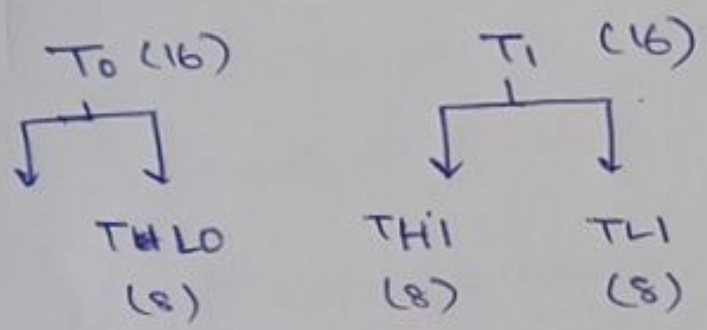
Microprocessors

Unit 5

* 8051 Timers

→ 8051 has 2 timers - T_0 and T_1 - this means that two delays can happen at the same time.

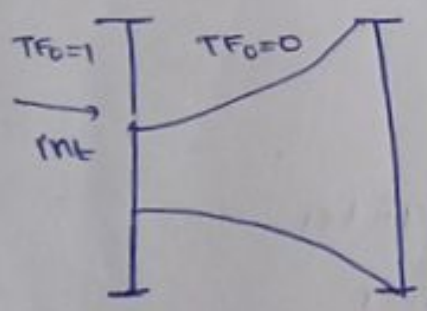
→ They are 16-bit timers, the count must be loaded, which is loaded into the SFRs.



Each timer has 2 SFRs.

→ They are up-counter, meaning it goes from the given count upto FFFF and then overflows.

→ The moment the timer finishes counting, it sends an interrupt. ISR



Once overflow happens, The TF_x flag is set to 1. It goes to the ISR that is to be carried out after the delay. On the way, it resets the TF_x flag back to 0 (so that looping does not happen when it returns from the ISR).

→ The TF_x flags are important because they help store the pending interrupts.

→ In the event of multiple interrupts, they are serviced in order of priority. All the pending interrupts would still have their TF_x flags 1, so that they are serviced next. After servicing, TF_x is made 0.

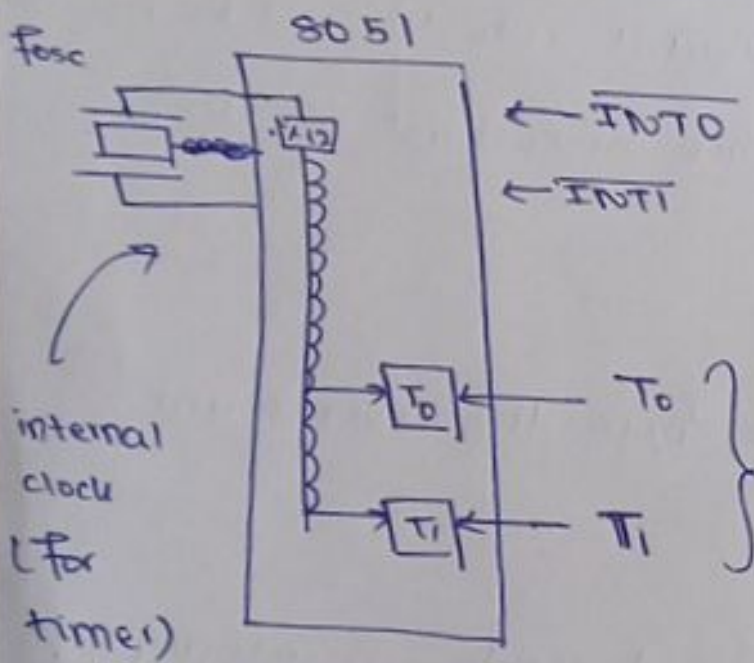
* Calculating the count — max count - desired count + 1
(FFFF)

* Counters vs. Timers — If the frequency is fixed ⇒ a timer

If the frequency is variable ⇒ counter.

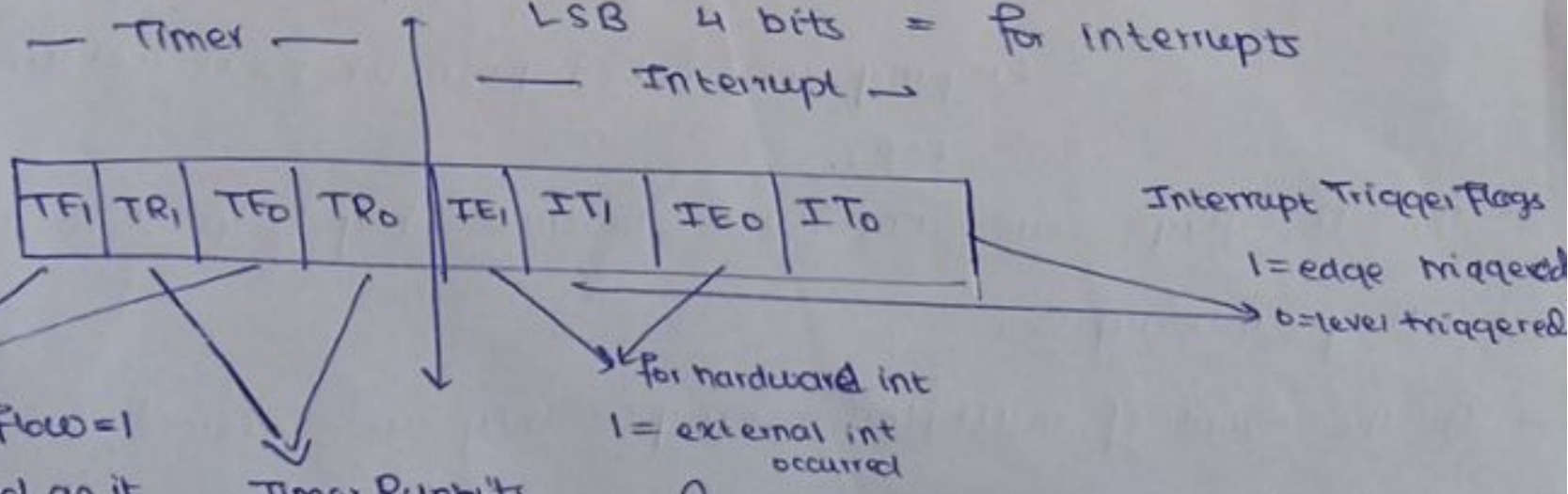
Also, if internal clock pulses are used (XTAL) ⇒ a timer

if external clock pulses are used ⇒ a counter



* Timer Registers — TCON & TMOD

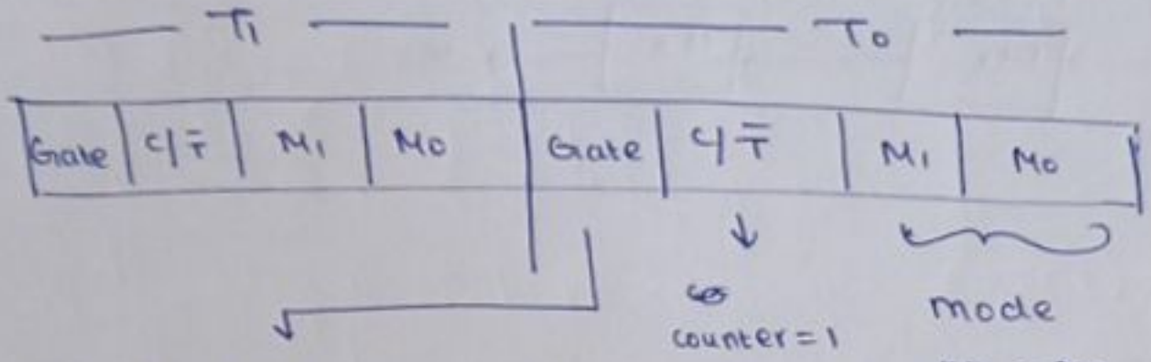
A. TCON — timer control — a bit addressable register
— 8 bits — MSB 4 bits = for timer
LSB 4 bits = for interrupts



8. TMOD - Timer mode

MSB 4 bits = for Timer 1

LSB 4 bits = for Timer 0



enables hardware control

use a button

C/T
counter = 1
timer = 0

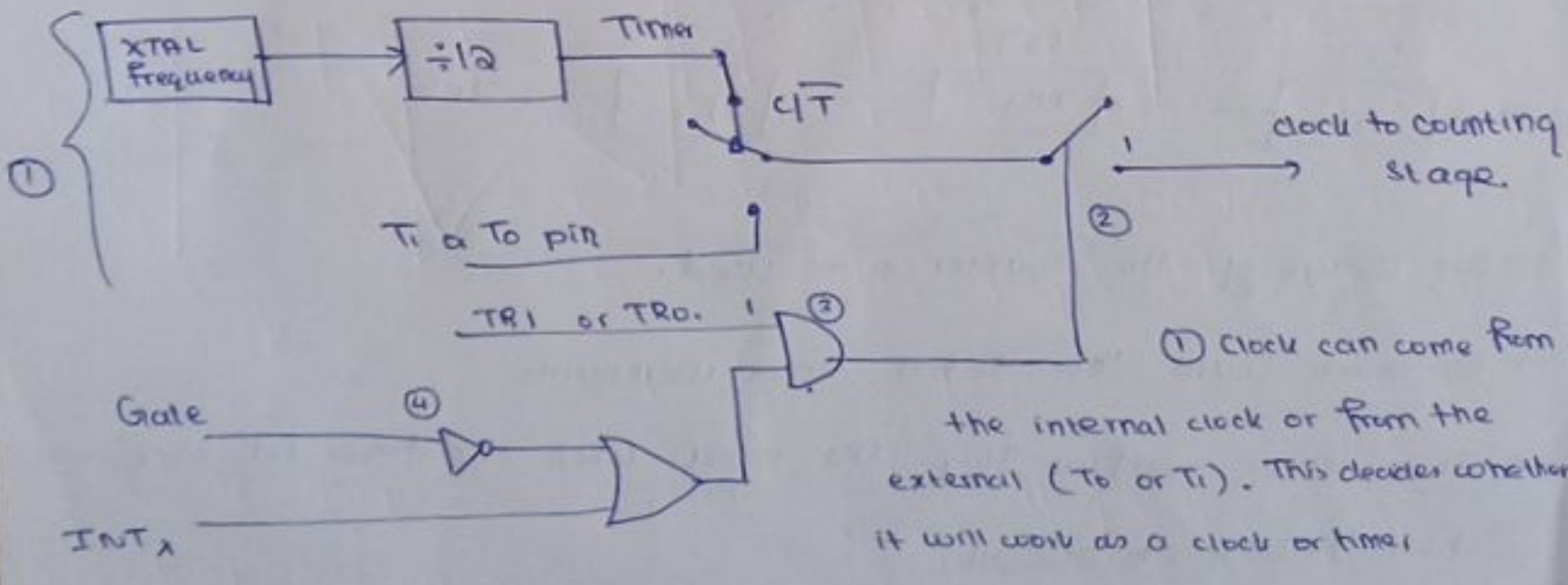
mode
00 - M0
01 - M1
10 - M2
11 - M3

1 = counting is controlled by INTx (connect button to INTx, which in turn controls Tx)

0 = counting independent of INTx

(Note that if hardware control is enabled, timers can be started/stopped, but interrupts can no longer happen on that line)

* Timer. Counter logic Diagram



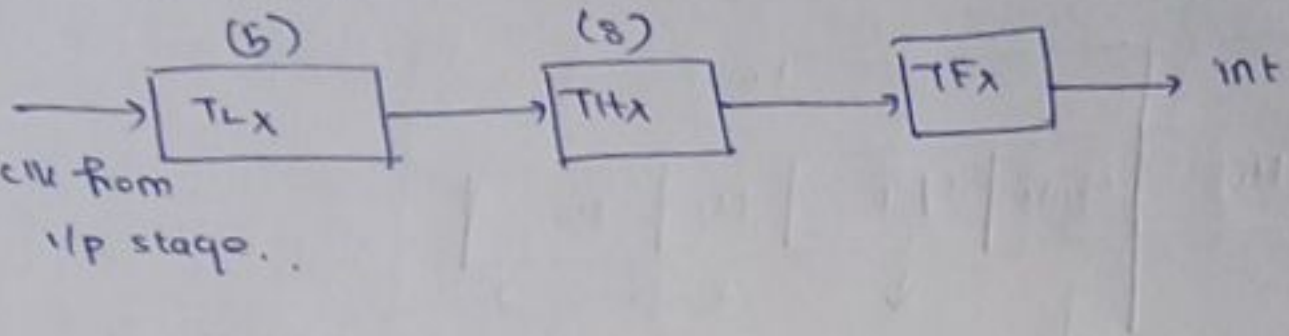
(2) Line should be 1, for clock to run.

