Module 6

The various ways of accessing

1) Direct Addressing The operand is specified as an 8 bit memory address. For 8051, only the lower 128 byte internal data memory and the SFR s are directly accessible

Eg MOV A, 07H

moves the content of memory location 07H to the accumulator. 07H refers to the address of data memory at location 7.

2) Indirect Addressing Mode

The operand is specified indirectly an indirect addressing. A register is used for holding the address of the memory variable on which operations are to be performed. R0 and R1 are used as indirect addressing registers

Eg MOV R0, #55H , Load R0 with 55H.
MOV A, @R0 ; Load accumulator with the contents of the memory location pointed by R0.
(for 16bit)
MOV DPTR, #0055H.
MOV A, @DPTR.

3) Register Addressing

Register Addressing mode involves use of register to hold data to be manipulated. In this case, the source and destination register must match in size.

MOV A, R1 ; copy content of Register R1 to register

Movement of data is possible only between Accumulator and other registers, but movement of data among register ($R_0 - R_7$) is not allowed.

4) Immediate Addressing

Constants can be an operand for some instructions. If an instruction makes use of a constant data as the operand, the type of addressing is called immediate addressing. An immediate constant is represented with a leading # symbol in the assembly code.

Eg MOV A, #09H;

) Indexed Addressing

Indexed Addressing is used for program memory access. This addressing mode is used for reading look up table from code memory. A 16 bit register is used to hold the base address of the ~~table~~ look up table. The offset of the table from which data to be returned is loaded in accumulator

Eg: MOV DPTR, #2008H
MOV A, #00H
MOV A, @A+DPTR.

INSTRUCTION SET AND CLASSIFICATIONS

1) Data transfer Instructions

They are used to copy data between register, memory location, accumulator and stack

i) MOV dest, Source

Moving data internally, ie b/w Internal RAM, SFR registers, general register etc

Eg MOV A, R1 (copy data in Register R1 to A)

ii) MOVC A, @ A + <base reg>. [for ROM]

This is code memory read instruction and is used for reading code memory contents to the accumulator.

Eg: MOV DPTR, #1200H

MOV A, #0FH

MOVC A, @ A+DPTR.

iii) MOVX dest, source.

External data memory instruction is used to transfer data between external memory and processor. The registers in external data transfer instructions are DPTR, indirect addressing registers R0/R1 and the accumulator.

Eg MOVX A, @R1
MOVX @R1, A
} If external memory address is 8 bit

MOVX A, @DPTR
MOVX @DPTR, A
} If external memory address is 16 bit.

PUSH

PUSH onto stack. The stack pointer is incremented by one.

Eg: PUSH 4CH: The content of RAM location 4CH is saved onto the stack.

13
+ 5
21

POP

POP from stack: The Stack pointer is decremented by one.

Eg POP 80H.

The content from current SP Address is copied to 80H and SP is decremented by one.

6) XCH

XCH A, Source: Exchange accumulator with byte variable

Eg: XCH A, R3.

XCH A, 50H.

XCH A, @R0.

7) XCHD A, @Ri: Exchange the lower nibble of A with the data pointed by the Indirect Addressing Register R0 or R1.

Eg XCHD A, @R1: Exchange lower nibble between A and RAM location address in R1.

0000 0001
0000 0101

I) Arithmetic Instructions.

Arithmetic Instruction perform basic arithmetic operations including addition, Subtraction, multiplication, division, increment and decrement.

i) ADD A, <location>

Adds content of accumulator and specified location and stores in accumulator. All addressing modes for locations are permitted.

Eg: ADD A, R2; $[A] \leftarrow [A] + [R_2]$.

ii) ADDC A, <location> : Adds the content of <location> with accumulator and carry bit; Store result in accumulator.

Eg: ADDC A, R3;

$[A] \leftarrow [A] + [R3] + [CY]$.

3) SUBB A, <location>

Subtract the content of location from accumulator and borrow bit ; store in accumulator.

