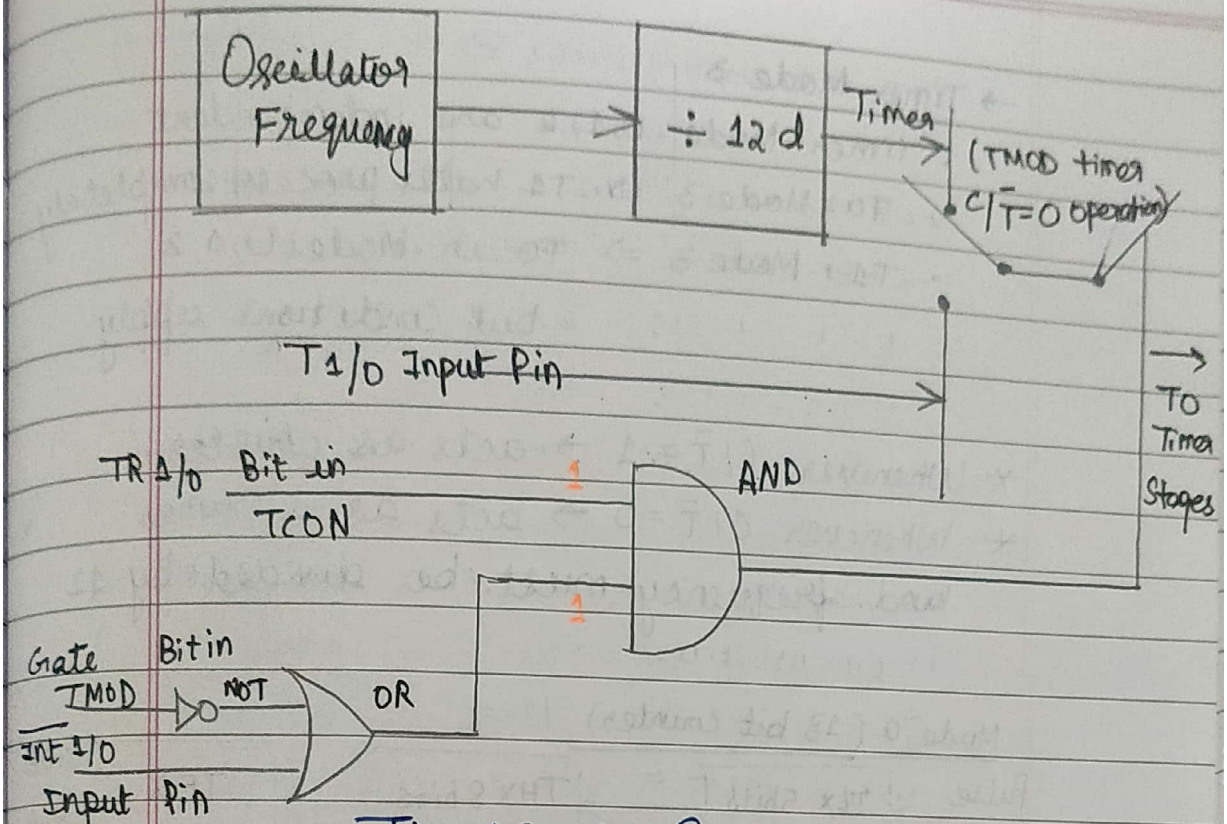


Timer/Counter Control Logic

→ Timer Mode 0 (5bits + 8bits)

- Lower bytes 8 bits for counting peerpose.
- Higher bytes 5 bits for counting peerpose.
- Total 13 bits for counting purpose.

→ Timer Mode 1 (8bits + 8 bits)

- Lower byte/nibble 8 bits for counting purpose.
- Higher byte/nibble 8 bits for counting purpose.
- Total 16 bits for counting purpose.

→ Timer Mode 2

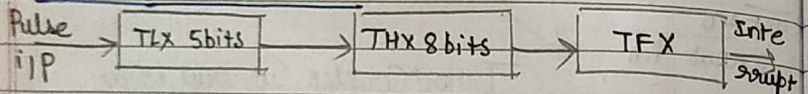
- Lower 8 bits used for counting 8 bits.
- Initialization value in lower byte register taken from higher register, auto reload.
- THO will be loaded to TLO, then counting starts.

→ Timer Mode 3

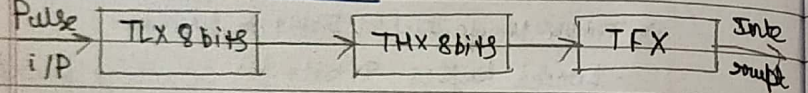
- Timer Mode 0, 1, 2 are independent.
- T0 = Mode 3 => T1 halts process completely
- T1 = Mode 3 => T0 in Mode 0, 1, 2, but conditions apply

* Whenever C/$\bar{T}$ = 1 → acts as counter
* Whenever C/$\bar{T}$ = 0 ⇒ acts as a timer and frequency must be divided by 12

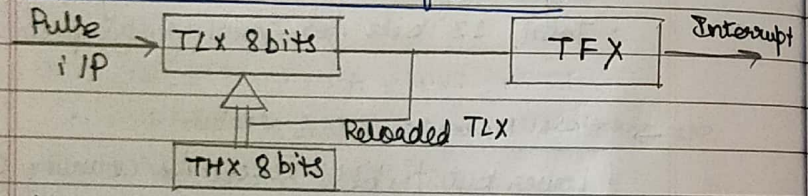
Mode 0 (13 bit counter):



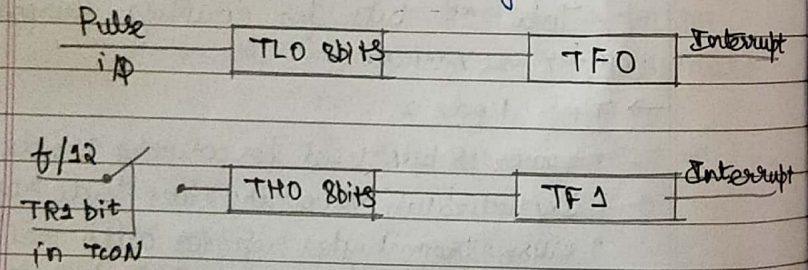
Mode 1 (16 bit counter):



Mode 2 (Auto Reloaded TL from TH):



Mode 3 (Two 8-bit timers using timer 0):



16-07-19

- Embedded Computing System: Any system that includes a programmable computer, but is not itself a general purpose computer, i.e., a hardware device into which CPU is inbuilt (hardware + software, processor is application specific but not general purpose).

- Complex Instruction Set Computer : CISC
- Reduced Instruction Set Computer : RISC

- More programming → advised to use RISC processor, instruction set is reduced, more efficiency.

- We use CISC processor in developing embedded systems because it is application specific, by running a simple program we can perform a specific task.

- Examples of Embedded Computing System
Cell Phone, Printer, Automobile (engine, brakes, dash, etc) Airplane (engine, flight controls, nav/comm), digital television, household appliances (washing machine, iron box, microwave, refrigerator, etc.)

