MICROCONTROLLERS EEE403

Module 2

Module – 2: Assembly programming And Instructions of 8051

Syllabus:

± Introduction to 8051 assembly programming,

± Assembling and running an 8051 program

± Data types and Assembler directives

± Arithmetic, logic instructions and programs

± Jump, loop and call instructions,

± IO port programming.

SESSION 1

Introduction to 8051 Assembly Programming

Machine language:

- ❑ The CPU can understand and work only on binary digits.
- ❑ In the early days of the computer, programmers coded programs in ***machine language*** which consists of 0s & 1s.
- ❑ Although the hexadecimal system was used as a more efficient way to represent binary numbers, the process of working in machine code was still cumbersome for humans.

Average of two numbers		
Copy value 26h into R7 register	0111 1111, 0010 0110	7Fh, 26h
Add contents of accumlator & R7	0010 1111	2Fh
Copy value 02h into B register	0111 0101, 1111 0000 0000 0010	75h, F0h, 02h
Divide A by B	1000 0100	84h

Introduction to 8051 Assembly Programming

Assembly language (Low-level language):

- Complexity in machine language programming, eventually, led to the development of ***Assembly Language*** that provided **mnemonics** for the machine code instructions, plus other features that made **programming faster and less prone to error**.
- The term ***mnemonic*** refer to codes and abbreviations that are relatively easy to remember.
- Assembly language programs must be translated into machine code by a program called an ***assembler***.
- Assembly language is referred to as a **low-level language** because it deals directly with the internal structure of the CPU.
- To program in Assembly language, the programmer must know all the registers of the CPU and the size of each, as well as other details.

Average of two numbers		
Copy value 26h into R7 register	**MOV R7, #26h** (0111 1111, 0010 0110)	7Fh, 26h
Add contents of accumlator & R7	**ADD A, R7** (0010 1111)	2Fh
Copy value 02h into B register	**MOV B, #02h** (0111 0101, 1111 0000 0000 0010)	75h, F0h, 02h
Divide A by B	**DIV AB** (1000 0100)	84h

Introduction to 8051 assembly programming

High-Level language:

- In high-level language programming the programmer does not have to be concerned with the internal details of the CPU.
- High-level language programs are translated into machine code by a program called a **compiler.**
- Ex.: BASIC, Pascal, C, C++, Java, etc.

Ex:

x = (x+y)/2;

Structure of Assembly language

[label:] mnemonic [Operands] [;comment]

- Brackets indicate that a field is optional, and not all lines have them. Brackets should not be typed in. Regarding the above format, the following points should be noted. .
- The **label field** allows the program to refer to a line of code by name. The label field cannot exceed a certain number of characters. Check the assembler for the rule.
- The Assembly language **mnemonic (instruction)** and **operand(s) fields** together perform the real work of the program and accomplish the tasks for which the program was written.
- The **comment field** begins with a semicolon comment indicator ";". Comments may be at the end of a line or on a line by themselves. The assembler ignores comments, but they are indispensable to programmers. Although comments are optional, it is recommended that they be used to describe the program and make it easier for someone else to read and understand, or for the programmer to remember what they wrote.
- Any label referring to an instruction must be followed by a **colon symbol, ":".**

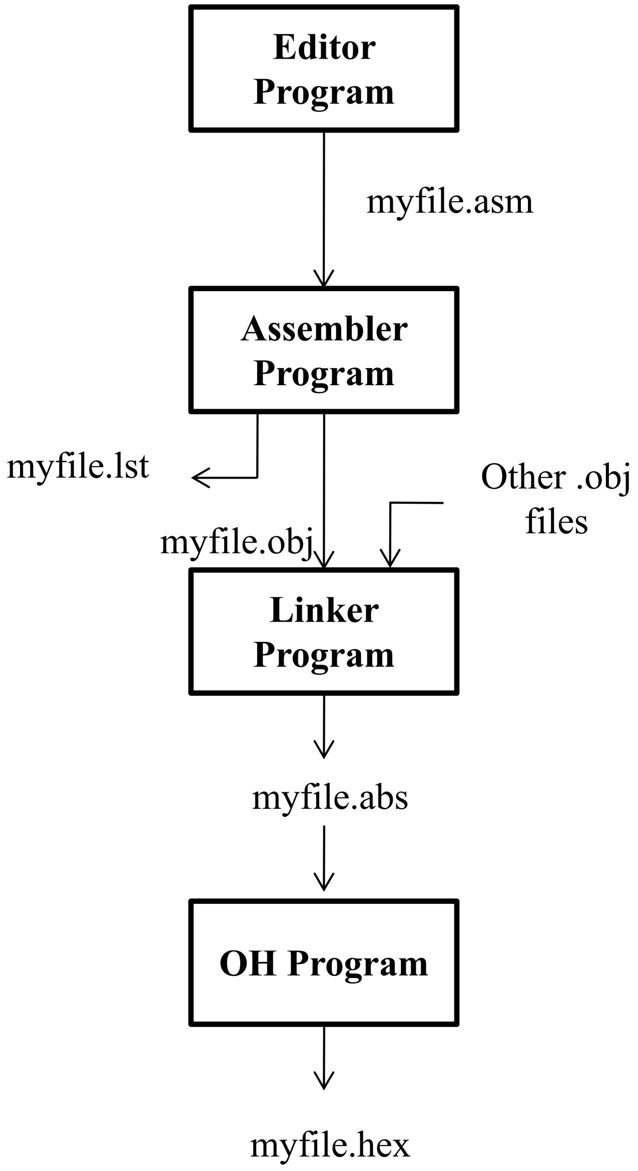
```
        ORG 0H            ; start (origin) at location 0
        MOV R3, #05h      ; load 05h into R3
Again:  MOV R5, #25h      ; load 25H into R5
        MOV R7, #34h      ; load 34H into R7
        MOV A, #0         ; load 0 into A
        ADD A, R5         ; add contents of R5 to A, now A = A + R5
        ADD A, R7         ; add contents of R7 to A, now A = A + R7
        ADD A, # 12       ; add to A value 12H, now A = A + 12H
        DJNZ R3, Again    ; Repeat 5 times
 Here:  SJMP here         ; stay in this loop
        END               ; end of asm source file
```

