Unit - I : The 8086 Microprocessor

Introduction to 8086 – Microprocessor architecture - Addressing modes - Instruction set and assembler directives - Assembly language Programming - Modular programming - Linking and Relocation - stacks Procedures - macros - Interrupts and Interrupt service routines - Byte & String manipulation.

Introduction to 8086

* 8086 µp is an enhanced version of 8085 µp that was designed by Intel in 1976.
* It is a 16-bit µp having 20 address lines and 16 data lines that provides up to 1MB storage.
* It consists of powerful instruction set, which provides operations like multiplication and division easily.
* It supports two modes
    1. Maximum mode
    2. Minimum mode
* maximum mode is suitable for system having multiple processors and minimum mode is suitable for system having a single processor.

Microprocessor Architecture

* Basically it is divided into two units namely,
    - Bus Interface unit
    - Execution unit

Bus Interface Unit (BIU):

- Performs all external bus operation
- BIU consists of
    - Instruction Queue
    - Segment Register & Instruction pointer

- Memory addressing logic
- Bus Interface logic

EU (Execution Unit)

* It Contains control circuitry, instruction decoder, ALU, General purpose Registers & flag register.
* The execution unit is responsible for decoding and executing all instructions
* It provides address to BIU for fetching data/instruction.

ALU:-

* 16 bit arithmetic and logic Unit performs arithmetic operations such as addition, subtraction, Multiplication, Division, Increment Decement etc, and Logical operations like AND, OR, XOR, NOT, etc.

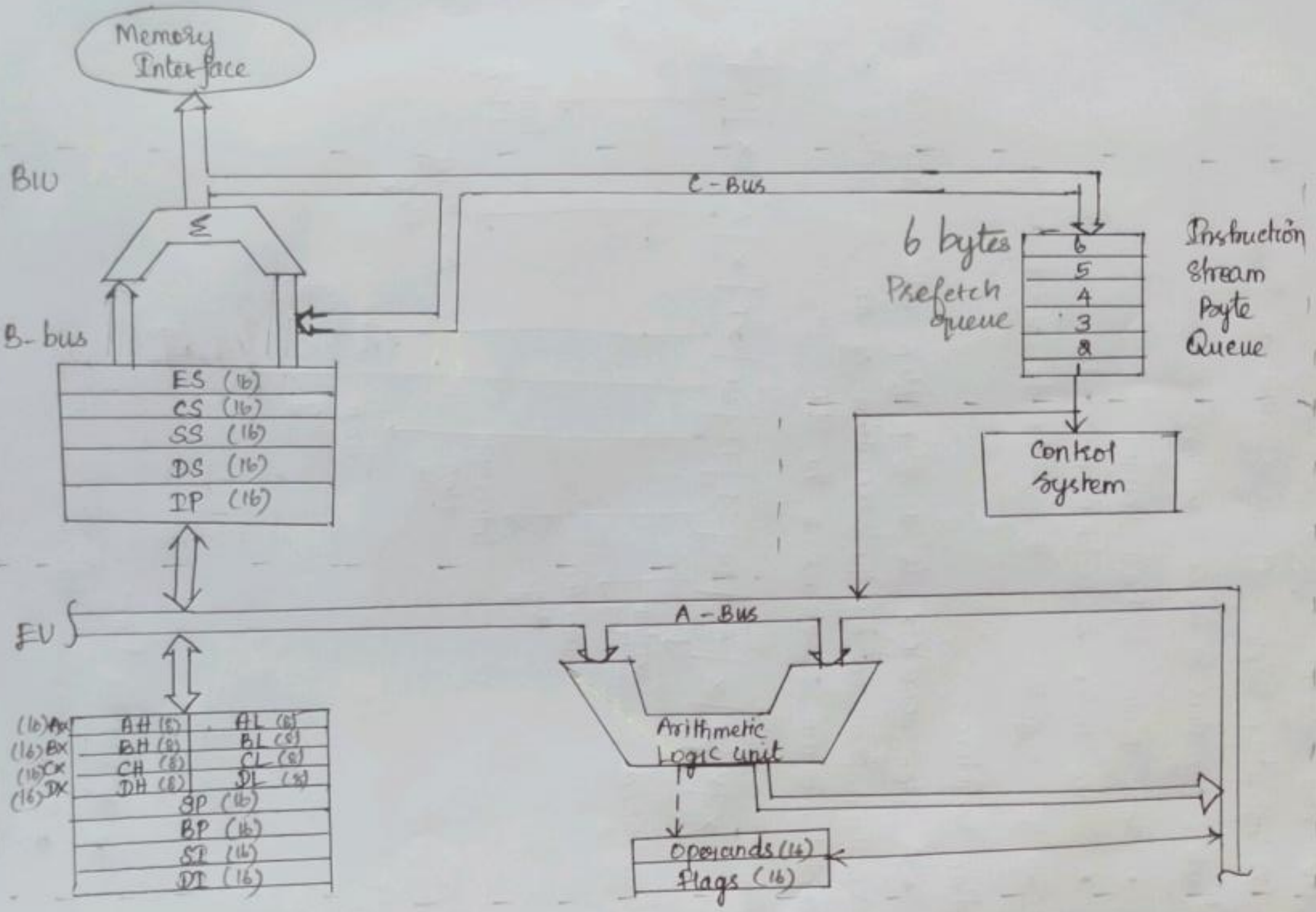
General Purpose Register:-

- These registers can be used as 8-bit registers individually or can be used as 16-bit in pair to have AX, BX, CX and DX.

1. AX register (accumulator Register) - stores operands for arithmetic operation.
2. Bx (Base register): Used as Base reg while computing data memory address.
3. Cx (Count register): It is defined as a counter.
4. Dx Register: Dx register is used to contain I/O post address for I/O instruction.

Segment base Register & an Instruction Pointer (IP)

(i) code Segment Register (16)
(ii) Data Segment Register (16 bit)
(iii) Stack Segment Register (16 bit)


Memory Interface
BIU
C-Bus
6 bytes Prefetch Queue
6
5
4
3
2
Instruction Stream Byte Queue
B-bus
ES (16)
CS (16)
SS (16)
DS (16)
IP (16)
Control System
EU
A-Bus
(16) AX
(16) BX
(16) CX
(16) DX
AH (8)
AL (8)
BH (8)
BL (8)
CH (8)
CL (8)
DH (8)
DL (8)
SP (16)
BP (16)
SI (16)
DI (16)
Arithmetic Logic unit
Operands (16)
Flags (16)

(iv) Extra Segment Register (16 bit)

(v) Instruction Pointer - It is referred as Program counter.

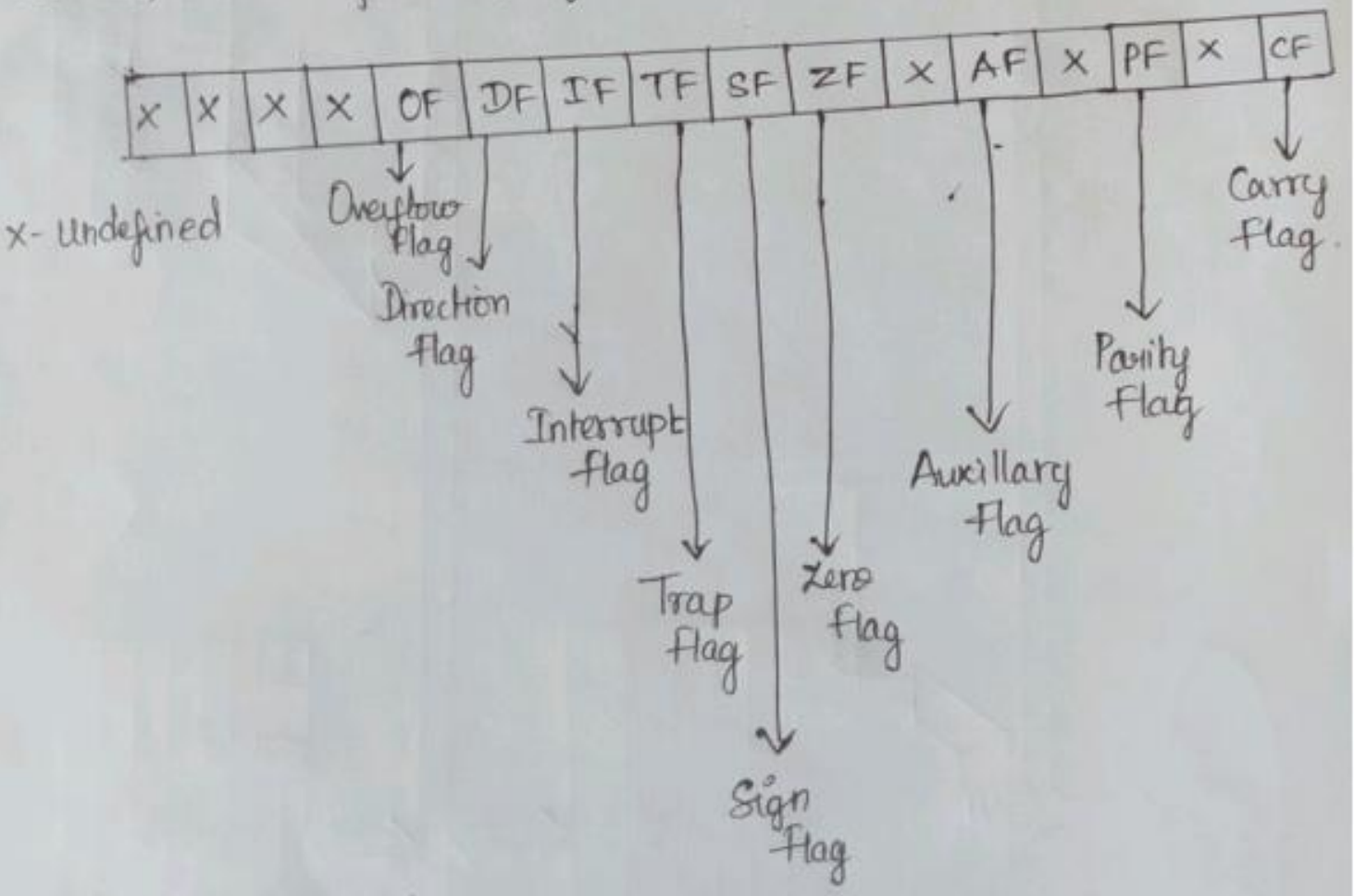
Pointers Registers & Index Registers

- Stack Pointer & Base Pointer
- Source Index & Destination Index

- all the registers are 16 bit - holds offset address.

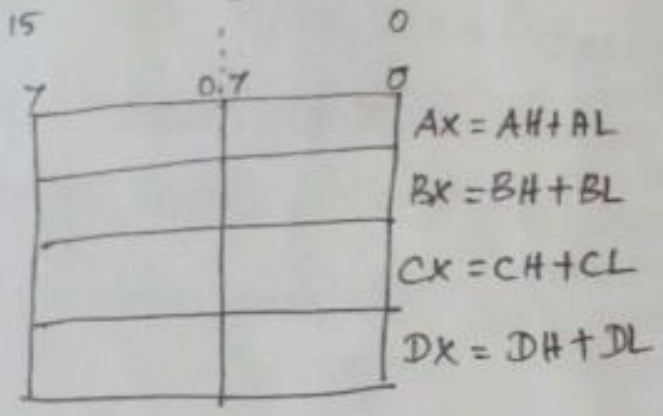
Flag Registers :

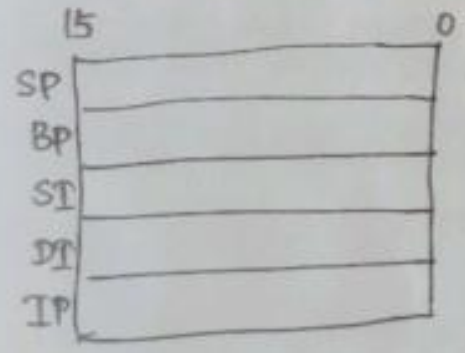
- A flag registers is a flip flop which indicates some condition produced by the execution of an instruction or controls, certain operations of the EU.

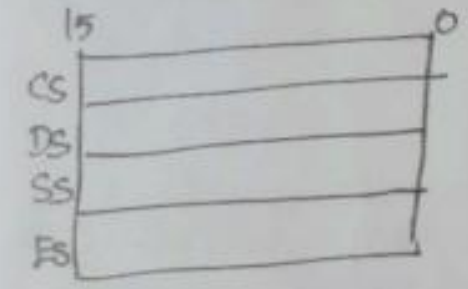


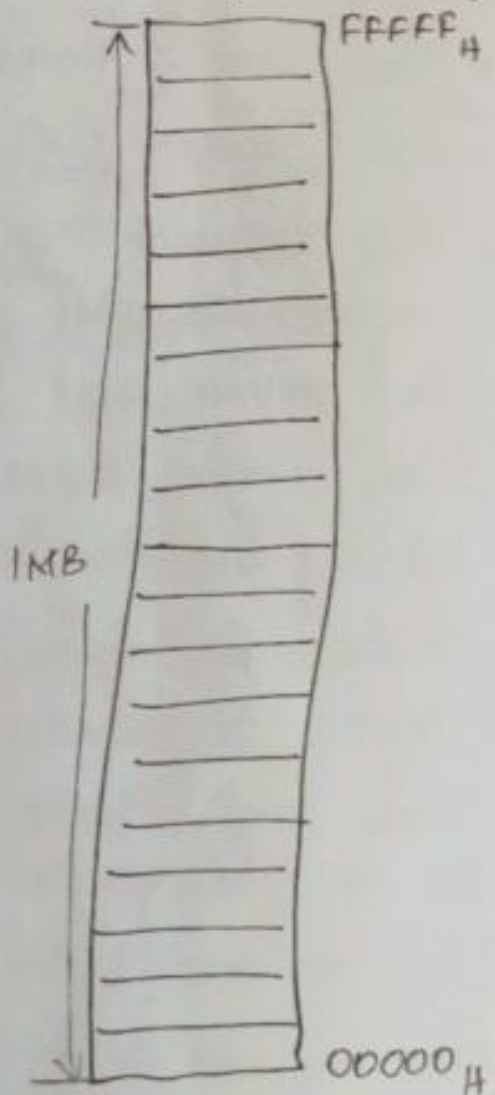
* 6 Status Flags
* 3 are control flag.

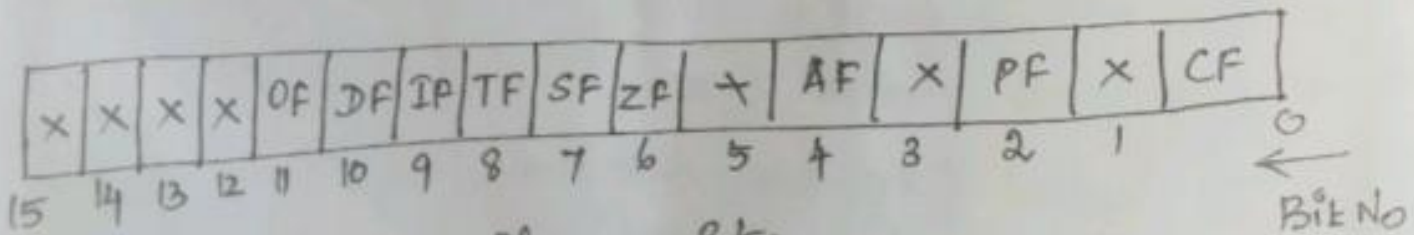
8086 programmer's model











Flag register.

* The 8086 has 20 addr lines & total 1MB of memory may be interfaced to it.
* The memory space is divided into 4 segments. each of 64KB

* The Segments are code segment, data segment, Sta Segment and Extra Segment.
* There are 4 segment registers namely code segment (C Data segment (DS), stack seg (SS) & Extra seg (ES).
* These Segment reg store the starting addr of the Segm in the memory.
* The pointer and index registers such as stack pointe Base pointer, Source index & Destination reg are the 16 bit reg ~~used to~~ holds the offset address
* Instruction pointer (IP) which is also known as Progra Counter. It provides the offset address to the code segm
* There are 4 General purpose registers. they are Ax, Bx, Cx Dx. All the registers are 16 bit.
* Flag reg are 16 bit registers.
* Flag reg are also known as Program Status word.
* It indicates the status of the processor.
* There are 9 flags in 8086. In which 3 are control flags and 6 are status flags.
* The control flags are direction flag, Trap flag & Interr flags.
* The status flags are Zero flag, Sign flag, parity fl Carry flag and Auxiliary Carry flag, overflow flag.

Instruction Set of 8086

* An Instr is a command to the µp to perform a given task on a specified data.
* Each Instr has two parts.
  1. Opcode (operation code) — Specifies the task (operation) to be performed
  2. operand — Specifies the data to be operated on.
* The 8086 has more than 20,000 instr.

Classification of Instruction set:

1. Data transfer
2. Arithmetic
3. Bit Manipulation
4. Program Execution transfer.
5. String
6. Processor control.

Data transfer Instructions

* used to transfer data from source to destination
* Mov Des, Src
* Exa: Mov AL, BL (Mov destination, source)
  - destination → reg/Mem
  - source → reg, mem, Immediate operand
* PUSH operand:
  - pushes operand into top of stack.
  - Exa: PUSH BX
* POP Des: (Des → destination)
  - pops the operand from top of stack to destination
  - Exa: POP AX.

- XCHG Des, Src :
  - * Exchanges Source & Destination
  - * Exa: XCHG DX, AX .
- IN accumulator, Port address :
  - * transfers the operand from specified port to accu reg
  - * Exa: IN AX, 0028_H.
- OUT port addr, accumulator .
  - * transfers the operand from accumulator to specified port
  - * Exa: OUT 0028_H, AX .
- LEA Reg, src : transfering offset addr from src reg to dest reg
  - * Loads the 16 bit Reg with the offset address .
  - * Exa: LEA BX, [DI] .
- LDS Reg, Src ;
  - * Loads the reg and DS with words from memory sour
  - * Exa : LDS BX, $[0301]_H$.
- LES reg, Src :
  - * Similar to LDA .
  - * Exa: LES Bx, $[0301]_H$:
- LAHF – Copies lower byte of flag reg to A_H.
- SAHF – copies content of AH to lower byte of flag reg.
- PUSHF – Pushes flag reg to top of stack
- POPF – pops the stack top to flag reg .

ithmetic Instructions :

- ADD dest, src :
  - * adds byte to byte or word to word
  - * Exa : ADD DX, AX - adds words
- ADC dest, src :
  - * add with carry
  - * Exa : ADC DX, AX (⇒ DX + AX + CF) → Carry flag content
- SUB dest, src :
  - * Subtracts byte from byte or word from word.
  - * Exa : SUB DX, AX.
- SBB dest, src :
  - * Subtracts with borrow.
  - * SBB DX, AX : DX - AX - CF →
  - * Carry flag (CF) acts as a borrow flag.
- INC src :
  - * Increments byte/word by one.
  - * Exa : INC AX.
- DEC src :
  - * Decrements byte/word by one.
  - * Exa : DEC AX.
- AAA (ASCII Adjust after addition) → Number (0-9) are represented as 30-39H in ASCII
  - * This instr allows us to add the ASCII codes.
- other ASCII Instr
  - * AAS (ASCII adjust after Subtraction)
  - * AAM (" " " Multiplication)
  - * AAD (" " " Division)

- DAA - (Decimal adjust after addition)
  * It is used to make sure that the result of adding BCD numbers is adjusted to be a correct BCD number
  * It works only on AL reg.
- DAS - (Decimal adjust after Subtraction)
  * 11th to DAA.
- NEG Src:
  * It creates 2's complement of a gn number.
  * It changes the sign of a number.