- First Microcontroller : Intel 4004 → Calculator

- Types of microprocessors (general purpose devices):
i) Microcontrollers (general purpose)
ii) Digital Signal Processor - DSP

- Functional Requirements (the requirements which give some output)
Non Functional Requirements (to implement it, how much time is required, what processor is required, etc.), power and energy required, etc.

- Any kind of application can be developed using microprocessor since it is a general purpose processor. So, it is useful to developed embedded system.

- FPGA: Hardware circuit used to develop later

- The Physics of Software:
  - Executing software/embedded system uses power and time, and generates heat.
  - Computing is a physical act.

- Characterizing performance
  - The hardware or software used affects performance.
  - Eg: Split large program to smaller modules to execute processor faster.
  - Multi Processor Systems and Multi Tasking Systems.

- Stepwise Refinement
  - i) Analyze - Requirements Analysis
    ii) Refine - Functional and Non Functional Requirements

- Simple Requirements Form

- Specification: More specific description of the system

- Architecture: Client Server Architecture

- Components: Design hardware and software components

- System Integration

Adressing Modes

- Adressing Modes specify how the operands (values) will be provided to the system.
- It indicates the way in which the instructions are specified.
- They are of five types:
  1) Immediate 2) Register 3) Direct
  4) Register Indirect 5) Indexed
- 1) Immediate Addressing Mode
  - Immediate data is specified in the instruction itself.
  - '#' represents immediate addressing mode.
  - Eg:

```
MOV   A, #65H
MOV   A, #'A'
MOV   R6, #65H
MOV   DPTR, #2343H   [DPTR length = 16 bit Transfer 4 chars]
MOV   P1, #65H
```

- 2) Register Addressing Modes
  - Transfer from register to accumulator or from accumulator to register. Do not transfer from register to register, or from register (8 bit) to DPTR (16 bit) but allowed to DPTRH = DPH (8 bit) or DPTRL = DPL (8 bit)
  - E.g.:
    MOV R$_n$, A } n = 0, ..., 7
    ADD A, R$_n$ }
    MOV DPL, R$_n$ }
    ~~MOV DPTR, A~~
    ~~MOV Rm, Rn~~
    ↳ Don't use
- 3) Direct Addressing Mode
  - Possible to access RAM data from ~~00~~ 30 to 7F only (RAM from 00 to 7F, but below that registers are present) => byte addressing.
  - Although the entire of 128 bytes of RAM can be accessed using direct addressing mode, it is most often used to access RAM loc 30-7FH
  - Eg:
    MOV R0, 40H → address, not data
    MOV 56H, A → address, not data
    MOV A, 4 ; ≡ MOV A, R4
    MOV 6, 2 ; copy R2 to R6
    * MOV R6, R2 is invalid
- 4) Register Indirect Addressing Mode
  - Indirect means two step process
  - Eg. MOV A, @ R1
    At location R1 in RAM, data is present, we must move that to register A, move content of RAM loc where address is held by Ri TO A, (i=0 or 1)

- Eg. MOV @ R1, B
  At location R1 in RAM, move data from register B and store it in that location, in other words, content of register R0 or R1 is source or target in MOV, ADD and SUBB instructions.
  - we can use only R0 or R1 registers here.
- 5) Indexed Addressing Mode
  - Can access ROM data also along with this
  - Instead of MOV, we use MOVC, where 'C' represents code.
  - This mode is widely used in accessing data elements of look-up table entries located in the program (code) space ROM at the 8051.
  - Eg: MOVC A, @A+DPTR
    A = Content of address A + DPTR from ROM
  - Note: Because the data elements are stored in the program (code) space ROM of the 8051, it uses the instruction MOVC instead of MOV. The 'C' means code.

17-07-19

Serial Data Input Output

→ Serial Data Interrupts
→ Data Transmission
→ Data Reception
→ Serial Data Transmission Modes
  - Mode 0 - Shift Register Mode
  - Mode 1 - Standard UART
  - Mode 2 - Multiprocessor Mode
  - Mode 3

→ Introduction
- Generally, serial data is slow.
- At the starting of data transmission / receiving and at the ending of data transmission / receiving, there are interrupts for the processor to help in serial data transfer.

→ Special Purpose Buffer
- We have only one buffer 'sbuf' that works in mutual exclusive manner to transmit or receive the data.
- Memory → Sbuf → External Device
  External Device → Sbuf → Memory

→ Special Purpose Registers
- Pins P3.0 and P3.1 are used
- RXD = 1 → receive data
- TXD = 1 → transmit data
- Both must not be 1 at the same time
- SCON - Serial Port Control
  - Controls Data Communication
- PCON - Power Port Control
  - Controls Data rates

→ Serial Data Interrupts
i) Transmit Interrupt Flag (Ti)
ii) Receive Interrupt Flag (Ri)

- Initially, both flags are 0. Transmit interrupt flag is set to 1 when bit 1 time / reached in Mode0 or beginning of stop bits in other modes.
- This is done so that time is given for the processor till it reaches last bit to transmit data
- So, transmission interrupt flag Ti is set to 1 whenever transmission is completed.
- So, receive interrupt flag Ri is set to 1

- Ti is set at the starting of stop bit.
- Ri is set at halfway of stop bit.

→ Data Transmission
- Data that we want to transmit must be written into sbuf. Once sbuf is full, then data transmission takes flag. Ti is set at the end of transmission.
- Ti = 0 → Transmission is occuring, sbuf full, data transmission is taking place, don't write into
- Ti = 1 → sbuf is empty, transmission completed, can write into it

→ Data Reception
- For data reception, receive enable flag (REN) must be set to 1.
- Also, Ri with above conditions for receiving.

→ Baud Rate is more (greater (highest)) and fixed in Mode0, it is variable in all other three modes and also lower.

→ SCON

Mode 2 & 3

SM0	SM1	SM2	REN	TB8	RB8	Ti	Ri
7	6	5	4	3	2	1	0

RB8: Stop bit in mode 1, not used in mode 0

Ri : Halfway through stop bit

• Bit 7 time in Mode 0, beginning of the stop bit in other modes

• Modes in SCON

SM0	SM1	Mode
0	0	0
0	1	1
1	0	2
1	1	3

Mode bits

• SM2 - Multiprocessor mode bit

i) Multiprocessor communication in Mode 2 and Mode 3 when set to '1' interrupt generated, i.e., Stop bit is '1' if bit 9 of received data is '1' and set to '0' if stop bit is '0'.

ii) Set to '1' for Mode 1 if no interrupt is generated (unless a valid stop bit is generated), otherwise set to '0'.

iii) Clear to '0' if mode '0' is in use, otherwise set to '1'

• REN = 1 while receiving data

• Ti and Ri are transmit and receive interrupt flags

→ PCON

SMOD	-	-	-	GF1	GF0	PD	IDL
7	6	5	4	3	2	1	0

• SMOD

= 0 → same frequency as timer 1 frequency

= 1 → double the baud width

Formula to find Baud Rate in Serial Mode 1

$$f_{i\,(baud)} = \frac{2^{SMOD}}{32} \times \frac{\text{oscillator}}{12[256 - (TH1)]}$$

• GF1, GF0 set by user for user applications

General Purpose User Flags 0 and 1

If something happens while performing a task, these flags work as interrupts.

