Micro-Processor Based System Design

The 8051 Microcontroller

Bilal Habib

DCSE, UET Peshawar

Slides are adapted from Chung-Ping Young lectures
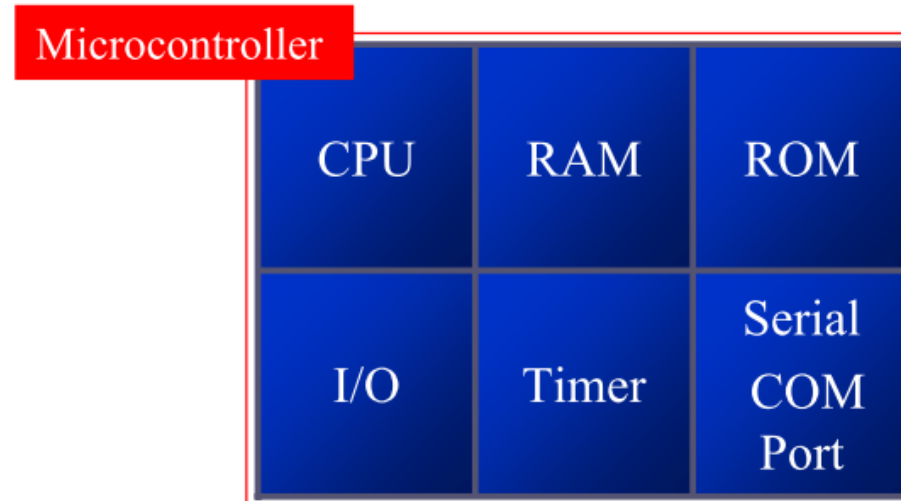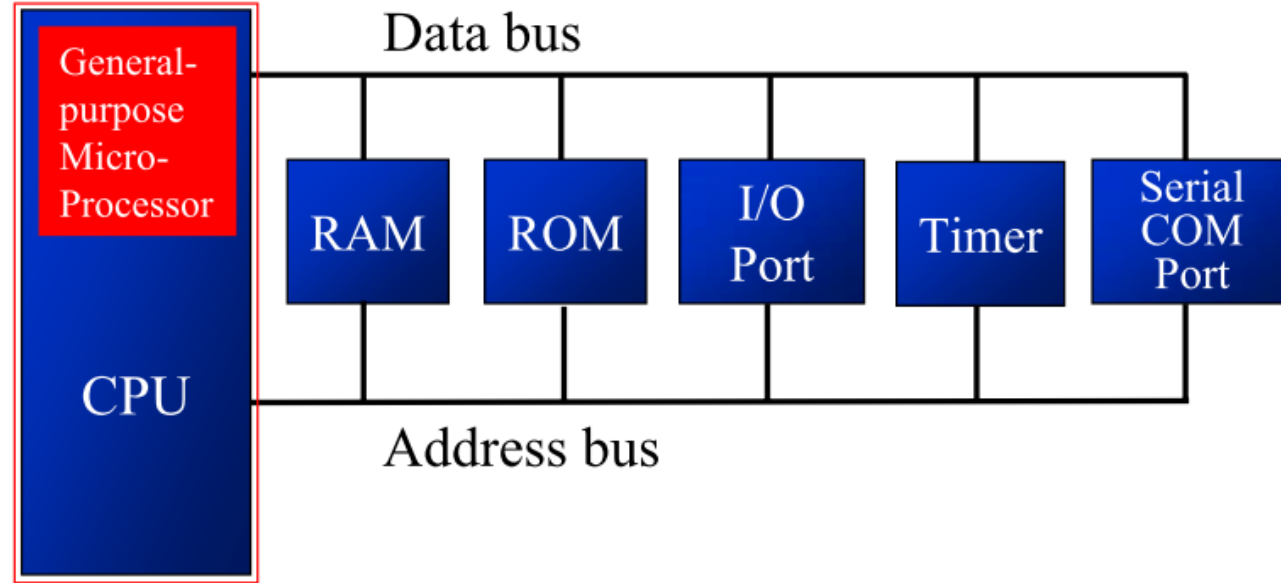
MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Microcontroller vs. General-Purpose Microprocessor

- General-purpose microprocessors contains
  - No RAM
  - No ROM
  - No I/O ports
- Microcontroller has
  - CPU (microprocessor)
  - RAM
  - ROM
  - I/O ports
  - Timer
  - ADC and other peripherals