Eg SUBB A, R2 ; [A] ← [A] - [R2] - [Cy]

4) INC <location> : Add one on to the source content and store the result in the same source register.

Eg INC A ; add it to the A register.

5) DEC <location> : Decrement one from the source content and store the result in the same source register.

Eg DEC A ; Subtract #01 from the location.

6) MUL AB ; Multiplier Accumulator with B register and Store result in Accumulator & B register.

[Lower order byte of resulting A and higher order byte in B]

Eg ; MOV A, #0A.

MOV B, #02H.

MUL AB ; product = 14H , A = 14H , B = 00H

OV flag = 0.

7) DIV AB ; Divides accumulator with B register and stores the result in accumulator and remainder in B register.

Eg: MOV A, #95 ; A = 95

MOV B, #10 ; B = 10

DIV AB ; A = 09 (Quotient),
B = 05 (remainder)

8) DAA : Decimal Adjust Accumulator.
Used in BCD arithmetic. (Add 6 if digit >9)

Eg: 47
+ 25
72

MOV A, #47H
ADD A, #25H
DAA A.

0100 0111
0010 0101

0110 1100 ⇒ bCH
+0000 0110

0111 0010 ⇒ 72H.

III) Logical Instructions

A) Byte level Logical Operations

i) ANL dest, source:

Performs bitwise logical AND operation between the variables indicated and stores the result in the destination variable.

Eg. ANL A, R2

ii) ORL dest, source

Performs logical OR between the variable.

Eg: ORL A, R2.

iii) XRL dest, source.

XOR performs logic XOR between the destination & source.

Eg: XRL A, R2.

iv) CPLA

Each bit of the accumulator is logically complemented

Eg: MOV A, #55H A = 01010101
CPLA
A = 10 101010

5) CLRA

Clear Accumulator.

Eg: MOV A, #95H. A = 10010101

CLR A A = 0000 0000H.

B) Bit level Logical Operations

i) CLRC

Clear carry : The carry bit is cleared.
No other flags are affected.

Eg CLR C ; C = 0

ii) CLR bit

Clear bit. The initialed bit is cleared.

MOV P3, #FFH. P3 = 1111 1111

CLR P3.2 P3 = 1111 1011 (P3.1, P3.0)

P3.2

iii) SETB C

Set the carry.

Eg: CLR C C = 0.

SETB C C = 1.

4) SETB bit.

Set the bit.

```
MOV P0, #00H
SETB P0.0
```

P0 = 0000 0000
0000 0001

5) CPL C

Complement the carry

```
CLR C   ; C = 0
CPL C   ; C = 1
```

) CPL bit

Complement bit.

```
MOV P0, #00H.   ---> P0 = 0000 0000
CPL P0.0        ---> P0 = 0000 0001
                P0 = 0000 0000
CPL P0.0        ---> P0 = 0000 0000
```

) ANL C, bit

Logical AND for carry.

Eg: SETB C

```
MOV A, #55H     ; A = 0101 0101
ANL C, ACC.7    ; C = 0 since ACC.7.
```

8) ANL C, /bit.

Logical AND Complement for Carry.

SETB C=1.

MOV A, #55H A=01 A = 0101 0101

ANL C, /ACC.7 C=1 Since /ACC.7=1.

9) ORL C, bit.
10) ORL C, bit } Same as AND.

11) MOV C, bit

Move specified bit to carry.