(3) In the AND gate & TRx is always 1 (since software control is mandatory)

(4) If Gate is kept as 1, after the inverter, it becomes 0. It is wholly dependent on INTx. Only if INTx is 1 - hardware control is enabled.

* Timer Modes

A. **Mode0** - 13 bit Timer / Counter.



→ Max size is 2^{13} = 8K.

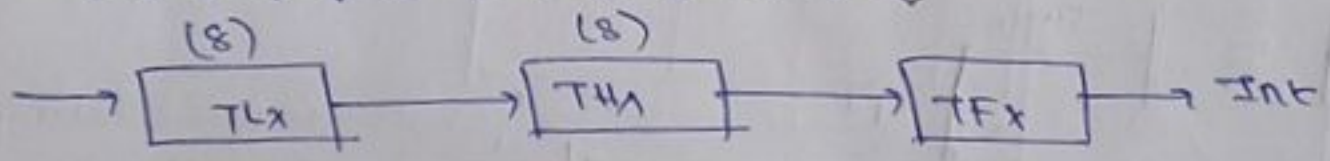
→ The mode is used to divide any natural frequency by 32 (That is why TLx has 5 bits)

→ On each count TLx increments.

→ Each time TLx rollsover (overFlows), THx increments (when THx goes from FFH to 00H)

→ TFx is set when THx overflows

B. **Mode1** - 16 bit Timer / Counter



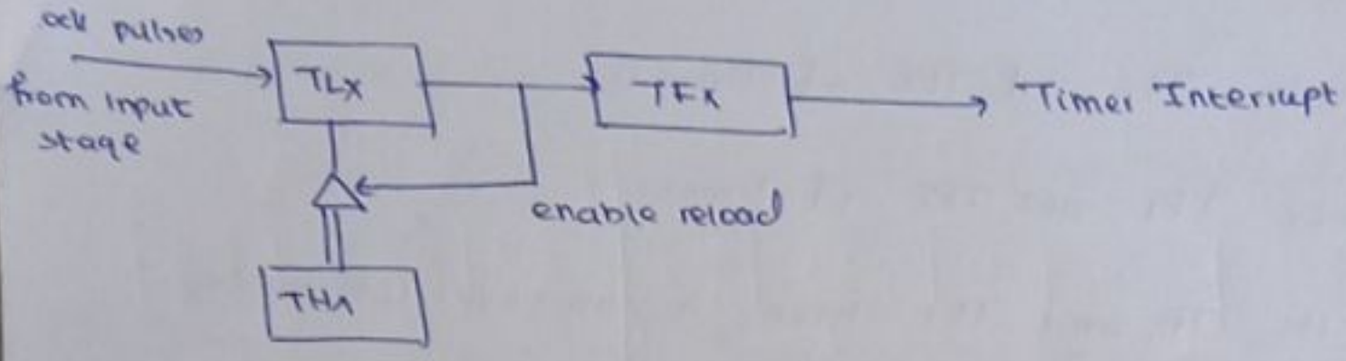
→ All 16 bits of the counter are used.

→ On each count the 16bit timer increments

→ The timer overflow flag TFx is set when the timer rolls over from FFFFH to 0000H.

→ Max count ⇒ 2^{16} = 16K

C. **Mode 2** — Auto-reload T_L from T_H



→ TLx is used as an 8-bit counter

→ THx holds the count value to be reloaded

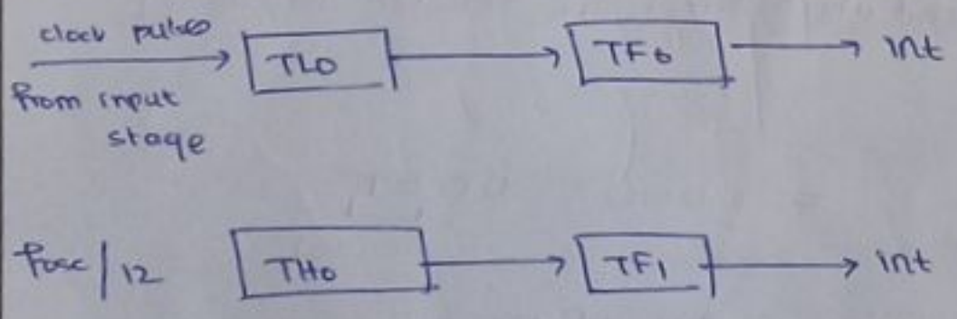
→ On each count TLx increments

→ When TLx rolls over from FFH to 00H, the following events happen:

1. Timer overflow flag TFx is set — timer interrupt happens
2. The value of THx is copied into TLx (auto-reload) count start again.

→ Max count = 2^8

D. **Mode 3** - (Two 8-bit timers from Time 0)



→ Timer 0 is used as 2 separate 8-bit timers TH_0 and TL_0 (3)
(This would give 3 timers - TL_0, TH_0, T_1)

→ TL_0 uses TF_0 and TR_0 of Timer 0

→ TH_0 uses TF_1 and TR_1 of Timer.

→ T_1 has its TR_1 and TF_1 stolen, it can never inform the processor.

→ This mode is used when T_1 is used in serial port communication (where it does not have to inform the processor). If 2 timers are still needed TL_0 and TH_0 are used.

→ Note that this mode (esp TH_0) can work only as a timer, not a counter. To be a counter, an external clock pulse must be given via T_0 or T_1 lines

Here T_1 would have its line

TL_0 would take the T_0 line

TH_0 would have no external line ⇒ cannot work as a counter

* Examples and Timer Programming Questions

(1) Indicate which mode and which timer are selected for each of the following:

a. MOV TMOD #01 = 0000 0001
(0001 = mode 1)

= mode 1 of timer 0

(b) mov TMOD, #20H = 0010 0000 (7)

⇒ mode 2 of Timer 1

(c) mov TMOD, #12H = 0001 0010

⇒ mode 1 of Timer 1 & mode 2 of Timer 0

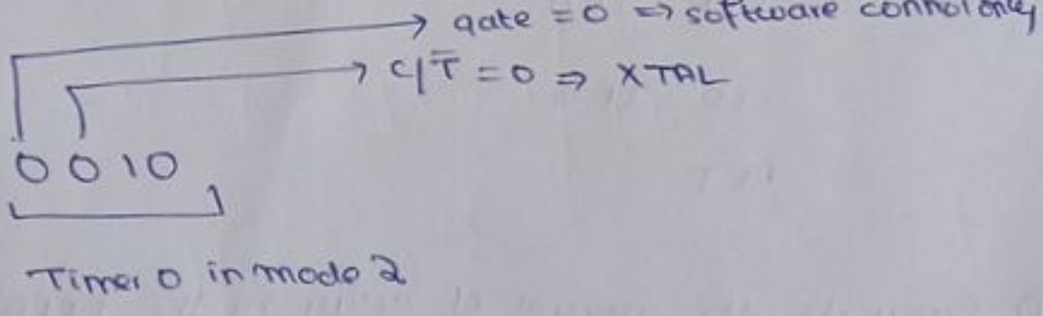
(2) Find the timer's clock frequency and its period for various 8051-based systems, with the crystal frequency 11.0592 MHz, when the C/T bit of TMOD is 0.

Frequency: $\frac{XTAL}{12} = \frac{11.0592}{12} = 0.9216 \text{ MHz} = 921.6 \text{ kHz}$

$T = 1/f = \frac{1}{921.6 \times 10^3}$

$= 1.085 \mu s$

(3) Find the value for TMOD if we want to program Timer 0 in mode 2. Use XTAL for the clock source, and use instructions to start/stop the timer.



(4) (i) Create a square wave of 50% duty cycle on the P1.5 bit. Timer 0 is used to generate the time delay, for 12 counts

FFFF - FFF2 + 1

Ans: mov TMOD, #01H

mov TL0, #0F2H

mov TH0, #0FFH

Setting the registers

To generate 14 ms Delay

do

65535 - 14 + 1

= 65522

= FFF2

Program

```
main :  CLR  P1.5
        ACALL  DELAY
        SETB   P1.5
        ACALL  DELAY
        SJMP HERE main

DELAY:  MOV  TMOD, #01H
        MOV  TL0 , #0F2H
        MOV  TH0,  #0FFH

wait :  JNB.  TF0, wait
        CLR  TR0
        CLR  TF0
        RET
```

(11) Calculate the amount of delay in the DELAY subroutine generated by the timer. (Exclude instructions)

$$F = \frac{11.0592}{12} = 921.6\ kHz$$

$$T = \frac{1}{921.6\ kHz} = 1.085\ \mu s$$

Delay = no. of counts x time period

= 14 x 1.085 μs

= 15.19 μs

This is for one call of the DELAY fn. ⇒ for half the pulse.