• PD = Power Down Mode

Set when power consumption is more

• IDL = Ideal Mode

→ Modes

i) Mode 0 - Shift Register Mode

• SM0 = 0, SM1 = 1

• Stuf is used to transfer 8 data bits

• Baud Rate = $\frac{1}{12}$ x oscillator frequency

ii) Mode 1 - Standard UART

• SM0 = 0, SM1 = 1

• Stuf is 10 bit full duplex receiver / transmitter

• 10 bits are: 1 start bit, 8 data bits, 1 stop bit

- Whenever we are sending stop bit, Ti flag is set.
- Receiver Data is loaded into sbuf register iff:
  i) RI must be 0 AND
  ii) SM2 = 0 OR Stop bit = 1
- Whenever we receive data, start bit is discarded, 8 data bits go to sbuf, stop bit is saved in RB8 (Receive bit # 8) of SCON register.
- Baud Rate: $f_{baud} = \frac{2^{SMOD}}{32} \times \frac{\text{Oscillator}}{12[256-(TH1)]}$
- $f_{baud} = \frac{2^{SMOD}}{32} \times$ timer 1 overflow frequency
  - Used when we use Timer Mode 2
  - Above formulas for other timer modes
  - Here, TH1 = Timer 1 frequency

iii) Mode 2 - Multiprocessor Mode
- Sbuf sets is 11 bit
  (1 Start bit + 9 data bits + 1 stop bit)
- 9th data bit is copied from TB8 in SCON during transmission and stored in RB8 of SCON
- Start and stop, both bits are discarded.
- Others similar to Mode 1
- Baud Rate: $f_{baud} = \frac{2^{SMOD}}{64} \times$ Oscillator frequency

iii) Mode 3
- Same as Mode 2.
- But use Baud Rate formula: $f_{baud} = \frac{2^{SMOD}}{32} \times \frac{\text{Oscillator}}{12[256-(TH1)]}$
  - the one for Mode 1

19-03-19

- Eg. Oscillator frequency = 6MHz
  → Baud Rate in Mode 0 = $\frac{1}{12} \times 6 = 0.5$ MHz
  = 500 KHz

Interrupts

- Generally, we are engaging in some tasks and in between, external or internal forces come and it is upto us whether we accept it or not.
- Interrupt halts the other process and executes the other tasks and after executing this task it goes back to old task and executes the process from the position where it has left it.
- If a process is being executed and an external signal halts it, it executes an address in an address / interrupt service routine, and then it continues executing the previous task.
- Interrupts in 8085:
  i) TF0 (Internal Interrupt)
  - Whenever we use timer 0, if it exceeds its capacity (becomes all 1s) it is set.

  ii) TF 1 (Internal Interrupt)
  - Whenever we use timer 1, if it exceeds its capacity (becomes all 1s) it is set.

  iii) Serial Port (TI, RI) (Internal Interrupt)
  - Whenever serial data transmission / reception occurs, the respective interrupts are set.

iv) $\overline{INT0}$ (External Interrupt)
- This is set when P3.2 value becomes 0.

v) $\overline{INT1}$ (External Interrupt)
- This is set when P3.3 value becomes 0.

- Whenever an interrupt occurs, PC value ~~becomes~~ has the address of the next location to be executed in the interrupt service routine.
- The stack pointer points to the instruction from where the execution has to continue, value of PC before interrupt
- After the interrupt service routine is executed, the value from the stack pointer goes to program counter and execution continues.

Interrupt Control

- Once interrupt occurs, it depends on the process whether or not to execute it. If the interrupt is executed, the value for that particular interrupt is cleared or reset.
- Reset is non maskable interrupt, whenever this option is used the interrupt has to be ~~enabled~~ executed and the programs runs from the beginning.
- Maskable interrupts may or may not be executed.
- IE0 - IT1 → Edge
  IT0 - IT1 → Signal Type
- ~~Logic 1 :~~

→

- Interrupt Enable / Disable
  - If interrupt enable (EA) IE is high, only then the interrupts are enabled, but if this flag is low, all the interrupts will not be executed

IE:

EA	-	ET2	ES	ET1	EX1	ET0	EX0
7	6	5	4	3	2	1	0

↳ Interrupt Enable Register (Special Function Register)

~~EX0 occured - :~~

- EX0 = 1 → external interrupt 0 occured
  = 0 → external interrupt 0 not occured
- ET0 = 1 → external timer 0 occured
  = 0 → external timer 0 not occured
- EX1 = 1 → external interrupt 1 occured
  = 0 → external interrupt 1 not occured
- ES = 1 → enable serial port (when TI = 1 or RI = 1)
  = 0 → disable serial port
- ET2 = Reserved
- ET1 = 1 → external timer ϕ^2 occured
  = 0 → external timer 1 not occured

- Software Enabled Interrupts
  - Possible to set any interrupts externally by programmer by using byte addresses
  - Set by action → normal interrupts
  - Set by programmer interferance → software generated interrupts

- Interrupt priority
- Interrupt priority is a special function register
- If flag = 0 → priority is low
  = 1 → priority is high
  ↳ set externally by programmer
- But, by default, the priority order is different. or, if many flags are set this priority order is followed:

IE0 — high
TF0
IE1
TF1
Serial EI and RI — low

- Interrupt destinations
  - IE0 - 0003
  - TF0 - 000B
  - IE1 - 0013
  - TF1 - 001B
  - Serial EI and RI - 0023

- IP: Interrupt Priority Register

-	-	PT2	PS	PT1	PX1	PT0	PX0
7	6	5	4	3	2	1	0

IP = 0 → low
     1 → high

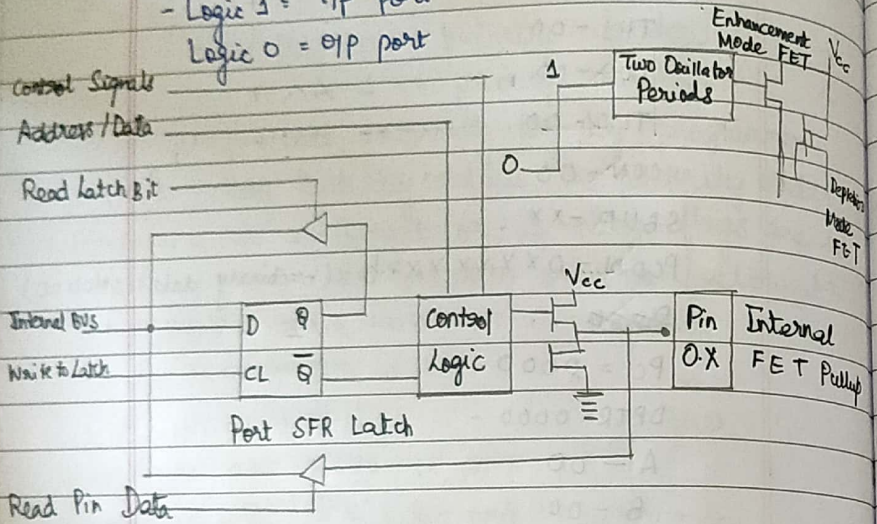
- Values changed when 'RESET' is pressed
  - TCON - 00
  - TH0 - 00
  - TH1 - 00
  - TL0 - 00
  - TL1 - 00
  - SCON - 00
  - SBUF - XX
  - PCON - 0XXXXXXXb (b = binary data) (SMOD = 0)
  - ~~Reset~~
  - PC = 0000
  - DPTR = 0000
  - A - 00
  - B - 00
  - SP - 07
  - PSW - 00
  - P0.3 - FF
  - IP - XXX00000b (b = binary data)
  - IE - 0XX00000b (b = binary data)