* CMP dest, src:
  * Compares dest & source data by subtracting the source from destination.
* MUL Src:
  * Unsigned Multiplication
  * Exa: MUL BX. -> AX * BX
* IMUL Src:
  * Signed Multiplication
  * Exa: IMUL BX -> AX * BX
* DIV src:
  * Unsigned division
  * Operand stored in AX; divisor is Src (reg/mem) result stored as AH = remainder; AL = quotient
* IDIV Src:
  * Signed division instr.
* CBW (Convert Byte to word)
  * Converts byte in AL to word in AX.
  * Conversion is done by extending the sign bit of AL throughout AH.

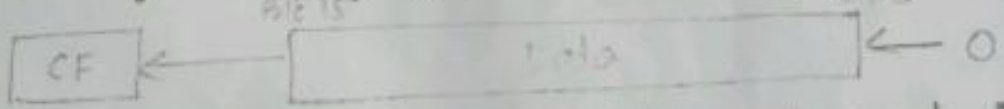
CWD - Convert word to double word.

* converts word in AX to double word in DX ; AX
* Conversion is done by extending the sign bit of AX throughout DX.

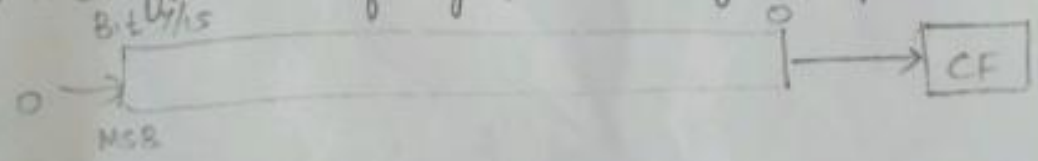
Bit Manipulation Instructions

* These instrs are used at the bit level.
* These instrs can be used for:
  - Testing a zero bit
  - Set/reset a bit
  - Shift bits across registers.

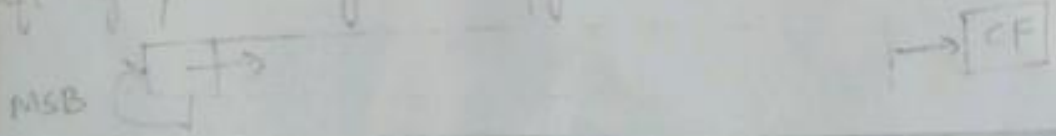
- **NOT src**
  * Complements (1's complement) src (reg/mem location)
- **AND dest, src :**
  * performs AND operation
  * Exa: AND AL, 08_H.
- **OR dest, src :**
  * performs OR operation
  * OR AL, 08_H.
- **XOR dest, src :**
  * performs xor operation
- SHL/SAL : shift bits of byte/word left ; put zero's in LSB's



- SHR : shift bits of byte/word right : put zero's in MSB's.

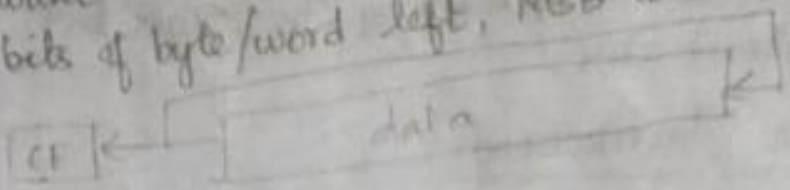


SAR
- shift byte/word right : copy old MSB into new

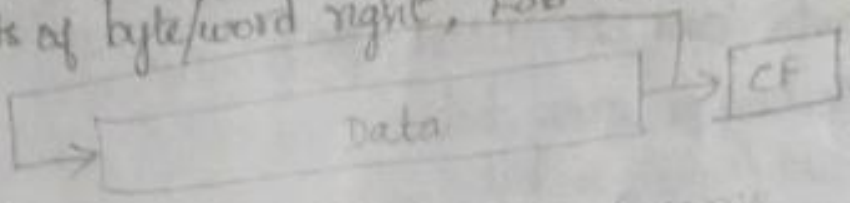


Rotate Instr

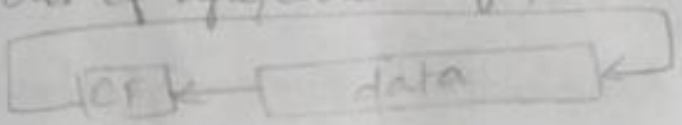
- ROL dest, count
  * rotate bits of byte/word left, MSB to LSB and to CF



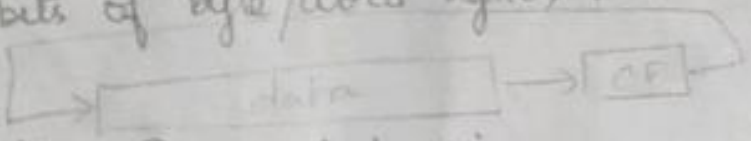
- ROR dest, count
  * rotate bits of byte/word right, LSB to MSB and to CF



- RCL dest, count:
  * Rotate bits of byte/word left, MSB to CF, & CF to LSB



- RCR dest, count:
  * rotate bits of byte/word right, LSB to CF & CF to MSB



Program Execution Tranfer Instructions

* These instr cause change in the sequence of the execution of instr.
* This change can be through a condition or sometimes unconditional
* The conditions are represented by flags.

- CALL dest:
  * This instr is used to call a subroutine or function or procedure.
- RET:
  * It returns the control from procedure to calling pgm.
  * Every CALL instruction should have a RET.

JMP dest:

* This instr is used for unconditional jump from one place to another.
* Exa: JMP 2000H.

Conditional Jumps – Jxx dest

* All the conditional jumps follows some conditional statements / any instr that affects the flag.

* JC – Jump if Carry : CF = 1
* JE – Jump if Equal : ZF = 1
* JNC – Jump if Not Carry : CF = 0
* JNE – Jump if Not Equal : ZF = 0
* JZ – Jump if zero : ZF = 1
* JNZ – Jump if not zero : ZF = 0
* JPE – Jump if Parity Even : PF = 1
* JPO – Jump if parity odd : PF = 0

Loop dest : —— looping Instr.

* The no. of times looping is required is placed in the CX-reg.
* with each iteration, the contents of CX are decrement

String Instructions

String
- Sequentially stored bytes/words.
- By using string instr, the size of the pgm is considerably reduced.

- **CMPS** dest, src ;
  * Compares the string bytes/words.
- **SCAS** string :
  * Scans string.
- **MOVS/MOVSB/MOVSW** :
  * moves ~~string~~ the byte/word from one string to another.
  * DS – Source string ; ES – Destination string.
- **REP** (repeat) :
  * used to repeat the given instr until $CX \neq 0$
  * Exa : REP MOVSB.

*Processor Control Instructions

- These instr controls the processor itself.

- STC – Sets the carry flag : CF = 1
- CLC – clears the carry flag : CF = 0
- CMC – complements the carry flag :
- STD – Sets direction flag : DF = 1 — string bytes are accessed from higher memory addr to lower memory addr.
- CLD – clears direction flag : DF = 0
  - string bytes are accessed from lower memory addr to higher memory address.

Operand types

Mov AL, BL
Mov BL, 01H
Mov BX, [2405]H
Opcode
Operand

* Types of operand
  - Bytes (8)
  - Words (16)
  - Short Integers (8)
  - Integers (16)
  - Double words (32)
  - Long Integers (32)

  Variables

  - Strings — is a series of bytes / a series of words stored in sequential memory locations. (alphanumeric characters)

Operand Addressing

* Addressing modes specifies the location of the operand and also how its location may be determined.

Assembler Directives :-

* Assembler Directives are the instructions that direct the assembler to do something.

* Assembler : is a program that accepts an ALP as i/p & converts it into an object module & prepares for loading the pgm into memory for execution.
* Loader (Linker) further converts the object module prepared by the assembler into executable form, by linking it with other object modules & library modules.

* Thus the basic task of an assembler is to generate object module & prepare the loading & linking

Directives

* also called as pseudo operations that control the assembly process.
* They indicate how an operand/section of a program to be processed by the assembler. They generate & store information in the memory.

• ASSUME:

* It is used to tell the assembler the name of logical segment it should use for a specified segment

Exa:

```
ASSUME CS: CODE
ASSUME DS: DATA.
```

• EQU:

* Equates a numeric, ASCII or label to another label

Exa:

```
Data SEGMENT
Num1 EQU 50H
Num2 EQU 60H
Data ENDS.
```

* Numeric value 50_H and 60_H are assigned to Num1 & Num2.

ORG

* changes the starting offset addr of the data in the data segment.

exa: ORG 100_H

100 data1 DB 10_H.

* It can be used for code too.

PROC & ENDP:

* indicate the start and end of the procedure. They require a label to indicate the name of the procedure
* NEAR : the procedure resides in the same code segment (local)
* FAR : resides at any location in the memory.
* Exa:

```
Add PROC NEAR.

    ADD AX, BX
    MOV CX, AX
    RET

Add ENDP
```

- PROC directive stores the contents of the register in the stack.

SEGMENT and ENDS:

* indicate the start & end of a segment
* Eg: INST (instruction) SEGMENT

```
    ASSUME CS:INST, DS:DATA W
    ____
    ____
    ____
INST ENDS
```

* informs the assembler that the names of procedures labels declared after this directive have been already de in some other assembly language modules.

- DB define Byte
- DW define Word
- DD define double word
- DQ define 10 bytes.

Exa:- Data1 DB $10_H, 11_H, 12_H$.

Data2 DW 1234_H.

- DUP – Memory is reserved for use in the future

* duplicate(dup) directive creates an array and stores a zero

exa:- Data1 DB 5 DUP (0) → array of 5 elements; Zero is stored in each

Data1 DB 5 DUP(?) → Reserves memory, but leaves uninitialized.

- ALIGN:

* align directive forces the assembler to align the next segment at a address divisible by specified divisor

Exa ALIGN Number

ALIGN 2 : Storing array from an even addr

Add0 XX
Add1 YY
Add2 XX.

The data XX is aligned to the even addr.

Lock — Lock Bus (2M)

- It is used to avoid two processors from updating the same data location
- This should only be used to lock the bus prior to XCHG, MOV, IN & OUT Instructions

NOP — No operation :

- This is a do-nothing Instruction
- It is most useful for patching code segments

Assembly Language Program :-

- Writing assembly language programs for 8086 is slightly different from that of writing assembly language programs for 8085.
- In addition to the Initruction that are mea for solving the problem, some addition Instructions are required to Complete the Programs.

Assembly Language Program :-

- An 8086 assembly Language program has five Columns namely
  - Address
  - Data or Code
  - Labels
  - Mnemonics
  - Operands
  - Comments

Address :-

- The address Column is used for the address or offset of a Code byte or a data byte

Data or Code :-

- The data bytes or Code bytes are put in the data or Code Column

Labels :-

- A Label is a name which represents an address referred to in a jump or Call instruction
- Labels are put in the labels Column

Operands :-

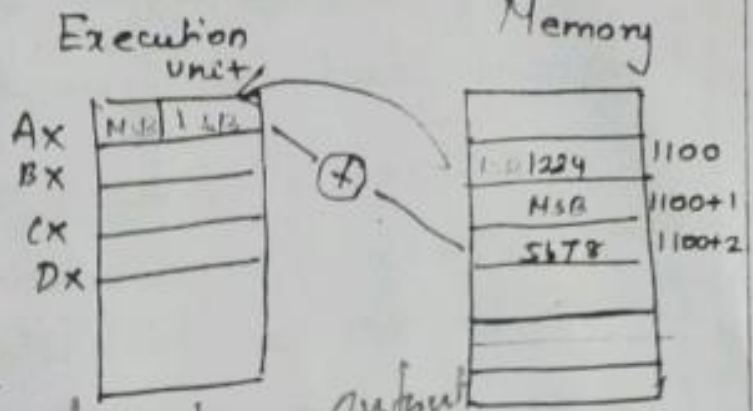
- Operands Column Contains the register memory location or data acted upon by the Instructions.

Comments: It gives space to decribe the fun of Instruction for future reference.

Assembly Language program:

Addition:-

```
Mov Ax, [1100]
ADD Ax, [1102]
Mov [1200], Ax
HLT
```



Input		Output	
1100	04	1200	07
1102	03		

Subtraction:-

```
Mov Ax, [1100]
SUB Ax, [1102]
Mov [1200], Ax
HLT
```

Input		ofp	
address	Data	addr	data
1100	04	1200	01
1102	03		

Multiplication:-

```
Mov Ax, [1100]
Mov Bx, [1102]
MUl Bx
Mov [1200], Ax
Mov [1202], Dx
HLT
```

Input	
addr	data
1100	04
1102	03

output	
addr	data
1200	12

Modular programming

• The Formulation of Complex programs from numerous Small Sequences Called Program modules (or simply modules)

• Each of which performs well-defined task. Such formulation of Computer Code is referred as Modular programming.

Linking and Relocation :-

The general process for creating and executing a program is shown in fig :-

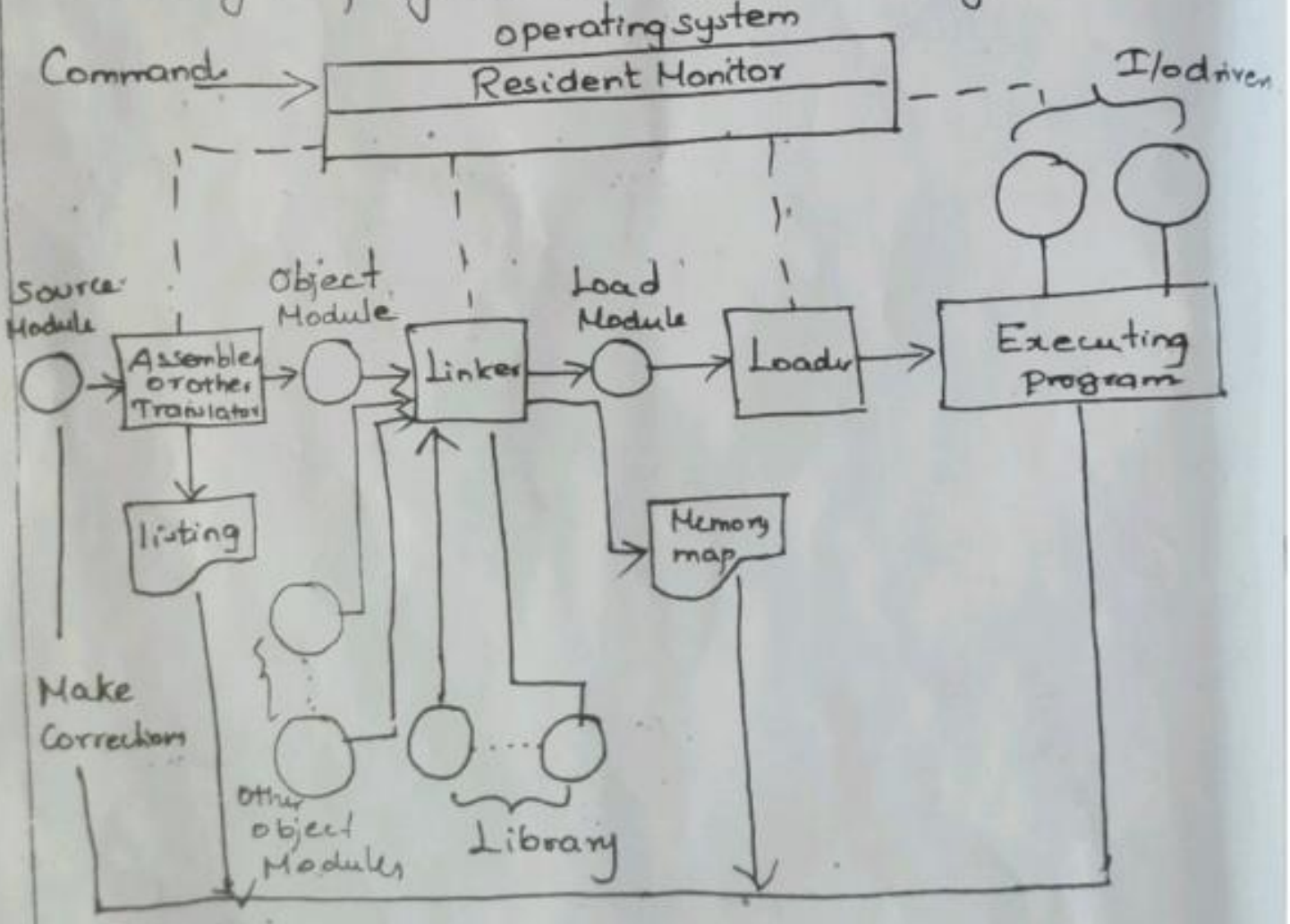


fig: Creation and Execution of a program

Modular Programming

* Programming task is divided into subtasks, Separate modules (programs) are written for performing subtasks.
* Each module is tested separately.
* The common routines required in modules are written in separate module and they are 'called' from individual modules as per requirements.
* Programming done in this fashion is called Modular programming

Advantages:-

* Code is short, simple & easy to understand
* Easier to test, design and debug.
* It is easy to do modifications
* Documentation can be easily understood.

There are two techniques are involved to combine the modules.

* control coupling : defines how and when the modules are ~~created~~ entered and exited.
* Data coupling : defines how information is communicated between modules.