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Microcontroller vs. General-Purpose Microprocessor (cont')

General-purpose Micro-Processor

CPU

Data bus

RAM | ROM | I/O Port | Timer | Serial COM Port

Address bus

Microcontroller

CPU	RAM	ROM
I/O	Timer	Serial COM Port

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Microcontroller vs. General-Purpose Microprocessor (cont')

❑ General-purpose microprocessors
- ➢ Must add RAM, ROM, I/O ports, and timers externally to make them functional
- ➢ Make the system bulkier and much more expensive
- ➢ Have the advantage of versatility on the amount of RAM, ROM, and I/O ports

❑ Microcontroller
- ➢ The fixed amount of on-chip ROM, RAM, and number of I/O ports makes them ideal for many applications in which cost and space are critical
- ➢ In many applications, the space it takes, the power it consumes, and the price per unit are much more critical considerations than the computing power

**MICRO-
CONTROLLERS
AND
EMBEDDED
PROCESSORS**

**Microcontrollers
for Embedded
Systems**

❑ An embedded product uses a microprocessor (or microcontroller) to do one task and one task only

  ➢ There is only one application software that is typically burned into ROM

❑ A PC, in contrast with the embedded system, can be used for any number of applications

  ➢ It has RAM memory and an operating system that loads a variety of applications into RAM and lets the CPU run them

  ➢ A PC contains or is connected to various embedded products

    ▪ Each one peripheral has a microcontroller inside it that performs only one task

5

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Choosing a Microcontroller

- ❑ 8-bit microcontrollers
  - ➢ Motorola's 6811
  - ➢ Intel's 8051
  - ➢ Zilog's Z8
  - ➢ Microchip's PIC
- ❑ There are also 16-bit and 32-bit microcontrollers made by various chip makers
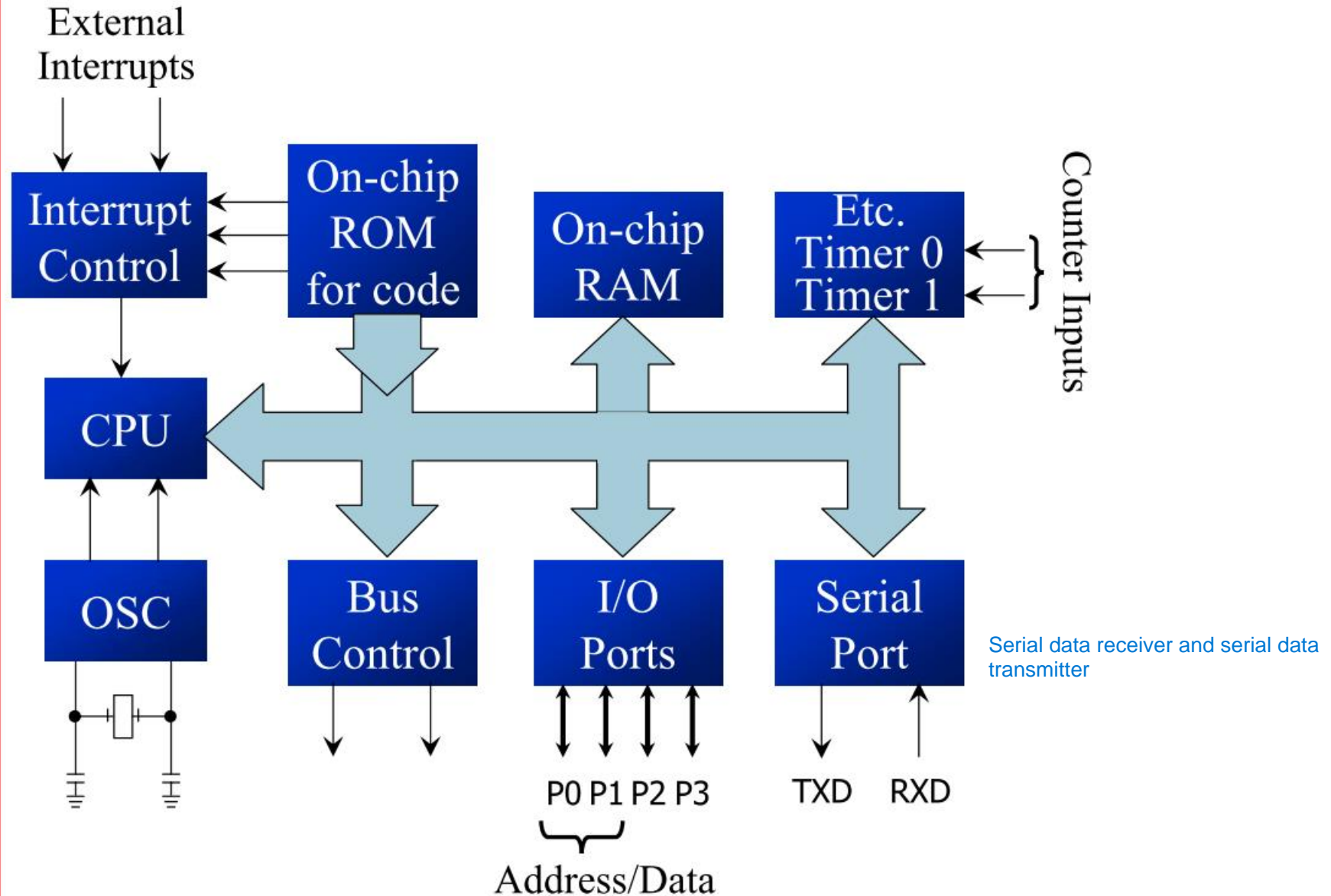
OVERVIEW OF 8051 FAMILY

