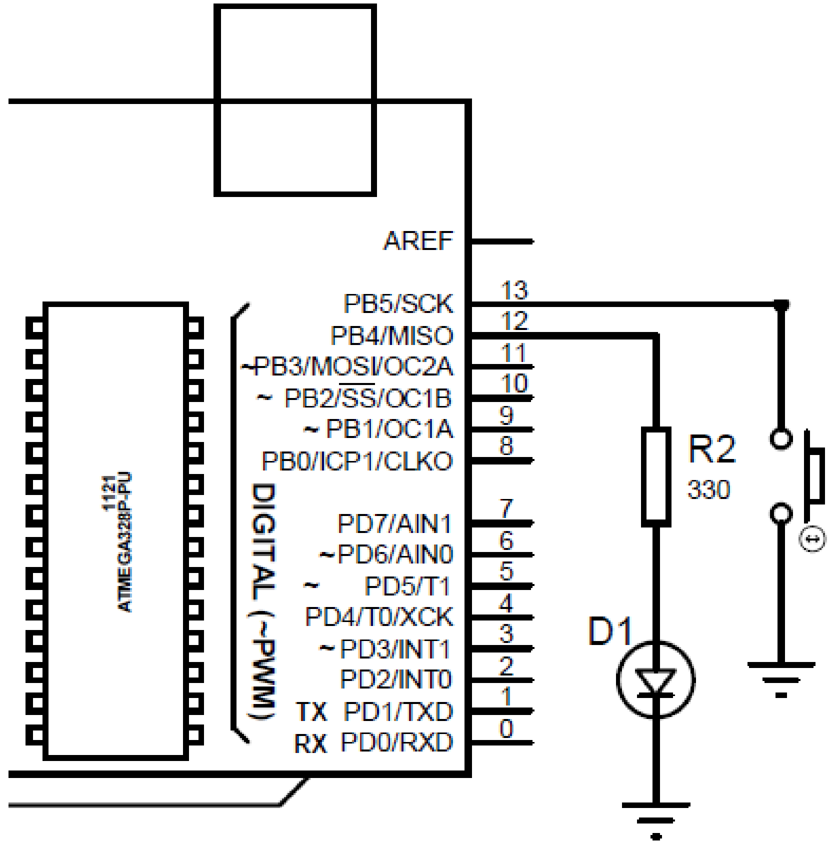
Task-4: PUSH Button and External Pull up Resistor

```
boolean a;
void setup()
{
pinMode (12, OUTPUT);
pinMode (13, INPUT);
}
void loop()
{
a=digitalRead(13);
If(a==HIGH)
digitalWrite(12,LOW);
else
digitalWrite(12,HIGH);
}
```



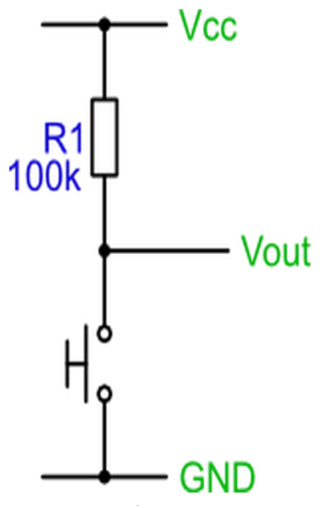
Task-5: Internal Pull up resistor

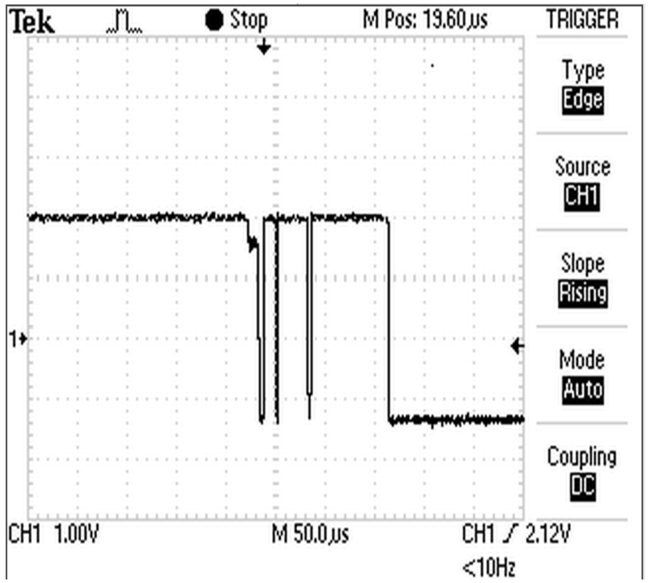
```
boolean switch;
void setup()
{
pinMode(12, OUTPUT);
pinMode(13,INPUT);
digitalWrite(13,HIGH);
}
Void loop()
{
switch=digitalRead(13);
if(switch==LOW)
digitalWrite(13,HIGH);
else
digitalWrite(13,LOW);
}
```



Skill Task-2: A press in switch blinks LED 10 times.

Bouncing Problem (Switch Debouncing)





Switch bouncing is another real-world problem that happens too quickly for human perception but which can doom an electronics project. When a switch is toggled, contacts have to physically move from one position to another. As the components of the switch settle into their new position, they mechanically bounce, causing the underlying circuit to be opened and closed several times. For embedded systems designers, the most common exposure to the problem is with user interface switches, in which proper care must be taken to correctly count the number of times a user presses and releases a switch.

Task-6 : A press in switch increase a 4-bit variable and it is shown by 4 LED’s.

```
#include <Bounce.h>
const int buttonPin = 12;
const int LED0=11, LED1=10, LED2=9,LED3=8;
Bounce button = Bounce(buttonPin,10);  // 10 ms
debounce
void setup() {
  pinMode(buttonPin, INPUT);
  digitalWrite(buttonPin, HIGH);
  pinMode(LED0,OUTPUT);
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);
  pinMode(LED3,OUTPUT);
}
byte a;
```

```
void loop() {
if (button.update())
{
if( button.risingEdge())
a+=1;
}

digitalWrite(LED0,a&1);
  digitalWrite(LED1,a&2);
  digitalWrite(LED2,a&4);
  digitalWrite(LED3,a&8);
}
```

Skill Task-3 : Let, two switches SW0 and SW1 are connected to microcontroller. Write a program to increase a 4-bit variable when user press SW0 and decrease the same four bit variable when user press SW1. Also show the 4-bit variable in 4 LED’s connected with microcontroller.

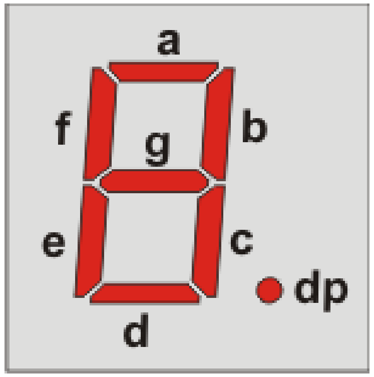
Exp-9
Interfacing with I/O devices

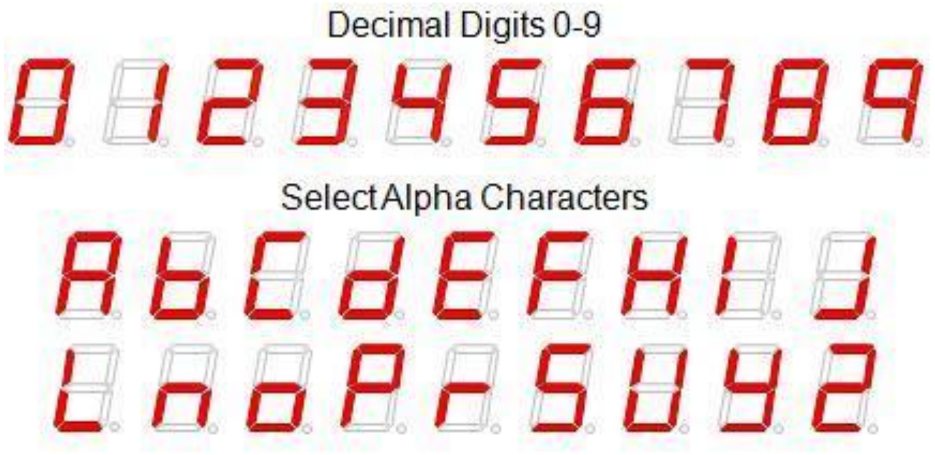
Objectives

- To familiarization with 7-segment display: Common anode and Common Cathode.
- To know the interfacing of 7-segment display with microcontroller.
- To know the interfacing of multiple 7-segment display with microcontroller.
- To develop a product counter.