* The Assembly Language components used for the development of modular program are,

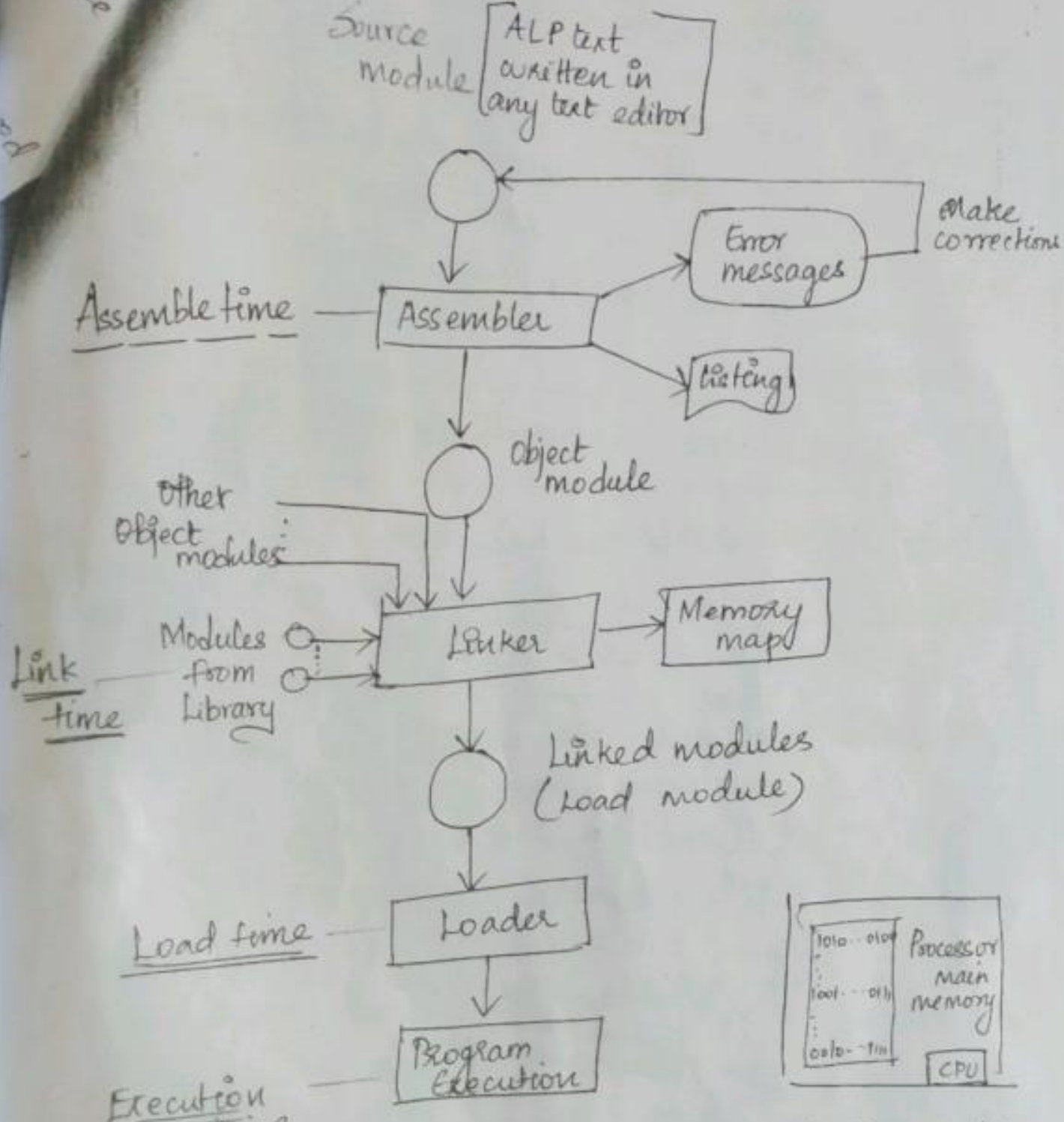
1. Structures : They allow data to be structured so that they can be readily accessed.
2. Procedures/subroutines
3. Macros.

Linking and Relocation:

* Generating machine code from Assembly Language Program manually is very time consuming and complex process.
* Thus assembly Language tools such as assembler, Linker, Loader and debugger are used to develop & execute assembly Language programs.

Steps in Program development and execution

1. Write an assembly language program which is an i/p to the assembler
2. The assembler translates the ALP to object code (Binary equivalent) * Time required to translate assembly code to object code is called assemble time.
3. * During assembling process, assembler checks for Syntax errors & display them.



Steps in program development & execution

3) At link time, the separately assembled modules are combined into one single module, by the linker.

4) At load time, the program loader copies the pgm into the computer's main memory, & at execution

time, program execution begins.

SEGMENT COMBINATION

* The Segment with different name can not be combined
* The Segment with the same name are joined together accordingly to the combine-type specified in the SEGMENT directives.

Segment name SEGMENT Combine-type

* There are 5 possible combine-types:

1. PUBLIC : Segments are concatenated into a single segment in the order specified by the Linker command.
2. COMMON : Segments are overlaid so that they have the same starting address.
3. STACK : Combined to form one large stack
4. AT : to determine the starting address of the segment in memory.
5. MEMORY : It is placed at the last of load module.

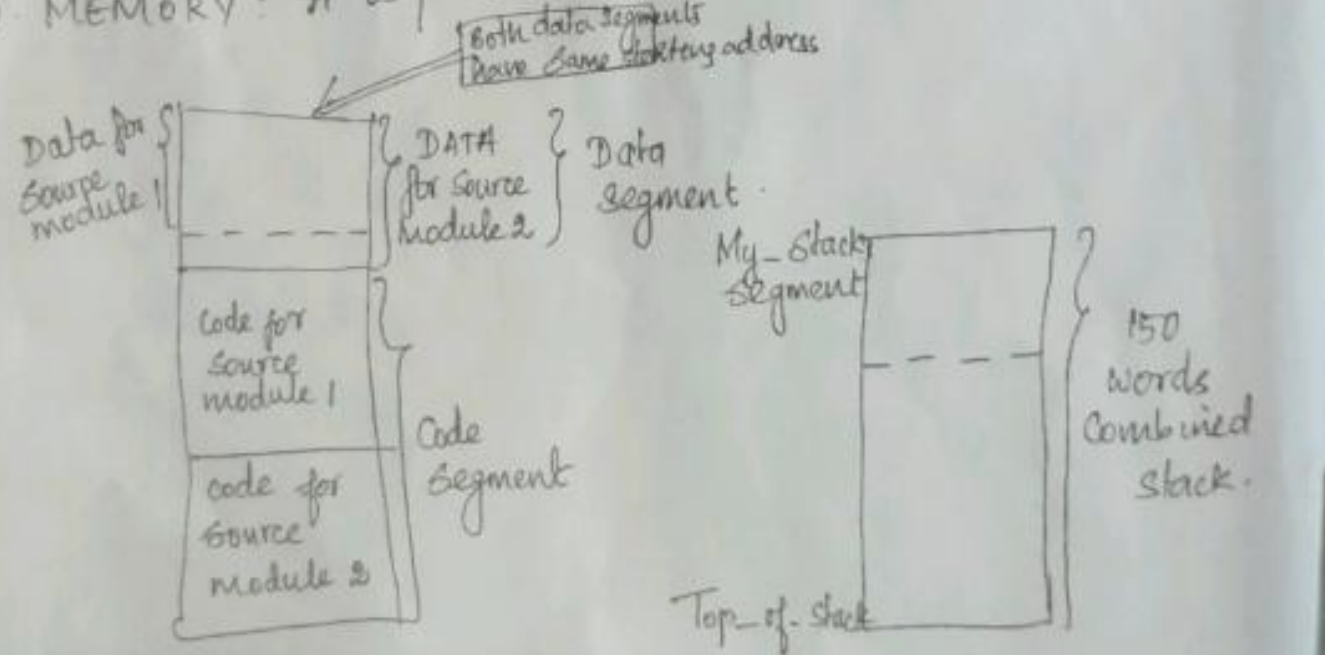


Fig : Generated load modules.

Source module 1

```
DATA SEGMENT COMMON
:
DATA ENDS
CODE SEGMENT PUBLIC
:
CODE ENDS
```

Source module 2

```
DATA SEGMENT COMMON
:
DATA ENDS
CODE SEGMENT PUBLIC
:
CODE ENDS
```

Source module 1

```
MY_STACK SEGMENT STACK
DW 50 DUP (?)
MY_STACK ENDS
:
```

Source module 2

```
MY_STACK SEGMENT STACK
DW 100 DUP (?)
MY_STACK ENDS
```

Access to External identifiers:

* The Variables and/or labels defined in the module itself are called local identifiers.
* If they are not defined in the module & defined in one of the other modules being linked, they are called external (global) identifiers.
* Two assembly directives are used: They are
  1. PUBLIC
  2. EXTRN
* If a variable/procedure is declared as PUBLIC – then the variable/procedure can be accessed from other modules.
* If a variable/procedure is not in this module but it has to access from other module, then EXTRN directive should be used.

STACK

* Stack is a portion of RAM set aside by the u for the purpose of storing information temporarily
* When the information is written on the Stack, the operation is called PUSH
* When the information is read from Stack, the operation Called pop.

Stack structure of 8086.

* 8086 has 16 bit stack pointer reg (SP)
* For stack operation, the physical address is produced by adding the content of stack pointer reg to the Segment base address in SS.

$$\text{Physical addr} = SS \times 10_H + SP$$

PUSH Operation

* The push instr decrements the stack pointer by and copies a word from some source to the loc in the stack pointed by the SP.
* Exa: PUSH AX.

Pop Operation:

* The POP inst copies a word from the stack location pointed by the stack pointer to the destination.
* After the word is copied to destination, the SP automatically incremented by two.
* Exa: POP DX

Procedures / Subroutine

* Procedure is a group of instructions stored as a separate program in the memory and it is called from the main program whenever required.
* If the procedure/subroutine is in the same code segment where the main program is stored, then it is called near procedure
* If it is in different segment, then it is called FAR procedure.

Instructions used

* CALL — Used to transfer execution to a subpgm/procedure
* RET — it will return the execution from a procedure to the next location after the CALL instr.

Reentrant Procedure :-

* In some situations it may happen that procedure1 is called from main pgm, procedure 2 is called from procedure 1 and procedure 1 is again called from procedure 2.

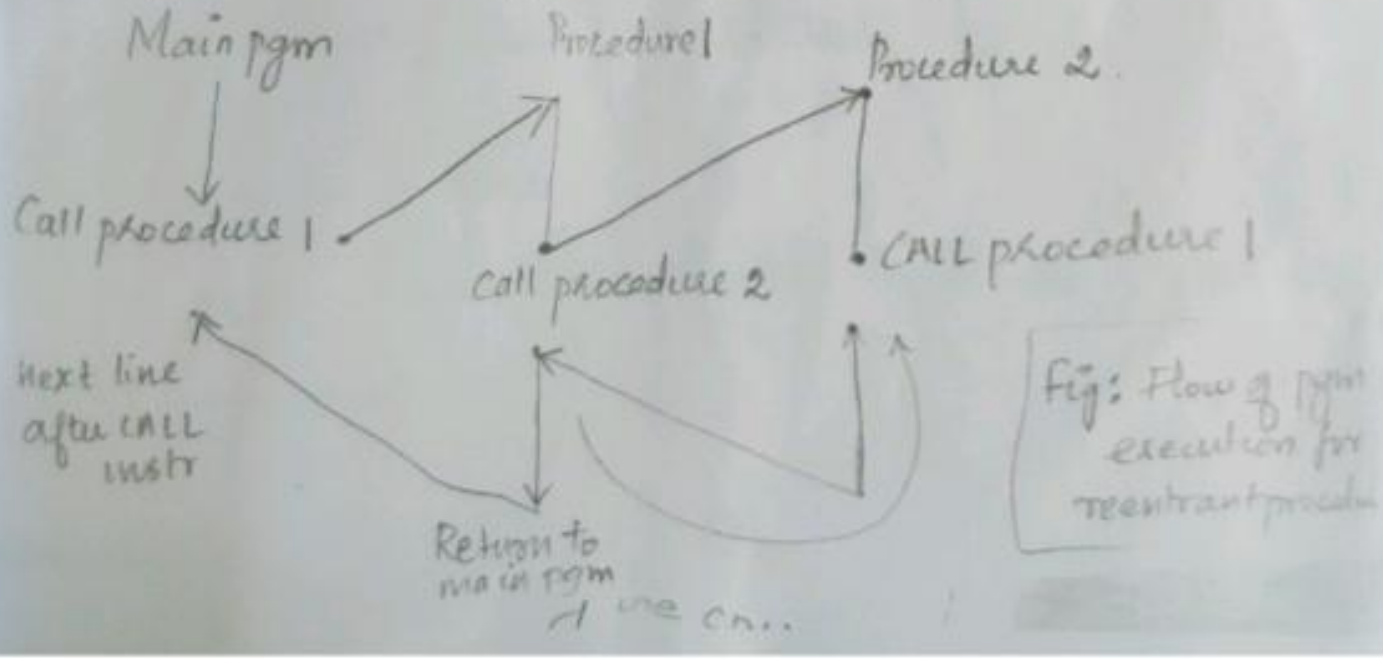


Fig: Flow of pgm execution for reentrant procedu

cont...

Recursive Procedure

* A recursive procedure is a procedure which calls itself.
* They are used to work with complex data structures called trees.

Exa: Pseudo_code

Procedure Recursive.

IF N ≠ 0

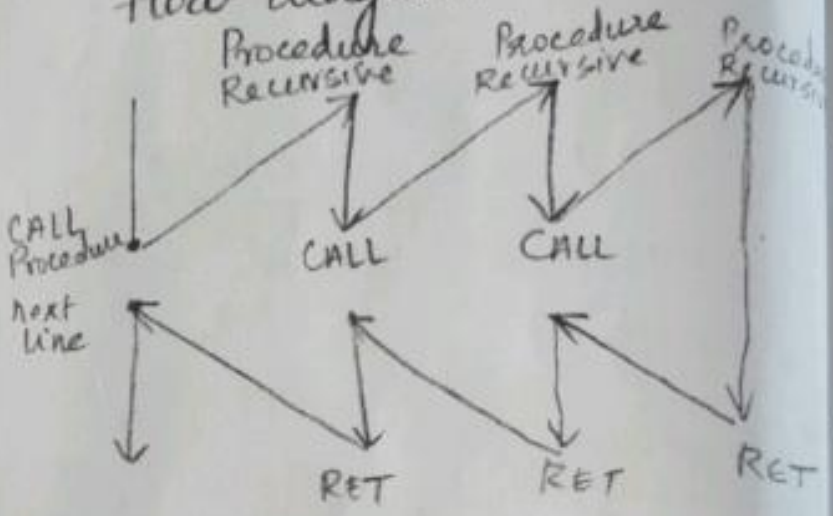
decrement N

CALL Recursive

else

Return.

Flow diagram



Passing Parameters

* there are 4 ways to pass parameters to and from the procedure.

1. Using Registers.
2. Using general memory.
3. Using pointers
4. Using Stack.

* Macro is a group of instructions
* The Macro assembler generates the code in the program each time where the marco is 'called'
* Macros can be defined by MACRO and ENDM assembler directives.
* Macro Sequences execute faster than procedures

Comparison b/o Procedure & Macro

Procedure	Macro
1. Accessed by CALL & RET instr during pgm execution	Accessed during assembly with name given to macro when defined.
2. machine code for instr is generated only once	* Machine code is generated each time when macro is called.
3. Less memory is required	* Macros require more memory.
4. Parameters can be passed in reg, memory locations or stack.	* Parameters passed as part of Statement which calls Macro.

Passing Parameters in Macro

In Macro, parameters are passed as a part of Statement which calls macro.

Example

```
.DATA
    DB 10,13,'studentname : $'
MES2 DB 10, 13,'student address :$'
.CODE
START :
MOV AX,@data  } initialise
MOV DS,AX     } data segment
PROMPT MES1
PROMPT MES2   } parameters
MOV AH,4CH
INT 21H
END START
```

dummy argument ↑

```
PROMPT MACRO MESSAGE ; with MESSAGE a parameter
MOV AH,09H
LEA MESSAGE
INT 21H
ENDM ; End macro
```

Body of macro definition

* The parameters can be passed in macro with the help of dummy argument.

Local Variable in a Macro :-

* A Local Variable defined in the Macro is available in the macro, however it is not available outside the macro
* It must be defined using LOCAL directive immediately after MACRO directive.

Nested Macros

* when macro call appears within a macro definitions, the macros are called nested macros.

Exa:-

```
INSIDE MACRO
MOV AH,09H.
ENDM

OUTSIDE MACRO
   MOV CH,CL
   INSIDE
   MOV BH,AH   ENDM
```

* In the body of the macro OUTSIDE, the macro INSIDE is called.

Interrupts and Interrupt Routines:

- It is necessary to execute one of some Special routines whenever Certain Conditions exist within a program or the Computer System by a Computer automatically
- The action that prompts the execution of one of these routines is called an "interrupt" and the routine that is executed is called an interrupt routine

There are two classes of Interrupts i.e

1) Internal Interrupts
2) External Interrupts

Internal Interrupts:-

- Internal interrupts are the interrupts which are initiated by the state of the cpu or by an instruction.

eg) Division by Zero or special Instructions.

External Interrupts:-

- External interrups are those interrupts which are caused by a signal being sent to the cpu from elsewhere in the Computer System.

eg) I/o devica to be serviced by the cpu

- An Interrupt Service routine is similar to a Procedure in that it may be branched to from any other program and a return branch is made to that program after the interrupt routine has executed.
- The PSW and the registers used by the routine must be saved & restored and the return must be made to the instruction following the last instruction executed before the interrupt.

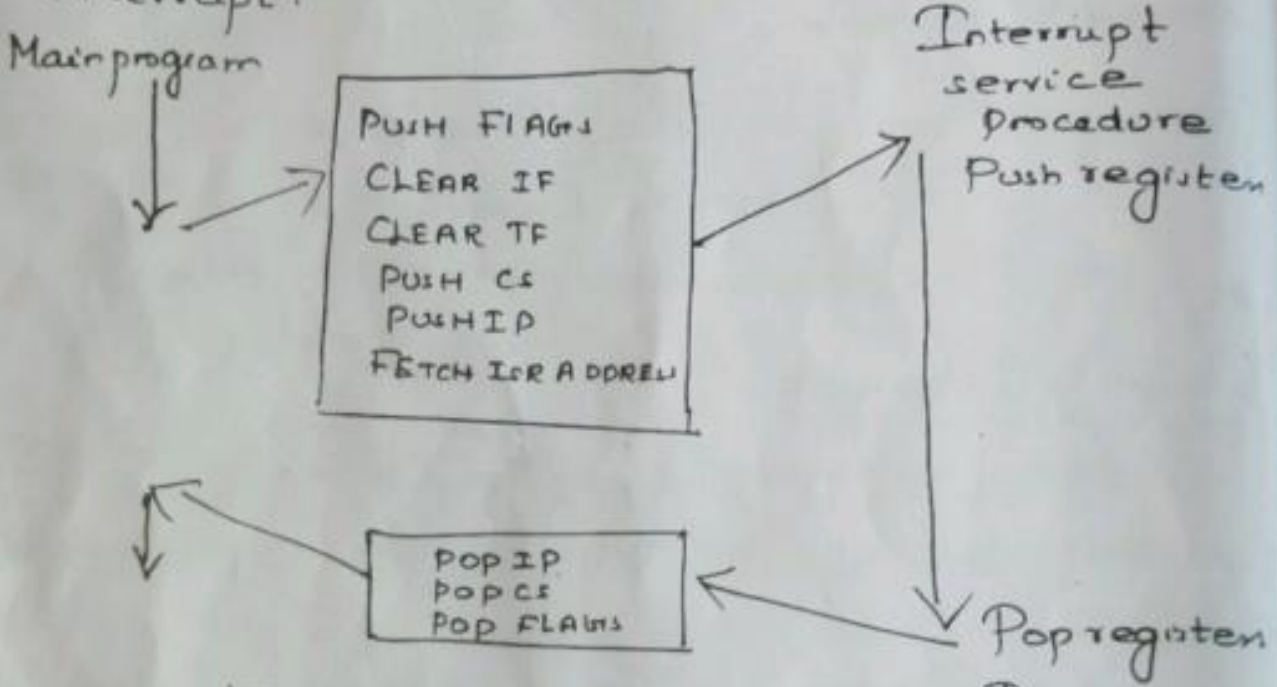


Fig: Interrupt Response.