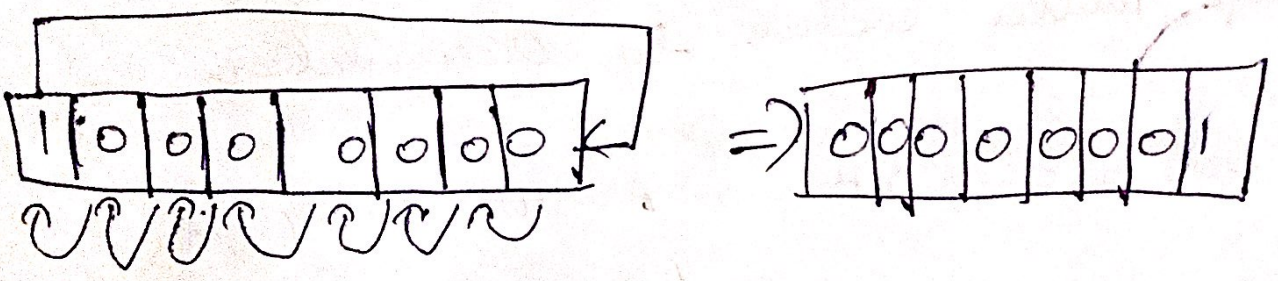
MOV P2, 55H P2 = 0101 0101.

MOV C, P2.7 C=0, Since P2.7=0

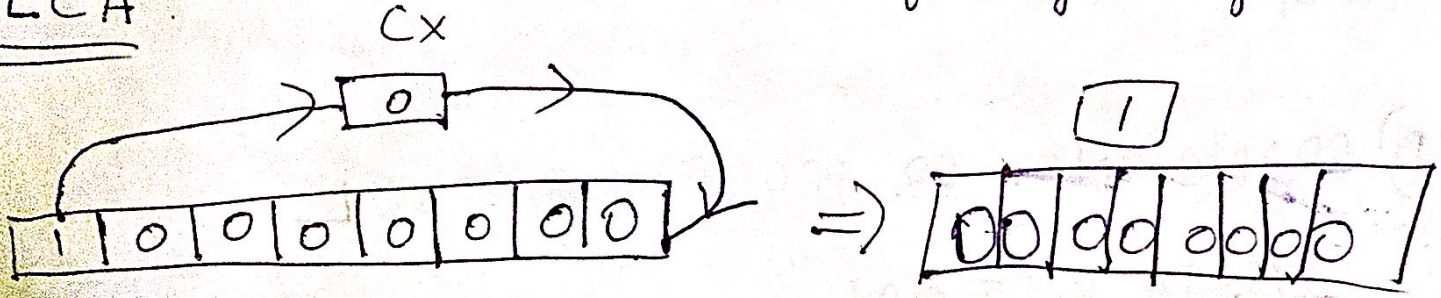
12) MOV bit, C: Move Specified carry to bit.