For the entire period, it would be T = 2 x 15.19 μs

= 30.38 μs

(5) (i) Calculate the delay created by the following code. Exclu overhead due to the instructions.

```
        CLR   P2.3
        mov   TMOD, #01
here :  mov TL0 , #3EH
        mov TH0 , #0B8H
        SETB  P2.3
        SETB  TR0
AGAIN : JNB   TF0, AGAIN
        CLR TR0
        CLR TF0
        CLR P2.3
```

Ans: Find no. of counts = FFFF - B83E + 1

= 47C1 + 1

= 47C2 H = 18370

Delay = count × time period

= 18370 × 1.085 μs

= 19.93145 ms

(ii) Modify the program given in the Q1 so that the largest delay possible is generated, and hence find the delay (w/o inst. overhead)

⇒ mov TL0, #00H

mov TH0 , #00H

⇒ count = 65,536

Delay = count × Time period

= 65,536 × 1.085 μs

= 71.1065 ms

(6) Calculate the time delay and frequency for the delay in the given program. Exclude overhead from instructions

```
        mov  TMOD, #10
again:  mov  TL1, # 34H
        mov  TH1, #76H
        SETB TR1
back :  JNB  TF1, back
        CLR  TR1
        CPL  P1.5
        CLR  TF1
        sjmp again
```

Ans: count = FFFF - 7634H + 1

= 89CB H + 1

= 89CC = 35 276

Delay = count × time period

= 35276 × 1.085 μs

= 38.274 ms

F = 1/Delay = 13.064 Hz

(7) Assume that XTAL = 11.0592 MHz. What value do you need to load into the timer's register if we want to have a time delay of 5ms? Write a program for timer 0 to create a pulse width of 5ms on P2.3

Ans: Delay = 5 ms = 5×10^{-3} s

Delay = count x Time period

5×10^{-3} = count x 1.085 μs

$$\text{count} = \frac{5 \times 10^{-3}}{1.085\ \mu s} = 4608.29 \approx 4608 \text{ counts}$$

To find the value to load onto the register:

FFFF - x + 1 = 4608

65535 - x + 1 = 4608

$x = (60,928)_{10} = EE00\ H$

$\Rightarrow$ TL0 = 00 H

TH0 = EE H

Program:

```
      CLR   P2.3
      ACALL DELAY
      SETB  P2.3
DELAY : MOV TMOD, #01
      MOV   TL0, #00H
      MOV   TH0, #0EEH
Wait: JNB   TF0, wait
      CLR   TR0
      CLR   TF0
```

} the only diff. from the sq. program is that you don't have to reset the counter P2.3 call a delay a second time to generate high & lows

(8) Assume that XTAL = 11.0592 Hz. write a program (13)

to generate a square wave of 2kHz frequency on pin

P1.5

① Calculate Time period of half the wave

Ans: Frequency = 2kHz = 2×10^3 Hz

$$T = \frac{1}{f} = \frac{1}{2 \times 10^3} = 0.5 \times 10^{-3}$$

$$= 500 \mu s$$

Total time period = 500μs

Delays must be generated for each half

$\Rightarrow D_{1/2} = 250 \mu s$

② Calculate the count value

Delay = count x Time period

$$\text{count} = \frac{D_{1/2}}{1.085 \mu s} = \frac{250 \mu s}{1.085 \mu s} = 230$$

③ Find the values to load onto the TL0 & TH0 registers

FFFF - x + 1

= FFFF - 230 + 1

= $(65\ 306)_{10}$

= FF1A H

$\Rightarrow$ TL0 = 1A
TH0 = FF

④ Write the program

```
main: CLR   P1.5
      ACALL DELAY
      SETB  P1.5
      ACALL DELAY
      SJMP  main

DELAY: MOV TMOD, #01H
       MOV TL0, #1AH
       MOV TH0, #0FFH
wait:  JNB TF0, wait
       CLR TR0
       CLR TF0
       RET.
```

⑨ Assume that XTAL = 11.0592 kHz. Write a program to generate a square wave of 50kHz frequency on P2.3

① Find Time (Delay) period

$T = 1/f = \frac{1}{50 \times 10^3} = 0.2 \times 10^{-3}$

$= 2 \times 10^{-5}$

$= 2 \times 10^{-3} \times 10^{-2} = 20\mu s$

② Find D1/2

$D1/2 = 20\mu s \div 2 = 10\mu s$

③ Find the no. of counts

D_{y2} = counts x Time period

counts:- $\frac{10 \times 10^{-6}}{1.085 \times 10^{-6}}$

counts = 9

load FFFF - x + 1 = 9

65,527 counts = FFF7

write the same program as Q8

⑩ Examine the following program & find the time delay in seconds Exclude overhead due to instructions

```
        mov TMOD, #10H
        mov R3, #200
again :  mov TL1, #0C1H
        mov TH1, #0H
        SETB TR1
back :   JNB TF1, back
        CLR TR1
        CLR TF1
        DJNZ R3, again
```

Ans: no. of counts : FFFF - 0108 +1

= FEF8

= 65272

Delay = count x Time period

= 65272 x 1.085 μs = 70.820ms

For 200 of them,

$= 200 \times 70.820\,ms$

$= 14164$ seconds

Mode 2 programming Qs

(11) Assume XTAL = 11.0592 MHz, find the frequency & the square wave generated on P1.0 in the following program

```
        mov TMOD, #20H
        mov TH1, #5
        SETB TR1
back :  JNB TF1, back
        CPL P1.0
        CLR TF1
        SJM back
```

Ans:- (i) Find the count value.

$\Rightarrow FF - x + 1 = 05\,H$

$\Rightarrow$ 251 counts

(ii) Find delayed 1/2 wave

Delay = counts x Time period

Delay = $251 \times 1.085\,\mu s$

$= 272.33\,\mu s$

(iii) Find T of whole wave

$T = D_{1/2} \times 2 = 544.67\,\mu s$

(iv) Find frequency $= 1/T = 1.83597$ kHz

(12) Find the frequency of a square wave generated on P1.0 (17)

```
    MOV  TMOD, #21H
    MOV  TH0, #0
again: MOV  R5, #250
       ACALL DELAY
       CPL  P1.0
       SJMP AGAIN
DELAY: SETB TR0
BACK:  JNB TF0, BACK
       CLR  TR0
       CLR  TF0
       DJNZ R5, delay
       RET
```

Ans: (i) Find count value

count = x

FF - x + 1 = 0H

x = 256

(ii) Find delay for ~~full~~ half wave

$T_{1/2}$ = count x Time period

= 256 x 1.085 μs

= 277.76 μs

(iii) Find delay for full wave

D = 277.76μs x 2

(v) Find for 250 times

T = 2 x 277.76μs x 250 = 138.88 ms

F = 7.2 Hz

(13) Assume that the timers are programmed in mode 2, the value in hex loaded into TH in each of the following cases again:

(a) mov, TH1, #-200

=38 H

(merely convert to hex on calculator and take last 2 digits

(b) mov TH1, #-60

= C4

(c) mov TH1, #-3

= FD H

(d) mov TH1, #-12

= F4 H

e) mov TH0, #-48

= D0 H

Questions on Counter Programming

★★★

(14) Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2, which connects to 8 LEDs

Ans:

```
// initialize mode
mov TMOD, #60H
mov TH1, #0
```

set C

00
01

0 1 10 0000

↓ mode 2

for Timer 1 Counter 0 - not relevant here

```
        // initialize an input port
        SETB  P3.5

again:  // start counter
        SET   TR1

back:   MOV   A, TL1      // get copy of TL
        MOV   P2, A       // show o/p on port 2
        JNB   TF1, back
        CLR   TR1
        CLR   TF1         } clean up
        SJMP  again       // go back to start
```

Additional Timer Programs.

(15) Write a program to generate a delay of 20 µs using Timer 0 of 8051. After the delay, send a 1 through Port 3.1

Ans Delay = 20 µs

Delay = counts × Time period

$$\text{counts} = \frac{20\ \mu s}{1.085\ \mu s}$$

≈ 18 counts

Register value:

FFFF − 18 + 1 = (65518)d = FFEE

*Serial Port

Program

```
        MOV   TMOD , #01 H
        MOV   TL0 , #0EE H
        MOV   TLH , # 0FFH
        MOV   TCON , #10 H      = equivalent to  SETB  TR0
WAIT :  JNB   TF0  wait
        SETB  P3.1
        CLR   TR0
        CLR   TF0
here :  SJMP here
```


D = Timer should stop to start

* Serial Ports

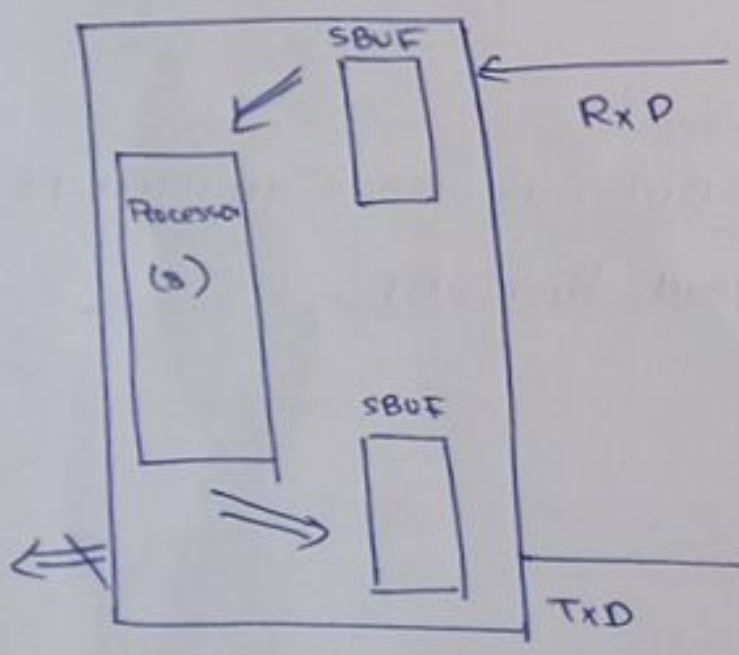
→ 8051 has a high speed, full duplex, software programmable serial port

→ Data is received serially through the RxD line, and transmitted through the TxD line

→ The SCON SFR mainly controls serial communication

→ The SMOD bit in the PCON SFR controls the baud rate

* Basic Transmission and Reception of Data



→ Generally, serial communication involves transmitting data bit by bit.

→ Hence, the transmission is very slow

→ 8051 has an 8-bit processor.

→ It uses SBUF, where all the 8 bit data is pushed into it.

Transmitting Data

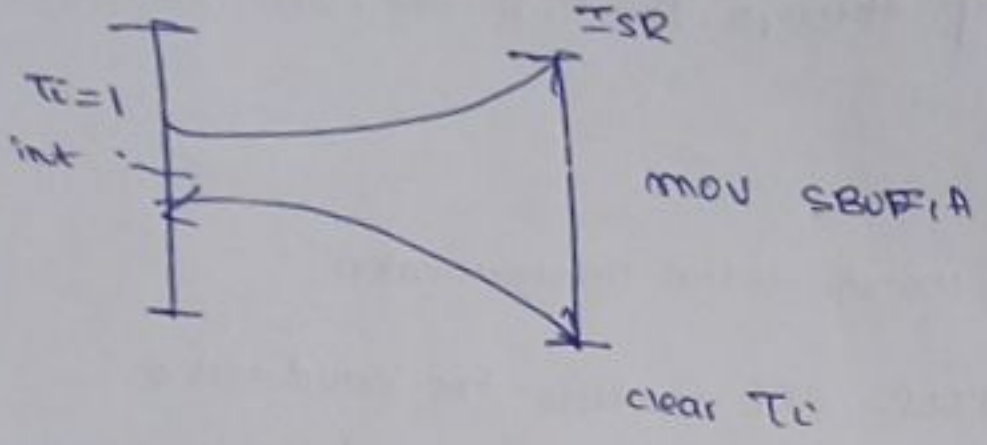
→ Once all the data is transferred to SBUF, the processor is no longer involved in the transmission of data.

