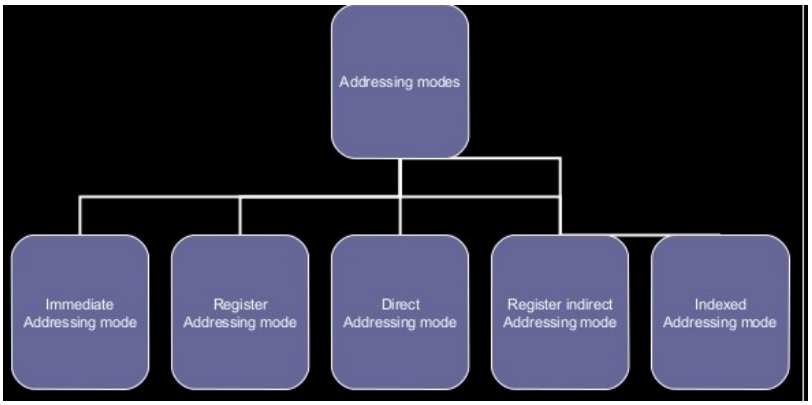
Module VI

8051 Addressing Modes, Different types of instructions and Instruction Set, Simple programs. Peripheral Chips for timing control - 8254/8253.

Addressing Modes

The way of accessing data is called addressing mode. The CPU can access the data in different ways by using addressing modes. The 8051 microcontroller consists of five addressing modes such as:



Immediate Addressing Mode:

In this addressing mode, the source must be a value that can be followed by the '#' and destination must be SFR registers, general purpose registers and address. It is used for immediately storing the value in the memory registers.

Syntax:

```
MOV A, #20h    //A is an accumulator register, 20 is stored in the A//
MOV R0,#15     // R0 is a general purpose register; 15 is stored in the R0 register//
MOV P0, #07h   //P0 is a SFR register;07 is stored in the P0//
MOV 20h,#05h   //20h is the address of the register; 05 stored in the 20h//
MOV DPTR,#4532H // DPTR=4532H.
```

Register Addressing Mode:

In this addressing mode, the source and destination must be a register, but not general purpose registers. So the data is not moved within the general purpose bank registers.

Syntax:

```
MOV A, B;      // A is a SFR register, B is a general purpose register//
MOV R0, R1     //Invalid instruction, GPR to GPR not possible//
```

Direct Addressing Mode

In this addressing mode, the source or destination (or both source and destination) must be an address, but not value.

Direct addressing mode In direct addressing mode, the data is in a RAM memory location whose address is known, and this address is given as a part of the instruction. Contrast this with the immediate addressing mode in which the operand itself is provided with the instruction.

Syntax:

MOV A,20h // 20h is an address; A is a register//

MOV 00h, 07h // both are addressed of the GPS registers//

Indirect Addressing Mode (Register Indirect mode):

In this addressing mode, the source or destination (or destination or source) must be a indirect address, but not a value

In the register indirect addressing mode, a register is used as a pointer to the data. If the data is inside the CPU, only register R0 and R1 are used for this purpose. In other words, R2-R7cannot be used to hold the address of an operand located in RAM when using this addressing mode.

When R0 and R1 are used as pointers, that is, when they hold the address of RAM locations, they must be preceded by the "@" sign.

Ex: - MOV A,@R0 // move contents of RAM location whose address is held by R0 into A.

MOV @R1, B // move contents of B RAM location whose address is held by R1

Base Index Addressing Mode:

This addressing mode is used to read the data from the external memory or ROM memory. All addressing modes cannot read the data from the code memory. The code must read through the DPTR register. The DPTR is used to point the data in the code or external memory.

Syntax:

MOVC A, @A+DPTR //C indicates code memory//

MOCX A, @A+DPTR // X indicate external memory//

Because the data elements are stored in the program space ROM of the 8051,it uses the instruction MOVC instead of MOV.

The 16-bit register DPTR and register "A" are used to form the data element stored in on-chip ROM.