ROTATE and SWAP Instructions

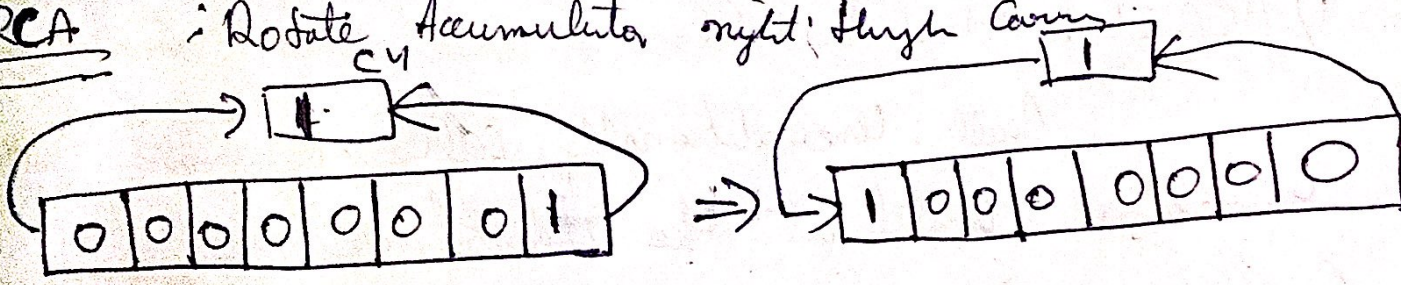
1) **RLA** : Rotate Accumulator to left



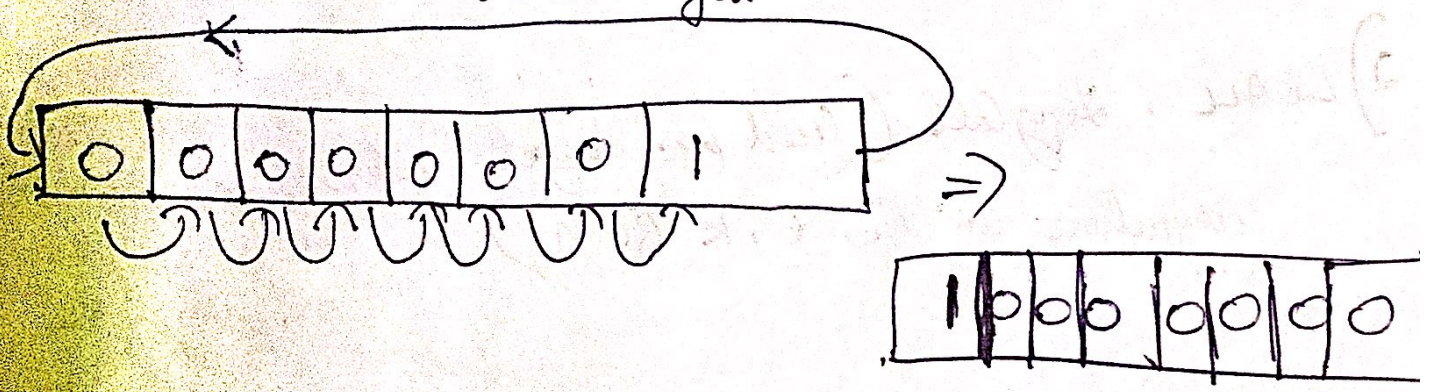
) **RLCA** : Rotate Accumulator left through Carry.



i) **RRCA** : Rotate Accumulator right through Carry



v) **RRA** : Rotate Accumulator right



5) SWAP A.

Swap nibbles within the accumulator.

Eg: MOV A, #0C5H A = 1100 0101, CS
A = 0101 1100; 5C

Eg SWAP A.

D) BRANCHING OR PROGRAM CONTROL INSTRUCTION.

1) ACALL: Absolute Call.

Acall: Unconditionally calls a subroutine located at the indicated address (within 2K).

2) LCALL: Long Call (used for target address anywhere in the 64K ROM).

3) RET (Return from Subroutine)

4) RETI (Return from Interrupt)

5) AJMP (Absolute Jump to Address (within 2K block).

6) LJMP (Long Jump (anywhere in 64K program memory)).

7) SJMP the jump is limited in the range -128 to 127. byte relative to the present PC.

8) JMP @ A+DPTR → Add 8 bit unsigned content of the accumulator with the 16 bit datapointer and load the resulting sum to the program Canter.

Conditional Jumps

JZ - Jump if accumulator is Zero

JNZ - Jump if accumulator is not zero

JC → Jump if carry is set

JNC → Jump if carry is not set

JB → Jump if bit is set

Eg: JB P1.0 Down, JUMP to DOWN if P1.0=1

i. JNB – Jump if bit is not set.

JBC → Jump if bit is set and clear bit.

Eg: JBC ACC.0, FIRST.

Jump to FIRST if ACC.0 = 1 and clear ACC.0.

//Jump hu na Return.

CJNE – Compare and JUMP if Not Equal.

Eg: CJNE A, R3, BACK.

Jump to Back if A ≠ R3.

DJNZ – Decrement and Jump if not equal to zero

Eg: DJNZ R0, BACK.

Decrement R0 and JUMP to BACK if R0 ≠ 0.

NOP – No Operation

Data Types and Directives

1) DB (Define byte)

Used to define 8 bit data

Eg: DATA1 DB 12

A memory location is taken and initialized with number 12.