8051 Microcontroller

- ❑ Intel introduced 8051, referred as MCS-51, in 1981
  - ➢ The 8051 is an 8-bit processor
    - The CPU can work on only 8 bits of data at a time
  - ➢ The 8051 had
    - 128 bytes of RAM
    - 4K bytes of on-chip ROM
    - Two timers   16 bits each
    - One serial port
    - Four I/O ports, each 8 bits wide
    - 6 interrupt sources
- ❑ The 8051 became widely popular after allowing other manufactures to make and market any flavor of the 8051, but remaining code-compatible

OVERVIEW OF 8051 FAMILY

8051 Microcontroller (cont')

External Interrupts

Interrupt Control

On-chip ROM for code

On-chip RAM

Etc. Timer 0 Timer 1

Counter Inputs

CPU

OSC

Bus Control

I/O Ports

Serial Port

Serial data receiver and serial data transmitter

P0 P1 P2 P3

Address/Data

TXD   RXD

OVERVIEW OF 8051 FAMILY

8051 Family

- ❏ The 8051 is a subset of the 8052
- ❏ The 8031 is a ROM-less 8051
  - ➢ Add external ROM to it
  - ➢ You lose two ports, and leave only 2 ports for I/O operations

Feature	8051	8052	8031
ROM (on-chip program space in bytes)	4K	8K	0K
RAM (bytes)	128	256	128
Timers	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

8051 Assembly Language Programming
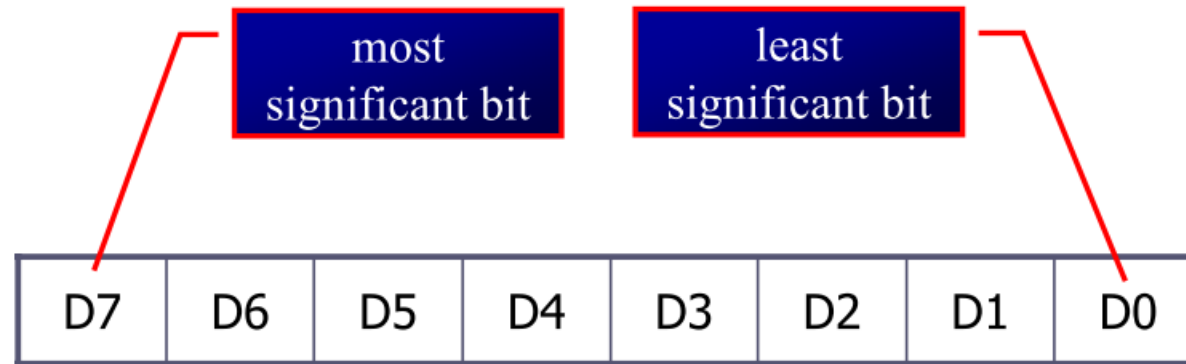
Bilal Habib

INSIDE THE 8051

Registers

❑ Register are used to store information temporarily, while the information could be