→ Rather, it becomes SBUF's responsibility to transmit bit after bit, while the processor remains free to carry out other operations.

→ After SBUF transmits all the 8 bits, it sends an interrupt to the processor by making the TI bit 1.

→ The processor moves into the ISR routine, if needed, new data can be moved into SBUF.

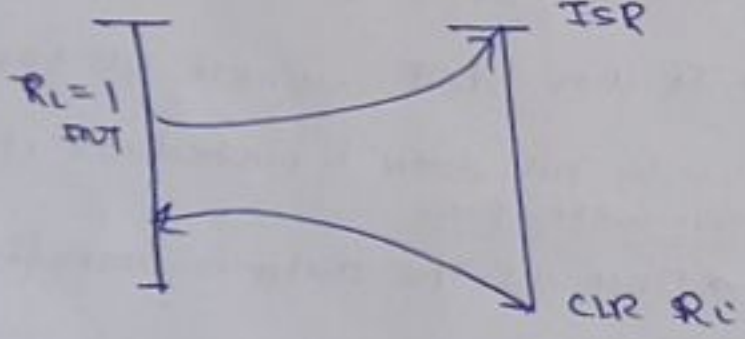
→ At the end of the ISR, set the Ti flag to 0. and return



Receiving Data

→ Data is received bit by bit (LSB) first into SBUF. This requires 8 cycles again.

→ Once all the data is received by SBUF, it sends an interrupt, making Ri = 1. . In the ISR, read from SBF.



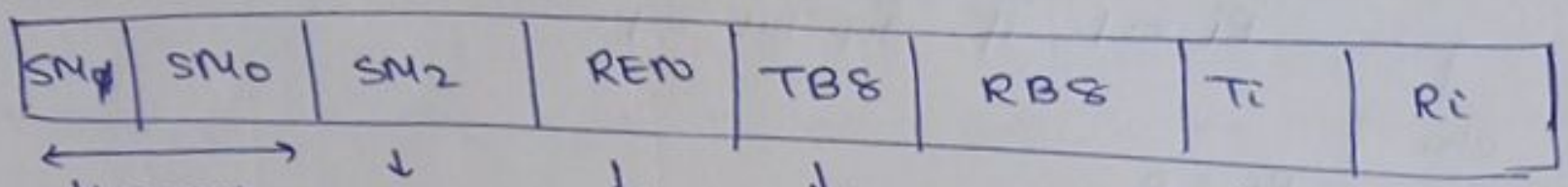
→ Before returning from the ISR, make Ri=0.

→ During reception, after receiving the data, but before giving it to the processor, SBUF would perform error checking.

→ If there is an error, Ri will remain zero. It will be discarded, by the next set of data overwriting it.

* SCON - Serial Control (SFR)

→ SCON is bit addressable.



SM0, SM1 — 4 serial modes
- 00 — mode 0
- 01 — mode 1
- 10 — mode 2
- 11 — mode 3

SM2 — a bit by which programmer decides to perform error checking
- if set to 1, only give data to processor if the data is valid.
- if set to 0, RI will automatically be 1 (that is, automatically call the interrupt)

REN — receiver enable (allows data to enter SBUF)
- 1 = ✓
- 0 = ✗

TB8 — 9th bit to be transmitted (m2,3)

RB8 — 9th bit to be received (m1,2,3)

TI — set to 1 when complete data has been transmitted

RI — set to 1 when the complete data is received

both have to be cleared by programmer after the ISR.

* Concept of Error Checking using SM2 and Mode Structures

→ In mode 0 - only 8 bit data is sent - called shift register.
- there is no error checking - SM2 has no use here

m0: |——— d(8) ———|

→ In mode 1 - use a start bit 0, data (8 bits), and an end bit (1)

(called the 8 bit UART)
↙
universal asynchronous receiver transmitter.

M1: |— 0 —|— d(8) —|— 1 —|

→ Note that the start & stop bit are system generated.

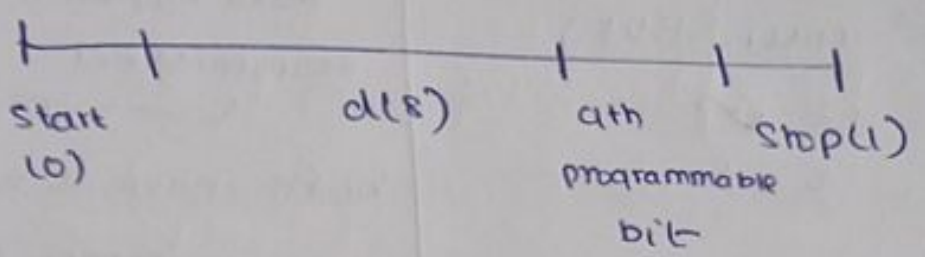
→ Error checking is done on the 9th bit - which is the stop

→ If stop is 1, then there is no error at all.

→ Thus,

```
if (SM#2 =1) {
        Ri ← 1  if stop bit =1
}
else {
        Ri =0
}
```

→ In mode 2,3 (called 9-bit UART) - give a start bit (0), data (8), 9th programmable bit, end bit (1)



→ Error checking is again done on the 9th bit - here it is the 9th programmable bit.

→ The 9th bit can be 0 or 1, since it is programmable. It is valid if the bit is set to 1.

→ Thus in mode 2,3

```
if (SM2 =1) {
        Ri ←  if 9th programmable bit = 1
}
else {
        Ri ← 0
}
```

* Usage of Mode 2, 3 – the 9th programmable bit

→ The 9th bit in 9-bit UART is programmable. It is considered to be valid only if set to 1.

→ However, it is set to 0, when data is not meant to be received. Specifically, this is for cases where are multiple recipients and data is meant only for a specific few ones.

* Multiprocessor Communication in Mode 2 and 3

→ Multiprocessor communication can be achieved using the 9th programmable bit, allowing for broadcast / selective transmission.

→ If the recipients have the SM2 bit set to 1, then checking will happen before receiving data – otherwise, if the SM2 bit is 0, no checking happens.

Broadcast Transmission

→ The sender sets the 9th bit to 1.

→ On the receiver's side, where SM2=0 – no checking – data is received

where SM2=1 → checking happens – has received a valid bit 1 from the sender – allows message through to recipients

→ Thus, all stations receive the message.

Selective Transmission

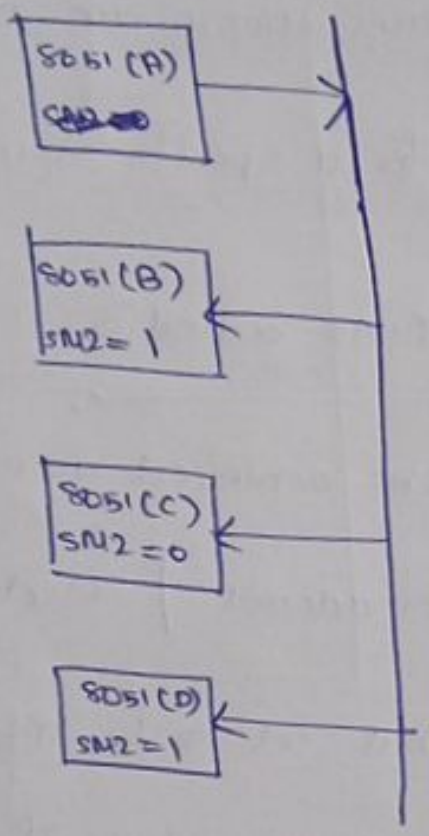
→ The sender sends the 9th bit to 0.

→ On the receiver's side, where the SM2=0, no checking. Synch

the data is received.

→ Where SM2=1, the receiver gets an invalid bit - does not let data through.

Example



→ For broadcast - A sets SM2 =1

→ For selective transmission - only C receives data, as it does no checking, B & D receive (0) - an invalid 9th bit ⇒ they block the data

* Terminology

A. Parallel and Serial Communication

Parallel - multiple lines used to transfer data to a device that is only a few feet away

Serial - transfer data to farther away locations
- data is sent one bit at a time.

B. Synchronous and Asynchronous Communication

Synchronous - transfer a block of data at a time
- sender and receiver operate on a common clock

Asynchronous - transfer a single byte at a time
- sender and receiver operate on a different clock though on the same frequency.

C. Dupler, Half-Duplex and Simplex Transmission

Dupler - data can be transmitted or received at the same time

Half-dupler - data can be transmitted one way at a time

Simplex - one way transmission

D. Baud Rate

- the rate of data transfer
- defined as the number of signal changes per second

* Modes of Operation

Model - 0 - d - 1 = 8-bit standard UART
start stop

→ a 10-bit full duplex mode

Transmission

→ TxD sends data. as start - d - stop

→ Ti flag is set after all 10 bits are received.

Reception → Data is received in the same order as it is transmitted.

→ During reception, the start bit is discarded, 8 data bits are received in SBUF, and stop bit is stored in RB8.

→ R_I is set only if SM2 = 0 (no checking) or
If RB8 = 1 (signifies that data is fully transmitted, line goes back to idle (high)).

→ Once R_I is set, the program is interrupted to receive the data.

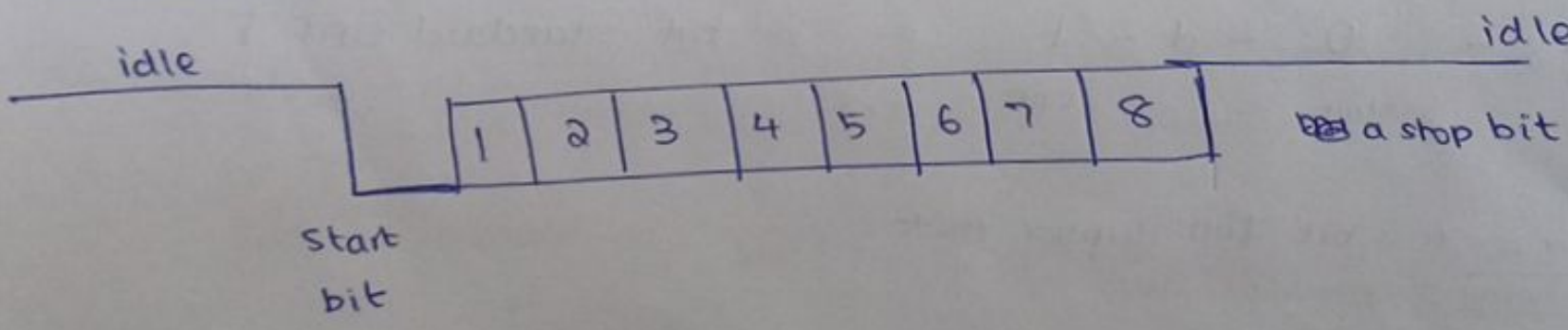
Baud Rate : variable baud rate - can be controlled by changing the overflow rate using different counts

$$f_{BAUD} = \frac{2^{SMOD}}{32} \times \text{Timer 1 overflow frequency}$$

(double baud rate by setting SMOD = 1)

(SMOD is from the PCON register)

Diagram



Mode2 – 9 bit UART – multiprocessor mode

→ similar to mode1, but there are a total of 11 bits

Format – Start (0) – data bits (8) – programmable 9th bit – Stop (0/1)

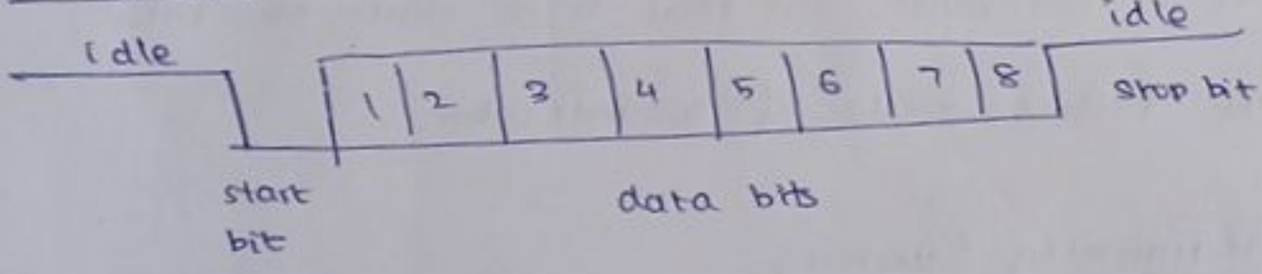
Transmission – transmit start + data bits
store 9th bit in TB8 of SCON

Reception – 8 data bits received in SBUF
9th bit in RB8 of SCON
start and stop bits are discarded

Baud Rate – fixed

$$f_{Baud} = \frac{2^{SMOD}}{64} \times \text{oscillator frequency}$$

Diagram



Mode3 – 9 bit UART – multiprocessor mode

– same as mode2, but has a variable baud rate

calculated as $f_{baud} = \frac{2^{SMOD}}{32} \times$ Timer1 overflow frequency

Mode 0 - Shift Register.