2) ORG (Origin)

Indicates the beginning of address

Eg: ORG 500H, address begins from location 500 H

3) EQU (equate)

Used to define a constant without applying a memory location Eg: V1 EQU 25 ; V1 = 25

4) END (end)

Used to indicate the end of the source file.

2) Write assembly language program for 8051 to find the largest no. from a group of 'n' number.

Soln Assume n=10, number stored from location 5000H.

```
MOV DPTR, #5000H
CLR C
MOV R2, #09H
MOVX A, @DPTR
MOV 30H, A
INC DPTR
MOVX A, @DPTR
CJNE A, 30H, DN
SJMP: NEXT
DN: JC NEXT
MOV 30H, A
NEXT: DJNZ R2, UP
INC DPTR
MOV A, 30H
MOVX @DPTR, A
```

DPTR → 5020H
C → 0
R2 → 09H

5000H: 9
5001H: 20
5002H: 15
5003H: 9
5004H: 7
5005H: 6
5006H: 12
5007H: 35
5008H: 17
5009H: 11

RESULT 5020H: 35.

DATA SORTING

Write assembly language program to sort an array of N numbers in ascending order

Solution: Let array size = 6, starting from 9000H.

```
        MOV R0, #05H
LOOP1:  MOV DPTR, #9000H
        MOV R1, #05H
LOOP2:  MOVX A, @DPTR
        MOV B, A
        INC DPTR
        MOVX A, @DPTR
        CLR C
        MOV R2, A
        SUBB A, B
        JNC NOEXCHG
        MOV A, B
        MOVX @DPTR, A
        DEC DPL
        MOV A, R2
        MOVX @DPTR, A
        INC DPTR
```

NO EXCHG : DJNZ R1, LOOP2

DJNZ R0, LOOP1

H : SJMP H.

Q) Write a program to copy the value 50H in to R
memory location 30H and 31H using

a) Direct Addressing Mode.

b) Register Indirect Addressing Mode without a loop.

c) with a loop.

Soln : a) MOV A, #50H.

MOV 30H, A.

MOV 31H, A.

b) MOV A, #50H.

MOV R0, #30H

MOV @ R0, A.

INC R0

MOV @ R0, A.

c) MOV A, #50H

MOV R0, #30H

MOV R2, #02H

AGAIN: MOV @R0, A

INC R0

DJNZ R2, AGAIN

---

2) Write ALP for 8051 to clear the lower 128 bytes of Internal RAM with help of DJNZ instruction.

CLR A

MOV R1, #00H

MOV R7, #128

AGAIN: MOV @R1, A

INC R1

DJNZ R7, AGAIN

END

Q) 15 bytes of data stored from location 6CH external RAM of 8051. Write ALP to count no of location which contain data 11H and to store the result to RAM location 6BH

```
        MOV R0, #6CH
        MOV R1, #15H
        MOV R2, #00H
        MOV R3, #6BH
BACK    MOV A, @R0
        CJNE A, #11H, NEXT
        INC R2
NEXT    INC R0
        DJNZ R1, BACK
        MOV A, R2
        END
```

a) Write an Assembly language program for 8051 microcontroller to find out how many bytes are zero out of 30 bytes stored in memory location starting from RAM location 45H.

Assume that ROM space starting at 250H contains "AMERICA", write a program to transfer the bytes into RAM location starting at 40H.

```
        ORG 0250H
MYDATA DB  "AMERICA"
        END

        ORG 0000H
        MOV DPTR, #MYDATA
        MOV R0, #40H
        MOV R2, #7
BACK    CLR A
        MOVC A, @A+DPTR
        MOV @R0, A
        INC DPTR
        INC R0
        DJNZ R2, BACK
HERE:   SJMP HERE
```