Instruction Set

The 8051 microcontroller can follow CISC instructions with Harvard architecture. In case of the 8051 programming different types of CISC instructions include:

- Data Transfer Instruction set
- Arithmetic Instruction set
- Branching Instruction set
  - Conditional Branching
  - Unconditional Branching
- Logical Instruction set
- Bit Oriented Instruction set

Data Transfer Instruction set

Mnemonics	Operational description	Addressing mode	No. of bytes occupied
Mov a,#num	Copy the immediate data num in to acc	immediate	2
Mov Rx,A	Copy the data from acc to Rx	register	1
Mov A,Rx	Copy the data from Rx to acc	register	1
Mov Rx,#num	Copy the immediate data num in to Rx	immediate	2
Mov A,addr	Copy the data from direct address to acc	direct	2
Mov addr,A	Copy the data from acc to direct address	direct	2
Mov addr,#num	Copy the immediate data num in to direct address	direct	3
Mov addr1,addr2	Copy the data from address2 to address1	direct	3
Mov Rx,addr	Copy the data from direct address to Rx	direct	2
Mov addr,Rx	Copy the data from Rx to direct address	direct	2
Mov @Rp,A	Copy the data in acc to address in Rp	Indirect	1
Mov A,@Rp	Copy the data that is at address in Rp to acc	Indirect	1
Mov addr,@Rp	Copy the data that is at address in Rp to address	Indirect	2
Mov @Rp,addr	Copy the data in address to address in Rp	Indirect	2
Mov @Rp,#num	Copy the immediate byte num to the address in Rp	Indirect	2
Movx A,@Rp	Copy the content of external add in Rp to acc	Indirect	1

Movx A,@DPTR	Copy the content of external add in DPTR to acc	Indirect	1
Movx @Rp,A	Copy the content of acc to the external add in Rp	Indirect	1
Movx @DPTR,A	Copy the content of acc to the external add in DPTR	Indirect	1
Movc A,@a+DPTR	The address is formed by adding acc and DPTR and its content is copied to acc	indirect	1
Movc A, @a+PC	The address is formed by adding acc and PC and its content is copied to acc	indirect	1
Push addr	Increment SP and copy the data from source add to internal RAM address contained in SP	Direct	2
Pop addr	copy the data from internal RAM address contained in SP to destination add and decrement SP	direct	2
Xch A, Rx	Exchange the data between acc and Rx	Register	1
Xch A, addr	Exchange the data between acc and given add	Direct	2
Xch A,@Rp	Exchange the data between acc and address in Rp	Indirect	1
Xchd A, @Rp	Exchange only lower nibble of acc and address in Rp	indirect	1

Arithmetic Instruction Set:

The arithmetic instructions perform the basic operations such as:

- Addition
- Multiplication

- Subtraction
- Division

MNEMONICS	DESCRIPTION	BYTES
ADD A,Rr	A+Rr-->A	1
ADD A,@Rp	A+(Rp)-->A	2
ADD A, #N	A+N-->A	1
SUB A,Rr	A-Rr-->A	2
SUB A,@Rp	A-(Rp)-->A	1
SUB A, #N	A-N-->A	2
DEC A	A-1-->A	1
DEC Rr	Rr-1-->	2
DEC ADD	(ADD)-1-->ADD	1
INC A	A+1-->A	2
INC Rr	Rr+1-->	1
INC ADD	(ADD)+1-->ADD	2
DIV AB	A/B-->AB	1
MUL AB	A*B-->AB	2

DAA- Decimal adjust after addition

Addition:

```
ORG 0000h
MOV R0, #03H     // move the value 3 to the register R0//
MOV A, #05H     // move the value 5 to accumulator A//
ADD A, 00H        // addA value with R0  value and stores the result inA//
END
```

Multiplication:

```
ORG 0000h
MOV R0, #03H     // move the value 3 to the register R0//
MOV A, #05H     // move the value 5 to accumulator A//
MUL A, 03H        // Multiplied result is  stored in the Accumulator A //
END
```

Subtraction:

```
ORG 0000h
MOV R0, #03H     // move the value 3 to register R0//
MOV A, #05H     // move the value 5 to accumulator A//
SUBB A, 03H     // Result value is stored in the Accumulator A //
END
```

Division:

```
ORG 0000h
MOV R0, #03H     // move the value 3 to register R0//
MOV A, #15H     // move the value 5 to accumulator A//
DIV A, 03H     // final value is stored in the Accumulator A //
END
```

Logical Instruction Set:

Mnemonics	Description	Bytes
ANL A, Rr	A AND Rr-->A	1
ANL A, ADD	A AND(ADD)-->A	2
ORL A, Rr	A OR Rr-->A	1
ORL A, ADD	A OR(ADD)-->A	2
XRL A, Rr	A XOR Rr-->A	1
XRL A, ADD	A XOR(ADD)-->A	2
CLR A	00-->A	1
CPL A	A bar-->A	2