SESSION 2

Rules for labels in Assembly language:

- ❑ Each label name must be unique.
- ❑ The names used for labels in Assembly language programming consist of alphabetic letters in both uppercase and lowercase, the digits 0 through 9, and the special characters question mark (?), period (.), at (@), underline (_), and dollar sign ($).
- ❑ The first character of the label must be an alphabetic character, it cannot be a number.
- ❑ Every assembler has some reserved words that must not be used as labels in the program.
- ❑ Foremost among the reserved words are the mnemonics for the instructions.

Assembling and Running an 8051 program



1.	0000			ORG 0H	; star (origin) at 0
2.	0000	7D 25		MOV R5, #25H	; load 25 into R5
3.	0002	7F 34		MOV R7, #34H	;Load 34H into R7
4.	0004	74 00		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 R57to A; now A = A + R7
7.	0008	24 12		ADD A, #12H	; add to A value 12H ; now A = A+12H
8.	000A	80 0A	HERE:	SJMP HERE	; stay in this loop
9.	000C			END	; end of asm source file

8051 Data types and Directives

DB (define byte):

```
        ORG   500H
DATA1:  DB    28                        ; DECIMAL (1C in hex)
DATA2:  DB    00110101B                 ; BINARY (35 in hex)
DATA3:  DB    39H                       ; HEX
        ORG   510H
DATA4:  DB    “2591”                    ; ASCII NUMBERS
        ORG   518H
DATA6:  DB    “I LOVE MY CULTURE”       ; ASCII CHARACTERS
```

Assembler directives

ORG (origin):

- ❑ The ORG directive is used to indicate the beginning of the address.
- ❑ The number that comes after ORG can be either in hex or in decimal. If the number is not followed by H, it is decimal and the assembler will convert it to hex.
- ❑ Based on the assembler ". ORG" or "ORG" notation is used.

  Ex: ORG 500h

EQU (equate):

- ❑ This is used to define a constant without occupying a memory location.
- ❑ The EQU directive does not set aside storage for a data item but associates a constant value with a data label so that when the label appears in the program.
- ❑ Its constant value will be substituted for the label.

```
Ex: COUNT EQU 25
. . .        . . . .
MOV    R3, #COUNT
```

Advantage of using EQU?

END:

- ❑ This indicates to the assembler the end of the source (asm) file.
- ❑ The END directive is the last line of an 8051 program, meaning that in the source code anything after the END directive is ignored by the assembler.
- ❑ Some assemblers use “. END” instead of “END”.

SESSION 3

Instruction set of 8051:

- Data Transfer Instructions
- Arithmetic instructions
- Logical instructions
- Bit manipulation instructions
- Byte manipulation instructions
- Program flow instructions

Data Transfer Instructions

Data transfer instructions move (copy) one source operand to another destination operand.

Destination operand	**Source operand**
Accumulator	Immediate (8-bit/16-bit)
Register (Rn)	Accumulator
Direct	Register (Rn)
Indirect (@Ri or @DPTR)	Direct
DPTR	Indirect (@Ri or @DPTR)
Output ports	Input ports

- The data transfers are not possible between
    - Register (Rn) & Register (Rn)
    - Indirect (@Ri) & Indirect (@Ri)
- Both the operands must be of same data type.
- All exchange instructions use Accumulator as one of the operand.
- One of the operand should be Accumulator for I/O transfers.
- No flags are affected on the execution of the data transfer instructions.