  ➤ a byte of data to be processed, or

  ➤ an address pointing to the data to be fetched

❑ The vast majority of 8051 register are 8-bit registers

  ➤ There is only one data type, 8 bits

❑ The 8 bits of a register are shown from MSB D7 to the LSB D0

  ➢ With an 8-bit data type, any data larger than 8 bits must be broken into 8-bit chunks before it is processed

most significant bit	least significant bit

D7	D6	D5	D4	D3	D2	D1	D0

8 bit Registers

❑ # The most widely used registers
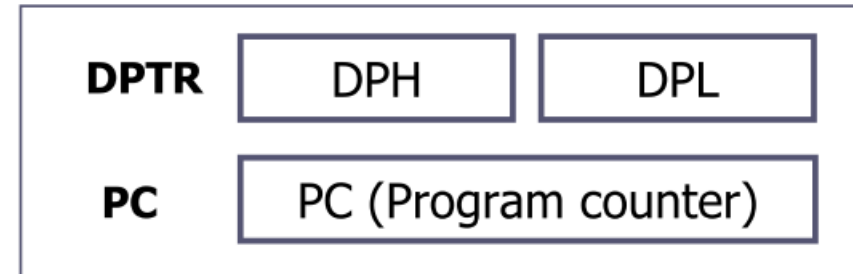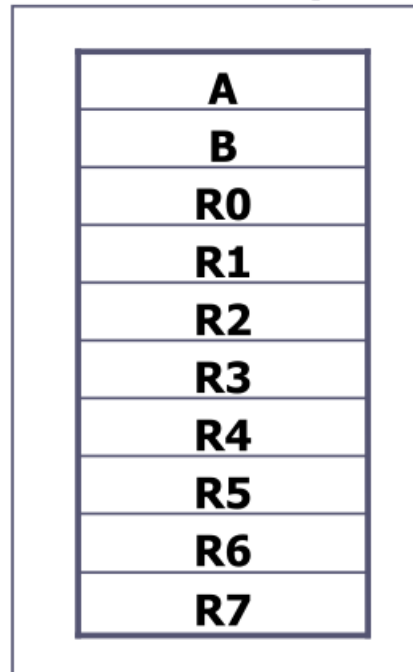
➢ A (Accumulator)

- For all arithmetic and logic instructions

➢ B, R0, R1, R2, R3, R4, R5, R6, R7   these are all 8 bits registers

➢ DPTR (data pointer), and PC (program   both are 16 bits external memory registers
counter)

A
B
R0
R1
R2
R3
R4
R5
R6
R7

	DPH	DPL
DPTR		
PC	PC (Program counter)	

INSIDE THE 8051

MOV Instruction

`MOV destination, source` ;copy source to dest.

➢ The instruction tells the CPU to move (in reality, COPY) the source operand to the destination operand

"#" signifies that it is a value

```
MOV   A,#55H      ;load value 55H into reg. A
MOV   R0,A        ;copy contents of A into R0
                  ; (now A=R0=55H)
MOV   R1,A        ;copy contents of A into R1
                  ; (now A=R0=R1=55H)
MOV   R2,A        ;copy contents of A into R2
                  ; (now A=R0=R1=R2=55H)
MOV   R3,#95H     ;load value 95H into R3
                  ; (now R3=95H)
MOV   A,R3        ;copy contents of R3 into A
                  ;now A=R3=95H
```

8051 ASSEMBLY PROGRAMMING

Structure of Assembly Language

- ❑ In the early days of the computer, programmers coded in *machine language*, consisting of 0s and 1s
  - ➢ Tedious, slow and prone to error
- ❑ *Assembly languages*, which provided mnemonics for the machine code instructions, plus other features, were developed
  - ➢ An Assembly language program consist of a series of lines of Assembly language instructions
- ❑ Assembly language is referred to as a *low-level language*
  - ➢ It deals directly with the internal structure of the CPU

8051
ASSEMBLY
PROGRAMMING

Structure of
Assembly
Language

❑ Assembly language instruction includes

➤ a mnemonic (abbreviation easy to remember)
- the commands to the CPU, telling it what those to do with those items