- Type of Interrupts:
  1) Diride by zero Interrupt (Type 0)
  2) Single step Interrupt (Type 1)
  3) Non markable Interrupt (Type 2)
  4) Breakpoint Interrupt (Type 3)
  5) Overflow Interrupt (Type 4)

i) Divide by Zero Interrupt (Type 0) :-

When the quotient from either a DIV or IDIV instruction is too large to fit in the result register; 8086 will automatically execute type 0 interrupt.

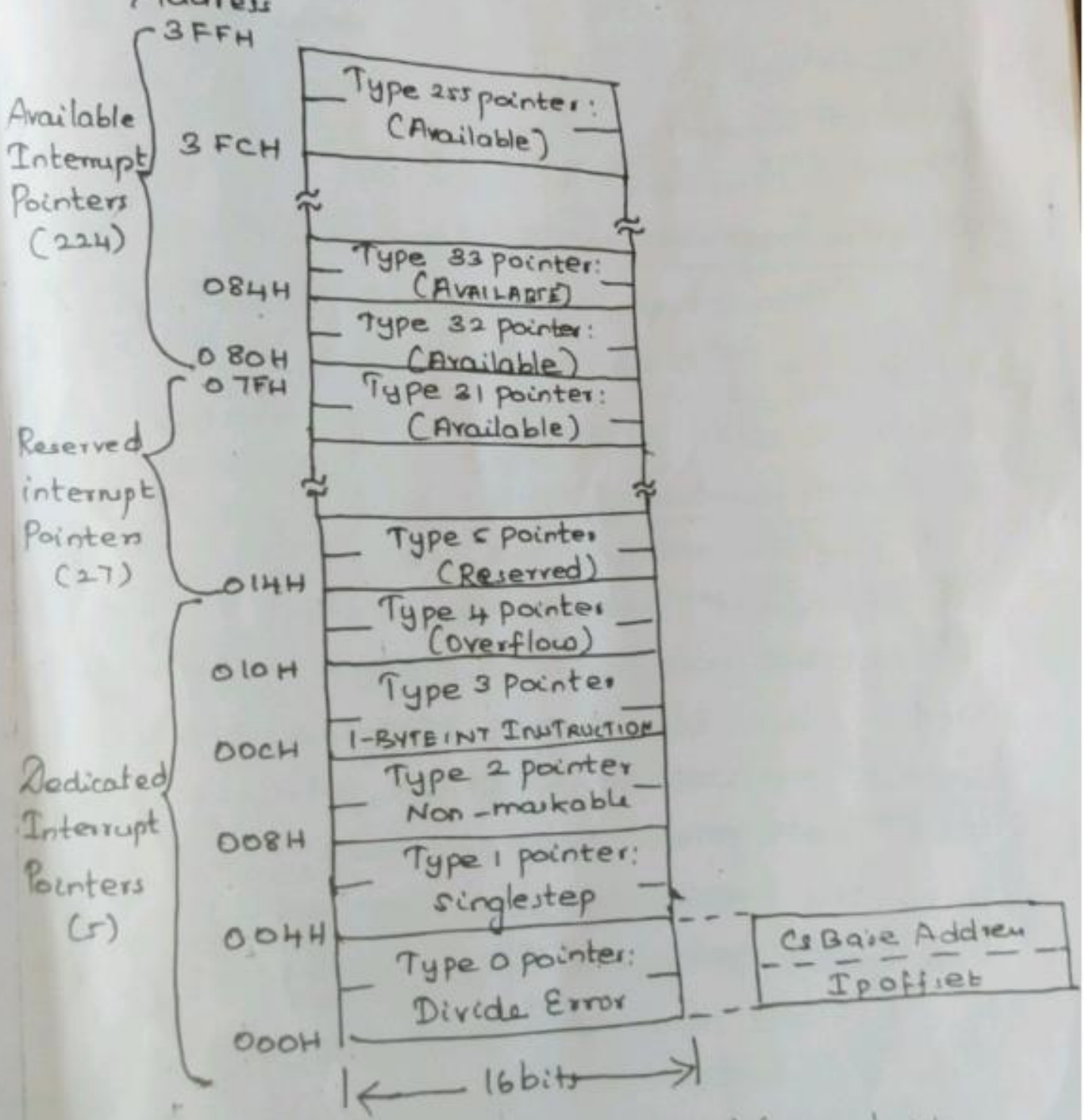


fig) 8086 Interrupt Vector table

2) Single step Interrupt (Type 1) :-

In single step mode, System will execute one instruction and wait for further direction from user. After examining the Contents & Memory location and if they are Correct then only User will tell the system to execute next Instruction. By setting the Trap flag we can enable the single step interrupt.

3) Non maskable Interrupt (Type 2)

This interrupt Cannot be disabled by any Software Instruction by setting NMI pin of 8086 mp. This interrupt will be enabled.

4) Breakpoint Interrupt (Type 3)

By the name itself tells that, it is used to implement Breakpoint function in the system Breakpoint fun is often used as a debugging aid in Cases when single stepping provides more detail than wanted. When we introduce a breakp into the system it will execute the instruction upto the breakpoint and then goes to the breakpoint procedure. This procedure can have the Contents of a reg, to be diplayed, Memory Contents and other informa required to debug the program.

5) Overflow Interrupt (Type 4)

The Type 4 Interrupt is used to check Overflow Condition after any signed arithmetic operation in the system. The 8086 Overflow flag of will be represented in the destination register or memory location.

Another way to detect and respond to an overflow error in a program is to put the jump if Overflow instruction (JO) immediately after the arithmetic Instruction.

6) Software Interrupt (Type 0-255)

The 8086 INT instruction can be used to cause the 8086 to do one of the 256 possible interrupt types. The interrupt type is specified by the number as a part of the Instruction. With the Software Interrupts we can call the desired routines from many different programs in a system

eg) BIOS in IBM PC

7) Maskable Interrupt: (INTR)

- The 8086 INTR input can be used to interrupt a program execution.
- This interrupt is implemented by using two Pins - INTR and $\overline{\text{INTA}}$.
- The interrupt can be enabled or disabled by STI (IF=1) or CLI (IF=0) respectively.

Byte and String Manupulation:-

String Instructions:

- The string Instructions can operate on only a single byte or word unless they are used with the REP Prefix & they are referred to as string primitives, or simply primitives.
- All of the primitives are 1 byte long. With bit 0 indicating whether byte (bit 0=0) or a word (bit 0=1) is being Manupulated.
- There are five basic primitives and each may appear in one of the following three forms:

Operation Operand(s)

(or)

Operation B

(or)

Operation W

- In the first form is used, whether byte or word are to be operated on is determined by the type of the operand(s)
- In the second and third forms indicate byte and word respectively.
- The actual operands are determined by the Contents of the SI, DI, Ds and Es register.

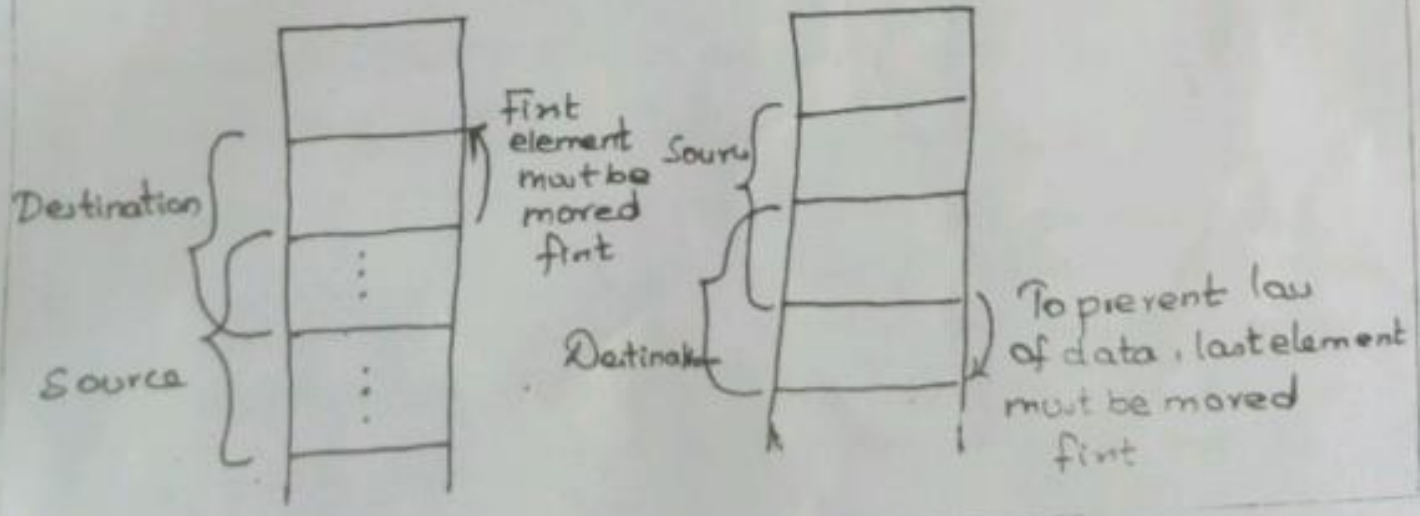
. When working with strings, the advantage of the movs and CMPs Instructions Over the Mov and CMP Instructions are:

1) They are only 1 byte long
2) Both operands are memory operands
3) Auto-Indexing obviates the need for Seperate incrementing or decrementing Instructions, thus decreasing overall processing time.

Example for String primitives

Name	Mnemonic & format	Description
Move string	Movs DST, SRC	(DI) ← (SI) Byte operands (SI) ← (SI) ± 1) (DI) ← (DI ± 1) Word operands (SI) ← (SI ± 2) (DI) ← (DI ± 2)
Move byte string	MOVSB	
Move word string	MOVSW	

Fig: Possible Cases when moving data between overlapping areas



• Source Operand :-

The address of the operand is the sum of (SI) and $(Ds) \times 16_{10}$, Unless the Ds register is Overridden by specifing the Es, cs or Ss register as the segment register

• Destination Operand :

The addres of the operand is always the sum of DI and $(Es) \times 16_{10}$

• The main reason for including an Operand(s) in a Primitive is that the inclusion of the Source (and destination) identifier(s) tends to make the program more readable.

• Using different segment register for the Source and destination allow the two string operands to reside in Separate Segments

• The primitive also Causes (SI) and or (DI) to be automatically incremented or decremented

• Auto-incrementing or auto decrementing is used is dictated by the Df flag in the PSW.

If DF=1, the index registers are auto-decremented and If DF=0, they are auto-incremented,

For Instance, if DF=0, then

MovsB

would Cause the byte in $(SI)+(Ds) \times 16_{10}$ to be moved to $(DI)+(Es) \times 16_{10}$ and the Content of both SI and DI to be incremented by 1.

. The two possible situations are for the source address to be greater than the destination address

In which case the first element in the source is to be moved first or for the source address to be less than the destination address, which requires the last element of the source to be moved first

String Comparison

```
        MOV  SI, OFFSET STRG1
        MOV  DI, OFFSET STRG2
        MOV  CX, LENGTH STRG1
        CLD
NEXT:   CMPS  STRG1, STRG2  ; Compare strG1
        JNE   EXIT          ; With STRG 2 AND
        LOOP  NEXT          ; BRANCH TO SAME IF
        JMP   NEAR PTR SAME ; They are equal
EXIT:   :
```

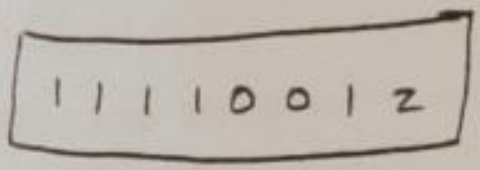
. The CMPS primitive can be used to Compare strings of bytes or words

If the two strings are equal and otherwise Continues at EXIT.

REP PREFIX:

The 8086 Machine language includes a prefix that simplifies the use of string primitives with loops.

• This prefix has the machine Code



where, for the CMPS and SCAS primitives, the z bit helps Control the loop. By prefixing MOVS, LODS and STOS, which donot affect the flags, with the REP prefix 11110011, they are repeated the no of times indicated by the CX register or until the z bit does not match the ZF flag.

• In Assembler Language the prefix is invoked by placing the appropriate repeat (REP) mnemonic before the primitive. The REP mnemonics are defined in fig

```
MOVE :   MOVS   STRING2, STRING1
         LOOP   MOVE
```

with

```
REP      MOVS   STRING2, STRING1
```

not only is the Code simplified, but the execution time is reduced from

18 + 17 = 35 Clock cycles per ibration

to

9 + 17 = 26 clock cycles.

```
NEXT :   CMPS   STRG1, STRG2
         JNE    EXIT
         LOOP   NEXT
         JMP    NEAR PTR SAME
```

Addressing modes of 8086 — unit 1

1) Immediate addressing mode :-
   * data is given in instr itself
   * MOV AX, 1234 H

2) Register addressing mode
   * Data is given through reg
   * MOV AX, BX

3) Direct addressing mode
   * Memory addr is given
   * MOV AX, [1234]H

4) Indirect addressing mode
   * register that has memory addr of data is given in instr
   * MOV AX, [BX]

5) Base plus Index Reg addr mode
   * Base reg — BX & BP
   * Index reg — DI & SI
   * MOV AX, [BX + DI]

6) Register Relative addr mode
   * MOV AX, [BX + 0003H] → displacement

7) Base plus index reg relative addressing mode
   * MOV AX, [BX + DI + 0003H]

8) String addressing mode
   DS & SI — Source string
   ES & DI — Destination string
   * MOVSB

Instruction Set of 8086 (Unit-I)

1) Data transfer instructions
2) Arithmatic and Logical instructions
3) Branch and control instructions

Data transfer Instructions :-

1) MOV – Move – MOV AX, BX
2) PUSH – Push into stack – PUSH BX
3) POP – POP from stack – Pop BX
4) XCHG – Exchange – XCHG AX, BX
5) IN – input port – IN AX, Port address
6) OUT – output port – OUT AX, Port address
7) LEA – Load effective address – LEA AX, [DI]
8) LDS – Load Data segment – LDA AX, $[1234]_H$
9) LES – Load Extra Segment – LES AX, $[1234]_H$.

Arithmatic instructions

1. ADD – Addition – ADD AX, BX
2. ADC – Addition with carry – ADC AX, BX
3. SUB – Subtraction – SUB AX, BX
4. SBB – Subtraction with borrow – SBB AX, BX
5. INC – Increment – Inc AX
6) DEC – Decrement – DEC AX
7) NEG – Negation – NEG AX
8) CMP – Compare – CMP AX, BX

Instr set 8086 - Unit - I

9) MUL - Multiplication ⇒ MUL BX
10) IMUL - Signed Multiplication ⇒ IMUL BX

11) DIV - Division ⇒ DIV BX
12) IDIV - Signed Division ⇒ IDIV BX

Logical Instructions (Bit Manipulation Instructions)

1) AND - AND operation - AND AX, BX
2) OR - OR operation - OR AX, BX
3) XOR - EXor operation - XOR AX, BX.
4) ROR - Rotate right - ROR AX
5) ROL - Rotate Left - ROL AX
6) RCR - Rotate right through Carry
7) RCL - Rotate Left through carry

Branch Instructions (Program Execution transfer instr)

Unconditional

1) JMP - Jump - JMP 2000_H
2) CALL - Call subroutine - CALL 2000_H
3) RET - return from subroutine - RET 3000_H

Conditional

1) JE - Jump if equal } ZF = 1
   JZ - Jump if zero }
   JNE - Jump if not equal } ZF = 0
   JNZ - Jump if not zero }
   JC - Jump if Carry ; CF = 1
   JNC - Jump if not carry ; CF = 0

Loop - no of times repeating is required is placed in CX register

Unit - II

8086 System Bus structure

8086 signals – Basic Configuration - System bus timing - System design using 8086 - IO programming - Introduction to multiprogramming - System Bus structure - Multiprocessor Configurations – Coprocessor, Closely Coupled and loosely Coupled Configurations - Introduction to advanced Processors.

8086 Signals (Pindiagram refer in Unit-I)

Basic Configuration:

There are two types of mode in the 8086 they are

i) Minimum mode
ii) Maximum mode

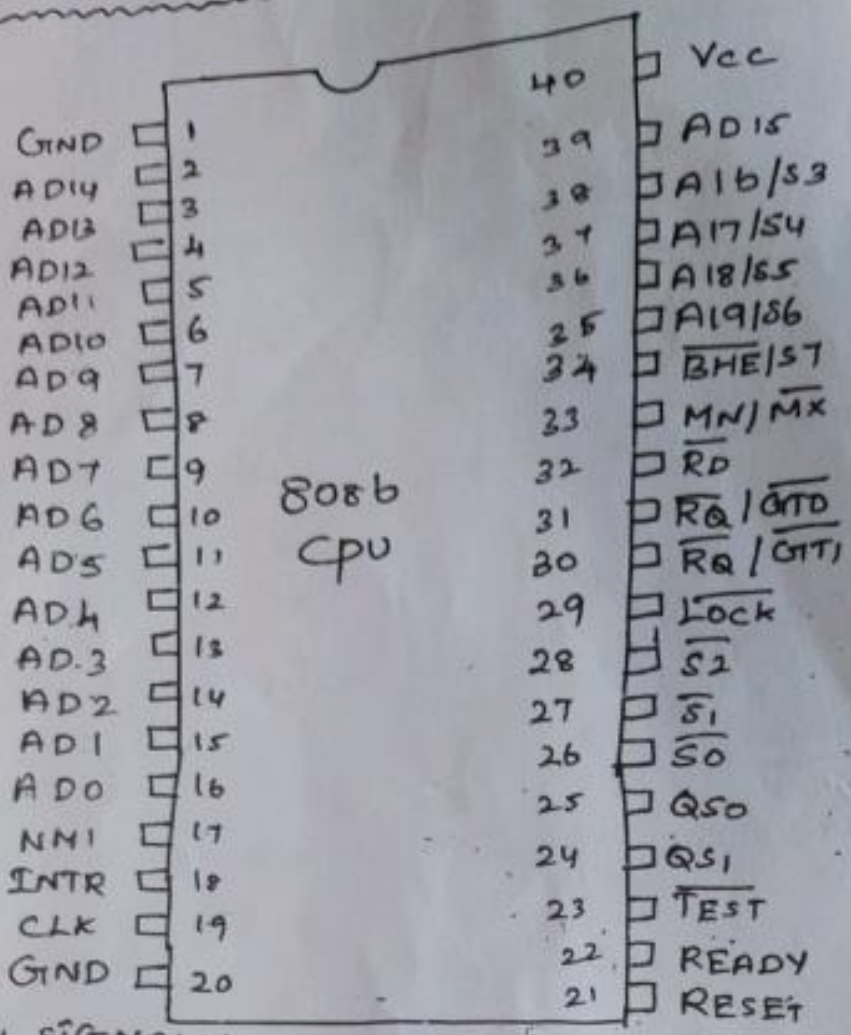
Minimum mode:

- The minimum mode is selected by applying logic 1 to the $MN/\overline{MX}$ input pin.
- This is a single microprocessor Configuration.