Sl. No.	Instruction	Operation	Sl. No.	Instruction	Operation
1	MOV A, Rn	A ← Rn	16	MOV DPTR, #imm16	DPTR ← imm16
2	MOV A, data_addr	A ← [data_addr]	17	XCH A, Rn	A ← Rn Rn ← A
3	MOV A, @Ri	A ← [Ri]	18	XCH A, data_addr	A ← [data_addr] [data_addr] ← A
4	MOV A, #imm8	A ← imm8	19	XCH A, @Ri	A ← [Ri] [Ri] ← A
5	MOV Rn, A	Rn ← A	20	MOVX A, @Ri	A ← [Ri]
6	MOV Rn, data_addr	Rn ← [data_addr]	21	MOVX @Ri, A	[Ri] ← A
7	MOV Rn, #imm8	Rn ← imm8	22	MOVX A, @DPTR	A ← [DPTR]
8	MOV data_addr, A	[data_addr] ← A	23	MOVX @DPTR, A	[DPTR] ← A
9	MOV data_addr, Rn	[data_addr] ← Rn	24	MOVC A, @A+PC	A ← [A+PC]
10	MOV data_addr, @Ri	[data_addr] ← @Ri	25	MOVC A, @A+DPTR	A ← [A+DPTR]
11	MOV data_addr, #imm8	[data_addr] ← imm8	26	PUSH data_addr	SP ← SP+1 [SP] ← [data_addr]
12	MOV data_addr, data_addr	[data_addr] ← [data_addr]	27	POP data_addr	[data_addr] ← [SP] SP ←SP-1
13	MOV @Ri, A	[Ri] ← A	28	MOV port, A	Port ← A
14	MOV @Ri, #data	[Ri] ← imm8	29	MOV A, Port	A ← Port
15	MOV @Ri, data_addr	[Ri] ← [data_addr]			

SESSION 4

Arithmetic Instructions

- The result of the most of the arithmetic operations is stored in the accumulator except a few decrement and increment operations.
- The arithmetic instructions except increment and decrement modify the flags.

Sl. No.	Instruction	Operation	Source operands	Flags affected
1	ADD A, Src	A ← A + Src	Imm8, Rn, data_addr, @Ri	All status flags
2	ADDC A, Src	A ← A+ Src + CF	Imm8, Rn, data_addr, @Ri	All status flags
3	SUBB A, Src	A ← A – Src – CF	Imm8, Rn, data_addr, @Ri	All status flags
4	INC dest	dest ← dest + 1	A, Rn, data_addr, @Ri, DPTR	No flags
5	DEC dest	dest ← dest – 1	A, Rn, data_addr, @Ri, DPTR	No flags
6	MUL AB	B:A ← A * B	No other operands used	OF = 1 when A*B > FFh

7	DIV AB	A ← A/B (Q)
		B ← A%B (R)
			No other operands used	OF = 1 on division by 0

8	DA A	i) If (A[LN]) >9 or AF=1; A ←A+06h ii) If (A[HN]) >9 or CF=1; A ←A+60h	No other operands used	No flags

Logical Instructions

- The logical instructions do not modify the flags
- Results are stored in accumulator in most of the cases, except in few cases where it is stored in RAM

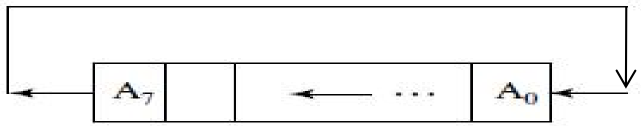
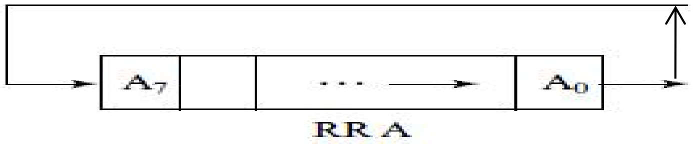
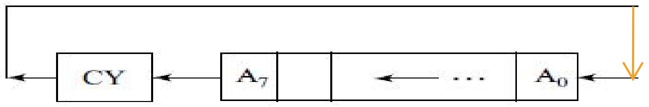
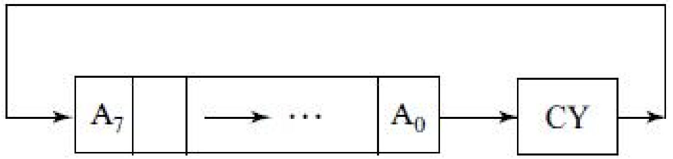
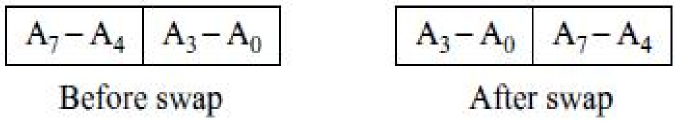
Sl. No.	Instruction	Operation	Source operands
1	ANL A, Src	A ← A & Src	Imm8, data_addr, @Ri
2	ANL data_addr, A	[data_addr] ← [data_addr] & A	No other operands used
3	ANL data_addr, #imm8	[data_addrt] ← [data_addr] & imm8	No other operands used
4	ORL A, Src	A ← A ¦ Src	Imm8, Rn, data_addr, @Ri
5	ORL data_addr, A	[data_addr] ← [data_addr] ¦ A	No other operands used
6	ORL data_addr, #imm8	[data_addr] ← [data_addr] ¦ imm8	No other operands used
7	XRL A, Src	A ← A^ Src	Imm8, Rn, data_addr, @Ri
8	XRL data_addr, A	[data_addr] ← [data_addr] ^ A	No other operands used
9	XRL data_addr, #imm8	[data_addr] ← [data_addr] ^ imm8	No other operands used

Bit Manipulation Instructions

Each bit of certain SFRs and an 16-byte internal RAM (20h – 2Fh) are assigned bit addresses in 8051 hardware.