→ Introduction
→ Timer Flag Interrupt
→ Serial Port Interrupt
→ External Interrupts
→ RESET
→ Interrupt Control
→ Interrupt Enable / Disable
→ Interrupt Priority
→ Interrupt Destinations
→ Software Generated Interrupts

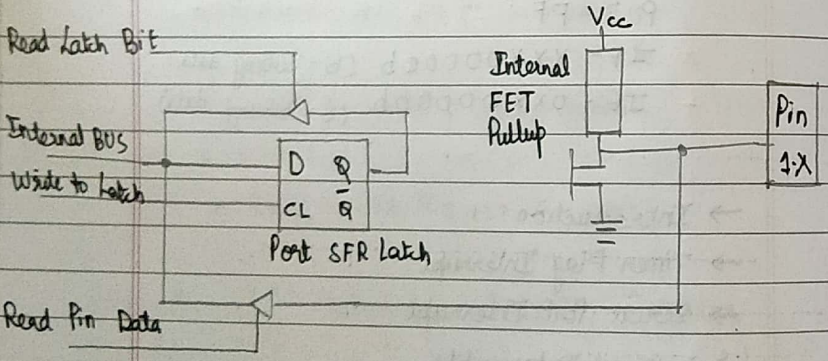
24-07-19

Circuit Diagram(s)

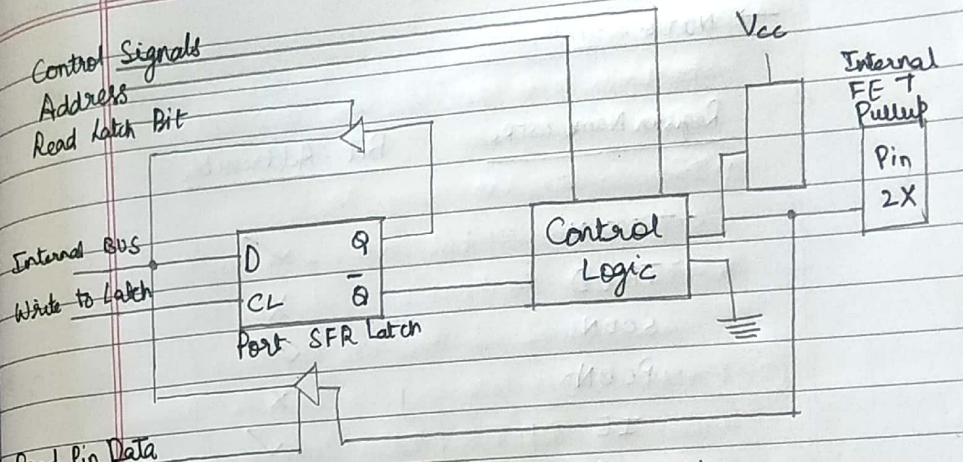
- Logic 1 = i/p port
  Logic 0 = o/p port



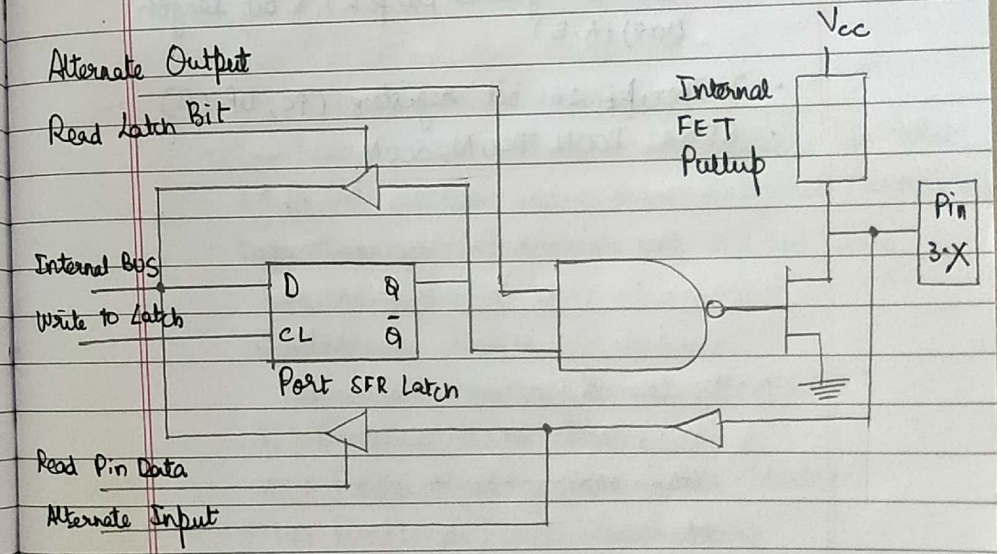
Port 0 Pin Configuration



Port 1 Pin Configuration



Port 2 Pin Configuration



Port 3 Pin Configuration

→ NOTE:

Register Name (SFRs)	Bit Addressible
TCON	✓
TMOD	X
SCON	✓
PCON	X
IE	✓
IP	✓

- Registers in 8051
  - 34 = 32 + 2 (general purpose), 8 bit length
    (4×8) (A, B)
  - 2 Specific 16 bit registers (PC, DPTR)
  - SFRs: PCON, TCON, SCON

25-07-'19

UNIT-II : PROGRAMMING USING 8051

Programming 8051

→ Data Transfer Instructions:

1) MOV destination, source
2) PUSH source (OR) POP destination
3) XCH destination, source

- These instructions are used to move data from one location to another.
- These locations can be of four types:
  i) Internal RAM
  ii) SFRs
  iii) External RAM
  iv) Internal or External ROM
- We have 28 data transfer instructions in total, which fall in the above three categories (MOV, PUSH, XCH)
- Data Transfer instructions are divided into five categories:
  i) Addressing Modes
  (Immediate, Register, Direct, Indirect)
  ii) External Data Modes
  iii) Code Memory ROM Data Modes
  iv) Push and Pop Instructions
  v) Data Exchange
- Data can be moved from accumulator or to accumulator, register to register moving (MOV R_1, R_2) is not possible.

① Addressing Modes

- Specifies how operands are divided in instruction

A) Immediate Addressing Mode

- # (pound) symbol represents immediate addressing mode
- No instruction uses register B directly, instead of it, we use its address directly.
- Here: R_n = Any register bank, A = Accumulator, DPTR = Data Pointer
- The instructions are:
  - → MOV R_n, #n (8 bit data) [Eg. MOV R0,#12]
  - → MOV A, #n (8 bit data)
  - → MOV DPTR, #nn (16 bit data)
- We select register bank using RS0, RS1 in program counter.
- Whenever we use reset, we by default the register used is R0, if we want to use something else, we must reset it which is bit addressable.