Maximum mode:

- The maximum mode is selected by applying logic 0 to the $MN/\overline{MX}$ input pin
- This is a multi processor Configuration.

i) Maximum mode:-



COMMON SIGNALS:

Symbol		Description
GND	–	Ground
AD14 – AD0	–	o/p address during the fint part of the bus cycle & Inputs or outputs data during the remaining part of the bus cycle
NMI	–	Nonmaskable Interrupt request – Paitive -going edge Triggered.

INTR – Maskable Interrupt request - level Triggered

Clk – Clock - 33%. duty cycle

RESET – Terminate activity clears PSW, IP, DS, SS, ES and the Instruction queue and sets CS to FFFF

READY – Acknowledgment from memory or I/O interface

$\overline{TEST}$ – Used in Conjunction with the wait Instruction

$\overline{RD}$ – Indicates a memory or I/O read is to be performed.

MN/$\overline{MX}$ – CPU is in minimum mode when Strapped to +5V and in the maximum mode when grounded

$\overline{BHE}$/S7 – If 0, the transfer is made on AD15–AD8 here Current transfer takes place, during the first part of bus cycle.

If 1, the transfer is made on AD7–AD0 status S7 is output during the latter part of bus cycle.

A19/S6 – During the first part of the
A16/S3 bus cycle the upper 4 bits of
the address are output.
During the remainder of the
bus cycle status is output S3
and S4 indicate the segment
register.

S_4	S_3	Register
0	0	ES
0	1	SS
1	0	CS or none
1	1	DS

S6 gives Current setting of IF
S6 is always 0

AD15 – Same as AD14 – AD0

Vcc – Supply voltage – ±5V ± 10%.

Maximum Mode signals (MN/$\overline{MX}$ = GND)

Symbol	Description
Q_{S1}, Q_{S0}	Reflects the status of the Instruction queue
$\overline{S0}, \overline{S1}, \overline{S2}$	Indicates the type of transfer to take place during the current bus cycle

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
0	0	0	– Interrupt acknowledge
0	0	1	– Read I/O port
0	1	0	– Write I/O port
0	1	1	– Halt
1	0	0	– Instruction fetch
1	0	1	– Read memory
1	1	0	– Write memory
1	1	1	– Inactive – passive

$\overline{Lock}$ – Indicates the bus is not to be relinquished to other bus masters. It is initiated by a lock Instruction Prefix.

$\overline{RQ/GT_1}$ – for inputting bus requests and Outputting bus grants

$\overline{RQ/GT_0}$ – Same as $\overline{RQ}/\overline{GT_1}$ except that a request on $\overline{RQ/GT_0}$ has higher priority.

ii) Minimum Mode :-

fig : pin definition for Minimum mode

Symbol	Description
$\overline{INTA}$	Indicates recognization of an interrupt request.
ALE	Indicate an address is available on address pin.

$\overline{DEN}$ – output during the latter Portion of the bus cycle and is to inform the transceiver that the cpu is ready to send or receive data.

DT/$\overline{R}$ – Indicates to the set of transceiver where they are to transmit (1) or receive (0) data.

M/$\overline{IO}$ – Distinguishes a memory transfer from an I/o transfer

$\overline{WR}$ – When 0, it indicates a write operation is performed

HLDA – Outputs a bus grant to a requesting master

HOLD – Receives bus request from bus master.

fig shows the pin assignment for the minimum mode of 8086 pin diagram

Min mode

System Bus Timing :-

I) Bus Timing for Minimum mode:

Timing for Read and write operation :

The Timing diagrams for input and Output transfer for 8086 minimum mode.

i) When processor is ready to initiate the bus cycle it applies a pulse to ALE during T_1 Before the falling edge of ALE, the address, BHE, M/IO, DEN and DT/R must be stable (i.e) DEN = high &

$DT/\bar{R}=0$ for input or $DT/\bar{R}=1$ for output.

ii) At the Trailing edge of ALE, ICs 74LS373 or 8282 latches the address.

iii) During T_2 the address signals are disabled and S_3-S_7 are available on $AD_{16}/S_3 - AD_{19}/S_6$ and $\overline{BHE}/S_7$. Also $\overline{DEN}$ is lowered to enable Transceiver

iv) In case of Input operation, $\overline{RD}$ is activated during T_2 and AD_0 to AD_{15} in high impedence

v) If Memory or I/o interface can perform the transfer Immediately. There are no waitstates and data is output on the bus during T_3.

vi) After the data is accepted by the processor $\overline{RD}$ is raised high at the begining of T_4.

vii) Upon detecting the transition during T_4, the memory or I/o device will diable its data signals.

viii) For output operation, processor applies $\overline{WR}=0$. and then data on the data bus during T_2.

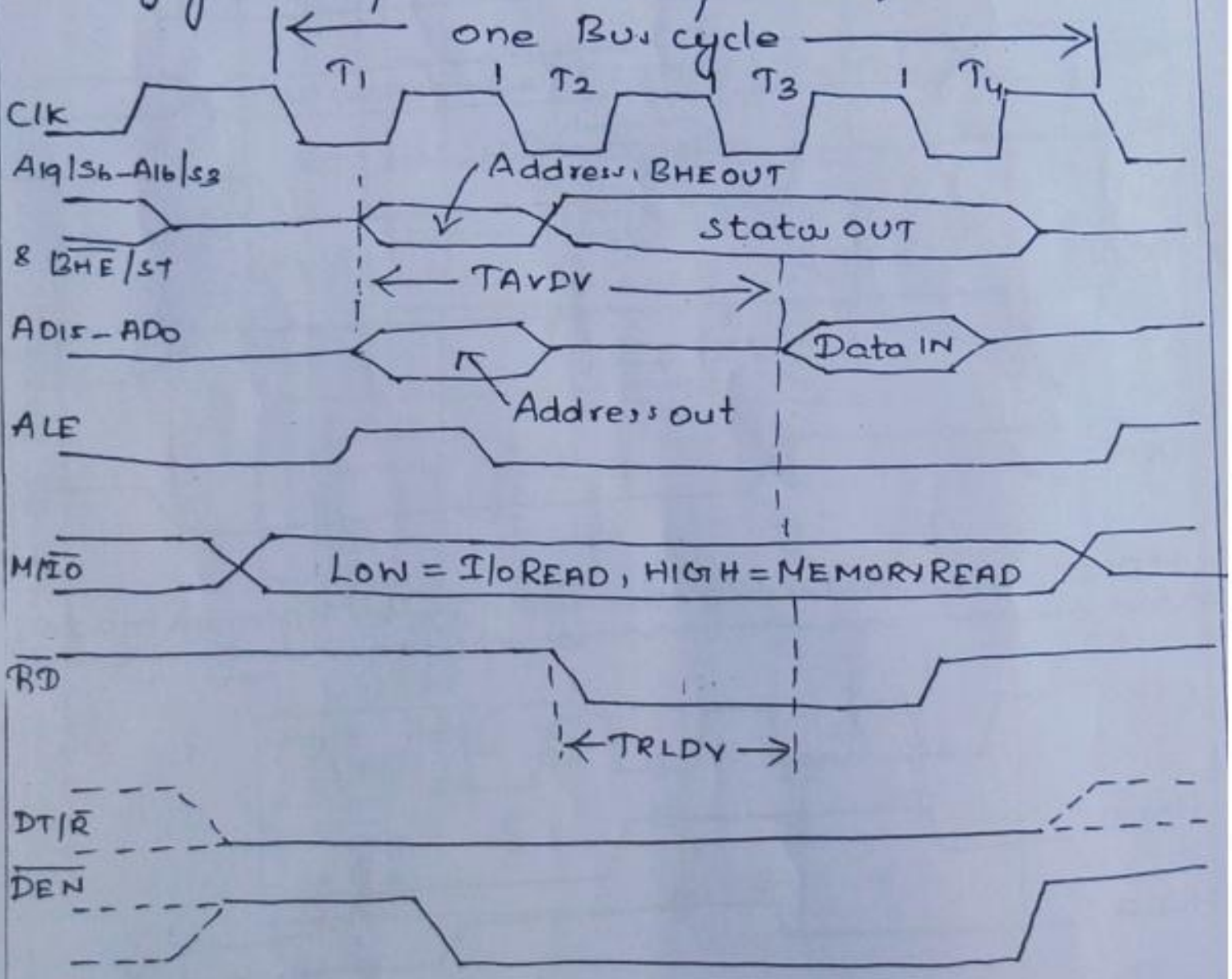
ix) In T_4, $\overline{WR}$ is raised high & data signals are disabled.

x) For either input or output operation, $\overline{DEN}$ is raised during T_4 to disable the Transceiver.

and also $M/\overline{IO}$ is set according to the next transfer during next T_1 state. Thus the length of the bus cycle in 8086 is four clock cycle.

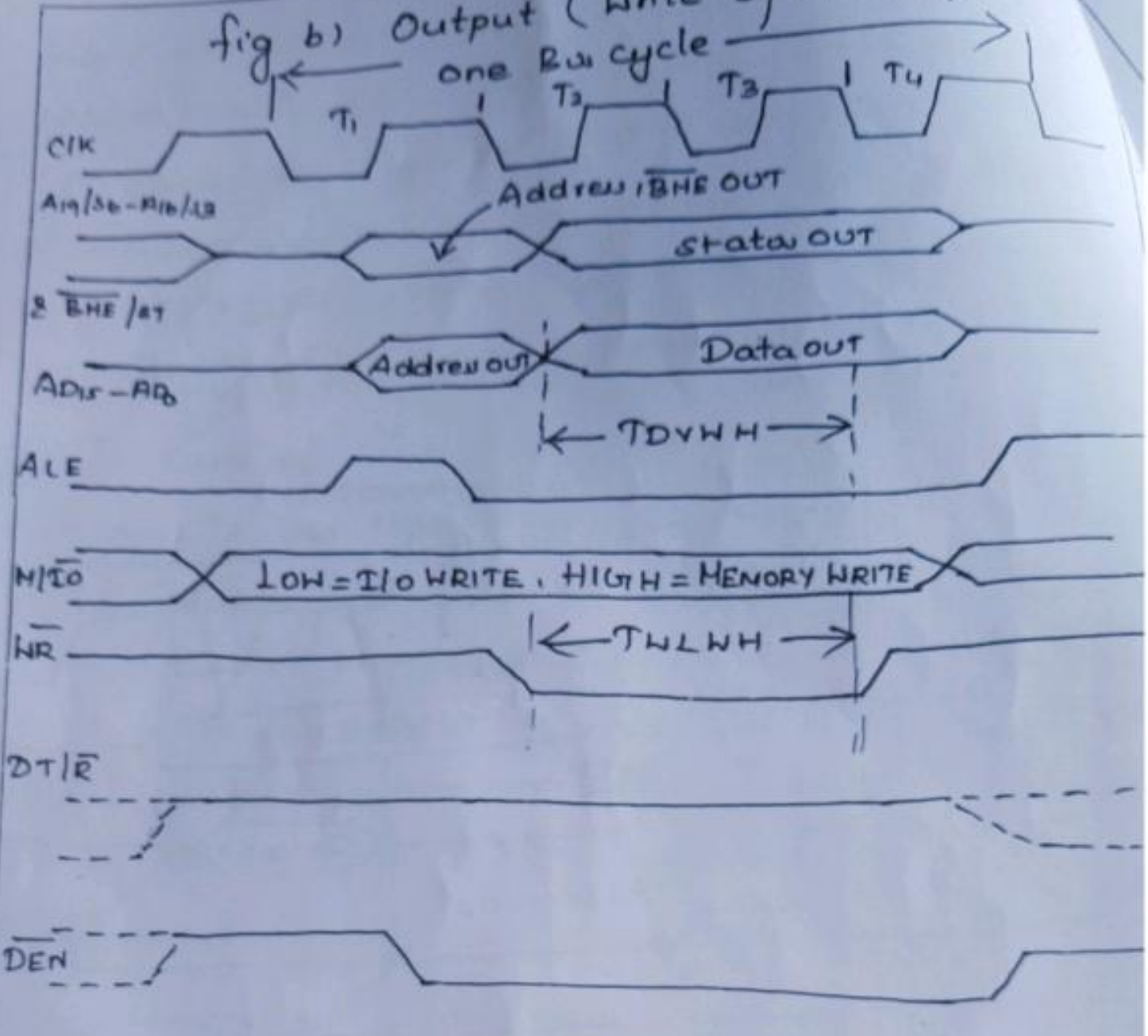
xi) When the memory or I/o device is not able to respond quickly during transfer, wait states (T_W) are inserted between T_3 and T_4 by disabling the ready input.

fig: Input (read operation)

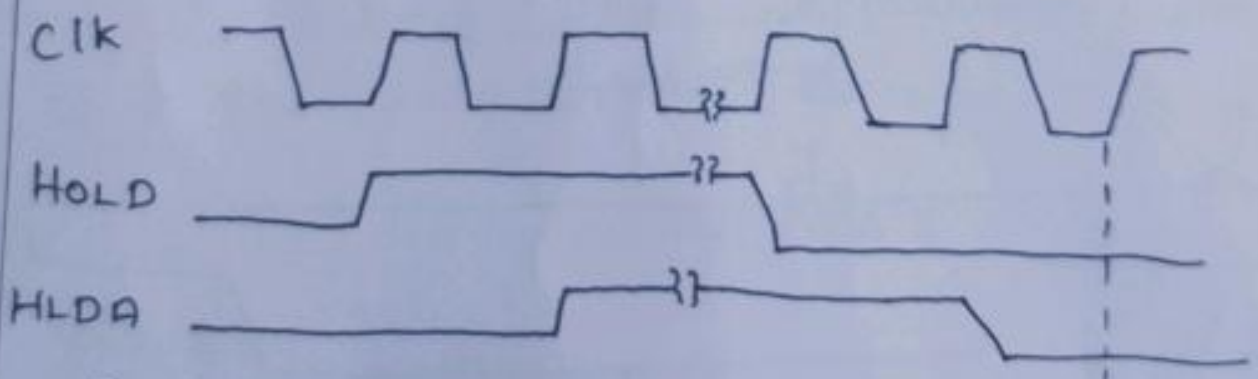


It shows the input (read operation)

fig b) Output (write operation)



HOLD and HLDA Signal Timings in Minimum mode



It shows the HOLD and HLDA signal timing in Minimum mode system.

Minimum Mode of 8086:

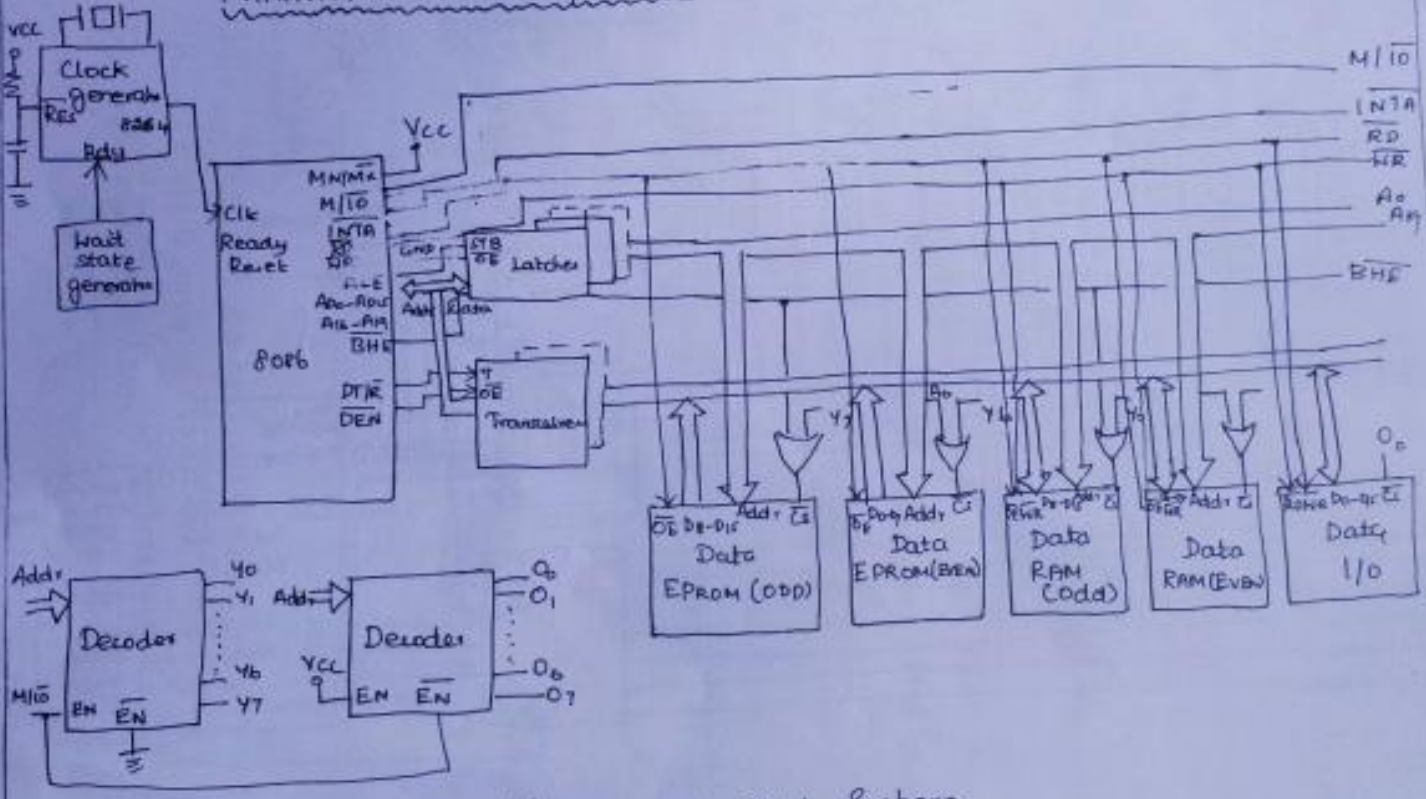


Fig) Minimum mode 8086 System

Maximum mode of 8086:

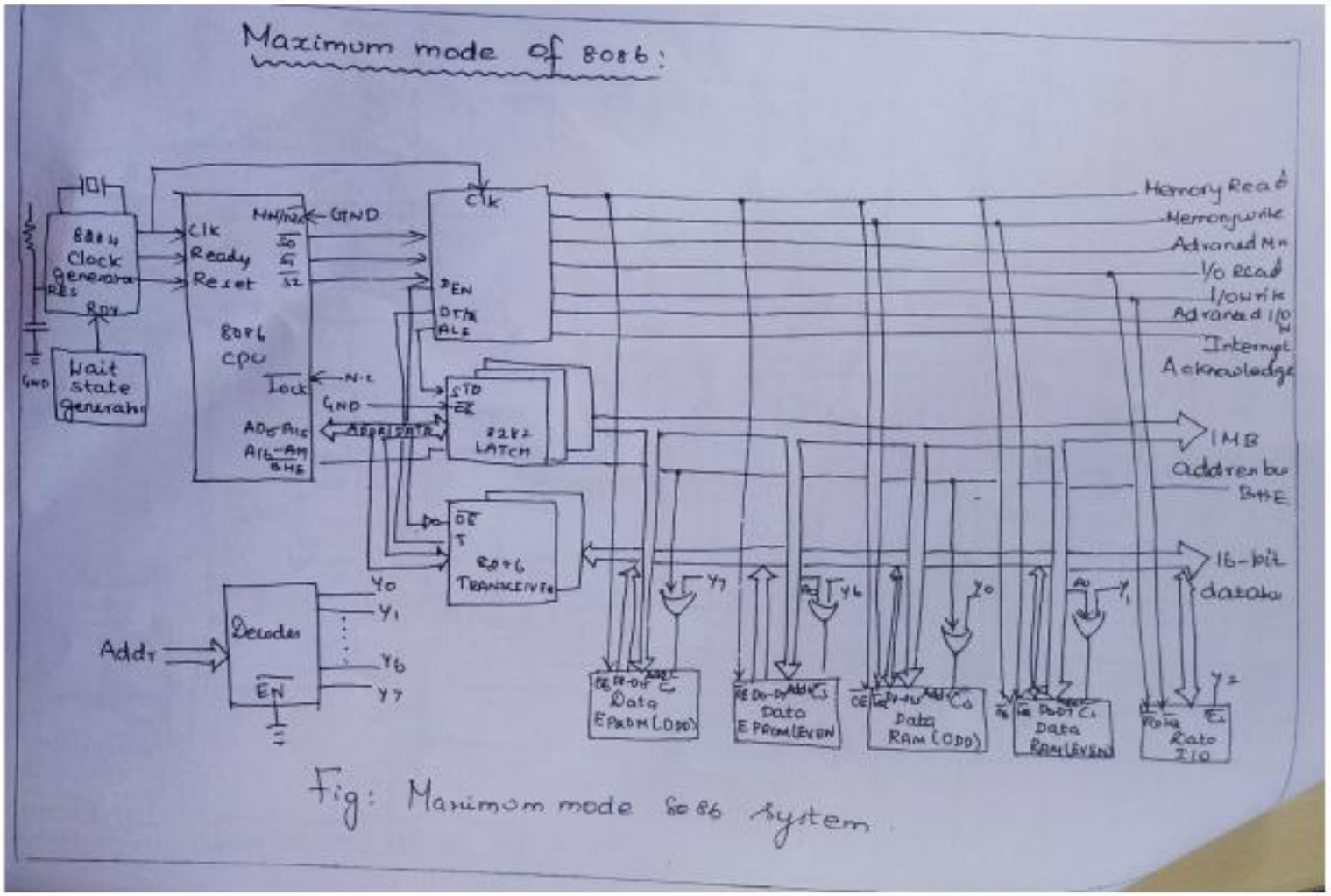


Fig: Maximum mode 8086 System

- The HOLD Pin is Sampled at leading edge of each clock pulse
- The 8086 activate HLDA in the next clock Cycle and for succeeding bus cycles. It Control of all buses and Control buses handled over to the requesting master.
- The Control of bus is not regained by 8086 Until the requesting master doesnot inactive the HOLD PIN.
- After inactivation of HOLD s/l, 8086 regains the Control of buse & inactivates the HLDA s/l.

II. Bus Timings for Maximum mode

Timing for Read and Write operation

i) $S_0, \overline{S_1}, \overline{S_2}$ are set at the begining of bus cycle. on detecting on change in passive state $\overline{S_0} = \overline{S_1} = \overline{S_2} = 1$,

ii) The 8288 bus Controller will output a pulse on its ALE and applied required s/l to DT/$\overline{R}$ Pin during T_1

iii) In T_2, 8288 will set DEN=1, thus enabling transceiver

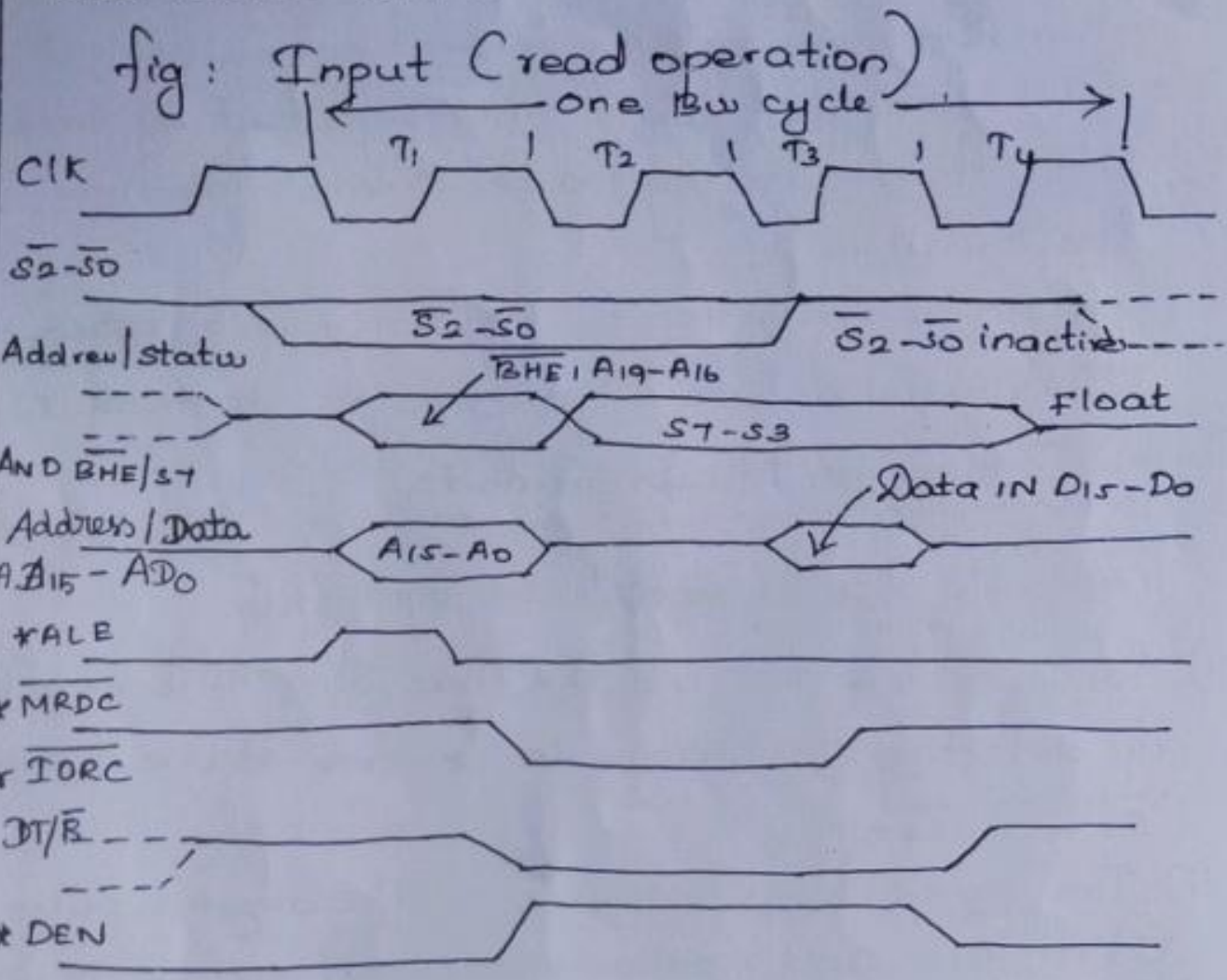
For an Input, 8288 will activate $\overline{MRDC}$ or $\overline{IORC}$. These signals are activated until T_4.

For an output, the $\overline{AMWC}$ or $\overline{AIOWC}$ is activated from T_2 to T_4 & $\overline{MWTC}$ or $\overline{IOWC}$ is activated from T_2 to T_4.

iv) The status bits $\overline{S_0}$ to $\overline{S_2}$ remain active Until T_3. and become Passive during T_3 & T_4.

v) If ready input is not activated before T_3, Wait state will be inserted between T_3 & T_4

fig: Input (read operation)



It shows the timing diagram of Input transfer of Maximode on the fig. It performs on read operation.

fig: output (write operation)

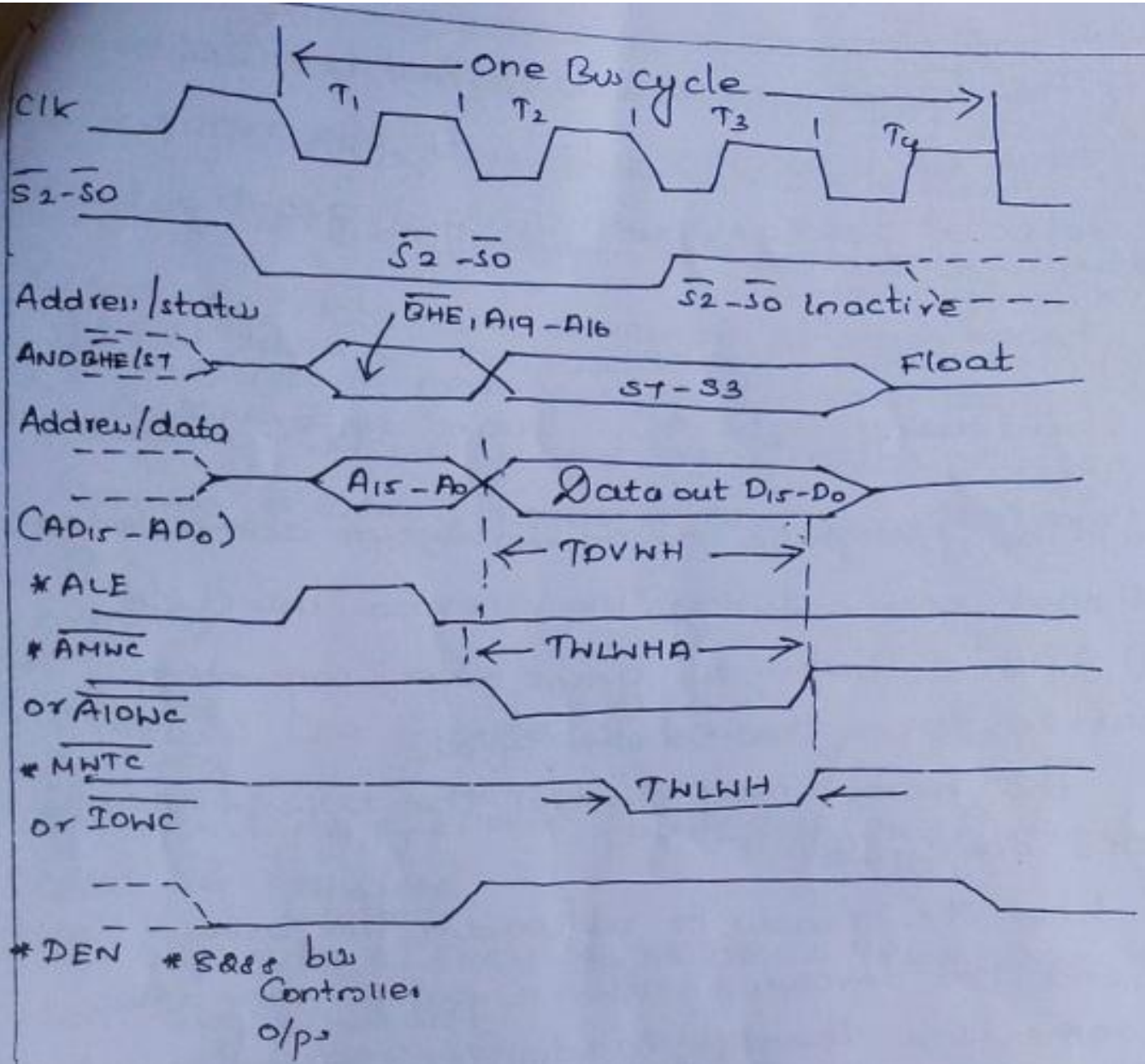
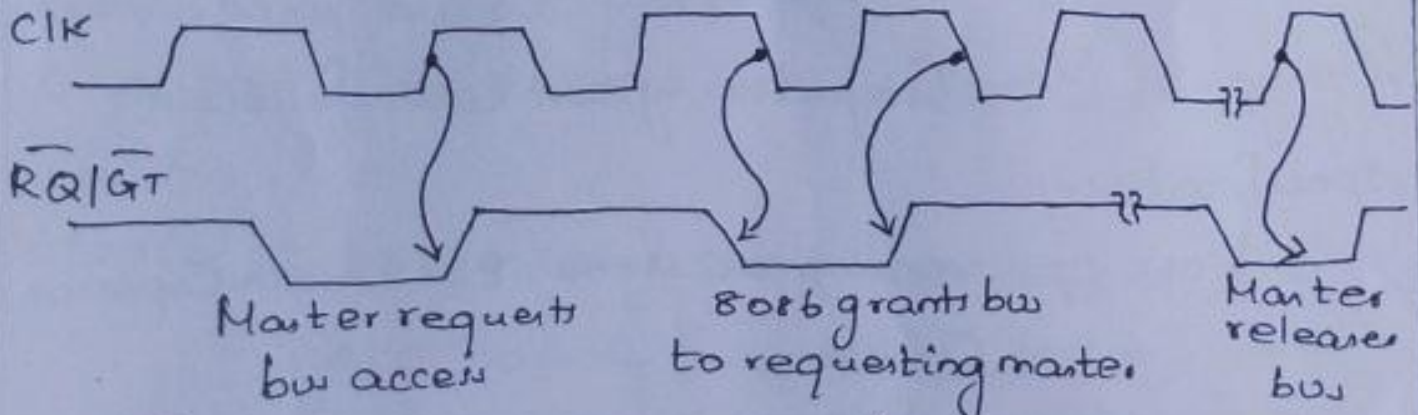


fig: Timing for $\overline{RQ}/\overline{GT}$ signals:



It shows the timings for bus request and bus grant signals.

i) The $\overline{RQ}/\overline{GT}$ pins are sampled at the rising edge of each clock pulse and if a request is detected the 8086 will apply a grant pulse to the $\overline{RQ}/\overline{GT}$

ii) The Grant signal will not be sent until the second byte of a word reference is accessed.

iii) The first pulse of an Interrupt acknowledgment didnot occur during the previous bus cycle.

iv) After activation of grant pulse, requesting master takes the control of bus.

This master may control the buse for only one bus cycle

When it is ready to relinquish the buses it will send the processor release pulse over the same line then it made its request.

System Design Using 8086:-

Design a Micro Computer System with following specifications:

a) 8086 cpu working at 5MHz. 8087 math-Coprocessor and Control Circuitry

b) 32k byte of EPROM to Store operating software Using 8k x 8 (2764) memory device

c) 64k byte of SRAM Using (6264) Memory chips and suitable decoding Circuitory

d) Input output device using PPI

1. Two 8 bit programmable input port
2. Two 8 bit programmable handshake outputport

Draw neat schematic diagram of each board, draw System memory and I/O map design decoding Circuitry and explain the System Assume memory requires one waitstate for the Cpu.

Soln:

step ①: Cpu operates at 5MHz with 8087

For 32k bytes of EPROM, four 2764 (8k x 8) memories will be required

For 64k bytes of SRAM, eight 6264 (8k x 8) memories will be required.

8255(1) required for two 8 bit programmable i/p port

8255(2) required for two 8 bit programmable handshake output port.