Sl. No.	Instruction	Operation	Sl. No.	Instruction	Operation
1	CLR C	CF ← 0	7	ANL C, bit_addr	CF ← CF & [bit_addr]
2	CLR bit_addr	[bit_addr] ← 0	8	ANL C, /bit_addr	CF ← CF & [~bit_addr]
3	SETB C	CF ← 1	9	ORL C, bit_addr	CF ← CF ! [bit_addr]
4	SETB bit_addr	[bit_addr] ← 1	10	ORL C, /bit_addr	CF ← CF ! [~bit_addr]
5	CPL C	CF ← ~CF	11	MOV C, bit_addr	CF ← [bit_addr]
6	CPL bit_addr	[bit_addr] ← [~bit_addr]	12	MOV bit_addr, C	[bit_addr] ← CF

Byte Manipulation Instructions

Sl. No.	Instruction	Operation	Flags affected
1	CLR A	A ← 00	No flags
2	CPL A	A ← ~A	No flags
3	RL A		No flags
4	RR A		No flags
5	RLC A		CF
6	RRC A		CF
7	SWAP A	 $A_7 - A_4$ | $A_3 - A_0$ Before swap $A_3 - A_0$ | $A_7 - A_4$ After swap	No flags

SESSION 5

Program Flow Control Instructions

- Normal sequential execution may be altered by using branching operations like JUMP & CALL.
- Branching instructions are classified in to ***unconditional branching instructions*** & ***conditional branching instructions***.

Unconditional branching instructions

Sl. No.	Instruction	Operation	Remarks

1	SJMP code_offset	PC← PC + 2
		PC←PC + code_offset
			Short jump
			8-bit signed offset is used

2	AJMP code_addr11	PC ← PC + 2 PC(10-0) ← code_addr11	Absolute jump 11-bit address limiting to within 2KB page
3	LJMP code_addr16	PC ← Code_addr16	Long jump, Entire 64 KB memory
4	JMP @A + DPTR	PC ← code_addr (A + DPTR)	Long jump, Entire 64 KB memory

Sl. No.	Instruction	Operation	Remarks
5	ACALL code_addr11	i) PC ←PC + 2 ii) SP ← SP + 1 iii) [SP] ← PCL iv) SP ←SP + 1 v) [SP] ←PCH vi) PC(10-0) ← code_addr11	Absolute jump 11-bit address limiting to within 2KB page
6	LCALL code_addr16	i) PC ←PC + 3 ii) SP ← SP + 1 iii) [SP] ← PCL iv) SP ←SP + 1 v) [SP] ←PCH vi) PC ← code_addr16	Long jump, Entire 64 KB memory
7	RET	i) PCH ←[SP] ii) SP ←SP – 1 iii) PCL ←[SP] iv) SP ←SP – 1	

Conditional branching instructions

In ***conditional branching instructions***, the content of the PC is modified, only if the condition specified in the instruction is true.

- ❑ Only relative addressing mode (Short Jumps) is used.
- ❑ The conditional branching instructions are normally used immediately after the arithmetic/logical operations so that branching can occur based on a condition specified.

Sl. No.	Instruction	Operation	Remarks
1	JZ code_offset	i) PC← PC + 2 ii) If (A = 0) PC←PC + code_offset Else, Next instruction is executed	Jump if result in A is zero
2	JNZ code_offset	i) PC← PC + 2 ii) If (A ≠ 0) PC←PC + code_offset Else, Next instruction is executed	Jump if result in A is not zero

Sl. No.	Instruction	Operation	Remarks
3	JC code_offset	i) PC← PC + 2 ii) If (CF = 0) PC←PC + offset_addr Else Next instruction is executed	Jump if carry flag
4	JNC code_offset	i) PC← PC + 2 ii) If (CF ≠ 0) PC←PC + offset_addr Else Next instruction is executed	Jump if no carry flag
5	JB bit_addr, code_offset	i) PC← PC + 3 ii) If ([bit] = 1) PC←PC + offset_addr Else Next instruction is executed	Jump if bit set
6	JNB bit_addr, code_offset	i) PC← PC + 3 ii) If ([bit] ≠1) {i.e. [bit]=0} PC←PC + offset_addr Else Next instruction is executed	Jump if no bit