Program to perform Matrix Addition.

```
        ORG   0000H
        MOV   R3, #02H
        MOV   R0, #50H
        MOV   R1, #56H
REPEAT: MOV R2, #02H
LOOP  : MOV A, @R0
        INC R0
        INC R0
        INC R0
        ADD   A, @R0
        MOV   @R1, A
        DEC   R0
        DEC   R0
        INC   R1
        DJNZ  R2, LOOP
        MOV   R0, #60H
        MOV   R1, #66H
        DJNZ  R3, REPEAT
        END
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$R_3=$

$R_2=2$

50	1	60 - 3
51	2	61 - 4
52	0	62 - 0
53	5	63 - 7
54	6	64 - 8

8253 Programmable timer interface

8253 is used as a timing and delay generation peripheral, the microprocessor becomes free from the tasks related to the counting process and can execute the program in memory, while the timer device may perform the counting tasks.

Architecture

→ 8253 contains 3 independent 16 bit counters, each with a maximum count rate of 2.6 MHz. Each

→ 3 counters in 8253 are independent of each other in operation, but they are identical to each other in organization.

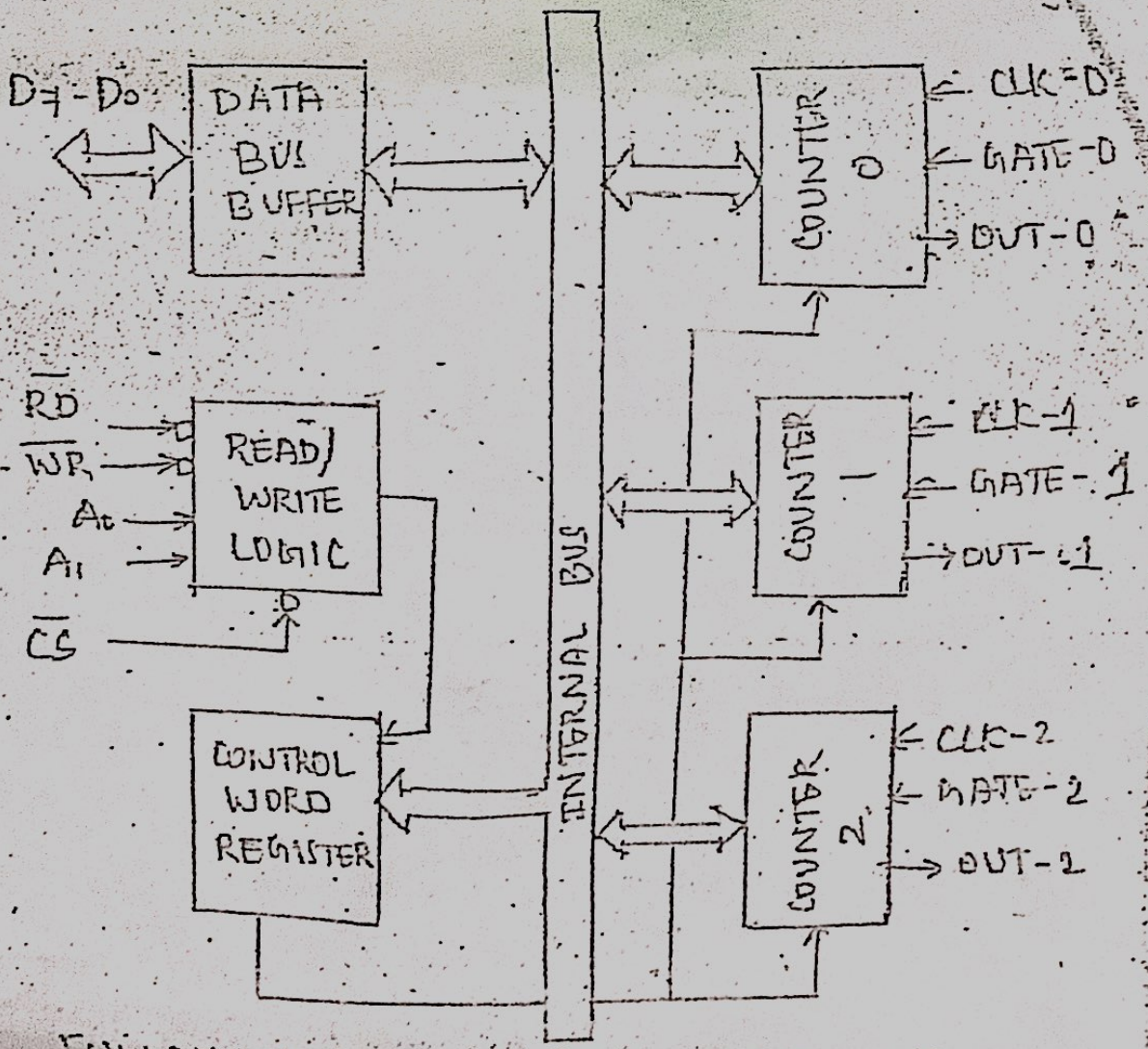
circuit of 8253 to microprocessor system bus

→ A_0, A_1 pins are the address input pins and are required internally for addressing the mode control word register and three counters.

A_0	A_1	
0	0	— counter 0
0	1	— counter 1
1	0	— counter 2
1	1	— control word register.

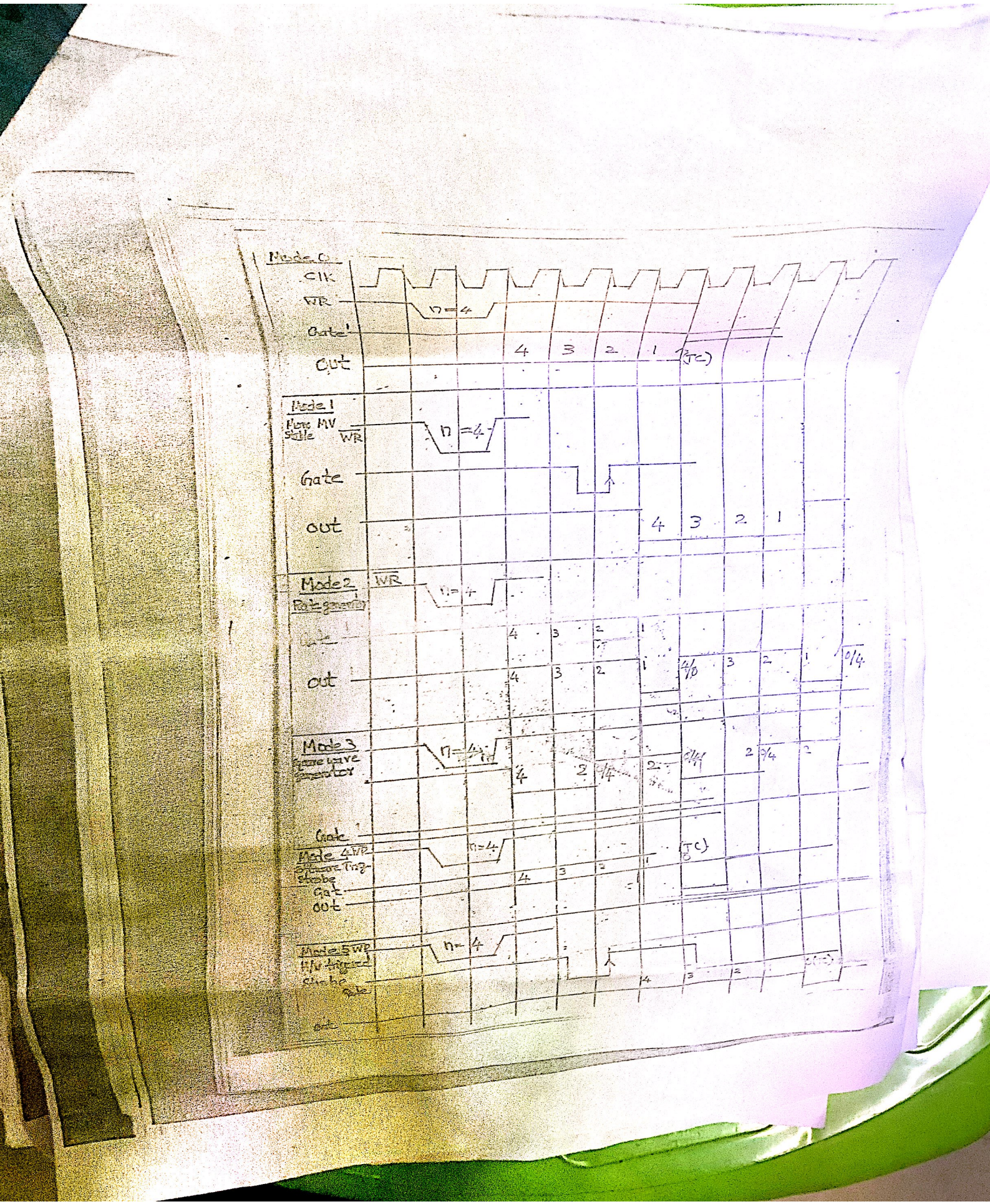
→ operating modes of 8253

1) Mode 0 : Interrupt on terminal count
2) Mode 1 : programmable monostable multivibrator
3) Mode 2 : Rate generator
4) Mode 3 : Square wave generator
5) Mode 4 : Software triggered strobe
6) Mode 5 : Hardware triggered strobe



FUNCTIONAL BLOCK DIAGRAM OF 8253 TIMER

8253 = WITH 8085 PROCESSOR.


Mode 0
CLK
WR
n=4
Gate
Out
4
3
2
1
0 (TC)
Mode 1
Retrig MV
Stable
WR
n=4
Gate
out
4
3
2
1
Mode 2
Rate generator
WR
n=4
out
Mode 3
Square wave generator
n=4
Gate
Mode 4 S/W
Software Trig-Strobe
n=4
Gate
out
Mode 5 H/W
H/W Triggered Strobe
n=4
Gate
out

8. Write an 8051 program to find the sum of digits of an 8 bit unsigned decimal number.