Syntax:

MOV A, #20H /00100000/

MOV R0, #03H /00000101/

ORL A, R0 //00100000/00000101=00000000//

Syntax:

MOV A, #20H /00100000/

MOV R0, #03H /00000101/

ANL A, R0

3. Syntax:

MOV A, #20H /00100000/

MOV R0, #03H /00000101/

XRL A, R0

The 8051 microcontroller consist **four shift operators**:

- RR —> Rotate Right
- RRC —>Rotate Right through carry
- RL —> Rotate Left
- RLC —>Rotate Left through carry

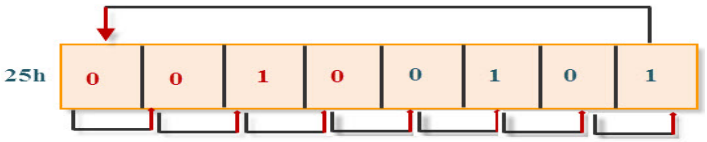
Rotate Right (RR):

In this shifting operation, the MSB becomes LSB and all bits shift towards right side bit-by-bit, serially.

Syntax:

MOV A, #25h

RRA



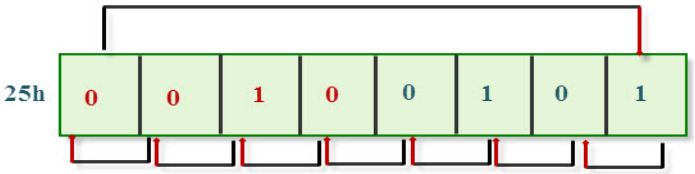
Rotate Left (RL):

In this shifting operation, the MSB becomes LSB and all bits shift towards Left side bit-by-bit, serially.

Syntax:

MOV A, #25h

RLA



RRC Rotate Right through Carry:

In this shifting operation, the LSB moves to carry and the carry becomes MSB, and all the bits are shift towards right side bit by bit position.

Syntax:

MOV A, #27h

RRC A

RLC Rotate Left through Carry:

In this shifting operation, the MSB moves to carry and the carry becomes LSB and all the bits shift towards left side in a bit-by-bit position.

Syntax:

MOV A, #27h

RLC A

Branch Instruction set

Conditional Instructions

The CPU executes the instructions based on the condition by checking the single bit status or byte status. The 8051microcontroller consists of various conditional instructions such as:

Instruction	Action
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and jump if register ≠ 0
CJNE A, data	Jump if A ≠ data
CJNE reg, #data	Jump if byte ≠ #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if bit = 0
JBC	Jump if bit = 1 and clear bit

Call and Jump Instructions:

The call and jump instructions are used to avoid the code replication of the program. When some specific code used more than once in different places in the program, if we mention specific name to code then we could use that name anywhere in the program without entering a code for every time. This reduces the complexity of the program.

CALL → ACALL & LCALL

The 8051 provides 2 forms for the CALL instruction:

- Absolute Call – ACALL
  - Uses an 11-bit address
  - The subroutine must be within the same 2K page.
- Long Call – LCALL
  - Uses a 16-bit address
  - The subroutine can be anywhere.

Both forms push the 16-bit address of PC on the stack and update the stack pointer.

LJMP (long jump)

LJMP is an unconditional long jump. It is a 3-byte instruction in which the first byte is the opcode, and the second and third bytes represent the 16-bit address of the target location.

SJMP (short jump)

In this 2-byte instruction, the first byte is the opcode and the second byte is the relative address of the target location.

AJMP

Function: Absolute Jump Within 2K Block

Syntax: AJMP *code address*

Loop Instructions:

The loop instructions are used to repeat the block each time while performing the increment and decrement operations. The 8051 microcontroller consist two types of loop instructions:

- CJNE —> compare and jump if not equal
- DJNZ —> decrement and jump if not zero

Compare and Jump if Not Equal – CJNE

- Compare the magnitude of the two operands and jump if they are not equal.
  - The values are considered to be unsigned.
  - The Carry flag is set / cleared appropriately.
  - CJNE A, direct, rel
  - CJNE A, #data, rel
  - CJNE Rn, #data, rel

Decrement and Jump if Not Zero – DJNZ

– Decrement the first operand by 1 and jump to the location identified by the second operand if the resulting value is not zero.