Sl. No.	Instruction	Operation	Remarks
7	JBC bit_addr, code_offset	i) PC← PC + 3 ii) If ([bit] =1) PC←PC + offset [bit]=0 Else Next instruction is executed	Jump if bit set & clear it
8	CJNE A, #imm8, code_offset	i) PC← PC + 3 ii) If (A ≠ [data]) PC←PC + offset Also, if A< [data], CF = 1 A > [data], CF = 0 Else Next instruction is executed	Compare & jump if not equal to Other forms, ii) CJNE A, data_addr, code_offset iii) CJNE Rn, #imm8, code_offset iv) CJNE @Ri, #imm8, code_offset
9	DJNZ Rn, code_offset	i) PC← PC + 2 ii) Rn ← Rn – 1 iii) If (Rn ≠ 0) PC←PC + offset Else Next instruction is executed	Decrement & jump if not zero ii) DJNZ data_addr, code_offset

SESSION 6

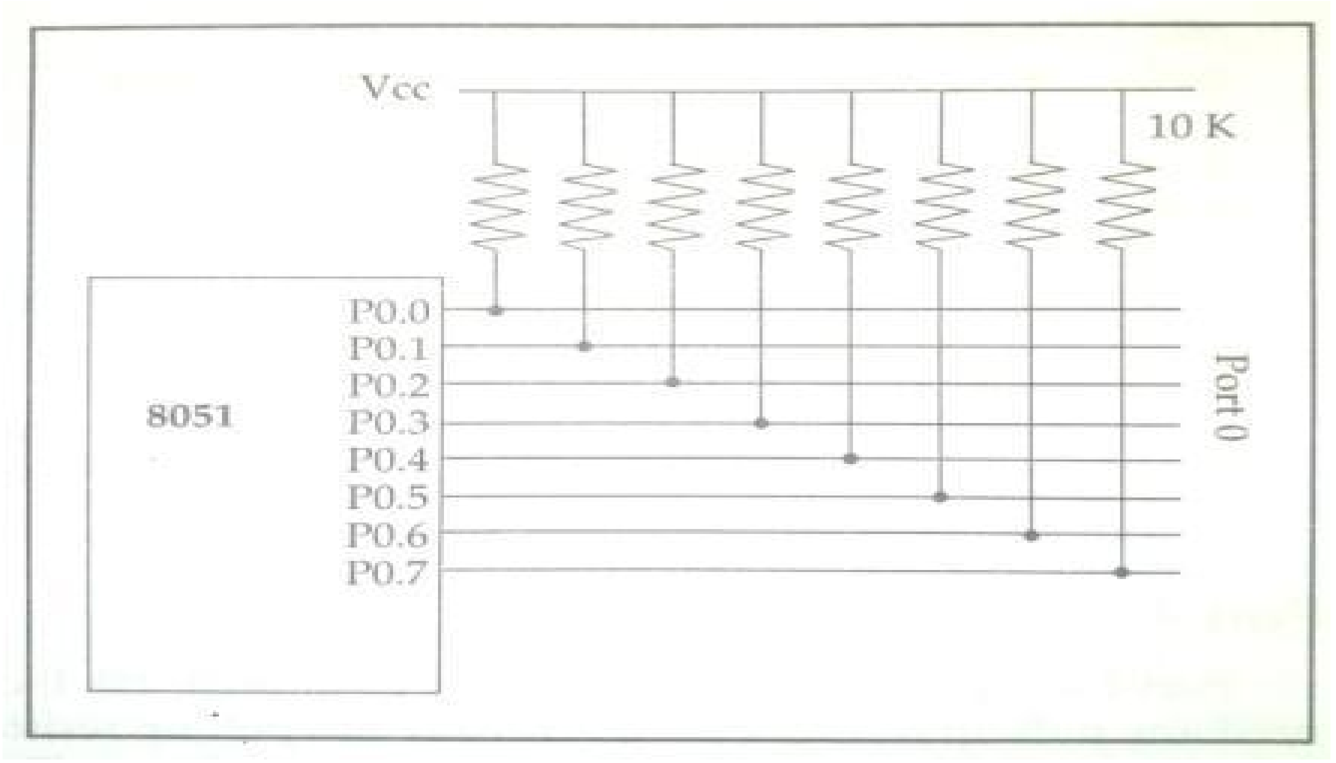
I/O port programming:

I/O port pins and their functions:

- The four ports PO, PI, P2, and P3 each use 8 pins, making them 8-bit ports.
- To use any of these ports as an input port, it must be programmed.
- All the ports upon RESET are configured as inputs, ready to be used as input ports.
- When the first 0 is written to a port, it becomes an output.
- To reconfigure as an input, a 1 must be sent to the port.

Port 0:

- Port 0 occupies a total of 8 pins (pins 32 - 39). It can be used for input or output.
- each pin must be connected externally to a 10K-ohm pull-up resistor to use port 0 as both input & output. This is due to the fact that P0 is an open drain, unlike P1, P2, and P3.
- **Ports 0 as input:**
  With resistors connected to port 0, in order to make it an input, the port must be programmed by writing 1 to all the bits.
- **Dual role of port 0:**
  Port 0 is also designated as AD0 - AD7, allowing it to be used for both address and data.