B) Register Addressing Mode

- Data must include / involve Accumulator.
- Two instruction to be used to move data from one register to another (R → A → R)
- Used to move data among / between registers
- The instructions are:
  - → MOV A, R_n
  - → MOV R_n, A

C) Direct Addressing Mode

- Here, we use the addresses of internal RAM.
- R0 is represented by 00h, R1 by 01h and so on.
- MOV A, 00h = MOV A, R0
- Whenever we have / give an address, it points to internal RAM address.
- MOV A, 41h → store value present in address 41 of RAM to accumulator.
- MOV 41h, A → move value present in accumulator to address 41h in
- Instructions:
  - → MOV A, add
  - → MOV add, A
  - → MOV R_n, add
  - → MOV add, R_n
  - → MOV add, #n
  - → MOV add1, add2
- Hash symbol must always be placed on the source side, but not on the destination side.
- Always, '#' indicates data, and other than '#' indicates addresses.

D) Indirect Addressing Mode

- In place of Rp, we can use only R0 or R1, but not R2 to R7.
- It is a two step process to access data.
- Eg. MOV @Rp, #n → The data #12h will be stored in the address stored by Rp, i.e. 2F for MOV @Rp, 12h

- Instructions:
  - → MOV @Rp, #n
  - → MOV @Rp, add
  - → MOV @Rp, A
  - → MOV add, @RP
  - → MOV A, @Rp

② External Data Modes
- We use external RAM here.
- MOVX → X represents external RAM.
- Here: @Rp and @DPTR contain addresses related to external RAM.
- Instructions:
  - → MOVX A, @Rp
  - → MOVX A, @DPTR
  - → MOVX @Rp, A
  - → MOVX @DPTR, A

③ Code Memory ROM Data Modes
- Used to access data in ROM (Internal / External)
- Instructions use C for code.
- We can access ROM memory using PC or DPTR
- Let A = 03, DPTR/SP = 1235, MOVC A, @A+DPTR, we move value in ROM address 1239 to accumulator
  Add the value in Accumulator to DPTR / PC we get a value, the value in that address of ROM will be copied to accumulator
- Possible to copy data to accumulator only, no other way.
- Source side address is available, destination side accumulator is available since data can be written only once in ROM, cannot be re-written / overwritten.
- Instructions:
  - → MOVC A, @A+DPTR
  - → MOVC A, @A+PC

④ Push and Pop Instructions
- Push - Stack pointer incremented by 1, data is placed inside the stack memory.
- Pop - Data is popped from the stack memory, stack pointer is decremented by 1.
- Use addresses only (even for registers).
- Eg. POP 00 → pop data from stack pointer to register R0
- Eg. PUSH 01 → push data from register R1 to stack
- Stack pointer will be initialized between 30 and 7F, but avoid 7F as we cannot increment after it.
- By default, whenever reset is pressed, stack pointer is initialized to 07.
- Instructions:
  - → PUSH add
  - → POP add

→

(5) Data Exchange

- Whenever we use moving instructions, no effect on source data.
- But, exchange instructions are used to exchange data between registers (s) and accumulator.
- 'D' → indicates that only the lower nibble data is exchanged.
- Instructions:
  - → XCH A, Rp
  - → XCH A, add
  - → XCH A, @Rp
  - → XCHD A, @Rp

NOTE:

TCON 88h

i) Copy the byte in TCON register to register R2 in four different methods.

ii) Put the number 34 in registers R5, R6, R7 in different ways.

MOV R0, #34

26-07-'19

→ Logical Instructions

- Generally we have four logical operations (AND, OR, XOR, NOT)
- 8051 microcontroller supports all these operations.
- Both data transfer operations and logical operations do not affect any of the flags.
- But, only in one case, i.e., whenever we use PSW in any of the operations, flags will be affected, otherwise, no change in flags.
- Logical instructions are divided into three categories:
  1) Byte Level Operations
  2) Bit Level Operations
  3) Rotate and Swap Operations
- Here:

	Rotate	
AND - ANL	RL	In RAM:
OR - ORL	RR	0 - 31: Page Bank
XOR - XRL	RLC	32 - 47: Bit Addresses
NOT - CPL	RRC	
	SWAP	

(1) Byte Level Operations

- → ANL A, #n
- → ANL A, add
- → ANL A, @Rp
- → ANL A, Rn

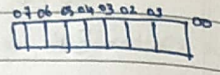
→ Value in acc AND with val in source, store val in Acc

- → CLR A [Accumulator = 00]
- → CPL A [One's complement of val in acc]
- → ANL add, A
- → ANL add, #n

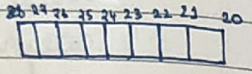
Replace ANL with ORL and XRL

② Bit Level Operations

- Addresses 0-31 in internal RAM are used for register banks.
- Addresses 32-47 (20-2F) are used for bit addressing
- Internal RAM



20	00-07
21	08-0F
22	10-17
23	18-1F
24	20-27
.	
.	
.	
2F	78-7F

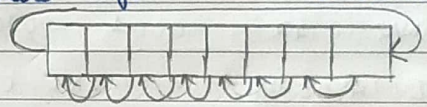


- Instructions
  - → ANL C,b
    - Here, small 'b' represents bit addressible area, the address, and 'C' represents the value of carry bit in accumulator. '\b' represents the complement value of 'b'.
  - → ANL C,/b b = address pointing to bit area of RAM
  - → CPL C
  - → CPL b
  - → CLR C
  - → CLR b
  - → SETB C
  - → SETB b

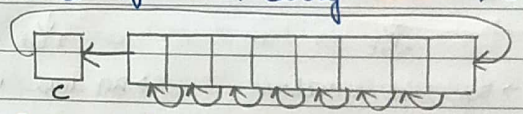
- SFR Bit Addresses
  - → A OE7 OE6 . . . . . OE1 OE0
  - → B OF7 OF6 . . . . . OF1 OF0
  - → 1E
  - → 1F
  - → P_0
  - → P_1
  - → P_2
  - → P_3
  - → PSW
  - → TCON
  - → SCON

③ Rotate and Swap Operations

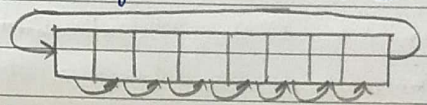
i) Rotate Left - RL - 8 bit rotate operation



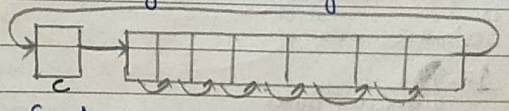
ii) Rotate Left with carry - RLC - 9 bit rotate operation



iii) Rotate Right - RR - 8 bit rotate operation



iv) Rotate Right with Carry - RRC - 9 bit rotate operation



v) Swap

MSB becomes LSB and LSB becomes MSB

Equivalent in performing 4×RL or 4×RR operations

NOTE:

- Rotate and swap operations can be done only with accumulator
- For every rotate left operation, the value in the accumulator doubles itself (wrt decimal)
- For every rotate right operation, the value in the accumulator halves itself (wrt decimal)
  - Consider lower nibble and higher nibble separately

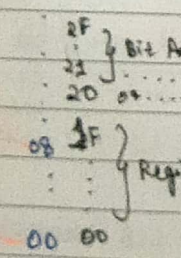
Eg. MOV A,#12h		RL	RL	RL	RL	SWAP
A →	12	0001	0010			1 SWAP
	24	0010	0100			= 4 X ROTATE
	48	0100	1000			LEFT
	90	1001	0000			(OR)
	21	0010	0001			4X ROTATE
SWAP	12	0001	0010			RIGHT

Eg. ANL A,08 — ANL C,08

Eg. ANL A,00 — ANL C,00

8 bit — 1 bit

→ Byte level instructions — → Bit level instruction



31-07-19

→ Arithmetic Operations

- In data transfer and logical instructions, generally, no flags will be affected unless in special cases:
  - → PSW is used
  - → Data transferred to CY bit, etc.
- Arithmetic Operations directly effect arithmetic flags (Carry, Parity, Overflow, Auxillary Carry) mainly.
- Instructions:
  - ADD → Addition
  - SUB → Subtraction
  - MUL → Multiplication
  - DIV → Division
  - INC → Increment
  - DEC → Decrement
  - DAA → Decimal Arithmetic

- Addition Operations
  - They add two 8 bits of data
    - → ADD A,#n
    - → ADD A,R
    - → ADD A,direct
    - → ADD A,@Ri (i=0/1)
  - Multi byte (8bit + 8 bit + CY bit) addition to solve multi byte
    - → ADDC A,#n
    - → ADDC A,R
    - → ADDC A,direct
    - → ADDC A,@Ri (i=0/1)

Eg. 01 FF
+ 01 01

- Subtraction Operation
- Subtract with borrow
- Value on R or S is subtracted from accumulator along borrow (0/1) is subtracted, and then the value is stored in accumulator.
- Before performing subtraction, you must clear carry flag to 0.
- Instructions:
  - → SUBB A,#n
  - → SUBB A,R
  - → SUBB A,direct
  - → SUBB A,@Ri

- Increment and Decrement operations

The value in Accumulator (A), value in Register/R value in direct address (direct), value in address held by Ri, where i=0/1 is incremented or decremented by 1.

- Data pointer can only be incremented, cannot be decremented, because it can only be used to access external data/addresses, given a starting address and can only be incremented, like program counter.
- Instructions:

→ INC A	→ DEC A
→ INC R	→ DEC R
→ INC direct	→ DEC direct
→ INC @Ri	→ DEC @Ri
→ INC DPTR	

- Multiplication operations
  - Value in Accumulator multiplied with the value in register B.
  - Value stored /Result stored as:

B A

higher bit data stored in B ← lower bit data stored in A

- Carry flag will always be 0 when multiplication is performed.
- Overflow flag may be set or reset.
  - when A×B > FF → overflow is set → must see register B also for higher nibble
  - when A×B < FF → overflow is reset → must see register A only for lower nibble
- Instruction:
  - → MUL AB

- Division Operations
  - Value in Accumulator divided with the value in the register B.
  - The quotient is stored in the Accumulator and the remainder is stored in register B.
  - The value in register B, i.e., the remainder must always be lesser than the value in register B, i.e., the divisor.
  - Instruction: → DIV AB

- Decimal Arithmetic
  - Must be used only when decimal numbers are used.
  - Must be used only after ~~Carry~~ ADD/ADC instruction(s).
  - Internal working:
    - ★ If CY = 1 or higher nibble > A
      → add 6 to higher nibble
    - ★ If Auxillary CY = 1 or lower nibble > A
      → add 6 to lower nibble
    - ★ If CY = 1 and Auxillary CY = 1, add 66 (6 to higher nibble and 6 to lower nibble)

- How flags are affected
  - ADD
  - ADDC
  - ANL c, direct
  - CJNE
  - CLR C
  - CPL C
  - DAA
  - DIV
  - MOV C, direct
  - MUL
  - ORL C, direct
  - RLC
  - RRC
  - SETB C
  - SUBB

NOTE: When we perform A/B, if B=00, overflow flag will be set, otherwise it will be reset.

NOTE: Unsigned Addition / Subtraction
→ Extreme values = 0 to 255
→ Only carry flag will be affected
→ Do not consider overflow flag.

NOTE: Signed Addition / Subtraction
→ Extreme values are from -127 to +127
→ Carry and overflow flags will be affected.
→ Overflow Flag: Check carry flag of bit 6 and 7
XOR them, XOR = 1 => OV = 1 XOR = 0 => OV = 0
↳ Complement sign flag

★ Unsigned Addition:

Eg. 95 + 189

128	64	32	16	8	4	2	1
0	1	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	0	0	0	1	1	0	0

★ Signed Addition:

-ve number => +ve no's 2's complement
= 1's complement of +ve no + 1

CY	OV	
0	0	
0	1	→ OV = 1 => Complement the sign
1	0	
1	1	

Eg: -01 => +1 = 0000 0001

+23 1's = 1111 1110

+ 1

1111 1111

0001 0011

1 1 1 1 1 1 1 0

1 0 0 0 1 0 1 0

1 XOR 1 = 0 => OV = 0

∴ Result = 00010010

Eg: +100

+50

→ Jump and Call Instructions

* Program Range:
  - Relative Range (-128d to +128d) (RJMP)
  - Short Absolute Range (Within the Page) (AJMP)
  - Long Range (Throught Memory) (LJMP)

* Jumps
  - Bit Jumps
  - Byte Jumps
  - Unconditional Jumps

Condition is tested, if condition is true, it jumps to a particular location, if condition is not true, program execution continues

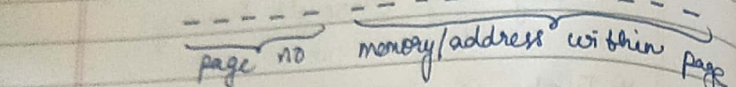
* Jump and Call instructions are called as decision codes, as they execute based on some decision. They alter the program execution sequence.

* Other three types of execution are called as action codes.

* Relative Range: From current location, we are able to jump from -128 to +128 decimal the range is relative range (small range).

* Short Absolute Range: We can jump anywhere within that page. (single page). We can jump from any instruction to any other instruction, but within the same page.

* Page Division:
Consider 16 bit memory: 0000 - FFFF in hexa decimal
Divide into $2^5 = 32$ pages.
5 bits for pages, remaining 11 bits for addresses in that page



00000 00000000000 } Page 0 : 0
00000 11111111111
0000 - 07FF
0800 - 0FFF → Page 1 : 1
1000 - 17FF → Page 2 : 2
1800 - 1FFF → Page 3 : 3
⋮
F800 - FFFF → Page 32 : 31

* Long Range: Any jump between 0000 - FFFF comes under the long range. Any Jump from any address to any address comes under this.