One port from 8255(1) will not be used. It will be Nc.

step ②: EPROM and SRAM has A_0 to A_{12} lines

8255 has A_0 and A_1 lines only

Map of 8086 System

Table1: Map of the system

A_{19} A_{18} A_{17} A_{16}	A_{15} A_{14} A_{13} A_{12}	A_{11} A_{10} A_9 A_8	A_7 A_6 A_5 A_4	A_3 A_2 A_1 A_0	Up lines
	A_{13} A_{12} A_{11}	A_{10} A_9 A_8 A_7	A_6 A_5 A_4 A_3	A_2 A_1 A_0	EPROM chip
	A_{14} A_{13} A_{12} A_{11}	A_{10} A_9 A_8 A_7	A_6 A_5 A_4 A_3	A_2 A_1 A_0	SRAM lines
Memory decoding					
	Required to decode I/O devices				

Address:

EPROM : F8000H onwards

SRAM : 00000H onwards

8255(1) : 0000H | 0002H | 0004H | 0006H

8255(2) : 0001H | 0003H | 0005H | 0007H

Decoder logic will use A_{19} to A_{14} address lines only.

step ③ : PAL decoder logic:

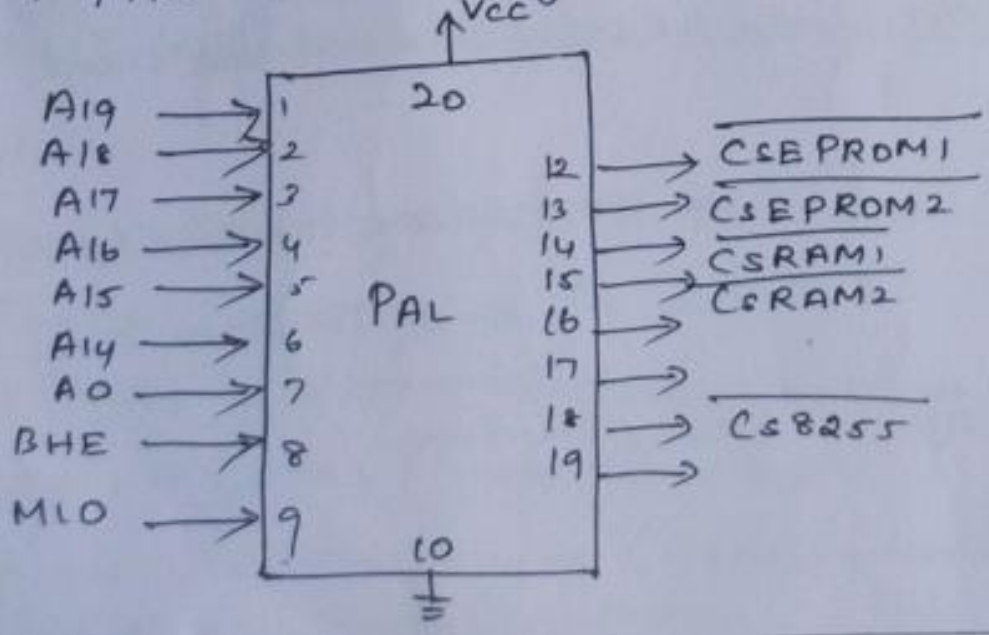


Table : 2

A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	Device enabled
1	1	1	1	1	0	EPROM set 1
1	1	1	1	1	1	EPROM set 2
0	0	0	0	0	0	RAM set 1
0	0	0	0	0	1	RAM set 2
x	x	x	x	0	0	8255 (1) / (2)

Step 4: Equations:

$\overline{CSEPROM1} = A_{19} \cdot A_{18} \cdot A_{17} \cdot A_{16} \cdot A_{15} \cdot \overline{A_{14}} \cdot M\overline{IO}$

[Memory device]

$\overline{CSEPROM2} = A_{19} \cdot A_{18} \cdot A_{17} \cdot A_{16} \cdot A_{15} \cdot A_{14} \cdot M\overline{IO}$

$\overline{CSRAM1} = \overline{A_{19}} \cdot \overline{A_{18}} \cdot \overline{A_{17}} \cdot \overline{A_{16}} \cdot \overline{A_{15}} \cdot \overline{A_{14}} \cdot M\overline{IO} \cdot A_0 +$

$\overline{A_{19}} \cdot \overline{A_{18}} \cdot \overline{A_{17}} \cdot \overline{A_{16}} \cdot \overline{A_{15}} \cdot \overline{A_{14}} \cdot M\overline{IO} \cdot \overline{BHE}$

$\overline{CSRAM2} = \overline{A_{19}} \cdot \overline{A_{18}} \cdot \overline{A_{17}} \cdot \overline{A_{16}} \cdot \overline{A_{15}} \cdot A_{14} \cdot M\overline{IO} \cdot A_0$

$+ \overline{A_{19}} \cdot \overline{A_{18}} \cdot \overline{A_{17}} \cdot \overline{A_{16}} \cdot \overline{A_{15}} \cdot A_{14} \cdot M\overline{IO} \cdot \overline{BHE}$

$\overline{CS8255} = + \overline{A_{15}} \cdot \overline{A_{14}} \cdot \overline{M\overline{IO}} \cdot \overline{A_0} + \overline{A_{15}} \cdot \overline{A_{14}} \cdot \overline{M\overline{IO}} \cdot \overline{BHE}$

(IO device)

Step 5: Circuit diagram:-

Here 8086 operates in a Maximum mode

∴ 8086 Interface is standard.

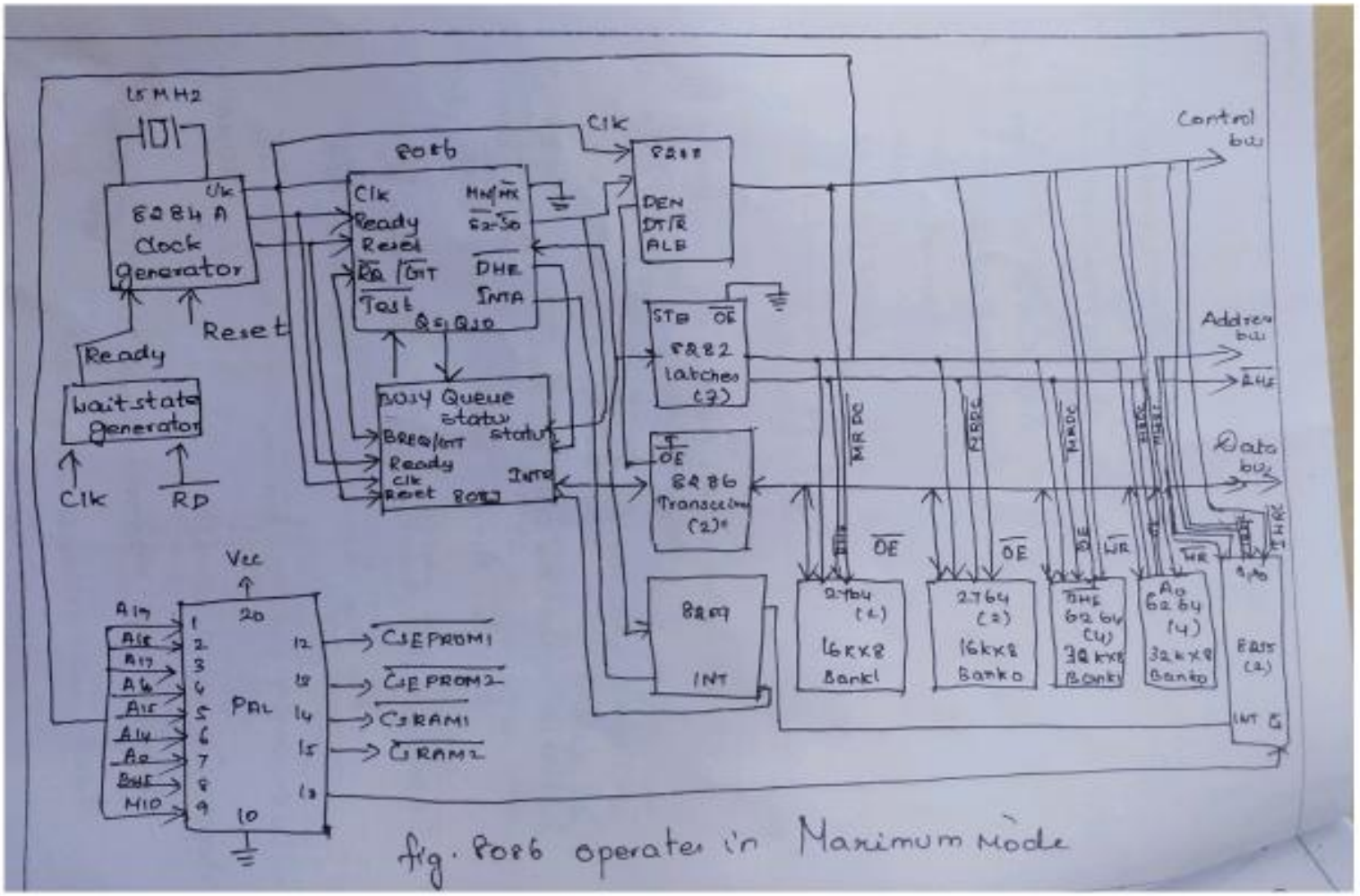
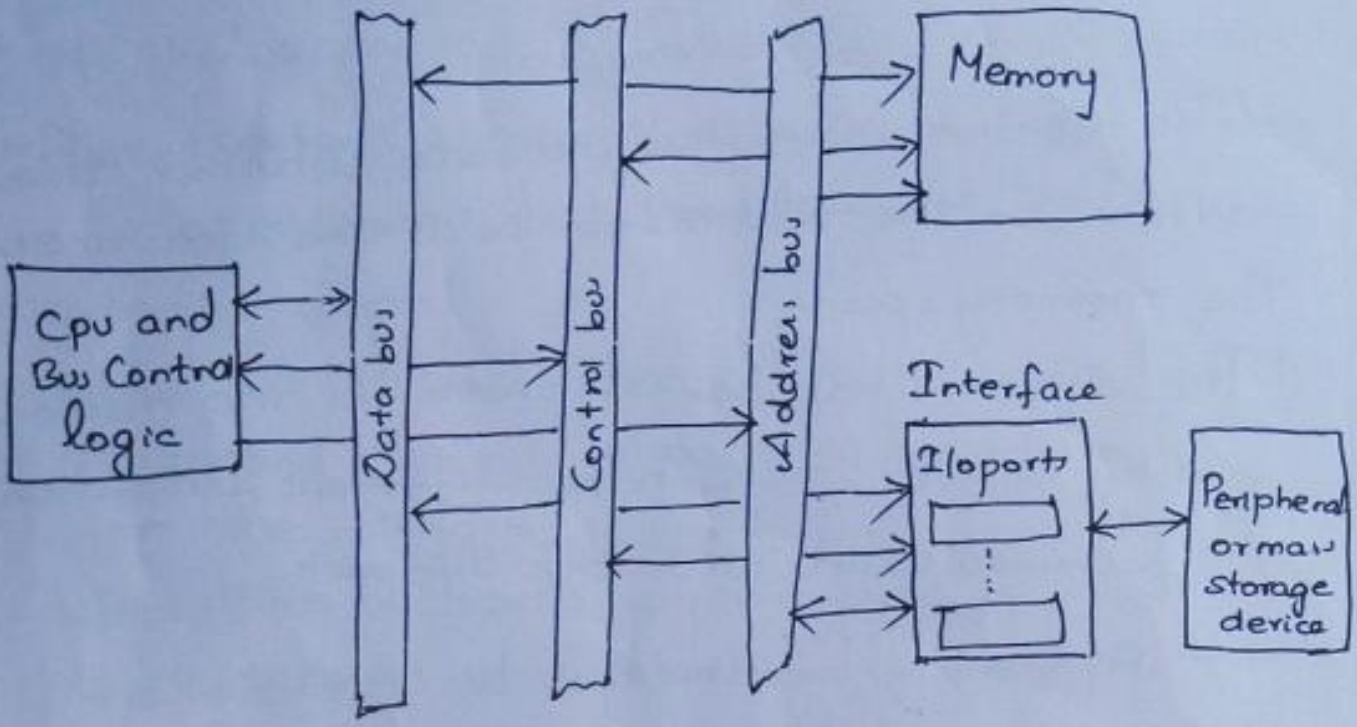


fig. 8086 operates in Maximum Mode

Ilo programming:-

The Basic Architecture of a Single bus Computer system.



- All peripheral and mass storage devices are Connected to the system bus through interfaces.
- Each interface Contains a set of registers, called Ilo ports, through which the cpu and memory Communicate with the interface's external device
- Some of the ports are for buffering data to and from the cpu and memory
- Some are for holding status information about the device and interface so that it can be examined by the cpu
- & some are for retaining the Commands sent

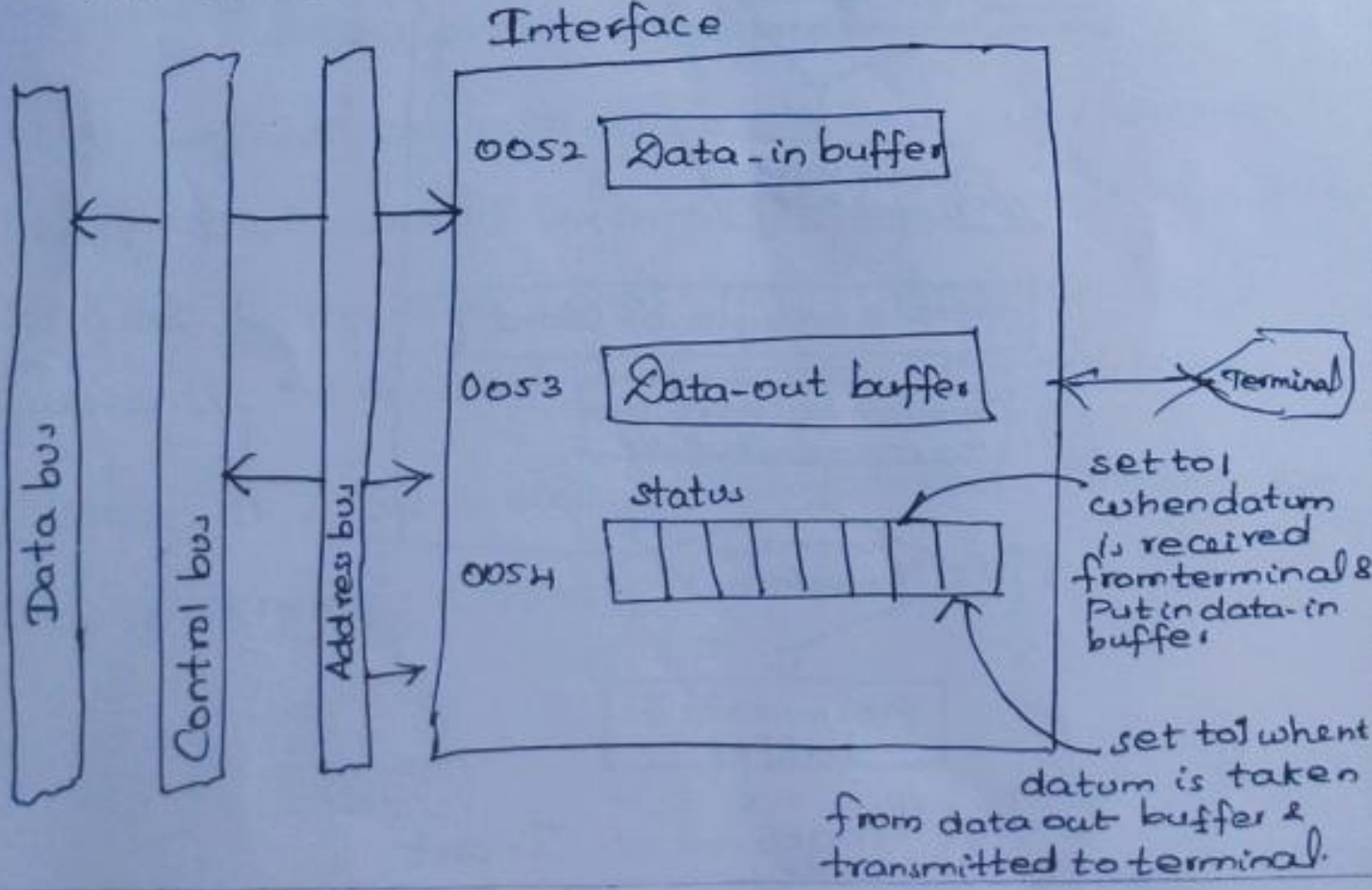
from the cpu to Control the actions taken by the interface and devices.

- The 8086-based systems, permit the establishment of two address space
  1) I/O space
  2) Memory space.
- The Control bus that indicate whether the address on the address bus is in the I/O space or the memory space.
- The system that has seperate I/O and memory address spaces must have different Instructions for Communicating with the I/O ports.
- An output from the cpu to a control or bus port is made by putting the address on the address bus
- An input from a input port is accomplished by putting the address and Control signals on their respective buses
- If an Interface has four ports, it must be designed to accept four addresses.
- The Types of I/O
  i) Programmed I/O
  ii) Interrupt I/O
  iii) Block Transfer & DMA

Interface for the programmed I/O Examples:

The address of the data-in buffer register being 0052 of the data-out buffer being 0053, and of the status register being 005H. A 1 in bit 1 of the status regiter Indicates that the data-in buffer register Contains a byte to be input and a 1 in bit 0 indicates that the data-out buffer regiter is empty.

- When Bit 1 is set when a byte is input from the device and Cleared when the byte is input from the interface.
- When Bit 0 is Cleared when a byte is output to the interface and set when it is output to the device.



i) Programmed Ilo :

- Programmed Ilo Consists of Performing an Ilo operation with the interface when its status indicates that it has data to be input or its data-out buffer register is ready to receive data from the Cpu.

A Typical programmed input operation is flowcharted. The flow chart assume that a sequence of bytes or words is to be input and each byte or word is brought into the cpu.