➤ optionally followed by one or two operands
- the data items being manipulated

❑ A given Assembly language program is a series of statements, or lines

➤ Assembly language instructions
- Tell the CPU what to do

➤ Directives (or pseudo-instructions)
- Give directions to the assembler

Structure of Assembly Language

❑ An Assembly language instruction consists of four fields:

```
[label:]  Mnemonic  [operands]  [;comment]
```

```
    ORG   0H                 ;start(origin) at location 0
    MOV   R5, #25H           ;load 25H into R5
    MOV   R7, #34H           ;load 34H in
    MOV   A, #0              ;load 0 int
    ADD   A, R5              ;add content
                            ;now A = A +
    ADD   A, R7              ;add contents of R7 to A
                            ;now A = A + R7
    ADD   A, #12H            ;add to A value 12H
                            ;now A = A + 12H
HERE: SJMP HERE             ;stay in this loop
    END                     ;en
```
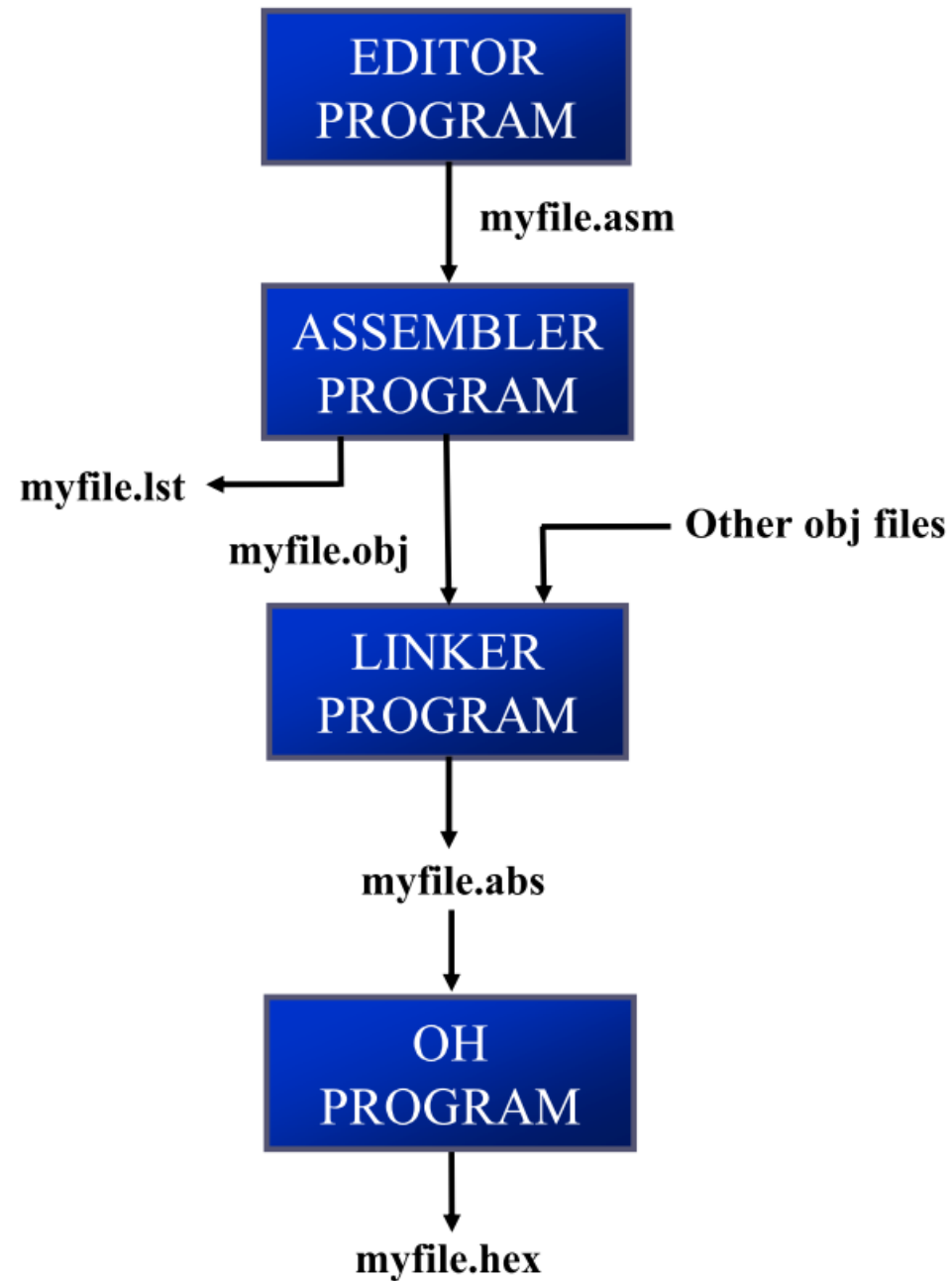
Mnemonics produce opcodes