7-segment Display

The 7-segment display, also written as "seven segment display", consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a **numerical digit (both Decimal and Hex) to be displayed**. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal point, (DP) when two or more 7-segment displays are connected together to display numbers greater than ten.

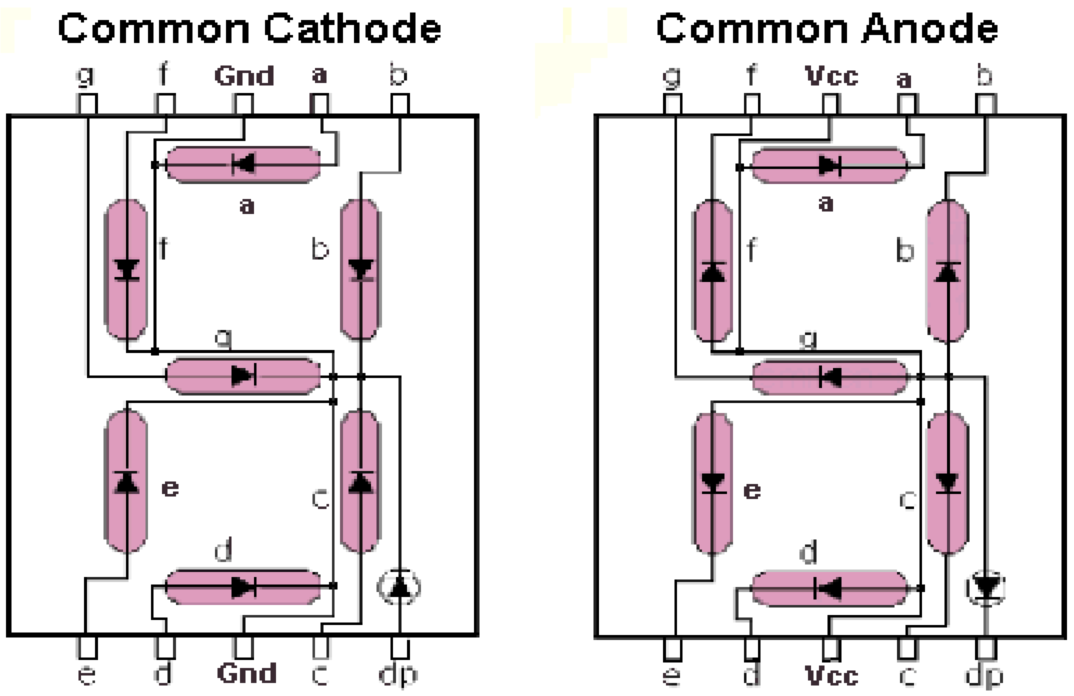




Common Anode and Common Cathode 7-segment Display

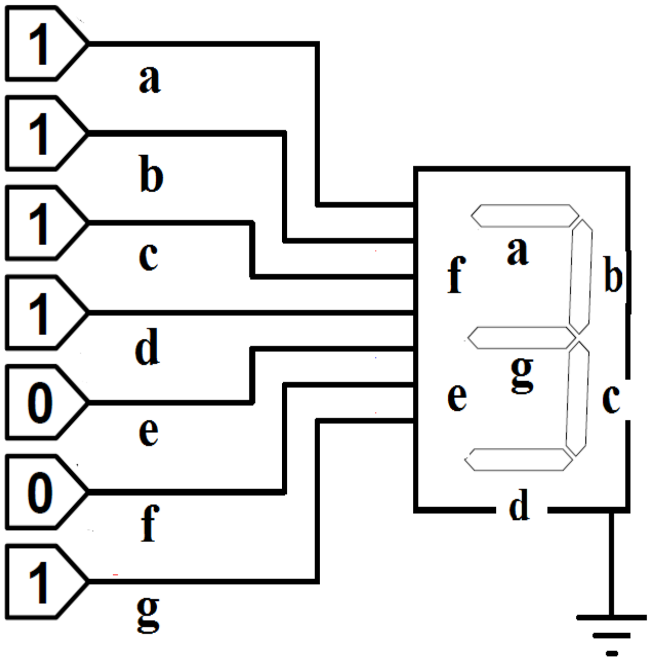
The Common Cathode (CC) – In the common cathode display, all the cathode connections of the LED segments are joined together to logic "0" or ground. The individual segments are illuminated by application of a "HIGH", or logic "1"

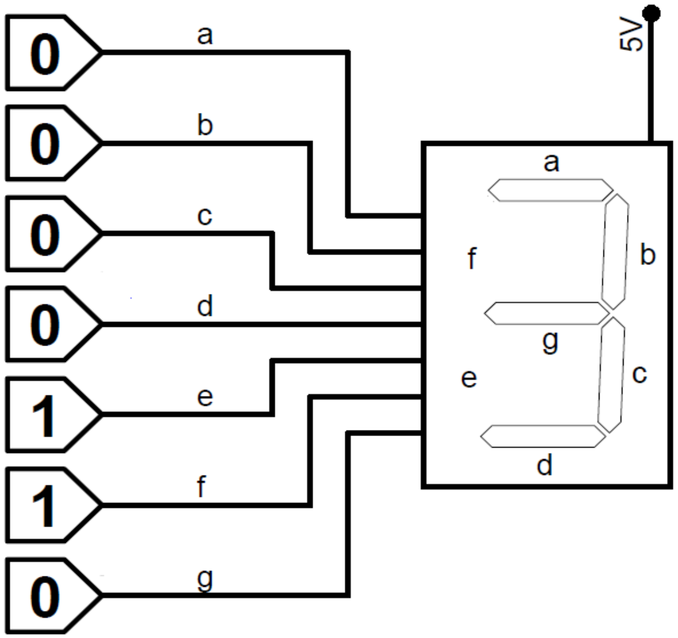
The Common Anode (CA) – In the common anode display, all the anode connections of the LED segments are joined together to logic "1". The individual segments are illuminated by applying a ground, logic "0" or "LOW"