→ only communicates in one direction - simplex

→ sender sends data on TxD, receiver gets on RxD

→ TxD carries the clock - on every clock pulse - bit is shifted out of RxD from sender, and shifts into the receiver.

→ Since TxD carries the clock for both sender & receiver

⇒ synchronous communication

→ The baud rate is fixed = clock frequency

$$= \frac{f_{osc}}{12}$$

→ cost is high, since there are exclusive lines for data and clock.

→ However, it is beneficial since there is no start - stop bit. - hence it is the fastest mode of communication

* Serial Programming Questions.

Derivation of Counts for standard baud rates

$$\text{baud rate} = \frac{2^{SMOD}}{32} \times T_1 \text{ overflow rate}$$

assume that SMOD = 0

T1 is used as a timer.

$$\text{counting rate} = \frac{f_{osc}}{12}$$

$$f_{osc} = 11.0592\ \text{MHz}$$

For a max baud rate, T1 must do 1 count

$$BR_{max} = \frac{1}{32} \times \frac{f_{osc}}{12} \times 1$$

$$= \frac{1}{32} \times \frac{11.0592 \times 10^6}{12}$$

$$= \underline{\underline{28800}}$$

Baud Rate	Count	Count to be Loaded
28800	1	FF
14400	2	FE
9600	3	FD
4800	6	FA
2400	12	F4
1200	24	E8

(1) Write a program to transmit 45H at a baud rate 9600.

Ans: For model: $BR = \frac{2^{SMOD}}{32} \times$ Timer 1 overflow rate

if SMOD = 0

$BR = \frac{1}{32} \times$ Timer 1 overflow rate

for max baud. rate: $BR_{max} = \frac{1}{32} \times \frac{f_{osc}}{12} \times 1$

$= 28800$

If the timer counts 3 pulses: $BR = \frac{28800}{3} = 9600$ which is the desired baud rate

count : $FF - x + 1 = 3$

$x = FD$

Program

```
main:   MOV  A, #45H
        ACALL Transfer
here :  SJMP here

transfer: MOV  SCON, #40H   //init mode 1
                            choose mode of
                            serial communicat
          MOV  TMOD, #20H   // select Timer 1.
```

```
        MOV TL1, #0FDH
        MOV TH1, #0FDH
        SETB TCON R1
        MOV SBUF, A
Wait:   JNB TI, wait
        CLR TR1
        CLR TF1
        CLR TI
        RET
```

} move baud count value.

(2) Write a program to transfer 35H serially at a baud rate of 9600 using 8051.

Ans: Like Q1 — BR = 9600

⇒ count = 3

⇒ value in register = 0FD H

Program

```
main:   MOV A, #35H
        ACALL Transfer
here:   SJMP here
transfer: MOV SCON, #40H        mode 1 serial comm
          MOV TMOD, #20H        select timer1
```

```
        mov TL1 , #0FDH
        mov TH1 , #0FDH
        SETB TCON R1
        mov SBUF , A
Wait :  JNB Ti , wait
        CLR TR1
        CLR TF1
        CLR Ti
        RET.
```

(3) Transfer the word 'SUCCESS' at 2400 baud

```
mov     A, #"S"
  ACALL Transfer
mov A, #"U"
ACALL Transfer
mov A, #"C"
ACALL Transfer
mov A, #"C"
ACALL Transfer
mov A, #"E"
ACALL Transfer
mov A, #"S"
ACALL Transfer
```

```
    MOV  A, #"S"
    ACALL Transfer
```

if continuously, enable REN ⇒ #50H

```
Transfer : MOV SCON, #40H      Mode 1 serial comm
           MOV TMOD, #20H      Timer 1 mode selection
           MOV TL1, #0F4H
           MOV TH1, #0F4H
           SETB  TCON R1
           MOV  SBUF, A
wait  :    JNB  Ti, wait
           CLR TR1
           CLR TF1
           CLR Ti
           RET
```

(4) Write a program to read 10 bytes from port 1 and send each byte serially at 2400 baud.

```
MOV R0 #0AH      → counter is 10
MOV 90H, #0FFH   → port 1 as input
```

```
Loop:  MOV A, 90H
       ACALL Transfer
       DJNZ R0, Loop
here:  SJMP here
```

write the same transfer block again

(5) Assume that XTAL = 11.0592 Hz. for the following prog

State (a) what the program does

(b) compute the frequency used

(c) compute Baud rate.

```
MOV   TMOD, #20H
MOV   PCON, A        SMOD = 1
MOV   TH1, -3

MOV   SCON, #50H

SETB  TR1

MOV   A, #"B"

A1 :  CLR TI
      MOV SBUF, A

H1 :  JBT1, H1
      SJMP A1
```

Answer (i) The program continuously transmits "B"

since SCON has #50H (REN enabled)

(ii) Since SMOD = 1 → Frequency = 2x regular frequency

= 28800 x 2

| f = 57,600 Hz |

(iii) Baud rate = 57600 | 3 (since TH1 has -3 loaded

= 19,200 into it)

* 8051 Interrupts

→ 8051 has 5 interrupts. (all are vectored interrupts)

(i) 2 External Interrupts – INT0 & INT1 (at 0003H and 0013H)
– have 2 flags – IE1 and IE0 from TCON
– set to 1 if interrupt – auto-cleared.

(ii) 2 Internal Timer Interrupts – Timer 1 overflow interrupt (TF_1)
Timer 0 overflow interrupt (TF_0)

1 = occurred – auto-cleared

when the timer overflows – TFx is set – cleared once control goes the ISR

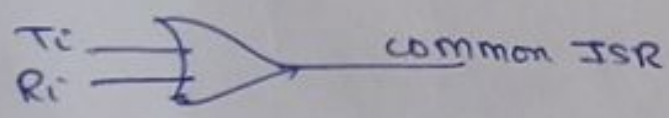
→ The addresses are 000BH and 001BH

(iii) Serial Port Interrupt – 2 interrupts, Ti & Ri

→ when the whole data has been transferred Ti becomes 1

→ when the whole data has been received Ri becomes 1.

→ whether Ti or Ri is 1 – one ISR is called, they are not cleared automatically, the programmer has to clear it.



→ address is 0023H

→ The reason Ti and Ri are not auto-cleared is because two different actions have to be taken for transmission & reception., and since they share an ISR.

→ A condition must be placed to check whether Ti is 1 or Ri is 1, they cannot be cleared before checking. Hence, it becomes the programmer's responsibility to clear the Ri / Ti

(Additionally, the RESET - BIOS program is called at 000 H (2) use a long jump to go to actual location of the program)

* Steps to Execute an ISR

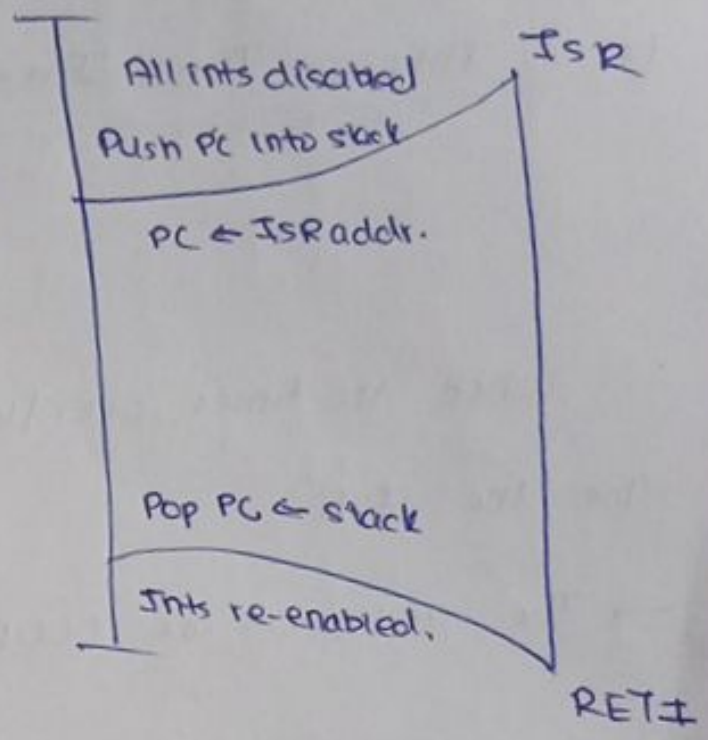
(i) All interrupts are disabled
Push PC into stack