DJNZ 20H,LOC1
DJNZ 30H,LOC2
DJNZ 40H,LOC3

- If internal RAM locations 20H, 30H and 40H contain the values 01H, 5FH and 16H respectively,
- the above instruction sequence will cause a jump to the instruction at LOC2, with the values 00H, 5EH, and 15H in the 3 RAM locations

Return instructions: RET and RETI(Return from subroutine and Return from interrupt service routine)

BIT-Oriented Instructions

Mnemonic	Description	Byte	Cycle
CLR C	Clears the carry flag	1	1
CLR bit	Clears the direct bit	2	3
SETB C	Sets the carry flag	1	1
SETB bit	Sets the direct bit	2	3
CPL C	Complements the carry flag	1	1
CPL bit	Complements the direct bit	2	3
ANL C,bit	AND direct bit to the carry flag	2	2
ANL C,/bit	AND complements of direct bit to the carry flag	2	2
ORL C,bit	OR direct bit to the carry flag	2	2
ORL C,/bit	OR complements of direct bit to the carry flag	2	2
MOV C,bit	Moves the direct bit to the carry flag	2	2
MOV bit,C	Moves the carry flag to the direct bit	2	3

Simple Programs using 8051

Write A Program To Add Two 8 Bit Number And Store The Result At External Memory Location 2050H.

```
ORG 0000H
 MOV A, #05H
 MOV R, #09H
ADD A, R1
MOV DPTR, #2050H      // Initialize DPTR with 2050 address
MOVX @ DPTR, A        // Move content of A into 2050. We can see output in 2050 location
```

AGAIN: SJMP AGAIN // this loop work infinitely, indicates end of the program

INPUT:

A=05H

R1=09H

OUTPUT:

2050 = 0EH..

Subtraction of two 8 bit numbers

```
ORG 4100
MOV A,#data1
SUBB A,#data2
MOV DPTR,#4500
MOVX  @DPTR,A   // X  indicate external memory
HERE: SJMP HERE
```

Input: 66

23

Output: 43 (4500)

Assembly program for Simple 2 16bit number addition using registers

```
ORG 0000H
MOV R7, #34H          // SAY R7 HAS LOWER BYTE 34
MOV R6, #24H          // R6 HAS HIGHER BYTE 24
                      // MAKING NUMBER 2434

MOV R5,#45H           // ANOTHER NUMBER SAY 3345
MOV R4, #33H          //WITH R5 LOWER BYTE 45
                      // R4 HIGHER BYTE 33
MOV A, R5
ADD A , R7
MOV R3, A
MOV A, R6
ADDC A, R4
MOV R2,A
END
```

Multiplication of two 8 bit numbers

MOV DPTR,#8600H // initialize DPTR with 8600

```
MOVX A,@DPTR            // Move content of DPTR into A
MOV B,A
MOV DPTR,#8601H         // Read the second operand from 8601 location
MOVX A,@DPTR
MUL AB
MOV DPTR,#8701H         // Initialize DPTR with 8701 for storing output data
MOVX @DPTR,A
MOV DPTR,#8700H         //  The least significant byte of the result is placed in the Accumulator and the
// most significant-byte is placed in the "B" register. So we need to display from B
MOV A,B
MOVX @DPTR,A
E: SJMP E
```

Factorial of a number using subroutine

```
ORG 0000
MOV R1,#05
MOV A, R1
LCALL FACT
E: SJMP E
        FACT:
            CJNE R1,#00,UP      // compare R1 and 00 if it is not equal go to UP
        RET
        UP: DEC R1              // Decrement R1 by one ie 4
        MOV B, R1
        MUL AB
        LJMP FACT
```

Sorting of n numbers

```
ORG 0000H
MOV R7,#4
loop1:MOV R0,#40H
   MOV R6,#04
loop:MOV A,@R0
  INC R0
  MOV 50H,@R0
  CJNE A, 50H, next
  SJMP down
next:JC down
```

```
 MOV @R0,A
 DEC R0
 MOV @R0,50H
down:DJNZ R6,loop
 DJNZ R7,loop1
END
```