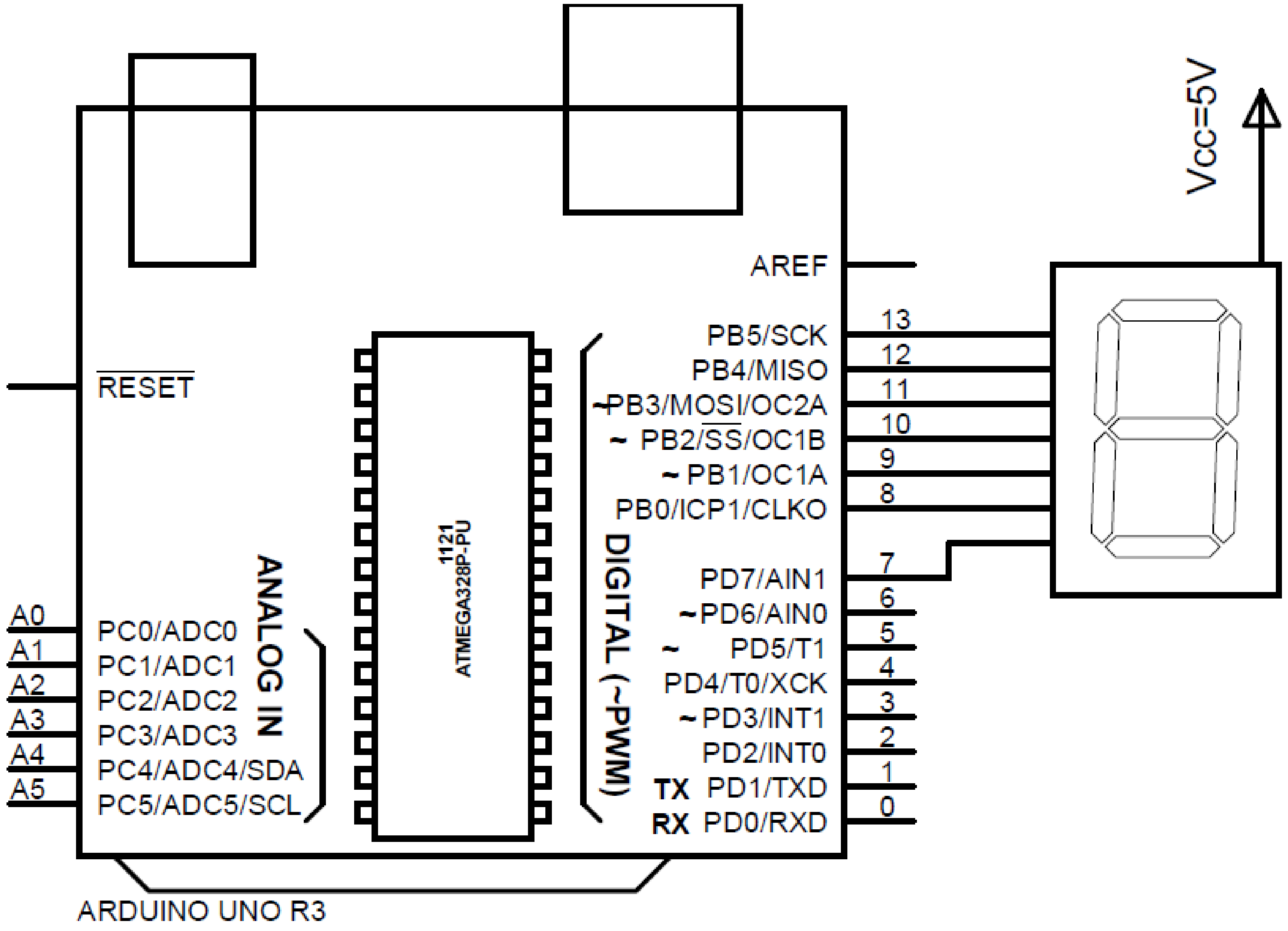
Common Anode and Common Cathode 7-segment Display (Cont.)

Displaying 3 in common cathode and common anode 7-segment display



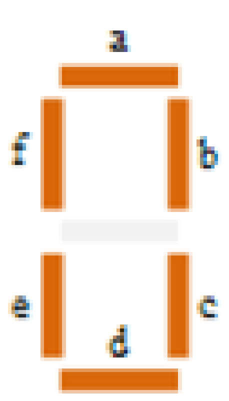


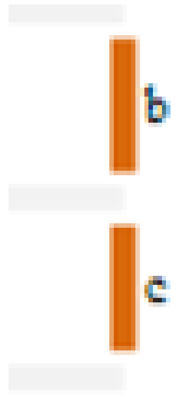
Task-1: Displaying a number in Common Anode 7-segment Display.



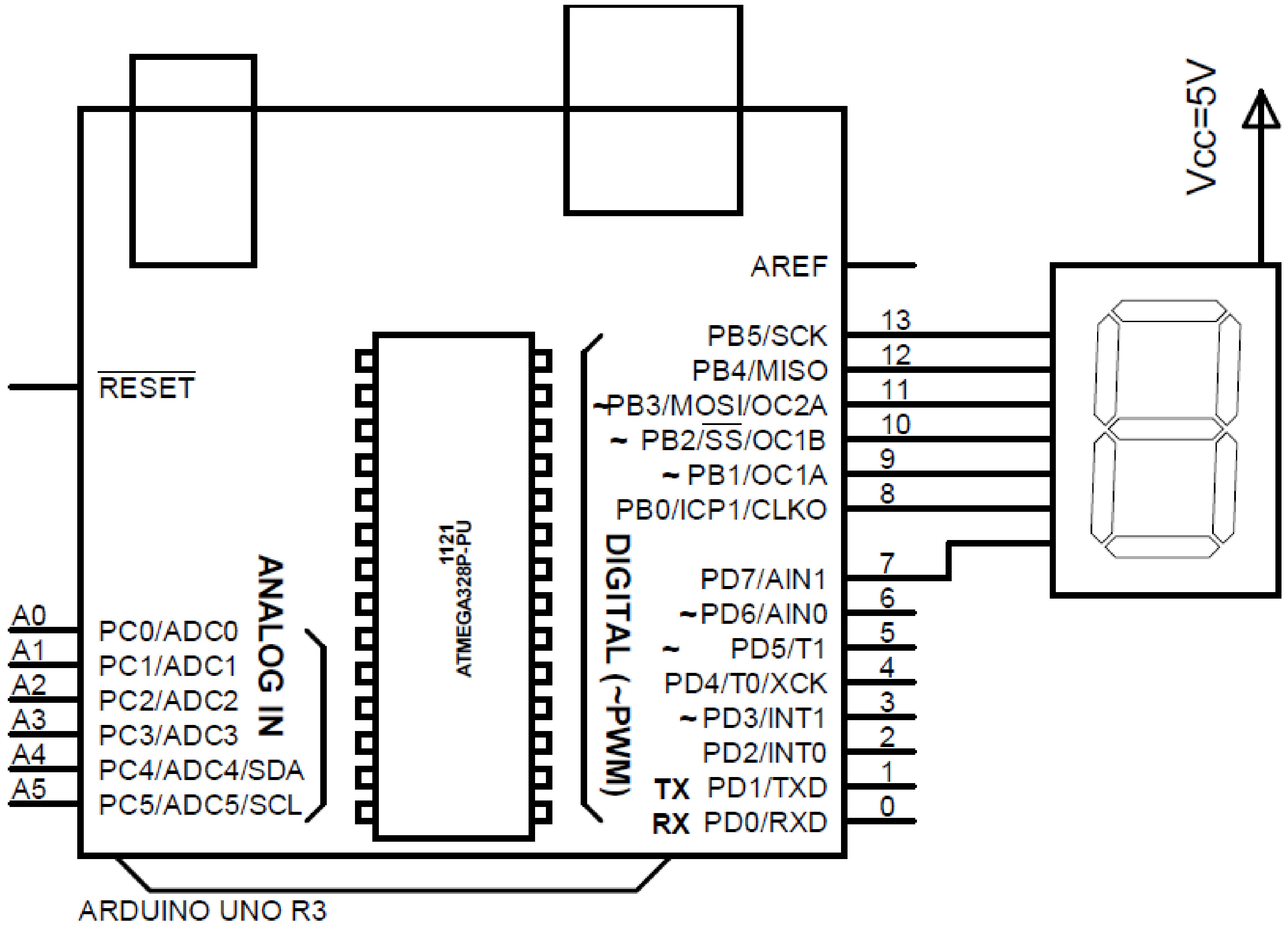
Program of Task-1

```
const int a=13,b=12,c=11,d=10,e=9,f=8, g=7;
void setup() {
pinMode(a,OUTPUT);
pinMode(b,OUTPUT);
pinMode(c,OUTPUT);
pinMode(d,OUTPUT);
pinMode(e,OUTPUT);
pinMode(f,OUTPUT);
pinMode(g,OUTPUT);
}
void loop() {
digitalWrite(a,LOW);
digitalWrite(b,LOW);
digitalWrite(c,LOW);
digitalWrite(d,LOW);
digitalWrite(e,LOW);
digitalWrite(f,LOW);
digitalWrite(g,HIGH);
delay(1000);
digitalWrite(a,HIGH);
digitalWrite(b,LOW);
digitalWrite(c,LOW);
digitalWrite(d,HIGH);
digitalWrite(e,HIGH);
digitalWrite(f,HIGH);
digitalWrite(g,HIGH);
delay(1000);
}
```