(ii) Put ISR address into PC

(iii) Go to the ISR, execute

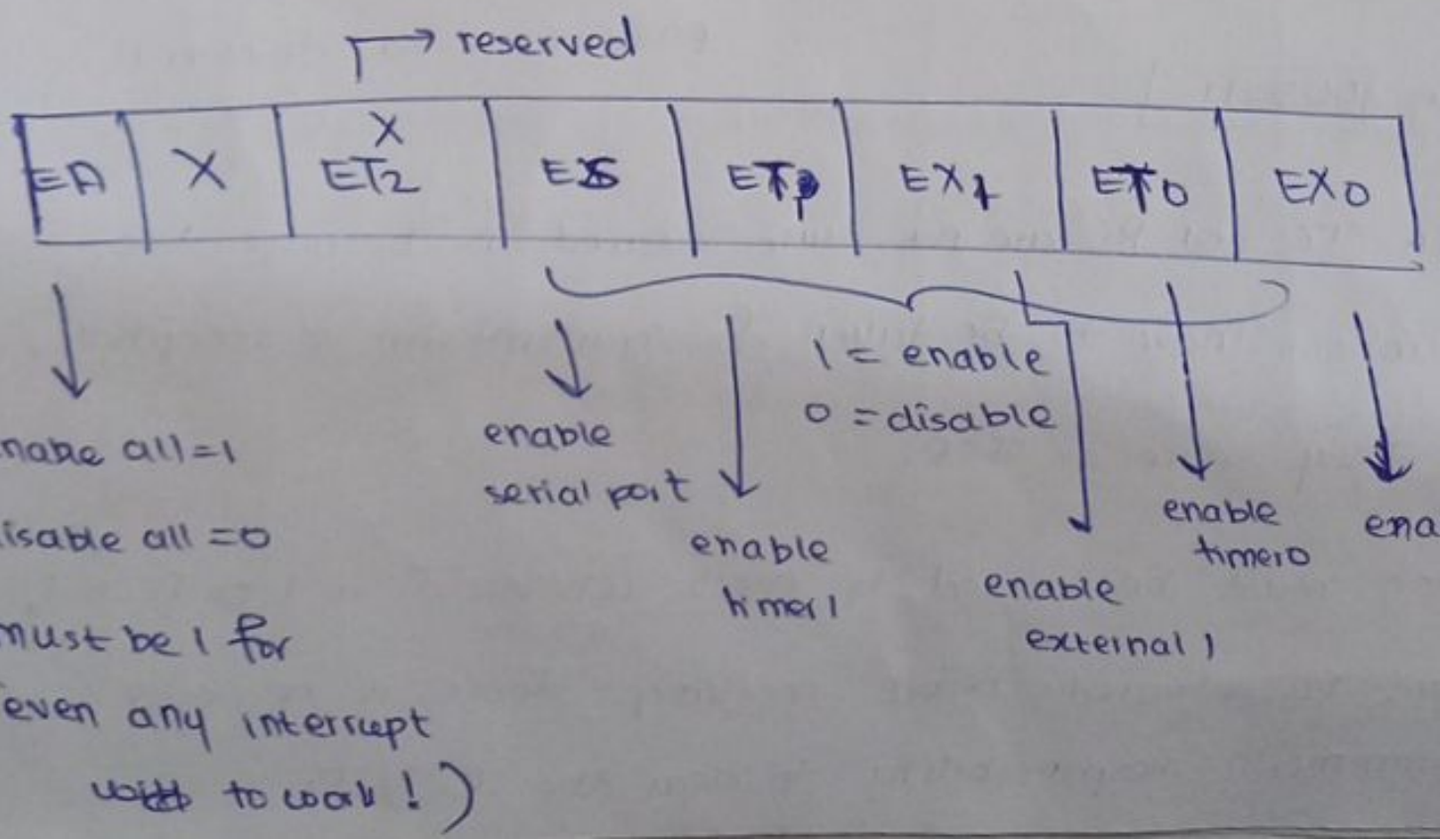
(iv) Pop return address from stack to go back to OG location.

All interrupts are re-enabled
(hence the RET-I)



* SFRs for Interrupts - IE and IP

① IE - Interrupt Enable - decides which interrupts are enabled or disabled.



② IP- Interrupt Priority

x	x	PT_2	PS	PT_1	PX_1	PT_0	PX_0
reserved							

1 => high priority

0 => low priority

→ A high priority interrupt can interrupt a low priority interrupt

→ If 2 or more interrupts at the same level occur simultaneously, it is a tie. Priorities are decided as

$\overline{INT0}$
TF0
$\overline{INT1}$
TF1
Serial (RI/TI)

* Interrupt Programming Questions

(1) Write instructions to:

(i) enable the serial, Timer 0 & external hardware interrupt 1

MOV IE, (10010110)B
96H

(ii) disable the timer 0 interrupt: CLR IE.1

(iii) disable all the interrupts: CLR IE.7

(8) ★★★★

(2) 8051 program for square wave using timer interrupt

Write a program to generate a square wave of 1kHz on P0.0 using timer interrupts. Also, keep reading data from port 1 & send it out on port 2.

Ans - $f = 1kHz$ $T = 1/10^3 = 10^{-3}\ s = 1ms$

Delay = 500 μs

⇒ 500 counts

Register count = FFFF - 500+1

= (65036)H

= (FE0C)H

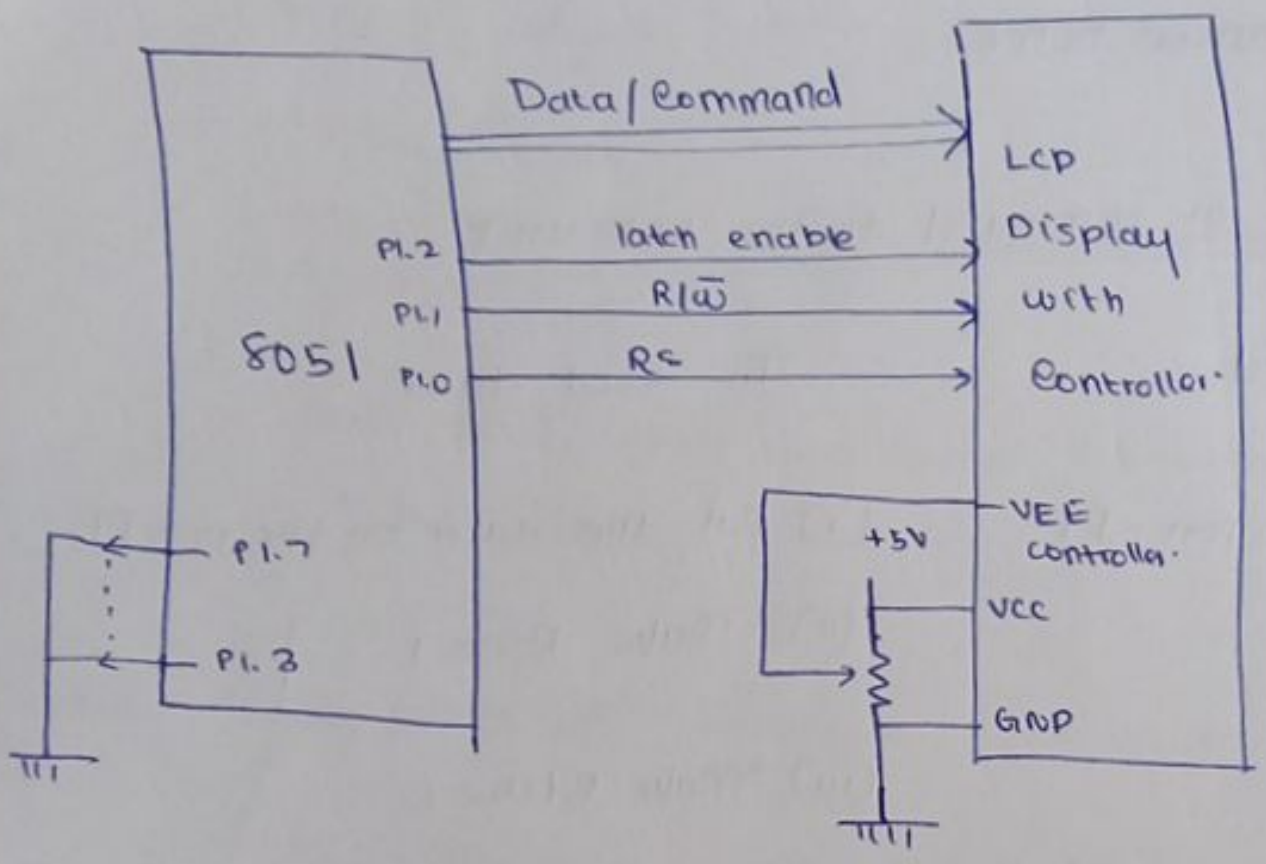
Program

```
        orig 0000H
        LJMP main
        orig 000BH
        LJMP TOISR
main:   MOV  P1,#0FFH   → make P1 an input port (by default ports are o/p ports)
        MOV  IE,#82H    → enable interrupts, and T0
        MOV  TMOD,#01H  → choose the mode of the timer
        MOV  TL0,#0CH   }
        MOV  TH0,#0FEH  } program register count
        SETB TR0        → start timer.
        CLR  P0.0
here:
        MOV  A,P1       }
        MOV  P2,A       } an infinite loop to get i/p from port 1 and output it at port 2
        SJMP here

TOISR:  CPL P0.0        ← complement P0.0 (to go low & high)
        CLR TR0
        MOV TH0,#0FEH   }
        MOV TL0,#0CH    } count
        SETB TR0
        RETI            ← an ISR - use RETI not RET
```

note that since we are going to the ISR TF0 gets auto-cleared

* LCD Interfacing

Interface Architecture



VEE Controller - brightness adjustment

RS = register select - to decide whether to send data or a command

RS = 0 ⇒ command

RS = 1 ⇒ data.

→ 14 bits in all - 8 bits for data - remaining pins for control

* Initialization of the LCD Display

38H - Set up display - as 2 lines 5x7 matrix display

0FH - display on
cursor on
cursor blinking

01 H - clear the display

06H - cursor increment mode

left to right

38
0F
01
06
80

80H - move cursor home

* Steps to send data or command to the LCD unit

To send command	To send data
(i) Put command on the port - P1	(i) Put the data on the port P1
(ii) Make RS = 0	(ii) Make RS = 1
(iii) Make R/W = 0	(iii) Make R/W = 0
(iv) Make Latch enable = 1	(iv) Make Latch enable = 1
(v) Make Latch enable = 0	(v) Make Latch enable = 0
(vi) Call a delay of 10ms	(vi) Call a delay of 10ms

* LCD Programming Questions

(1) Write a program to display "Hello World" on the LCD display.

Ans: mov DPTR, #0400H → initialize start of array

init: mov A, #38H → set up display.

ACALL cmd

```
mov  A , #0FH  → turn on the display

ACALL cmd

mov  A, #01H  →  clear the display

ACALL cmd
mov  A, #06 H  →  set cursor increment mode
ACALL
      cmd

mov  A, #80H  → move the curror home
 ACALL cmd


cmd :  mov  P2 , A  → move cmd to port

       CLR  P1.0  → make RS =0

       CLR  P1.1  → make R/W =0

       SETB P1.2
                    } set and reset latch
       CLR  P1.2

       ACALL delay  → call delay of 10ms

       RET

Display :  mov  R0, #00H  ←index of array

           mov  R7, #0BH  ← length of array=len(Hello
                                                world) =11

displayloop :  mov  A, R0

               movc  A, @DPTR +A

               ACALL DATa  ← call function to display data
```

```
        INC  R0          ← increment index

        DJNZ R7, display loop  → decrement counter and jump to
                                 the start of loop

here :  SJMP here.
```

Data : MOV P2, A

SETB P1.0 } enable RS to make it display character

CLR P1.1

SETB P1.2

CLR P1.2

ACALL delay

RET

} exactly the same as command, except u.o SETB RS P1.0 to set RS

* Keypad

(2) LCD Program to display while checking the busy flag (i.e display only one character at a time)

→ use the same program as before - modify the cmd function as follows:.

```
cmd :  MOV  P2, A
       SETB P1.7      ← set the busy flag
       CLR  P1.0
       CLR  P1.1
back:  SETB P1.2      } Latch enable 2 disable
       CLR  P1.2
       JB   P1.7 back  - if bit 7 (busy flag) is set, the LCD
       ACALL delay, RET   is busy and no. info should be sent to it
```

* Keyboard Interfacing with 8051

→ Keyboards are organized in a matrix of rows and columns.