Directives do not generate any machine code and are used only by the assembler

The label field allows the program to refer to a line of code by name

Comments may be at the end of a line or on a line by themselves
The assembler ignores comments

17

ASSEMBLING AND RUNNING AN 8051 PROGRAM

Steps to Create a Program

EDITOR PROGRAM
myfile.asm
ASSEMBLER PROGRAM
myfile.lst
myfile.obj
Other obj files
LINKER PROGRAM
myfile.abs
OH PROGRAM
myfile.hex

PROGRAM COUNTER AND ROM SPACE

Power up

- All 8051 members start at memory address 0000 when they're powered up
  - Program Counter has the value of 0000
  - The first opcode is burned into ROM address 0000H, since this is where the 8051 looks for the first instruction when it is booted
  - We achieve this by the `ORG` statement in the source program

❑ Examine the list file and how the code is placed in ROM

```
1 0000                  ORG 0H              ;start (origin) at 0
2 0000    7D25          MOV R5,#25H         ;load 25H into R5
3 0002    7F34          MOV R7,#34H         ;load 34H into R7
4 0004    7400          MOV A,#0            ;load 0 into A
5 0006    2D            ADD A,R5            ;add contents of R5 to A
                                            ;now A = A + R5
6 0007    2F            ADD A,R7            ;add contents of R7 to A
                                            ;now A = A + R7
7 0008    2412          ADD A,#12H          ;add to A value 12H
                                            ;now A = A + 12H
8 000A    80EF          HERE: SJMP HERE     ;stay in this loop
9 000C                  END                 ;end of asm source file
```