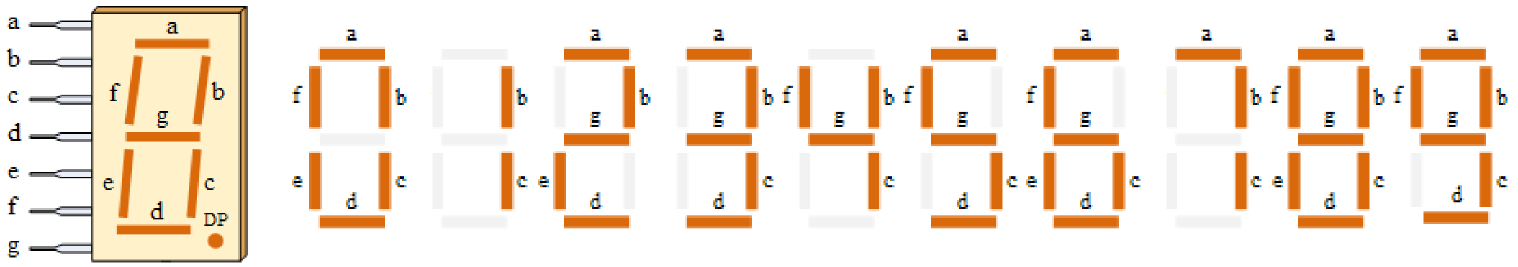




Task-2: Displaying 0-9 in Common Anode 7-segment Display



Number Conversion (BCD to 7-segment code)



Number	g Pin 7	f Pin 8	e Pin 9	d Pin 10	c Pin 11	b Pin 12	a Pin 13	Nember in decimal
0	1	0	0	0	0	0	0	64
1	1	1	1	1	0	0	1	121
2	0	1	0	0	1	0	0	36
3	0	1	1	0	0	0	0	48
4	0	0	1	1	0	0	1	25
5	0	0	1	0	0	1	0	18
6	0	0	0	0	0	1	0	2
7	1	1	1	1	0	0	0	120
8	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	16

Bit separation and sending to pin

number = g f e d c b a

to separate bit 'a' from number

AND operation →

$$
\begin{array}{ccccccc}
g & f & e & d & c & b & a \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & a
\end{array}
$$

Example

to separate 5th bit from number 1111001

$$
\begin{array}{ccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 1 & 0 & 0 & 0 & 0
\end{array}
$$

(5th bit is 1)

To separate 1st bit,	a= (number & 1)	Sending to pin: digitalWrite(13,a)
To separate 2nd bit,	b= (number & 2)	Sending to pin: digitalWrite(12,b)
To separate 3rd bit,	c= (number & 4)	Sending to pin: digitalWrite(11,c)
To separate 4th bit,	d= (number & 8)	Sending to pin: digitalWrite(10,d)
To separate 5th bit,	e= (number &16)	Sending to pin: digitalWrite(9,e)
To separate 6th bit,	f= (number &32)	Sending to pin: digitalWrite(8,f)
To separate 7th bit,	g= (number &64)	Sending to pin: digitalWrite(7,g)

Program ofTask-2

```
const int aa=13, bb=12, cc=11,dd=10;
const int ee=9,ff=8,gg=7;
boolean a,b,c,d,e,f,g;
unsigned char i=0;
unsigned char number;
void setup() {
pinMode(aa,OUTPUT);
pinMode(bb,OUTPUT);
pinMode(cc,OUTPUT);
pinMode(dd,OUTPUT);
pinMode(ee,OUTPUT);
pinMode(ff,OUTPUT);
pinMode(gg,OUTPUT);
}
void loop() {
/* Number Conversion */
 if(i==0) number=64;
else if(i==1) number=121;
else if(i==2) number=36;
else if (i==3) number=48;
else if (i==4) number=25;
else if (i==5) number=18;
else if (i==6) number=2;
else if(i==7) number=120;
else if(i==8) number=0;
else if(i==9) number=16;
/*Number Separation */
a=number & 1;
b= number & 2;
c= number & 4;
d = number & 8;
e= number & 16;
f = number & 32;
g = number & 64;
/* Sending to Pin */
digitalWrite(aa,a);
digitalWrite(bb,b);
digitalWrite(cc,c);
digitalWrite(dd,d);
digitalWrite(ee,e);
digitalWrite(ff,f);
digitalWrite(gg,g);
delay(1000);
 i=i+1;
  if(i>9) i=0;
}
```

Bit Separation Function

bitRead(number, bit_position)

Example: To read 1st bit , seg_bit=bitRead(number,0);

Bit Separation and sending to pin

```
k=13;
for(j=0;j<7; j++)
{
seg_bit=bitRead(number,j);
digitalWrite(k,seg_bit);
k=k-1;
}
```

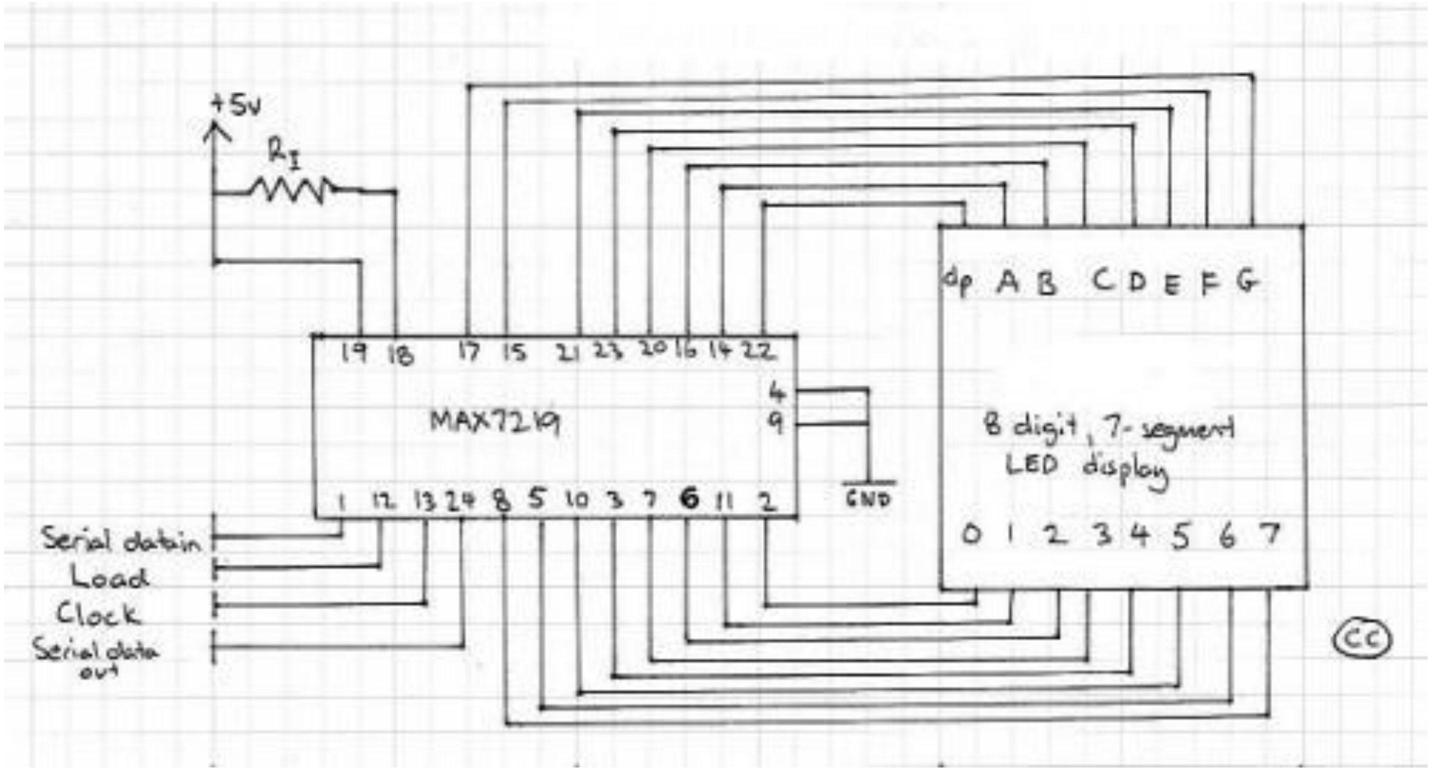
Program of Task-2 using Bit Separation Function

```
boolean seg_bit;
unsigned short i=0;
unsigned short number, j,k;
void setup() {
for(byte pin=7;pin<=13;pin++)
{
 pinMode(pin,OUTPUT);
}
}
void loop() {
/* Number Conversion */
 if(i==0) number=64;
else if(i==1) number=121;
else if(i==2) number=36;
else if (i==3) number=48;
else if (i==4) number=25;
else if (i==5) number=18;
else if (i==6) number=2;
else if(i==7) number=120;
else if(i==8) number=0;
else if(i==9) number=16;
/*Number Separation and Sending to pin */
k=13;
for(j=0;j<7; j++)
{
seg_bit=bitRead(number,j);
digitalWrite(k,seg_bit);
k=k-1;
}

delay(1000);
 i=i+1;
  if(i>9) i=0;
}
```