To add n bytes stored in external RAM(Sum of n numbers)

```
        MOV R0, #0A
        MOV R1, #00
        MOV DPTR, #9000
        Loop: MOVX A, @DPTR
        ADD A, R1
        MOV R1, A
        INC DPTR
        DJNZ R0, LOOP
```

Fibonacci series (Refer Page 87 in solved Text book)

```
 MOV R1, 30 H
MOV R7,#40H
MOV @R7, #00H
INC R7
MOV @R7, #01H
MOV R5, #42H
DEC R1
DEC R1
DEC R7
LOOP: MOV A, @R7
        INC R7
        ADD A, @R7
        MOV @R5,A
        INC R5
        DJNZ R1, LOOP
STOP: SJMP STOP
```

8254/8253 Programmable Interval Timer

The Intel 8254 is a counter/timer device designed to solve the common timing control problems in microcomputer system design.

It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz.

All modes are software programmable.

The 8254 is a superset of the 8253.

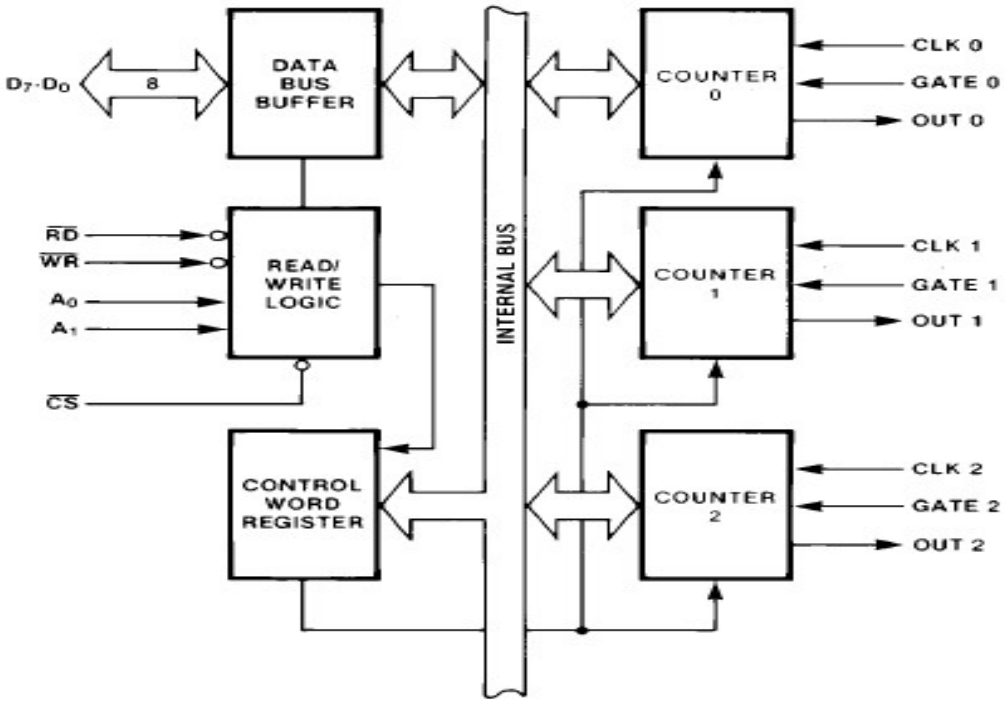
The 8254 uses HMOS technology and comes in a 24-pin plastic or CERDIP package.

- These three counters can be programmed for either binary or BCD count.
- It is compatible with almost all microprocessors.
- 8254 has a powerful command called READ BACK command, which allows the user to check the count value, the programmed mode, the current mode, and the current status of the counter.

Data Bus Buffer

It is a tri-state, bi-directional, 8-bit buffer, which is used to interface the 8253/54 to the system data bus. It has three basic functions −

- Programming the modes of 8253/54.
- Loading the count registers.
- Reading the count values.



Read/Write Logic

It includes 5 signals, i.e. RD, WR, CS, and the address lines A_0 & A_1. Address lines A_0 & A_1 of the CPU are connected to lines A_0 and A_1 of the 8253/54, and CS is tied to a decoded address. The control word register and counters are selected according to the signals on lines A_0 & A_1.

A1	A0	Function
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control

Control Word Register

This register is accessed when lines A_0 & A_1 are at logic 1. It is used to write a command word, which specifies the counter to be used, its mode, and either a read or write operation.