* Range: Difference between new address and the old address.

→

* Conditional jumps (bit level and byte level) are used only for relative range, i.e., the relative address given by user. We can move -128 bits backward or +127 bits forward, if more is given, error is occurs.

* Conditional jumps can use short absolute range and long absolute range.

* Bit Level Jump Instructions
(no flags affected in general)

→ JC radd (if CY=1, jump to relative address)
→ JNC radd (if CY=0, jump to relative address)
→ JB b, radd
(if bit (value), i.e., value in address specified by b = 1, jump to radd)
→ JNB b, radd
(if bit (value), i.e., value in address specified by b = 0, jump to radd)
→ JBC b, radd
(if bit (value), i.e., value in address specified by b=0, make b = 1 and if b=1, make it b=0, and jump to relative address).
(Flag affected only in this instruction, if b value is value of some flag in SP)
Here: b → bit level address
r → relative address

* Byte Level Jump Instructions:

- Comparision performed on byte level instructions.
- CJNE = Compare Jump Not Equal
- Instructions:

→ CJNE A, add, radd
(If val in acc and given address are not equal, jump to relative address, else, continue execution
execution ~~if~~ A-add (val in) is performed
A-val in add < 0 → CY = 0
A-val in add > 0 → CY = 1
} affects carry flag)

→ CJNE A, #n, radd
→ CJNE Rr, #n, radd
→ CJNE @Rp, #n, radd
→ DJNZ Rr, radd
(After decrementing by 1 the value present in the register, if the value becomes 0, jump to relative address. else, continue execution)
→ DJNZ add, radd
(Decrement jump no zero: after decrementing the value present in the ~~register~~ RAM address, if the value becomes, jump to relative address, else, continue execution)
→ JZ radd
(If accumulator value = 0, jump to radd)
→ JNZ radd
(If accumulator value = 1, jump to radd)

Doesn't affect any flags

* Unconditional Jumps:

- Action / Decision must take place, no condition.
- Instructions:

→ JMP @A+DPTR
(Add the value in accumulator and data pointer, jump to that particular relative address (sum of A+DPTR) present in accumulator).
→ AJMP sadd
(Jump instructions for short absolute range)
→ LJMP ladd
(Jump instructions for long range)
→ SJMP radd
(Jump instructions for relative
→ NOP
(No operation, used to waste time, moves to next instruction)

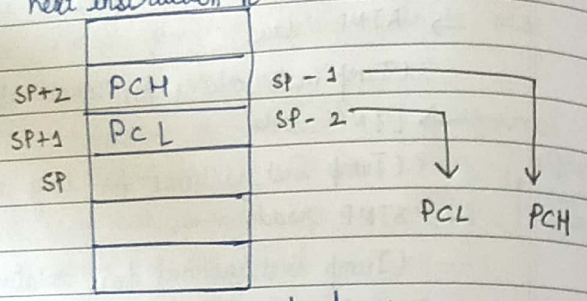
* Call Instructions:

- Used to return to main program after the execution of instructions called by JUMP.
- Instructions:

→ ACALL sadd
(Returns to address in short absolute range)
→ LCALL ladd
(Returns to address in long absolute range)

→ RET

(Return instruction. Used to go back to place where main execution programme stopped, (when subroutine was called) from subroutine. Two values are popped from stack and pushed to program counter, so that it knows about the address of the next instruction to be executed.)



(Subroutines - Similar to functions
Used for modularity and code reusability)

→ RETI

(Return from Interrupt - Whenever an interrupt occurs, control goes to a particular location to execute the interrupt service routine. The last instruction of the interrupt service routine will be RETI to go back to the original program, from where the execution stopped).

07-08-'19

→ Example Problems on Instructions

1) Copy the number 8Dh to RAM location 30h to 34h.

→
```
MOV 30h, #8D        MOV A, #8D      → 2 bytes memory
MOV 31h, #8D        MOV 30h, A      → 2 bytes memory
MOV 32h, #8D        MOV 31h, A
MOV 33h, #8D        MOV 32h, A      6x2 = 12 bytes
MOV 34h, #8D        MOV 33h, A
                    MOV 34h, A
```
↳ 3 bytes
(op, dest add, source value)

3 x 5 = 15 bytes

2) Use the stack to execute the above program.

→

3.) Swap the contents of register R_7, R_6 of register bank '0'.

→
```
MOV 30h, R6          MOV 30h, 06
MOV 31h, R7   (OR)   MOV 31h, 07
MOV R6, 31h          MOV 06, 31h
MOV R7, 30h          MOV 07, 30h
```

NOTE: 00 ← R0
01 ← R1
02 ← R2
03 ← R3
04 ← R4
05 ← R5
06 ← R6
07 ← R7

4.) Use three different instructions to clear the contents of A register

→
```
MOV A, #00h
CLR 0E0
CLR A
XRL A
ANL A,00
```

5.) Complement / Invert every bit in register R_6 of register bank 0.

→
```
MOV A, R6
CPL A
MOV R6, A
```

6.) OR the contents of ports 1 and 2. Put the result in external RAM location 0100 H.

→ P0 ← 80H P1 ← 90h P2 ← 0A0h P3 ← 0B0h
```
MOV A, 90h  //Port 1 data copied to accumulator
ORL A, 0A0  //OR value in accumulator with val in P2
MOV DPTR,#0100h // Move 16 bit address to DPTR
MOVX @DPTR, A //Move val in A to external RAM
  ↳ external RAM
```

7.) Add the unsigned numbers found in RAM location 25h, 26h, 27h together and put the result in RAM location 31h(MSB) and 30h(LSB).

→
```
MOV R0, 25h        ADD  A, R0
MOV R1, 26h        ADDC A, R1
MOV R2, 27h        ADDC A, R2
MOV A, 00h         MOV  30h, A
                   MOV  31h, PSW
```

21-08-19

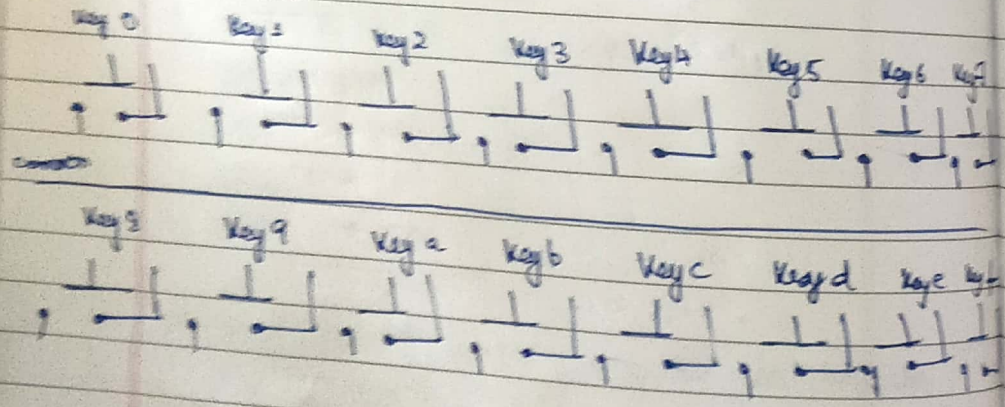
8051 Applications

→ Keyboards
→ Displays
→ A/D (Analog to Digital) and D/A (Digital to Analog) conversions
→ Multi-closure Interrupts