Program of Task-2 using Array to convert number

```
boolean seg_bit;
unsigned short i=0;
unsigned short number, j,k;
const byte numbers[10]={64,121,36,48,25,18,2,120,0,16};
void setup() {
for(byte pin=7;pin<=13;pin++)
{
 pinMode(pin,OUTPUT);
}
}
void loop()
{
/* Numbers Conversion */
number=numbers[i];
/*Number Separation and sending to
pin */
k=13;
for(j=0;j<7; j++)
{
seg_bit=bitRead(number,j);
digitalWrite(k,seg_bit);
k=k-1;
}

delay(1000);
 i=i+1;
  if(i>9) i=0;
}
```

Driving 7-segment Display by MAX7219



lc.setDigit(A, B, C, D);

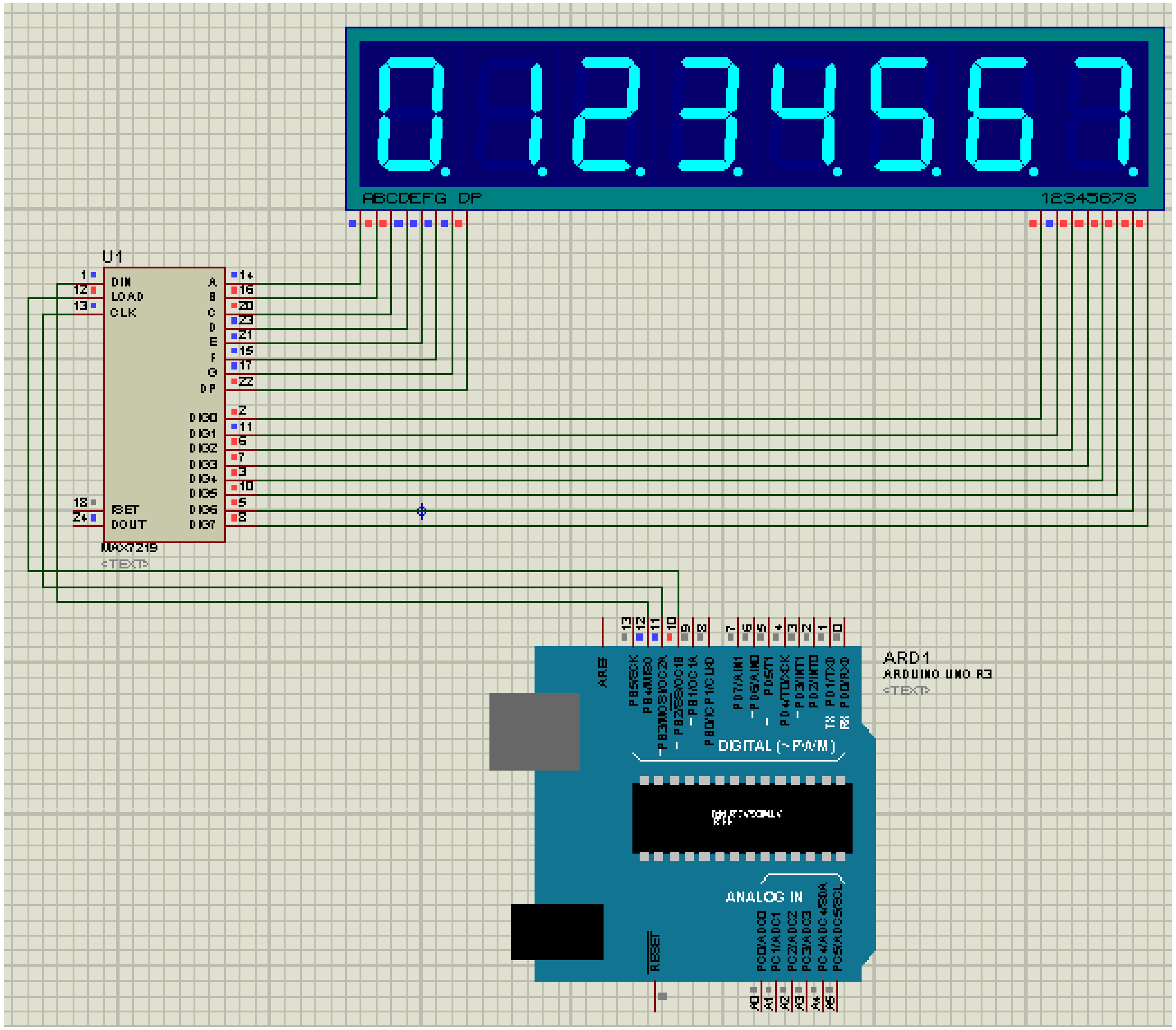
here A is the MAX7219 we're using, B is the digit to use (from a possible 0 to 7), C is the digit to display (0~9… if you use 10~15 it will display A~F respectively) and D is false/true (digit on or off). You can also send basic characters such as a dash "-" with the following:

lc.setChar(A, B,'-',false);

Problem: Interfacing eight 7-segment display by MAX7219.

```
#include "LedControl.h"
// Din=12,clk=11,cs=10
LedControl lc=LedControl(12,11,10,1);
void setup()
{
 lc.shutdown(0,false);
 lc.setIntensity(0,8);
 lc.clearDisplay(0);
}
void loop()
{
 for (int a=0; a<8; a++)
  {
   lc.setDigit(0,a,a,true);
   delay(100);
  }
for (int a=0; a<8; a++)
  {
   lc.setDigit(0,a,8,1);
   delay(100);
  }
for (int a=0; a<8; a++)
  {
   lc.setDigit(0,a,0,false);
   delay(100);
  }
 for (int a=0; a<8; a++)
  {
   lc.setChar(0,a,' ',false);
   delay(100);
  }
 for (int a=0; a<8; a++)
  {
   lc.setChar(0,a,'-',false);
   delay(100);
  }
 for (int a=0; a<8; a++)
  {
   lc.setChar(0,a,' ',false);
   delay(100);
  }
}
```

Simulation in PROTEUS



In the name of Allah, Most Gracious, Most Merciful.