→ The CPU accesses both rows and columns through ports

→ When a key is pressed, a row and column make contact- otherwise there is no connection between rows and columns.

Scanning and Identifying the Key

→ A 4x4 matrix keyboard is connected to 2 ports

→ The rows are connected to an output port and the columns are connected to an input port.

Steps to Detect a Key Press and Identify a Key

1. The microcontroller scans the keyboard continuously to detect and identify the key pressed

2. To detect a pressed key - ground all the rows by providing a 0 to the output latch. If the data read from the columns is 1111, it means that no key press has occurred.

3. However, if there is a 0, it means that a key has been pressed, and the microcontroller goes through the process of identifying the key.

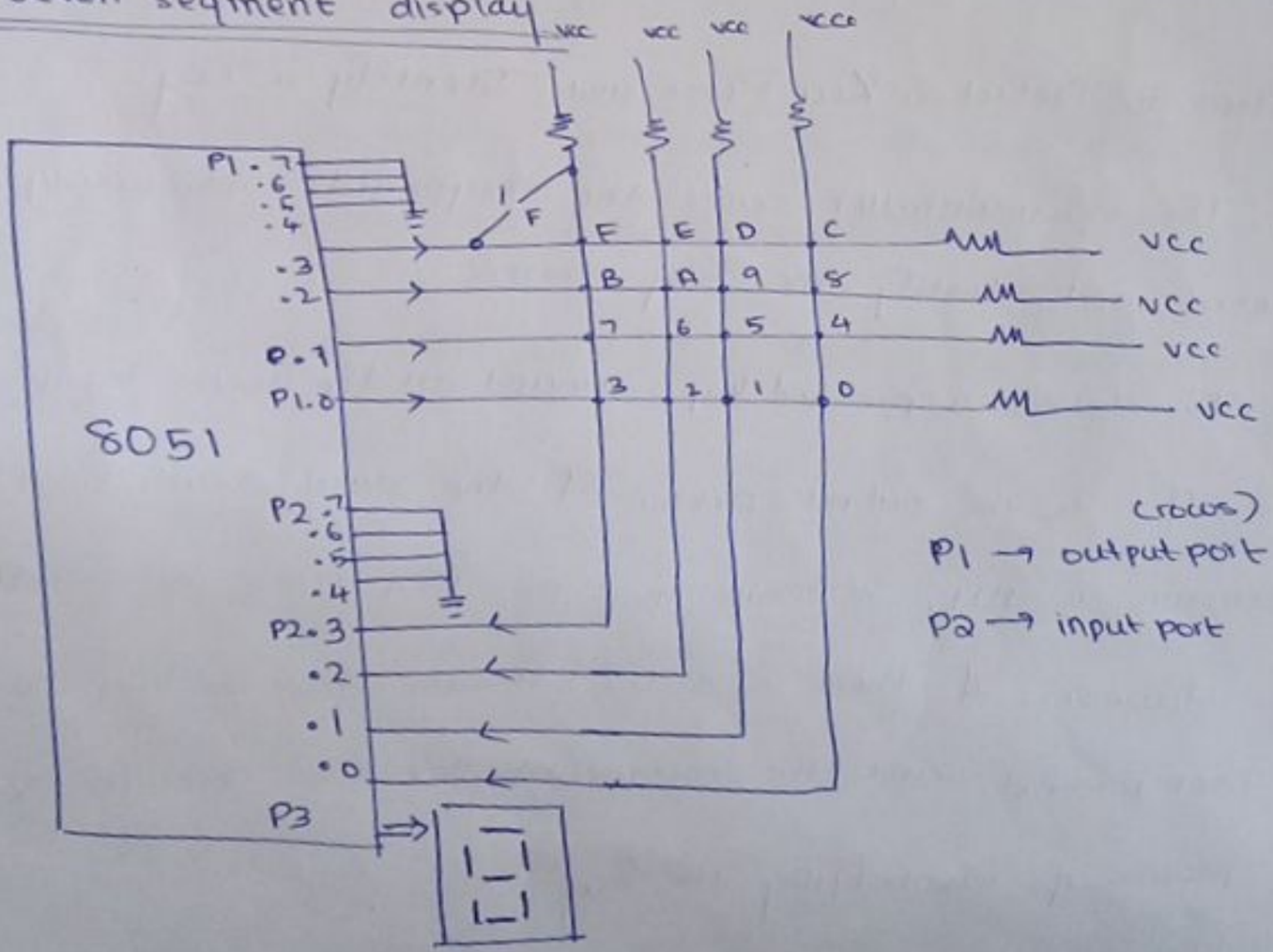
4. To detect which row the key press belongs to, it grounds one row at a time, and reads the column one by one.

5. If all the column values are high, it means that the key press has not occurred in that row.

6. It continues to ground one row after the other.

7. To find the key press, rotate the column bits of the chosen row one by one, into the carry flag, and checks if it is zero - if so - get the ASCII value from the lookup table.

* (ex) Program to Detect a Key Press and Display it on a seven segment display



Program

```
MOV   DPTR, #0400H      → start of lookup table
MOV   P2, #0FFH         → explicitly set P2 as an input port
```

// phase1 - keep checking if a key is pressed

```
check:  MOV  P1, #00H       → make all the rows 0
        MOV  A, P2          → read input into A, if any
        CJNE A, #0FH,       → if there is no key press it
             pressed          would be 0000 1111 (0F)
        SJMP check          → otherwise a key has been pressed
                              infinite loop to keep checking
```

// phase 2 - go row by row to identify row of key press

```
pressed:  MOV  R0, #00H     →load smallest value of 1st row
          MOV  P1, #0E      → set o/p port to choose row1
                              = 0000 1110 = 0EH
          MOV  A, P2
          CJNE A, #0FH, colcheck  → check if all bits are1,
                                    if not go to colcheck

          MOV  R0, #04 H
          MOV  P1, #0DH     → row 2 grounding
                              0000 1101 = 0D
          MOV  A, P2
          CJNE A, #0FH, colcheck
```

```
mov  R0, #08H
mov  P1, #0BH
mov  A, P2
CJNE  A, #0FH, colcheck
```

row 3 check
0000 1011

```
mov  R0, #0CH
mov  P1, #07H
mov  A, P2
```

row 4 check
0000 0111

// phase 3 - identify the column for the row chosen

```
colcheck :  RRC  A
            JNC  Display
            INC  R0
            SJMP colcheck
```

→ keep moving bits into carryflag

→ if there is a 0 ⇒ that key is pressed, move to display section

→ R0 holds base value for each row, keep incrementing it

↳ loop until a zero is found

// phase 4 - display section

```
display:
  mov  A, R0
  mov  A, @DPTR +A
  mov  P3, A
  SJMP check
```

→ move value of row pointer into A

→ move to that location in array

→ show o/p through port 3

→ go back to checking if a key is pressed.

*ADC - Analog to Digital Conversion

ADC 0809

→ an 8-channel, 8 bit ADC.

→ converts analog voltage input into an 8 bit digital data output

To use the ADC:

→ Use one of the select lines (A, B, C) to choose one out of 8 options

→ Put the channel no. on these lines and latch using ALE.

→ Give SOC - indicating the start of conversion (start of conversion)

→ The channel voltage is internally sampled and held into a capacitor.

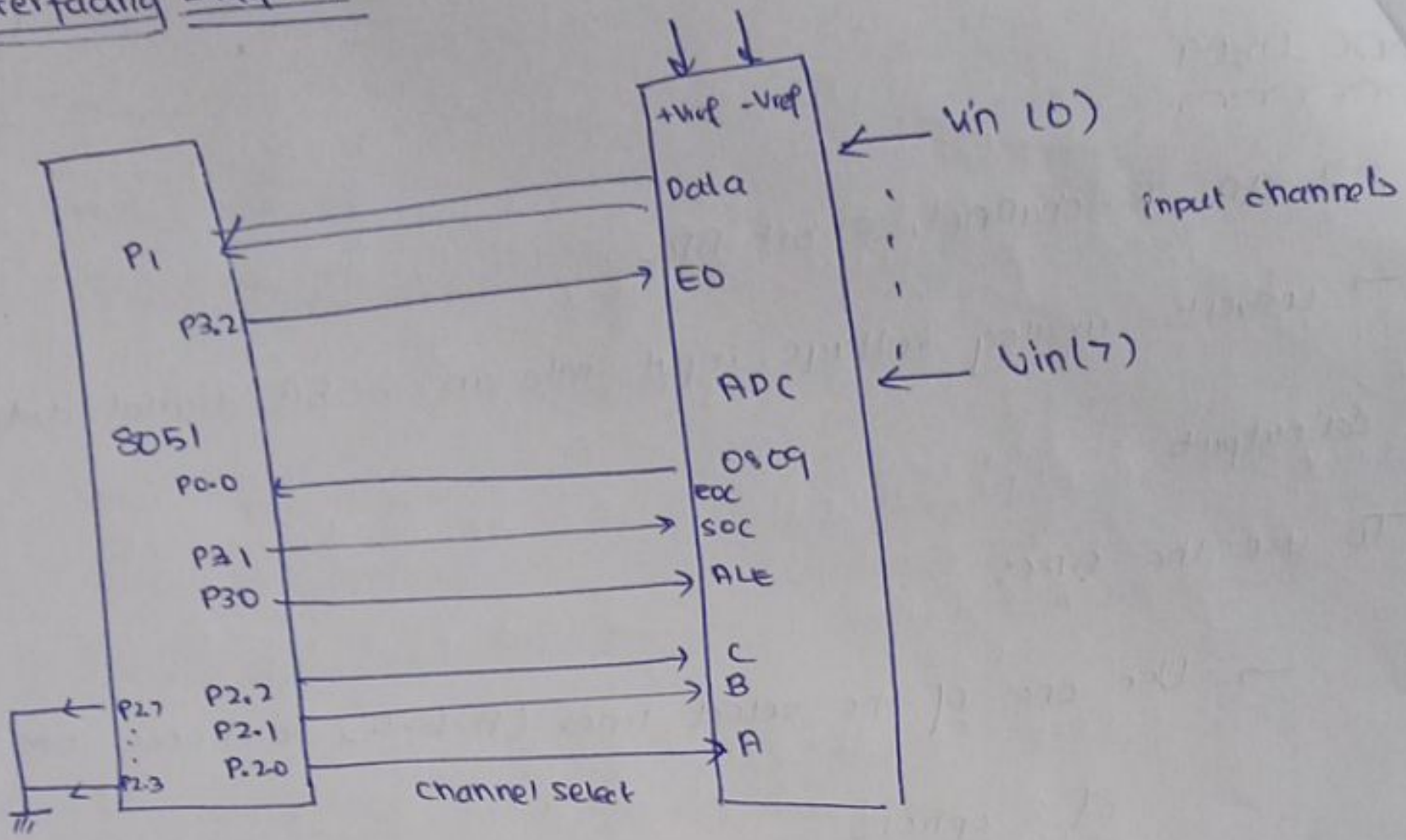
→ Conversion takes place internally using the Successive Approximations Algorithm.