Port 1

- Port 1 occupies a total of 8 pins (pins 1 to 8). It can be used as input or output.
- In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally.
- Upon reset, port 1 is configured as an output port.
- If port 1 has been configured as an output port, to make it an input port again, it must be programmed as such by writing 1 to all its bits.

Port 2

- Port 2 occupies a total of 8 pins (pins 21 to 28). It can be used as input or output.
- Just like P1, port 2 does not need any pull-up resistors since it already has pull-up resistors internally.
- On reset, port 2 is configured as an output port.
- To make port 2 an input, it must programmed as such by writing 1 to all its bits.
- Port 2 is also designated as A8 - A15, indicating its dual function. When the 8051/31 is connected to external memory, P2 is used for the upper 8 bits of the 16-bit address, and it cannot be used for I/O.

Port 3

- ❑ Port 3 occupies a total of 8 pins, pins 10 through 17. It can be used as input or output.
- ❑ P3 does not need any pull- up resistors, just as P1 and P2 did not.
- ❑ Port 3 has the additional function of providing some extremely important signals such as interrupts.

Port 3 Alternate Function

P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	/INT0	12
P3.3	/INT1	13
P3.4	T0	14
P3.5	T1	15
P3.6	/WR	16
P3.7	/RD	17

Test program to toggle all the bits of P0, P1, and P2 every 1/4 of a second. Assume a crystal frequency of 11.0592 MHz

```
                ORG       0
        BACK:   MOV       A, #55H
                MOV       P0, A
                MOV       P1, A
                MOV       P2, A
                ACALL     QSDELAY                ;Quarter of a second delay
                MOV       A, #0AAH
                MOV       P0, A
                MOV       P1, A
                MOV       P2, A
                ACALL     QSDELAY
                SJMP      BACK
; -----------------------1/4 SECOND DELAY
    QSDELAY:
                MOV       R5, #11
          H3:   MOV       R4, #248
          H2:   MOV       R3, #255
          H1:   DJNZ      R3, H1
                DJNZ      R4, H2
                DJNZ      R5, H3
                RET
                END
```

Delay = 11x248x255x4MCx90ns = 250,430μs

SESSION 7

8051 Instruction Set:

Hex codes, no. of bytes & Machine cycles

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
1	NOP		00	1	1
2	AJMP	Addr11	01	2	2
3	LJMP	Addr16	02	3	2
4	RR	A	03	1	1
5	INC	A	04	1	1
6	INC	Data_ addr	05	2	1
7	INC	@R0	06	1	1
8	INC	@R1	07	1	1
9	INC	R0	08	1	1
10	INC	R1	09	1	1
11	INC	R2	0A	1	1
12	INC	R3	0B	1	1
13	INC	R4	0C	1	1
14	INC	R5	0D	1	1
15	INC	R6	0E	1	1
16	INC	R7	0F	1	1
17	JBC	Bit_addr, code_addr	10	3	2
18	ACALL	Code_ addr	11	2	2
19	LCALL	Code_addr	12	3	2
20	RRC	A	13	1	1

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
21	DEC	A	14	1	1
22	DEC	Data_ addr	15	2	1
23	DEC	@R0	16	1	1
24	DEC	@R1	17	1	1
25	DEC	R0	18	1	1
26	DEC	R1	19	1	1
27	DEC	R2	1A	1	1
28	DEC	R3	1B	1	1
29	DEC	R4	1C	1	1
30	DEC	R5	1D	1	1
31	DEC	R6	1E	1	1
32	DEC	R7	1F	1	1
33	JB	Bit_addr, Code_addr	20	3	2
34	AJMP	Code_addr	21	2	2
35	RET		22	1	2
36	RL	A	23	1	1
37	ADD	A, #data	24	2	1
38	ADD	A, data_addr	25	2	1
39	ADD	A, @R0	26	1	1
40	ADD	A, @R1	27	1	1

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
41	ADD	A, R0	28	1	1
42	ADD	A, R1	29	1	1
43	ADD	A, R2	2A	1	1
44	ADD	A, R3	2B	1	1
45	ADD	A, R4	2C	1	1
46	ADD	A, R5	2D	1	1
47	ADD	A, R6	2E	1	1
48	ADD	A, R7	2F	1	1
49	JNB	Bit_addr, code_addr	30	3	2
50	ACALL	Code_addr	31	2	2
51	RETI		32	1	2
52	RLC	A	33	1	1
53	ADDC	A, #data	34	2	1
54	ADDC	A, data_addr	35	2	1
55	ADDC	A, @R0	36	1	1
56	ADDC	A, @R1	37	1	1
57	ADDC	A, R0	38	1	1
58	ADDC	A, R1	39	1	1
59	ADDC	A, R2	3A	1	1
60	ADDC	A, R3	3B	1	1