EEE-314 Microprocessor and Interfacing Sessional

Experiment-10

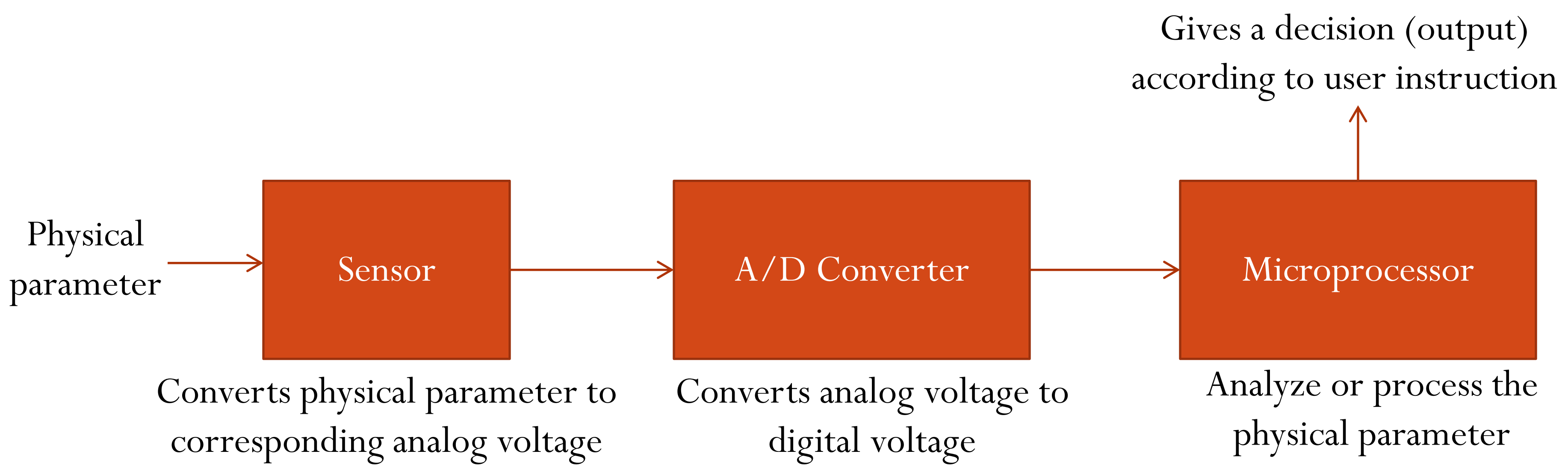
Interfacing with Analog World

Objectives

- To become familiar with analog to digital conversion process.
- To know about the internal A/D converter of ATmega328P Microcontroller.
- To convert some analog parameters like voltage, temperature by A/D converter of ATmega328P.

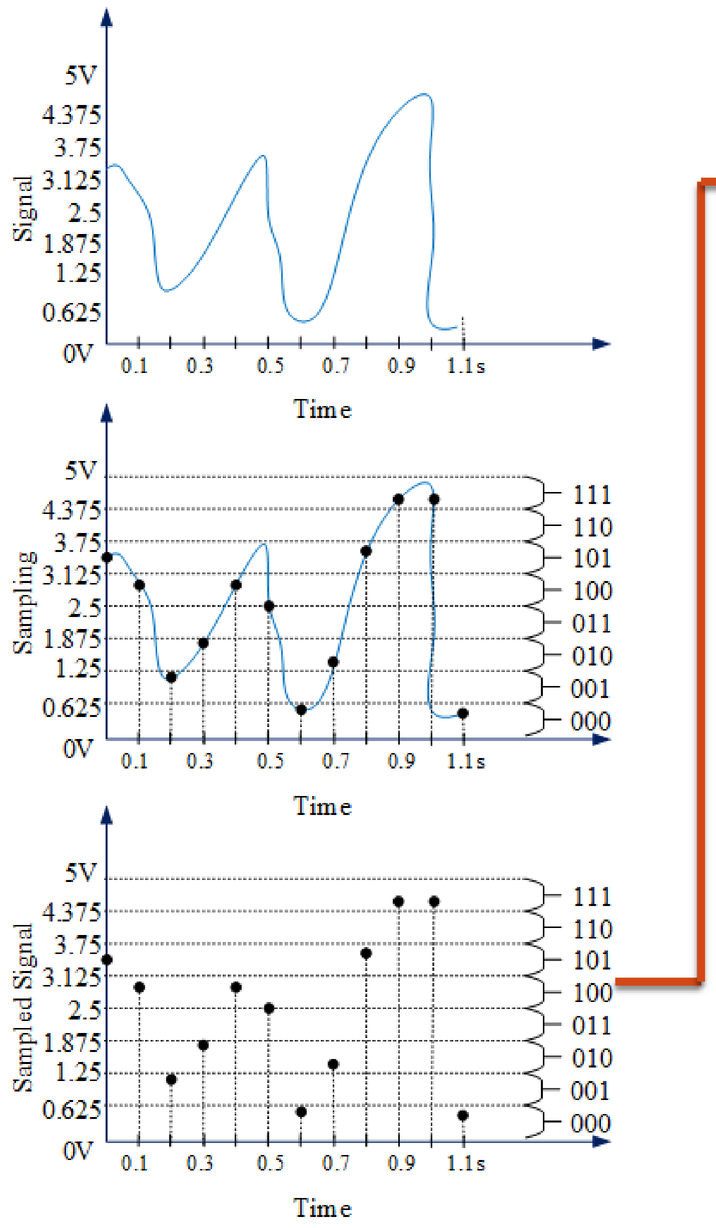
ADC (A/D Converter) in Embedded System

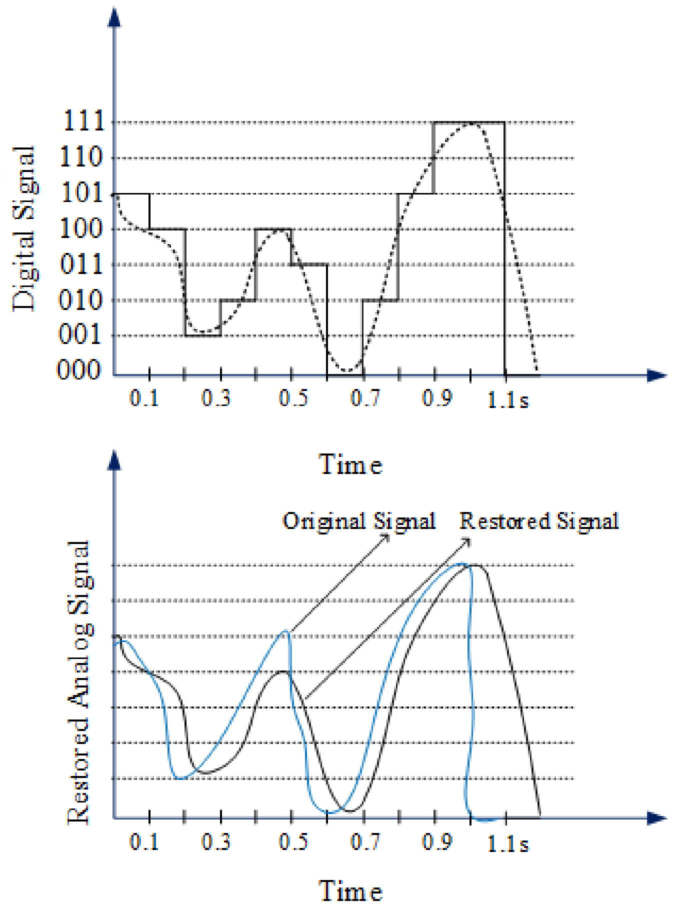
- The main use of ADC in embedded system is to measure the voltage outputs of sensors.
- Most electronic sensors produce a voltage that corresponds to temperature, pressure, acceleration or other phenomenon.
- Music, speech, or other signals can be converted to digital form by A/D converters for storage or additional processing.