```
        ORG  0000H
        MOV DPTR,#1000H     ; Move 1000H to DPTR
        MOVX A,@DPTR        ; Move content of DPTR to A
        MOV B,A             ; Move A to B
        MOV R0,0FH          ; Move higher nibble Load 0FH to R0
        ANL A,R0            ; AND content of A and R0 (mask Higher nibble)
        MOV R1,A            ; Move lower nibble to R1
        MOV A,B             ; move B to A
        MOV R0,F0H          ; move F0H to R0
        ANL A,R0            ; mask Lower nibble
        RLA                 ; rotate accumulator left
        RLA                 ;  "
        RLA                 ;  "  SWAP A
        RLA                 ;  "
        ADD A,R1            ; Add A and R1
E:      SJMP E              ; wait.
        END
```

Ext mem

45H	1000H

A group of 8 LEDs connected to the lower 4 bits of port P0 of 8051 microcontrollers. Assume that the LEDs shall glow if the corresponding bit is 1. Write an 8051 program which makes the group of LEDs to function as 4-bit ring counter. The program should illustrate to display the ring counter sequence five times continuously and then exit.

(Hint: 4 bit Ring counter sequence is 1000, 0100, 0010, 0001)

```
        ORG 0000H
        MOV R0,#05H
        MOV R1,#04H
        MOV A,#08H
        MOV P0,A
        ACALL DELAY
        RRA
        DJNZ R1,BACK
        DJNZ R0,JMP
HERE:   SJMP HERE

DELAY:  MOV R3,#50
HERE2:  MOV R4,#255
HERE:   DJNZ R4,HERE
        DJNZ R3,HERE2
        RET
        END
```