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
61	ADDC	A, R4	3C	1	1
62	ADDC	A, R5	3D	1	1
63	ADDC	A, R6	3E	1	1
64	ADDC	A, R7	3F	1	1
65	JC	Code_addr	40	2	2
66	AJMP	Code_addr	41	2	2
67	ORL	Data_addr, A	42	2	1
68	ORL	Data_addr, #data	43	3	2
69	ORL	A, #data	44	2	1
70	ORL	A, data_addr	45	2	1
71	ORL	A, @R0	46	1	1
72	ORL	A, @R1	47	1	1
73	ORL	A, R0	48	1	1
74	ORL	A, R1	49	1	1
75	ORL	A, R2	4A	1	1
76	ORL	A, R3	4B	1	1
77	ORL	A, R4	4C	1	1
78	ORL	A, R5	4D	1	1
79	ORL	A, R6	4E	1	1
80	ORL	A, R7	4F	1	1

SESSION 8

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
81	JNC	Code_addr	50	2	2
82	ACALL	Code_addr	51	2	2
83	ANL	Data_addr, A	52	2	1
84	ANL	Data_addr, #data	53	3	2
85	ANL	A, #data	54	2	1
86	ANL	A, data_addr	55	2	1
87	ANL	A, @R0	56	1	1
88	ANL	A, @R1	57	1	1
89	ANL	A, R0	58	1	1
90	ANL	A, R1	59	1	1
91	ANL	A, R2	5A	1	1
92	ANL	A, R3	5B	1	1
93	ANL	A, R4	5C	1	1
94	ANL	A, R5	5D	1	1
95	ANL	A, R6	5E	1	1
96	ANL	A, R7	5F	1	1
97	JZ	Code_addr	60	2	2
98	AJMP	Code_addr	61	2	2
99	XRL	Data_addr, A	62	2	1
100	XRL	Data_addr, #data	63	3	2

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
101	XRL	A, #data	64	2	1
102	XRL	A, data_addr	65	2	1
103	XRL	A, @R0	66	1	1
104	XRL	A, @R1	67	1	1
105	XRL	A, R0	68	1	1
106	XRL	A, R1	69	1	1
107	XRL	A, R2	6A	1	1
108	XRL	A, R3	6B	1	1
109	XRL	A, R4	6C	1	1
110	XRL	A, R5	6D	1	1
111	XRL	A, R6	6E	1	1
112	XRL	A, R7	6F	1	1
113	JNZ	Code_addr	70	2	2
114	ACALL	Code_addr	71	2	2
115	ORL	C, bit_addr	72	2	2
116	JMP	@A+DPTR	73	1	2
117	MOV	A, #data	74	2	1
118	MOV	Data_addr, #data	75	3	2
119	MOV	@R0, #data	76	2	1
120	MOV	@R1, #data	77	2	1

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
121	MOV	R0, #data	78	2	1
122	MOV	R1, #data	79	2	1
123	MOV	R2, #data	7A	2	1
124	MOV	R3, #data	7B	2	1
125	MOV	R4, #data	7C	2	1
126	MOV	R5, #data	7D	2	1
127	MOV	R6, #data	7E	2	1
128	MOV	R7, #data	7F	2	1
129	SJMP	Code_addr	80	2	2
130	AJMP	Code_addr	81	2	2
131	ANL	C, bit_addr	82	2	2
132	MOVC	A, @A+PC	83	1	2
133	DIV	AB	84	1	4
134	MOV	Data_addr, data_addr	85	3	2
135	MOV	Data_addr, @R0	86	2	2
136	MOV	Data_addr, @R1	87	2	2
137	MOV	Data_addr, R0	88	2	2
138	MOV	Data_addr, R1	89	2	2
139	MOV	Data_addr, R2	8A	2	2
140	MOV	Data_addr, R3	8B	2	2

SESSION 9

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
141	MOV	data addr, R4	8C	2	2
142	MOV	Data_addr, R5	8D	2	2
143	MOV	Data_addr, R6	8E	2	2
144	MOV	Data_addr, R7	8F	2	2
145	MOV	DPTR, #data	90	3	2
146	ACALL	Code_ addr	91	2	2
147	MOV	Bit_addr, C	92	2	2
148	MOVC	A, @A+DPTR	93	1	2
149	SUBB	A, #data	94	2	1
150	SUBB	A, data_addr	95	2	1
151	SUBB	A, @R0	96	1	1
152	SUBB	A, @R1	97	1	1
153	SUBB	A, R0	98	1	1
154	SUBB	A, R1	99	1	1
155	SUBB	A, R2	9A	1	1
156	SUBB	A, R3	9B	1	1
157	SUBB	A, R4	9C	1	1
158	SUBB	A, R5	9D	1	1
159	SUBB	A, R6	9E	1	1
160	SUBB	A, R7	9F	1	1