Gives a decision (output) according to user instruction
Physical parameter
Sensor
A/D Converter
Microprocessor
Converts physical parameter to corresponding analog voltage
Converts analog voltage to digital voltage
Analyze or process the physical parameter

Basic of A/D Conversion

3-bit A/D converter with sampling rate of 10 samples/sec

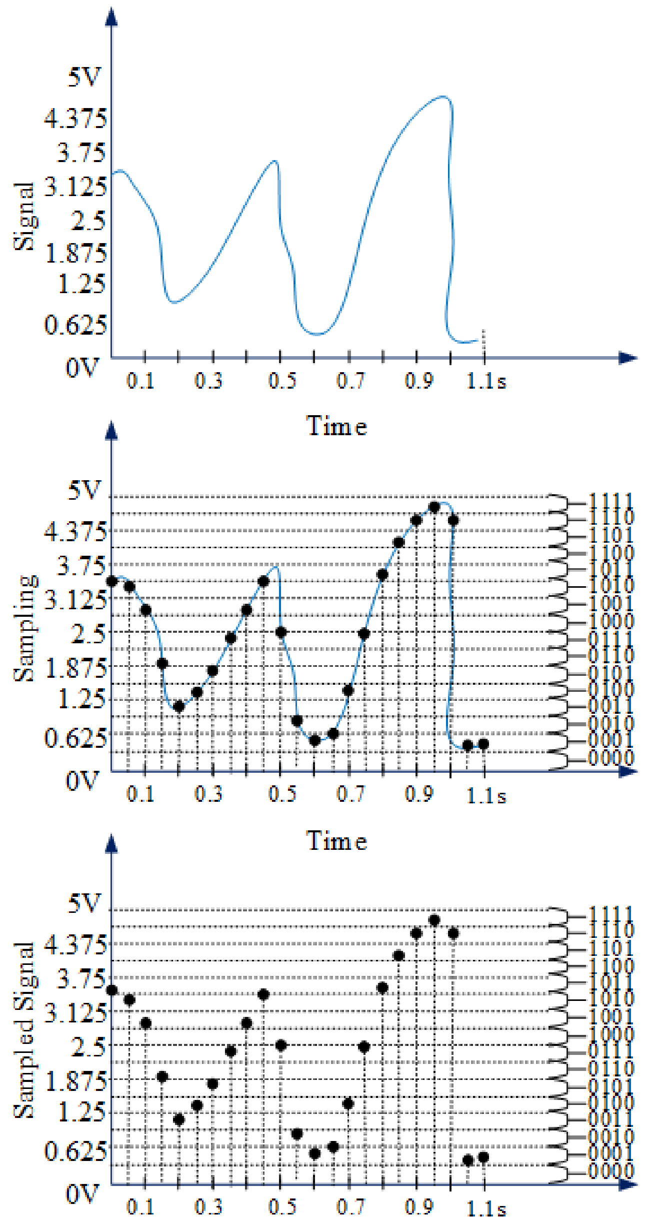


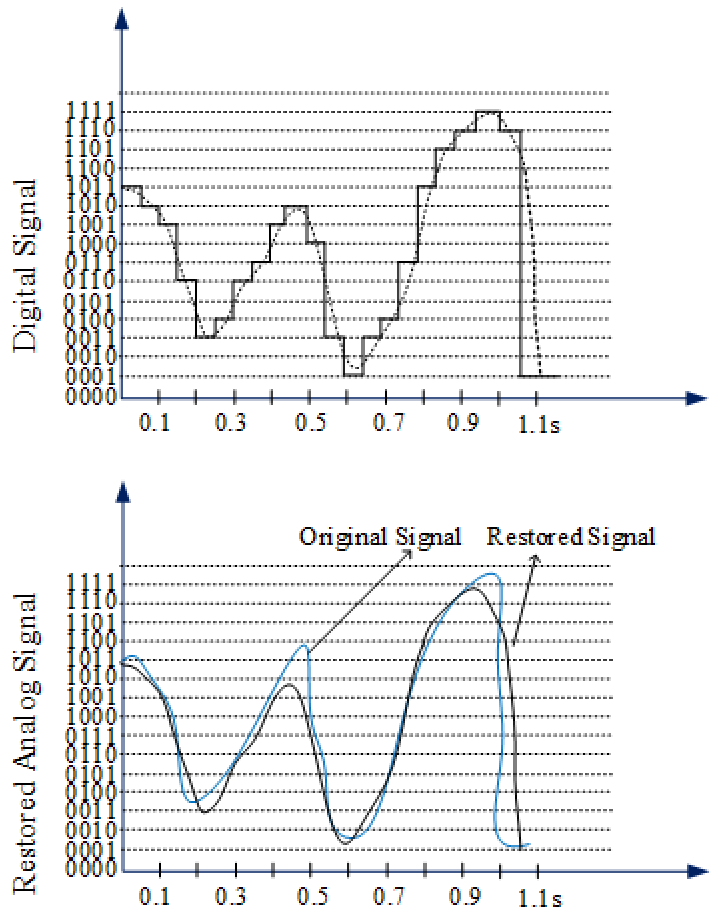


Sampling Rate= 10 samples/Sec
3-bit A/D converter
So, to store the signal of length 1.1 s we need (10*1.1*3) = 33 bit memory.

Basic of A/D Conversion

4-bit A/D converter with sampling rate of 20 samples/sec

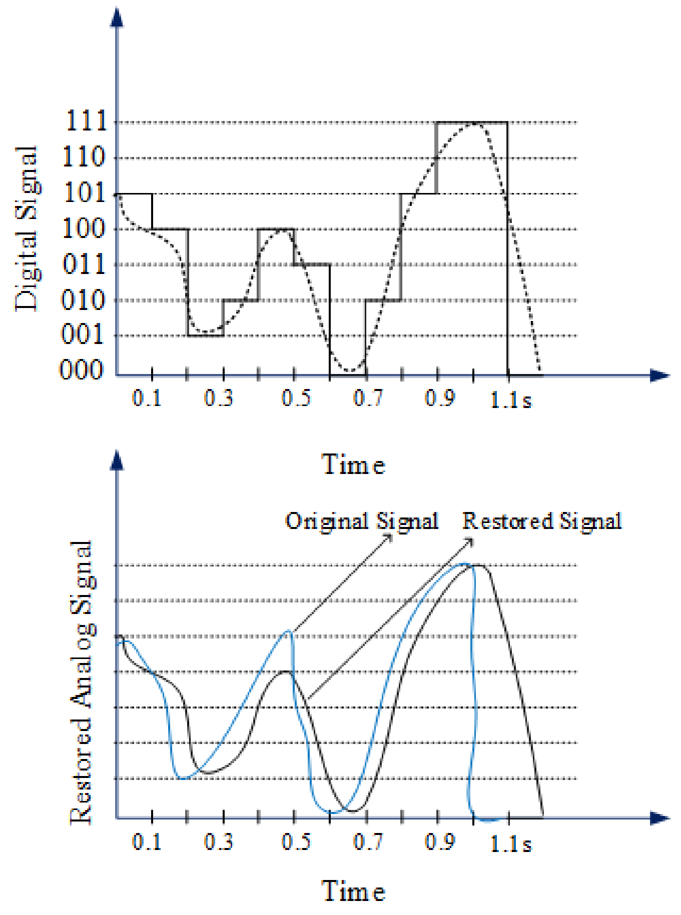




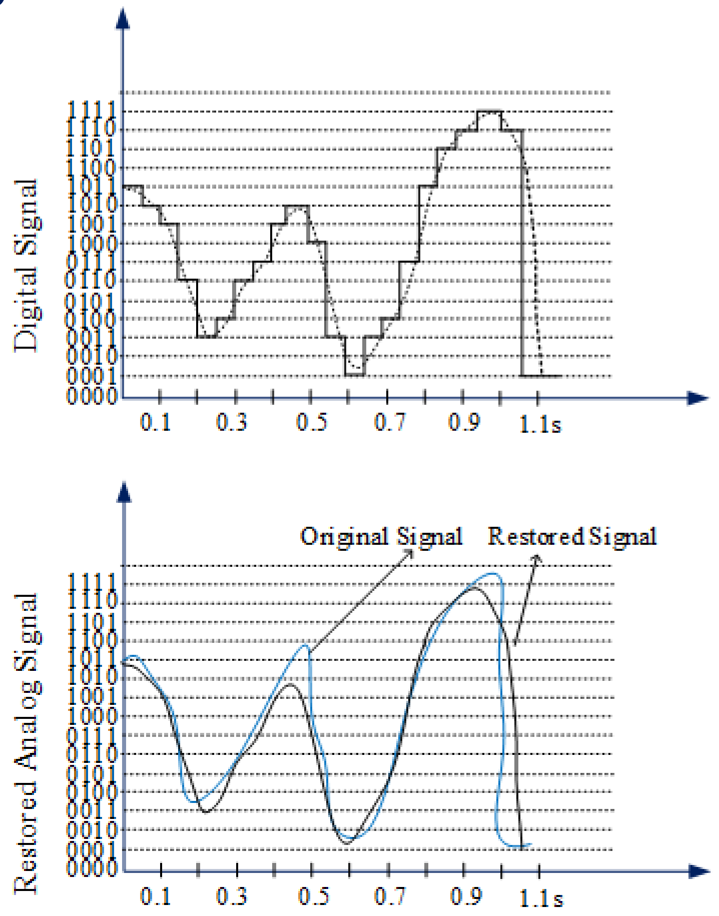
Sampling Rate= 20 samples/Sec
4-bit A/D converter
So, to store the signal of length 1.1 s we need
(20*1.1*4) = 88 bit memory. = 11 Byte