ROM Address	Machine Language	Assembly Language
0000	7D25	MOV R5, #25H
0002	7F34	MOV R7, #34H
0004	7400	MOV A, #0
0006	2D	ADD A, R5
0007	2F	ADD A, R7
0008	2412	ADD A, #12H
000A	80EF	HERE: SJMP HERE

```
$NOMOD51
$INCLUDE (8051.MCU)

        ; Reset Vector

        org     0000h
        jmp     Start


        org     0100h
Start:
        ; Write your code here
    CLR P2.0
AGAIN :
    SETB P2.0
    ACALL DELAY
    CLR P2.0
    ACALL DELAY
    SJMP AGAIN



DELAY : MOV R0,#0FFH
GO : MOV R1,#0F9H
HERE : DJNZ R1,HERE
    DJNZ R0,GO
    RET


Loop:
        jmp Loop

    END
```
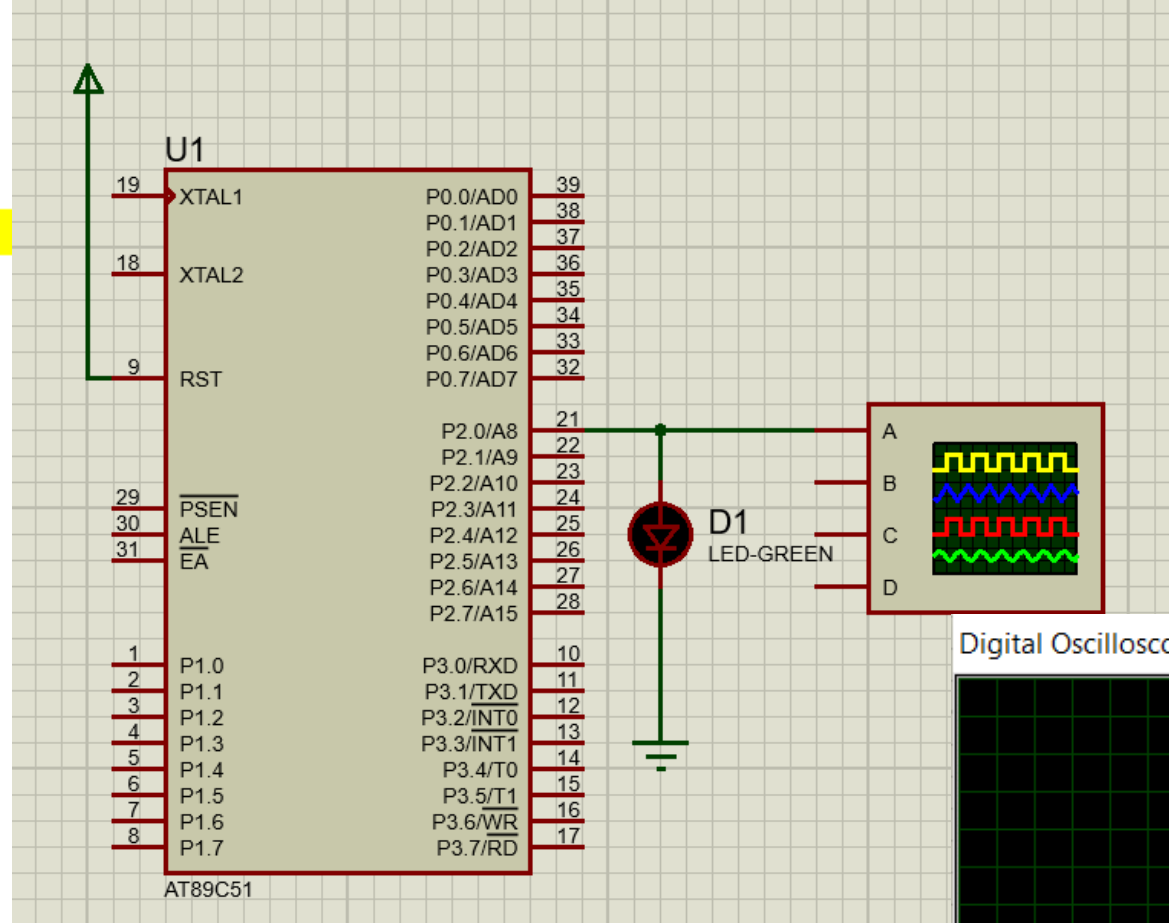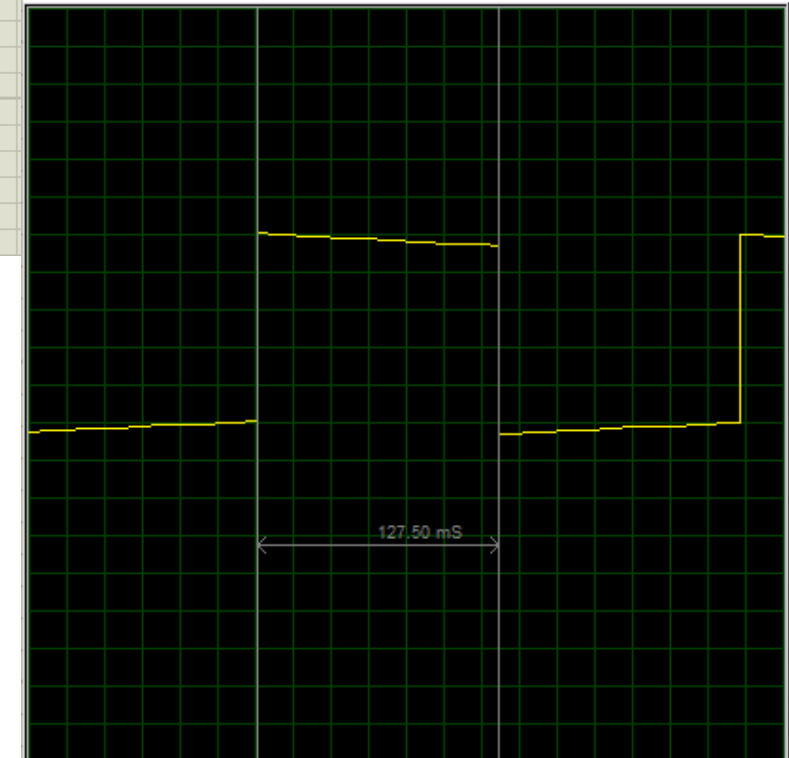

Digital Oscilloscope

127.50 mS

To do:
- Read Chapter **0,1 and 2** from Mazidi book
- **Task1**: Complete it before the deadline