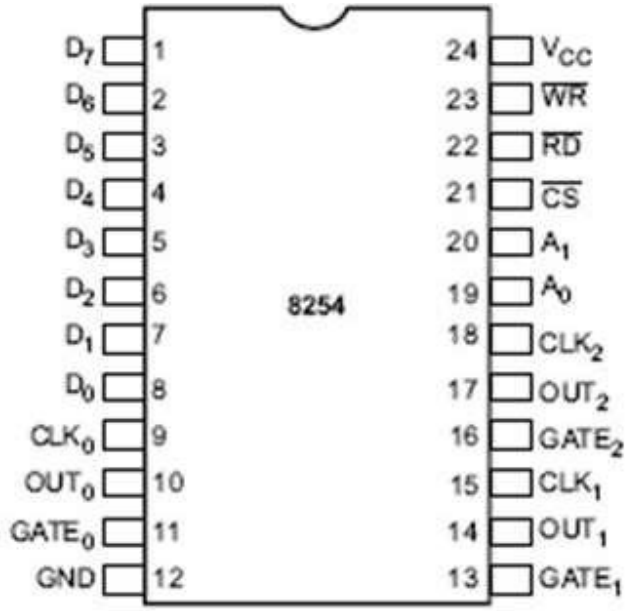
Counters

Each counter consists of a single, 16 bit-down counter, which can be operated in either binary or BCD. Its input and output is configured by the selection of modes stored in the control word register. The programmer can read the contents of any of the three counters without disturbing the actual count in process.

Each counter has 2 input signals CLK and GATE and one output signal OUT.

GATE is used to enable or disable counters

Pin Diagram



D7-D0: Data lines
A1,A0: Selects one of the 4 internal registers in the 8254
CS: Enables the 8254

CLK: Timing source for each of the 3 timers. e.g. pulses applied to CLK0 are used to decrement counter 0.

G: Gate = 1, enables the counter.

e.g. when G0=1, counter 0 decrements on CLK

OUT: Timer waveform output (a clock or a pulse)

RD: Causes data to be read from the 8254 (IORC)

WR: Causes data to be written to 8254 (IOWC)

Difference between 8253 and 8254

8253	8254
Its operating frequency is 0 - 2.6 MHz	Its operating frequency is 0 - 10 MHz
It uses N-MOS technology	It uses H-MOS technology
Read-Back command is not available	Read-Back command is available
Reads and writes of the same counter cannot be interleaved.	Reads and writes of the same counter can be interleaved.

Programming the 8254(Control word register)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Counter Latch Command

- Selected counter has its content transferred into temporary latch, which can be read by CPU.

Read back command

A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{COUNT}$	$\overline{STATUS}$	CNT 2	CNT 1	CNT 0	0

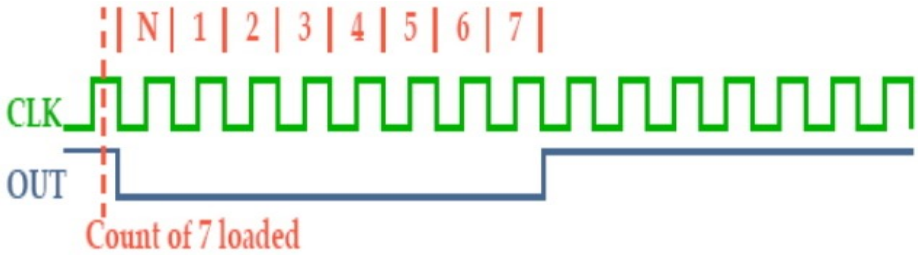
D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counters(s)
D_3: 1 = Select Counter 2
D_2: 1 = Select Counter 1
D_1: 1 = Select Counter 0
D_0: Reserved for future expansion; Must be 0

Modes of operation of 8254

MODE 0: INTERRUPT ON TERMINAL COUNT

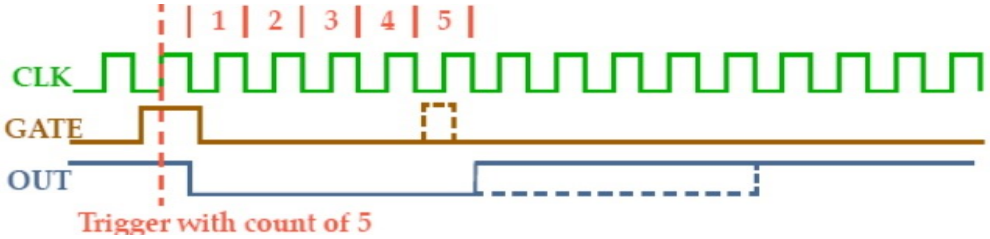
In this mode OUT is low. Once a count is loaded the counter is decremented after every cycle, and when count reaches zero, the OUT goes high.