Basic of A/D Conversion: Performance analysis of two A/D converters

3-bit A/D converter with sampling rate of 10 samples/sec



4-bit A/D converter with sampling rate of 20 samples/sec



4-bit A/D converter with sampling rate 20 sample/sec is better than 3-bit A/D converter with sampling rate 10 sample/sec because converter output of 2nd one is more closer to original signal. But, 2nd one need 88-bit memory to store the signal where 1st one need only 33-bit memory.

A/D Converter in ATmega328P (ARDUINO UNO)

- 10-bit resolution (Divide 0 to Ref Volt into 1024 levels)
- 6 channel
- Internal or, External reference selection

Functions

analogRead()

Reads the analog voltage from the specified analog pin and returns the digital value (0 to 1023) of the analog voltage.

Syntax: analogRead(pin)

analogReference()

Configures the reference voltage used for analog input (i.e. the value used as the top of the input range). The options are:

DEFAULT: The default analog reference is 5V for Arduino UNO.

INTERNAL: an built-in reference equal to 1.1 Volts on the ATmega328

EXTERNAL: the voltage applied to the AREF pin (0 to 5V only) is used as the reference.

Task-1: Digital Voltmeter (Read analog voltage and show it in serial monitor)

For a A/D converter of 10-bit resolution with a conversion range of 0 to Ref (Ref $\leq$5V), we can write (Suppose, Ref=5V)

$$5 \text{ V is equivalent to } 1023\ (2^{10})$$

$$\text{So, Vo is equivalent to } \frac{1023}{5} \times Vo$$

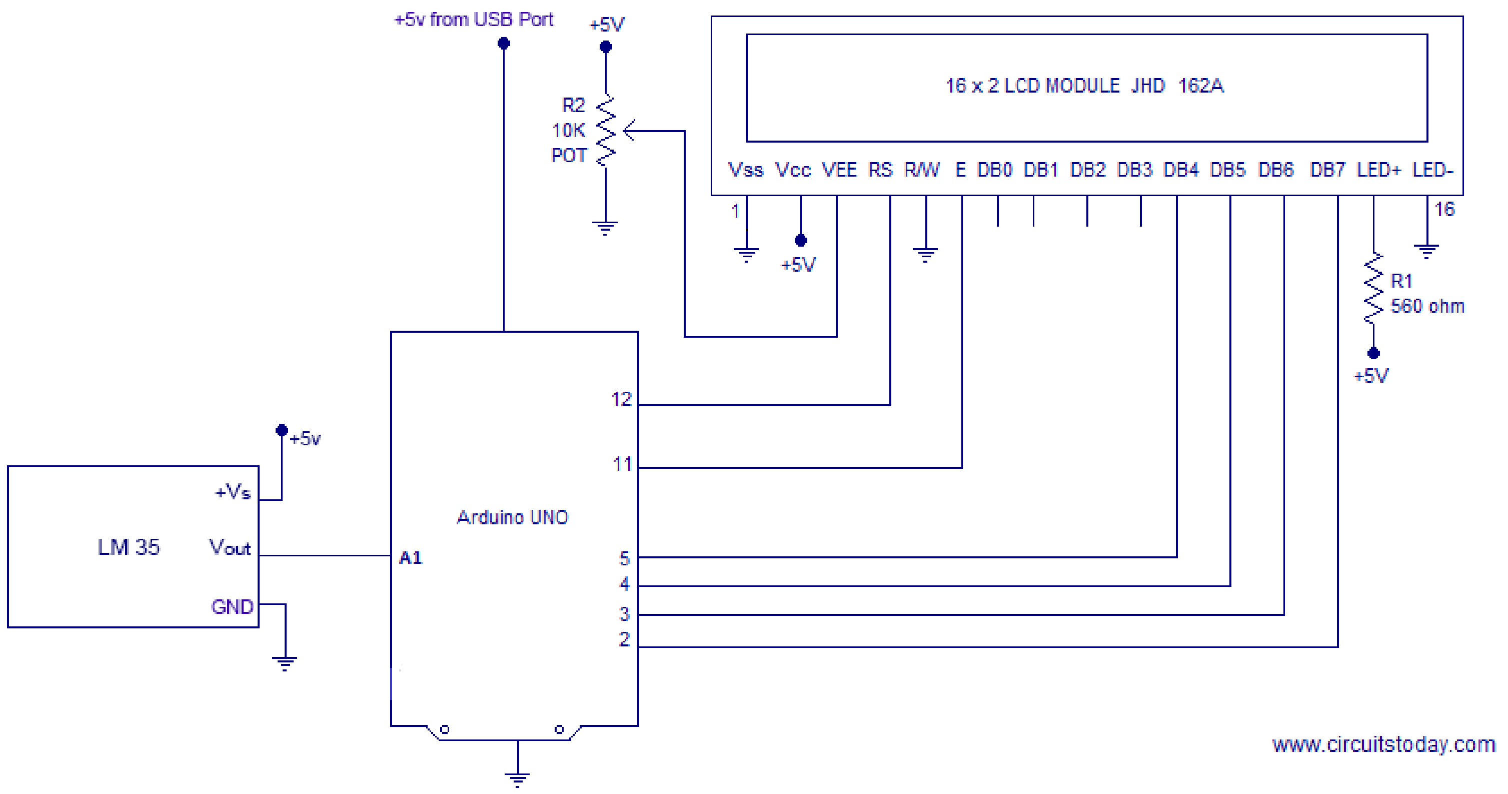
$$\text{ADC Value} = \frac{1023}{5} \times \text{Vo} \Rightarrow \text{Vo} = \frac{ADC\ Value}{1023} \times 5 \text{ Volt}$$

Program:

```
float ADC_value;
float voltage;
void setup() {
Serial.begin(9600);
}
void loop() {
ADC_value=analogRead(A0);
voltage=(ADC_value/1023)*5;
Serial.print("The measured Voltage is:");
Serial.print(voltage);
Serial.println(" Volt");
delay(1000); }
```

Task-2 :Temperature measurement by LM35

LM35 is an analog, linear temperature sensor whose output voltage varies linearly with change in temperature. LM35 is three terminal linear temperature sensor from National semiconductors. It can measure temperature from -55 degree celsius to +150 degree celsius. The voltage output of the LM35 increases 10mV per degree Celsius rise in temperature. LM35 can be operated from a 5V supply and the stand by current is less than 60uA.



Program ofTask-2

```
#include<LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
const int sensor=A1;
float tempc;
float tempf;
float vout;
void setup()
{
pinMode(sensor,INPUT);
Serial.begin(9600);
lcd.begin(16,2);
delay(500);
lcd.clear();
}
void loop()
{
vout=analogRead(sensor);
tempc=((vout/1023)*5000)/10;
tempf=(tempc*1.8)+32;
lcd.setCursor(0,0);
lcd.print("in DegreeC= ");
lcd.print(tempc);
lcd.setCursor(0,1);
lcd.print("in Fahrenheit=");
lcd.print(tempf);
delay(1000);
}
```