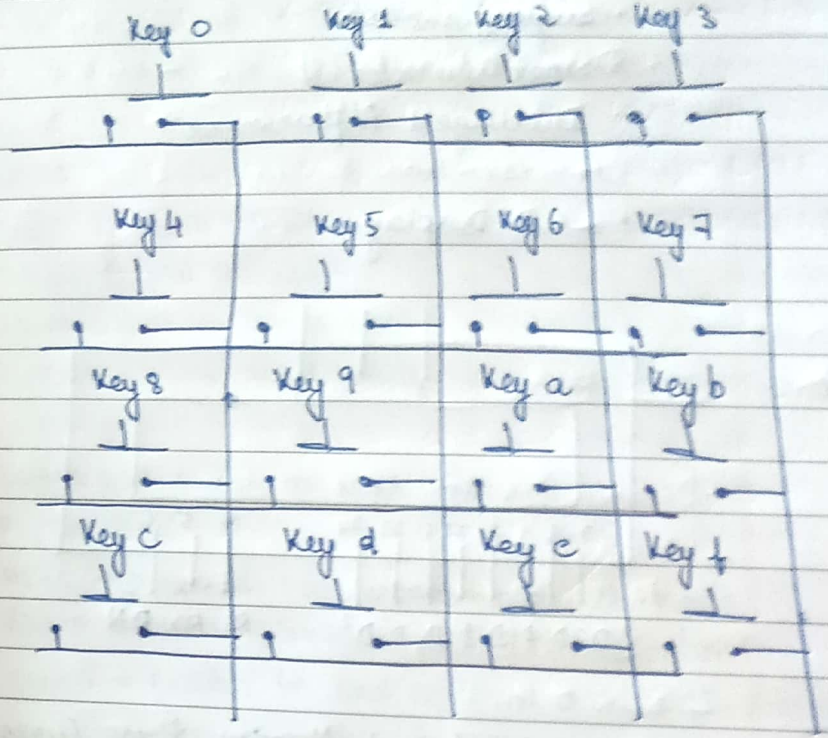
→ Keyboards

- Human Factors
- Key Switch Factors
  i) Lead-per-Keyboard
  ii) Matrix Keyboard
  iii) Hexa Coded Keyboard

Lead Per Key Keyboard



XY Matrix Keyboard



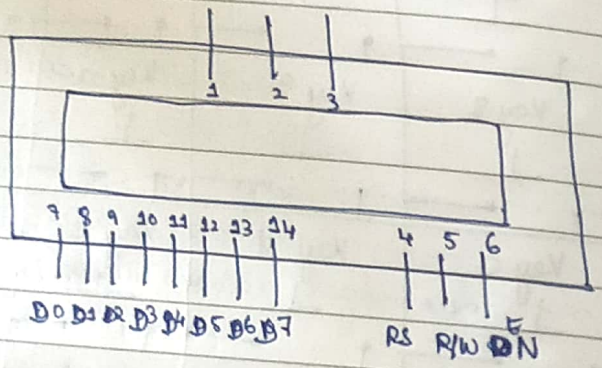
Coded Keyboards

Programs for Keyboards
→ Bounce
→ Multiple Keys
→ Key Held
→ Rapid Key Hit

→ Displays

- Single Light (S)
- Single Character (S)
- Intelligent Alphanumeric

- LCD Display



0X02 Row 1: Clear LCD and Memory, Home Cursor
Row 2: Clear and Home Cursor Only
Row 3: i) s = 1/0
1 → Shift Screen, 0 → Shift Cursor
ii) id = 1/0
1 → Cursor right, screen left
0 → Cursor left, screen right

b c d
d = 1/0 → Screen on (1) / off (0)
c = 1/0 → Cursor on (1) / off (0)
b = 1/0 → cursor blink (1) / non blink (0)
S/C = 1/0 → Screen (1) / cursor (0)
R/L = 1/0 → Shift one (1) space right / shift one (0) space left

f n dl
dl = 1/0 → eight (1) / four (0) bits per character
n = 1/0 → two (1) / one (0) rows of characters
f = 1/0 → 5/10 dots per (1) character / 5/7 dots per (0) character

0001 — → Write to character RAM address after this
001 — → Write to display RAM address after this
01 bf current address → bf=1 → busy bf=0 → not busy
10 char byte → Write byte to last RAM chosen
11 → Read byte from last RAM chosen

D0 to D7 → given to Data Register
RS, R/W, EN
RS=0 → command mode RS=1 → data mode
R/W=0 → write operation R/W=1 → Read operation
EN=0 → possible to send command data/ data register da
LCD is able to receive
EN=1 → LCD is not able to receive

→ D/A Conversions

- Converts digital values to analog signals
- Dout = - D reference x (byte in / 100H)
where D reference = ± 10V
- Conversion time = 5 micro seconds
- Control Sequence = $\overline{CS}$ $\overline{WR}$ then

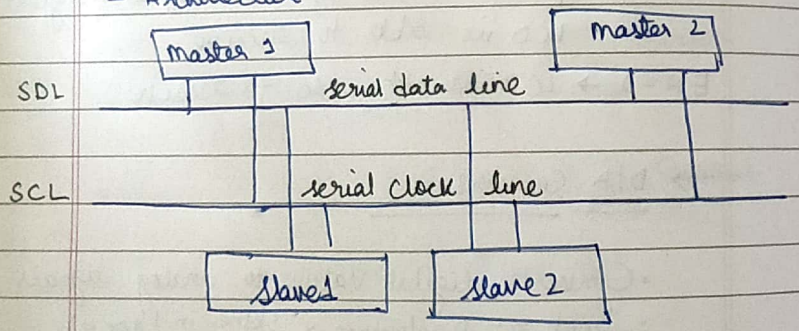
→ A/D Conversion

- Converts analog values to digital signals.
- Data D_{in} = Vreference − when data = 00H
  + when data = FFH
- Conversion time period = 1 micro second
- Control Sequence = $\overline{CS}$ then $\overline{WR}$ then $\overline{RD}$

22-08-19

→ I^2C Bus

- Designed for low-cost, medium data rate applications.
- Characteristics:
  - Multiple Master
  - Fixed Priority Arbitrato
  - Serial (Data)
- Architecture



- Several microcontrollers come with builtin I^2C controllers

→ CAN Bus

- More advantageous than I^2C Bus.
- I^2C Bus: Speed / Data Rate = 40-80 kbps
- Can Bus: Speed / Data Rate = 1 Mbps

→ ARM Processor

→ SHARC Processor

NOTE:

- RISC Architecture, CISC Architecture
- ARM processor consists 32 bit data register, each byte has 4 ~~bytes~~ words of data (4×8 = 32)

Big Endian mode → data stored from MSB to LSB
Little Endian mode → data stored from LSB to MSB

Instructions of RISC processor