→ Reference voltage for conversion is provided with +Vref & -Vref.

→ End of conversion indicated by the EOC signal

→ Give the OE signal to enable 8-bit data output from ADC to a port of 8051.

Interfacing Diagram



Program: Write a program to read 10 samples from channel 3 and store them from internal RAM location 30H

// Phase1 - Initialization

```
mov  R7, #0AH     ⟶ 10 sample count
mov  R0, #30H     ⟶ start of array

mov  P0, #0FFH    } set P0 & P1 as input ports
mov  P1, #0FFH
```

// Phase2 - Steps to carry out ADC

```
back: CLR  P3.0   } Make sure all of the ports are deactivated
      CLR  P3.1
      CLR  P3.2

      mov  P2, #03H   ⟶ choose channel
```

```
SETB  P3.0
CLR   P3.0      } Latching
SETB  P3.1      ⟶ SOC
```

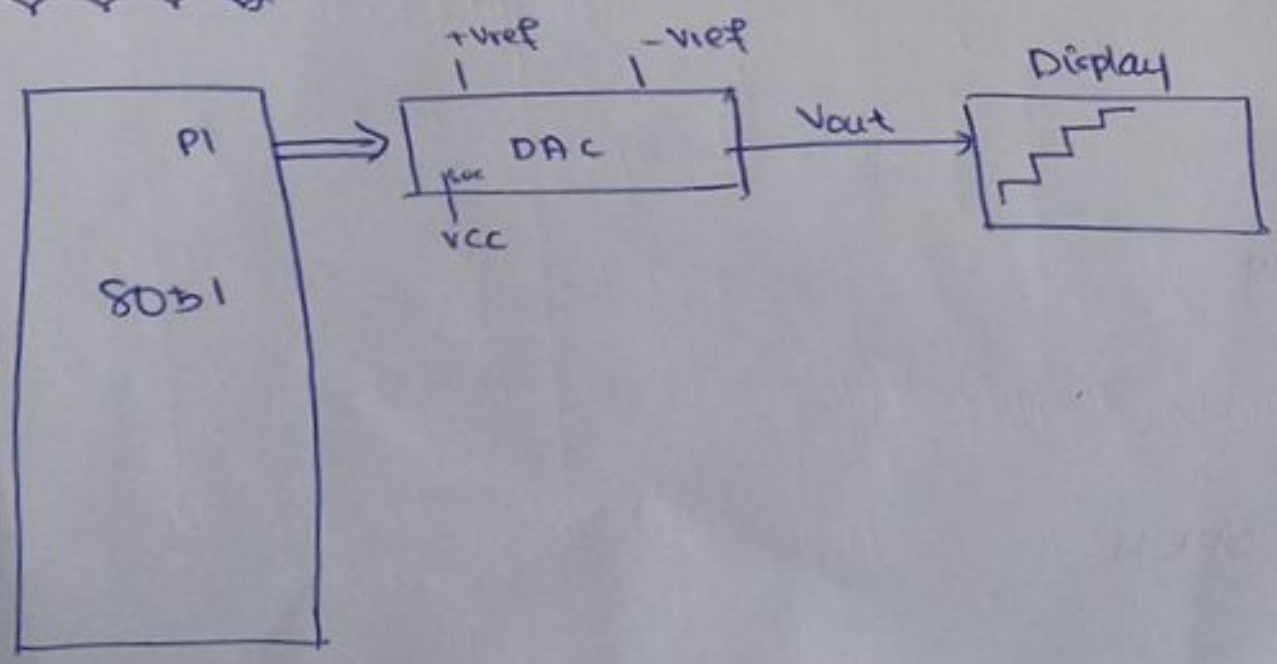
```
conversion wait : JNB  P0.0   conversion wait

        SETB  P3.2   → ask for data with output enable
        MOV   A, P1
        MOV   @R0, A   } movement into array
        INC   R0
        DJNZ  R7, back

here    : SJMP here
```

*DAC – Digital to Analog

→ DAC is an 8 bit digital to analog converter

→ It can convert an 8 bit digital data input into an analog voltage output. The output is in current form.

Interfacing



Staircase Wave Generation

```
mov A, #00H
back : mov P1, A
       INC A
       ACALL DELAY
       JNZ back
here: SJMP here.
```

Ramp waveform - same as staircase - just use a much smaller delay

Sawtooth Waveform

```
mov A,#00H
back = mov A,A
       INC A
       ACALL DELAY
       SJMP back    (unconditional jump)
here : SJMP here.
```

Triangular Wave

```
    mov A, #00H
back1: mov P1,A
       ACALL delay
       INC A
       JNZ back1
       mov A, #0FEH
```

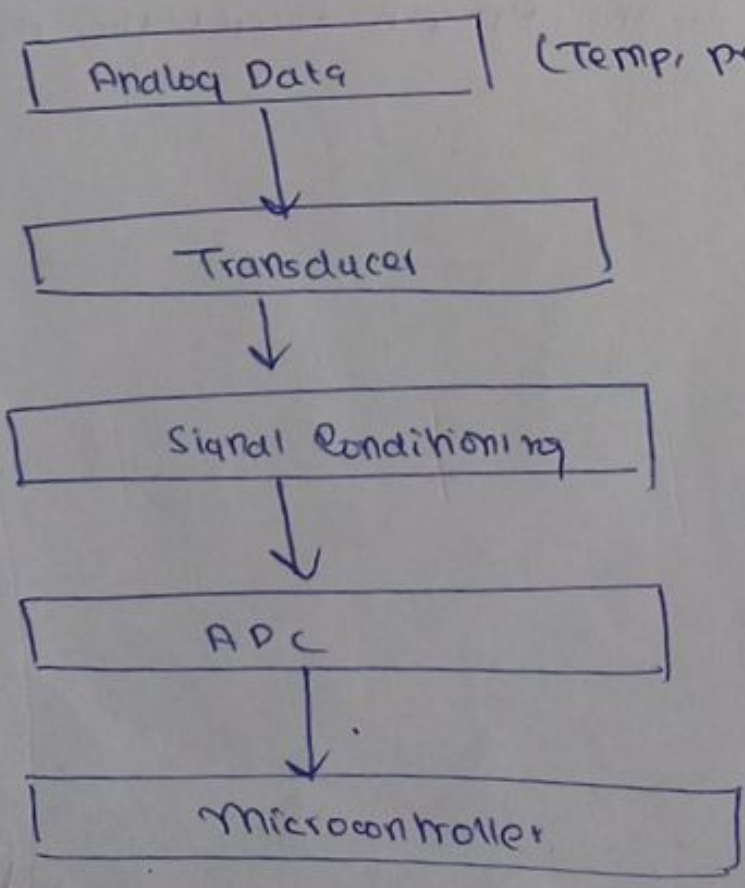
```
42:     MOV   P1 , A
        ACALL delay
        DEC   A
        JNZ   back 2
        SJMP  back 1
```

* Interfacing Temperature Sensor

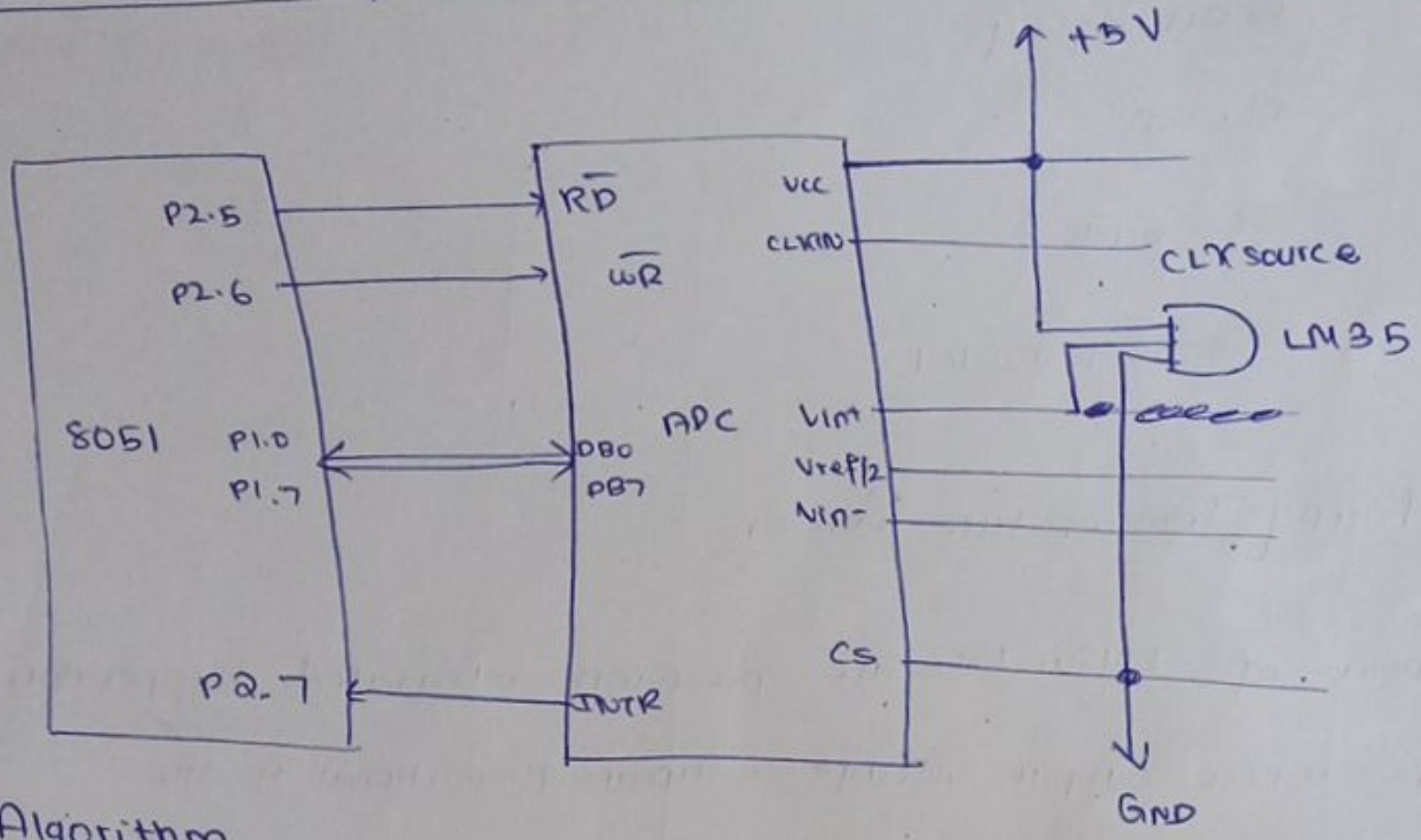
→ Sensors of LM34 /35 are precision- integrated temperature sensors whose output voltage is linear proportional to the Farenheit / Celsius temperature.

→ It outputs 10mV for each degree of F/C temperature.

→ Signal conditioning - conversion of the signal produced by tranducer to voltage - which is sent to an ADC.



LM35 Interfacing with 8051



Algorithm

→ Make CS = 0

→ Send low to high pulse to pin WR to start conversion

→ Keep monitoring the INTR pin - if low, the conversion is

→ finished, if high - keep polling until it goes low

→ Send high to low pulse to the RD pin to get the data out of the ADC