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
161	ORL	C, /bit_addr	A0	2	2
162	AJMP	Code_addr	A1	2	2
163	MOV	C, bit_addr	A2	2	1
164	INC	DPTR	A3	1	2
165	MUL	AB	A4	1	4
166	reserved		A5		
167	MOV	@R0, data_addr	A6	2	2
168	MOV	@R1, data_addr	A7	2	2
169	MOV	R0, data_addr	A8	2	2
170	MOV	R1, Data_addr	A9	2	2
171	MOV	R2, Data_addr	AA	2	2
172	MOV	R3, Data_addr	AB	2	2
173	MOV	R4, Data_addr	AC	2	2
174	MOV	R5, Data_addr	AD	2	2
175	MOV	R6, Data_addr	AE	2	2
176	MOV	R7, Data_addr	AF	2	2
177	ANL	C, /bit_addr	B0	2	2
178	ACALL	Code_addr	B1	2	2
179	CPL	Bit_addr	B2	2	1
180	CPL	C	B3	1	1

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
181	CJNE	A, #data, code addr	B4	3	2
182	CJNE	A, data_addr, code_addr	B5	3	2
183	CJNE	@R0, #data, code_addr	B6	3	2
184	CJNE	@R1, #data, code_addr	B7	3	2
185	CJNE	R0, #data, code_addr	B8	3	2
186	CJNE	R1, #data, code_addr	B9	3	2
187	CJNE	R2, #data, code_addr	BA	3	2
188	CJNE	R3, #data, code_addr	BB	3	2
189	CJNE	R4, #data, code_addr	BC	3	2
190	CJNE	R5, #data, code_addr	BD	3	2
191	CJNE	R6, #data, code_addr	BE	3	2
192	CJNE	R7, #data, code_addr	BF	3	2
193	PUSH	Data_addr	C0	2	2
194	AJMP	Code_addr	C1	2	2
195	CLR	Bit_addr	C2	2	1
196	CLR	C	C3	1	1
197	SWAP	A	C4	1	1
198	XCH	A, data_addr	C5	2	1
199	XCH	A, @R0	C6	1	1
200	XCH	A, @R1	C7	1	1

SESSION 10

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
201	XCH	A, R0	C8	1	1
202	XCH	A, R1	C9	1	1
203	XCH	A, R2	CA	1	1
204	XCH	A, R3	CB	1	1
205	XCH	A, R4	CC	1	1
206	XCH	A, R5	CD	1	1
207	XCH	A, R6	CE	1	1
208	XCH	A, R7	CF	1	1
209	POP	Data_addr	D0	2	2
210	ACALL	Code_addr	D1	2	2
211	SETB	Bit_addr	D2	2	1
212	SETB	C	D3	1	1
213	DA	A	D4	1	1
214	DJNZ	Data_addr, code_addr	D5	3	2
215	XCHD	A, @R0	D6	1	1
216	XCHD	A, @R1	D7	1	1
217	DJNZ	R0, code_addr	D8	2	2
218	DJNZ	R1, code_addr	D9	2	2
219	DJNZ	R2, code_addr	DA	2	2
220	DJNZ	R3, code_addr	DB	2	2

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
221	DJNZ	R4, code_addr	DC	2	2
222	DJNZ	R5, code_addr	DD	2	2
223	DJNZ	R6, code_addr	DE	2	2
224	DJNZ	R7, code_addr	DF	2	2
225	MOVX	A, @DPTR	E0	1	2
226	AJMP	Code_addr	E1	2	2
227	MOVX	A, @R0	E2	1	2
228	MOVX	A, @R1	E3	1	2
229	CLR	A	E4	1	1
230	MOV	A, data_addr	E5	2	1
231	MOV	A, @R0	E6	1	1
232	MOV	A, @R1	E7	1	1
233	MOV	A, R0	E8	1	1
234	MOV	A, R1	E9	1	1
235	MOV	A, R2	EA	1	1
236	MOV	A, R3	EB	1	1
237	MOV	A, R4	EC	1	1
238	MOV	A, R5	ED	1	1
239	MOV	A, R6	EE	1	1
240	MOV	A, R7	EF	1	1

Sl. No.	Mnemonic	Operands	Hex Code	No. of Bytes	No. of Machine cycles
241	MOVX	@DPTR, A	F0	1	2
242	ACALL	Code_addr	F1	2	2
243	MOVX	@R0, A	F2	1	2
244	MOVX	@R1, A	F3	1	2
245	CPL	A	F4	1	1
246	MOV	Data_addr, A	F5	2	1
247	MOV	@R0, A	F6	1	1
248	MOV	@R1, A	F7	1	1
249	MOV	R0, A	F8	1	1
250	MOV	R1, A	F9	1	1
251	MOV	R2, A	FA	1	1
252	MOV	R3, A	FB	1	1
253	MOV	R4, A	FC	1	1
254	MOV	R5, A	FD	1	1
255	MOV	R6, A	FE	1	1
256	MOV	R7, A	FF	1	1

Thank You