This can be used as an interrupt. The OUT remains high until a new count or command word is loaded.



MODE 1: HARDWARE RETRIGGERABLE ONE SHOT

In this mode OUT is initially high. When gate is triggered, the OUT goes low, and at the end of count it goes high again, thus generating a one shot pulse.

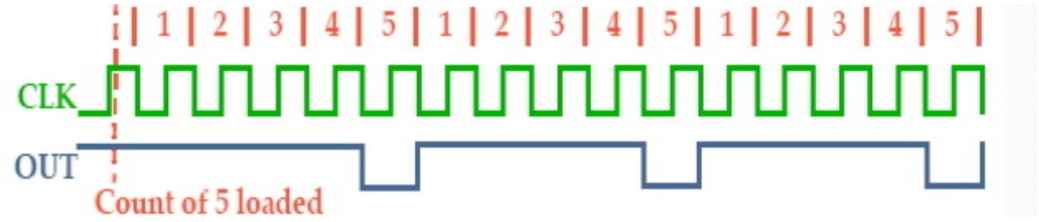


A pulse input has to be sent to gate input in order to trigger the counter

MODE 2: RATE GENERATOR

The mode is used to generate a pulse equal to given clock period at a given interval.

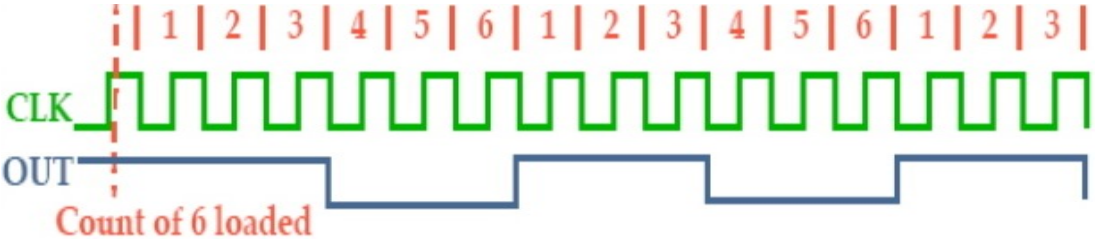
When a count is loaded, the OUT stays high until count reaches 1 and then OUT goes low for 1 clock period then gets reloaded automatically and this is how pulse gets generated continuously.



MODE 3: SQUARE WAVE GENERATOR

In this a continuous square wave with period equal to count is generated.

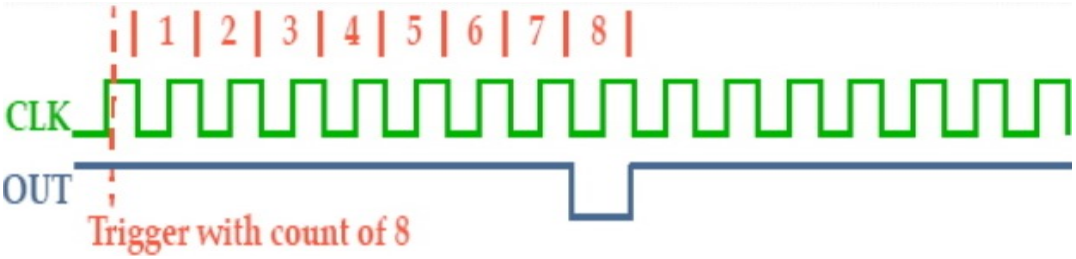
The frequency of square wave = frequency of clock divide by count. if count (N) is odd pulse stay high for (N + 1)/2 and low for (N – 1)/2.if count is even output remains high for half the count and low for the rest half of the count.



MODE 4: SOFTWARE TRIGGERED STROBE

In this mode OUT is initially high, it goes low for one clock period at the end of count.

The count must be reloaded for subsequent outputs.



These pulses can be used as strobe for interfacing MP with peripherals

MODE 5: HARDWARE TRIGGERED STROBE

Same as MODE4 except that it is triggered by rising pulse at gate.