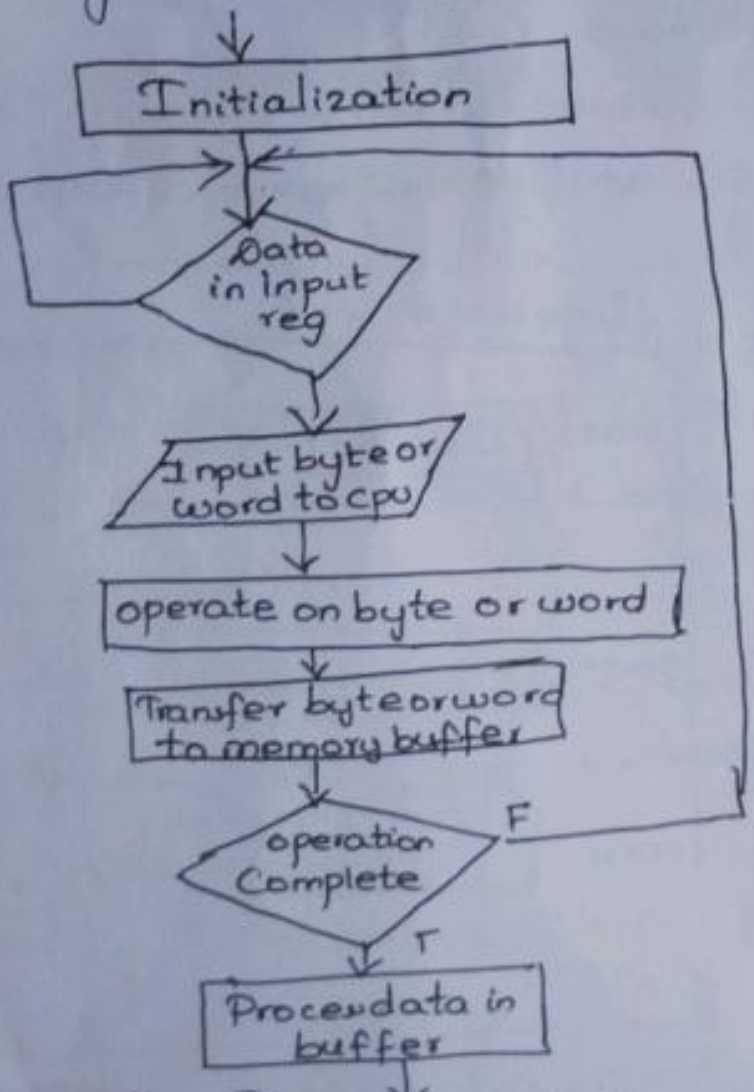
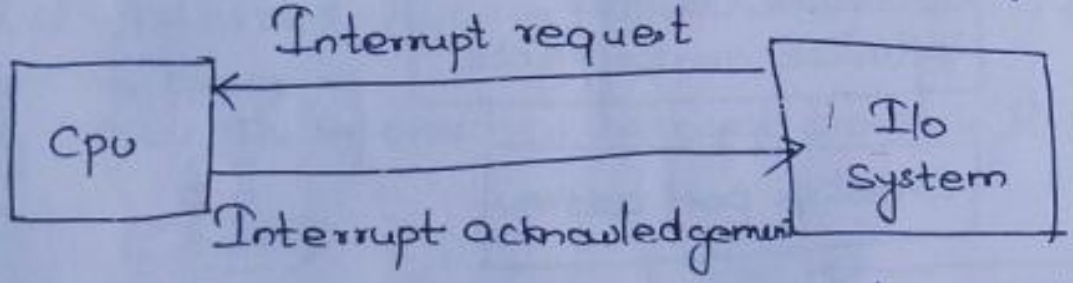


fig) Programmed Input

Interrupt Driven Ilo:-

When the cpu is asked to Communicate with devices, it Services the devices. e.g each time you type a character on a keyboard, a keyboard Service routine is called. It transfers the character you typed from the keyboard Ilo port in to the Processor and then to a data buffer in memory. The Interrupt driven Ilo technique allows the Cpu to execute it main program and only stop to service Ilo device as shown in fig



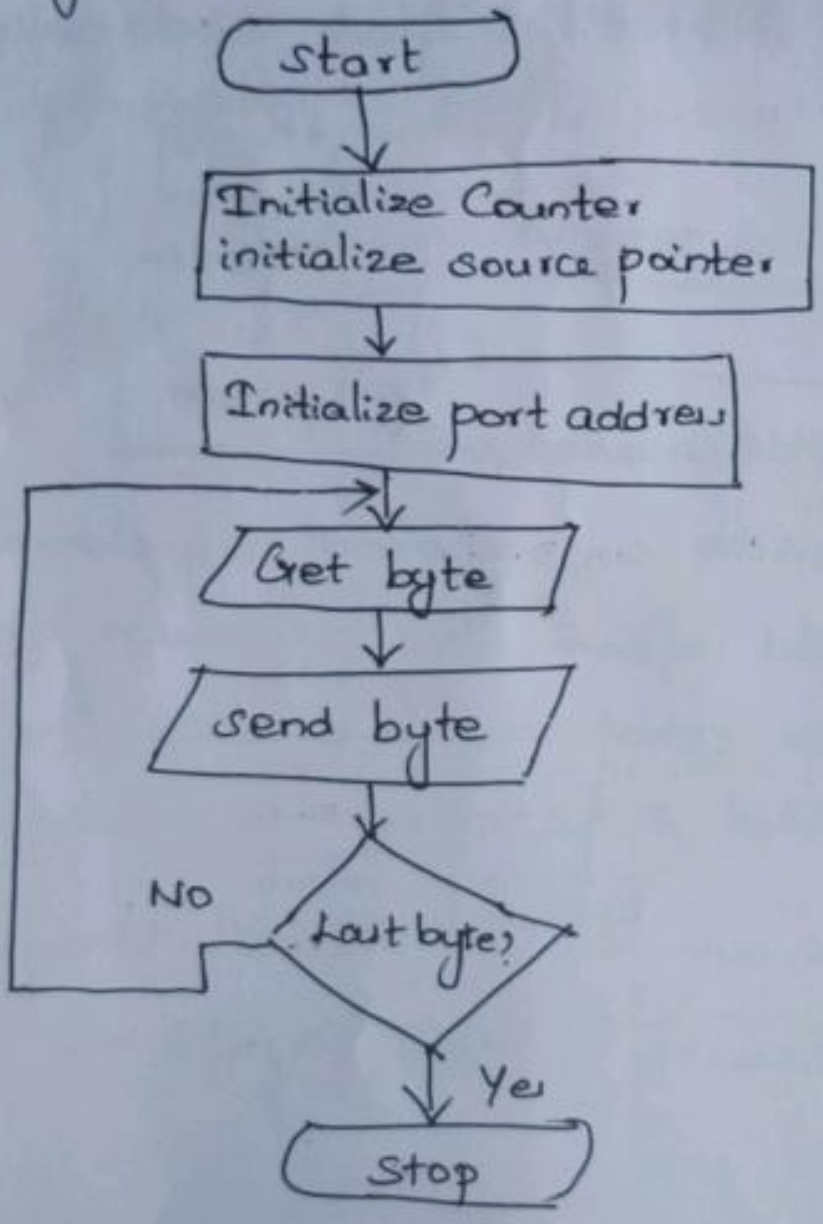
This method provides an external asynchronous input that would inform the processor that it should Complete whatever instruction (ie) Currently executed & fetch a new routine.

Once this Servicing is Completed, the processor would resume exactly where it left off.

Direct Memory access (DMA) Transfer

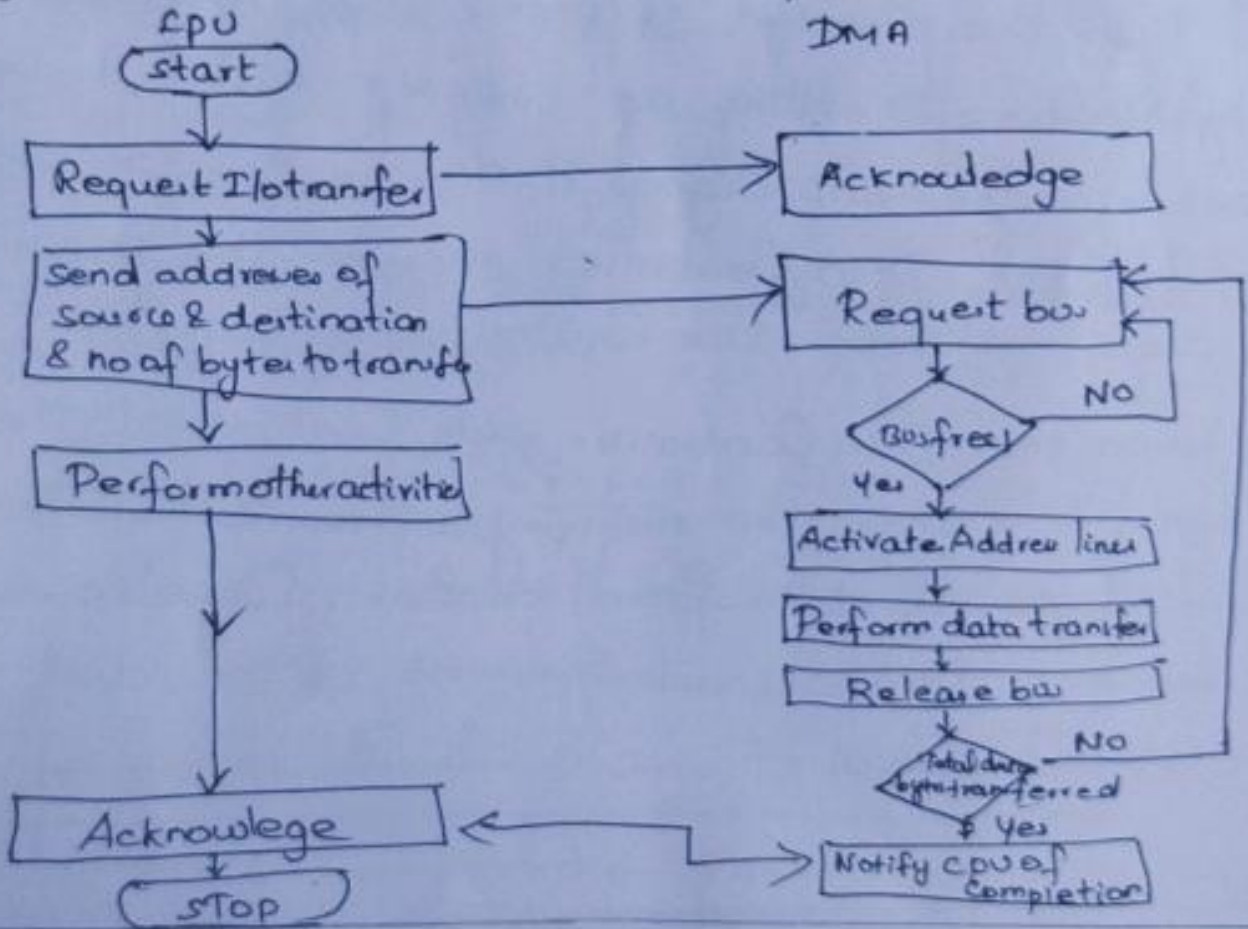
- In saftware Control data transfer, Processor executes a series of Instruction to Carry out data transfer
- for each Instruction execution fetch, decode and execute phases are required

fig shows the flowchart to transfer data from memory to I/o devices.



- Finally it asserts both I/O read and memory write signals on the Control bus.
  - ✓ Asserting the I/O read signal enables the disk Controller
  - ✓ Asserting the Memory write signal enables the addressed Memory
- In this technique data is transferred directly from the disk Controller to the Memory location with out passing through the processor or DMA Controller.
- After Completion of data transfer, the HOLD SIGNAL is deasserted to give Control of all buses back to the processor.

fig shows the interaction between processor & DMA



DMA operation:-

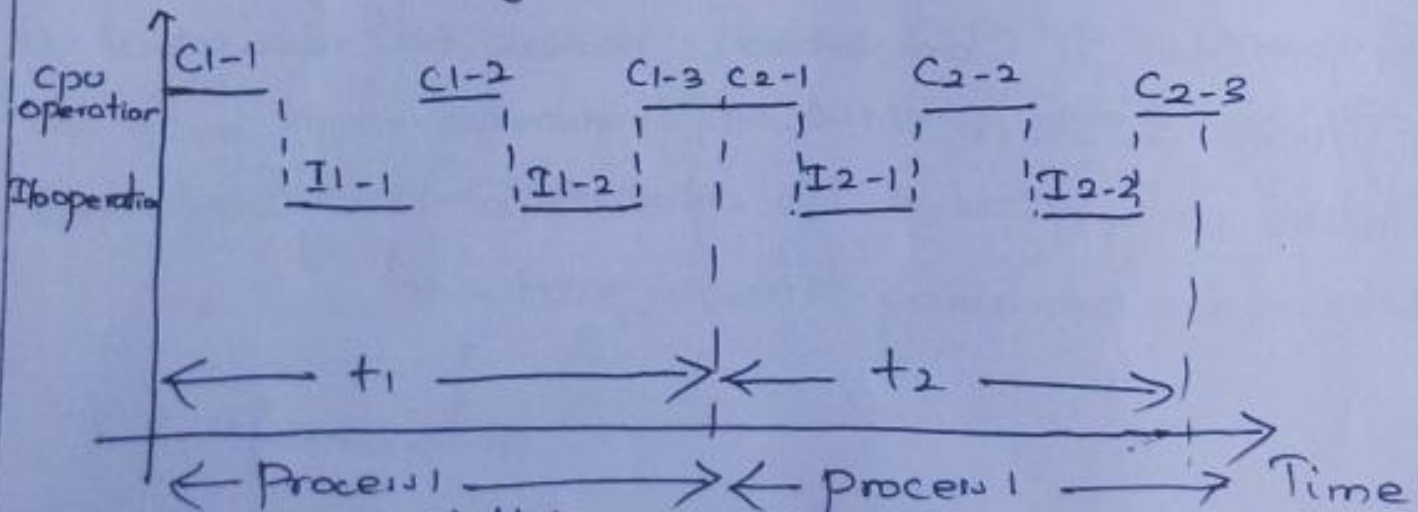
- DMA Controlled data structure is used for large data transfer for(eg) To read bulk amount of data from disk to Main memory.
- To read a block of data from the disk Processor Sends a Series of Commands to the disk Controller device telling it to Search & read the desired block of data from the disk.
- When disk Controller is ready to transfer first byte of data from disk, it Sends DMA request DRQ signal to the DMA Controller
- The DMA Controller Sends a hold request HRQ, Signal to the processor HOLD input.
- Then the processor responds this HOLD signal by floating its buses and Sending out a hold acknowledgement signal HLDA, to the DMA Controller
- When the DMA Controller receives the HLDA signal. it takes the Control of system bus.
- When the DMA Controller gets Control of the buses, it Sends the memory address where the First byte of data from the disk. It also Sends a DMA acknowledgement to the disk Controller device.

Introduction to Multiprogramming

Introduction:

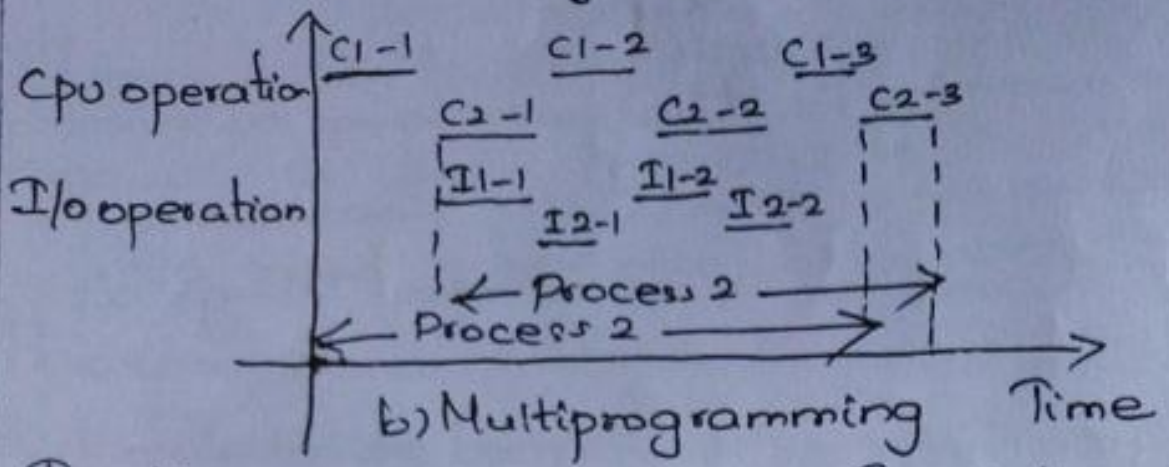
- Multiprogramming is the technique of running several processes at a time using timesharing. It allow a Computer to do several things at the same time.
- Multiprogramming creates logical parallelism
- The Concept of multiprogramming is that the Operating system keeps several processes in memory simultaneously
- The operating system selects a process from the process pool & starts executing a process
- Multiprogramming improves the system Performance by overlapping the I/o operation & cpu operation.

Fig a) shows how two processes are executed in a Uni-processing manner



a) Uniprogramming approach

Fig b) Shows how processes are executed using Pr multiprogramming approach.



- In Uniprogramming, After Completion of subtask C1-1 cpu has to wait for I/o operation to Complete. During this period, Process1 is not utilizing the cpu time because it is busy in I/o operation.
- After Completion of I/o operation I1-1 cpu starts Processing of subtask C1-2.

Similar description applies to I1-2 and Subtask C1-3, In this way, Process1 is executed & then process 2 is executed.

- In Multiprogramming, after Completion of subtask C1-1 cpu, The cpu time is utilized by subtask C2-1 of Process2. At the time, I/o operation I1-1 of process1 is Completed & hence it is resumed to process Subtask C1-2 of Process1.

In this way cpu & I/o operations are overlapped & overall processing time is reduced.

Multiprocessor Configuration:

* If a µp s/m contains two or more components that can execute instr independently, then the s/m is called multiprocessor system.

Advantages:

* Improves cost/performance ratio.
* Tasks are divided among the modules. If failure occurs it is easier and cheaper to find and replace the malfunctioning processor.

Types:

1. closely coupled configuration
2. loosely coupled configurations

Closely coupled Configuration:

* In the closely coupled s/m (ccs) the processors or supporting processors (coprocessor, maths processor) share ① clock generator, ② bus control logic, ③ entire memory and ④ I/O subsystem
* Communicate through a shared main memory.

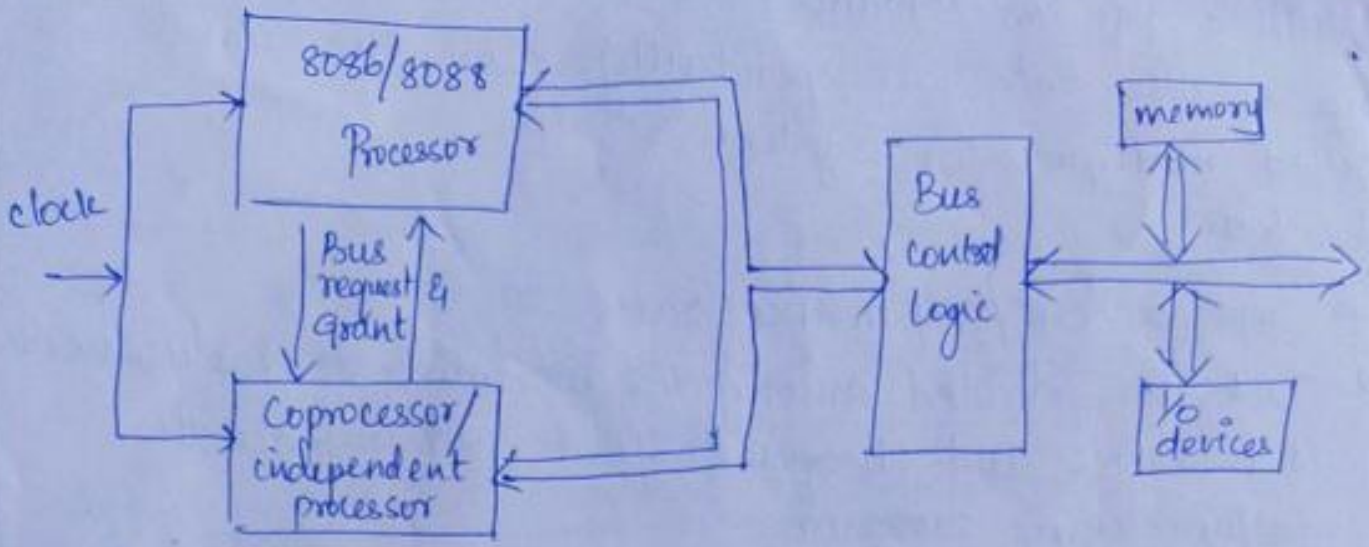
Disadvantage

* Performance degradation due to memory contentions

Uses (Advantages)

* High speed & real-time processing

Closely Coupled S/m:-



* The CPU (8086) is the master.
* The supporting processor is the slave.
* In CCS, no special instr such as WAIT or ESC is used.
* Communication b/w host & independent processor is done through memory space.

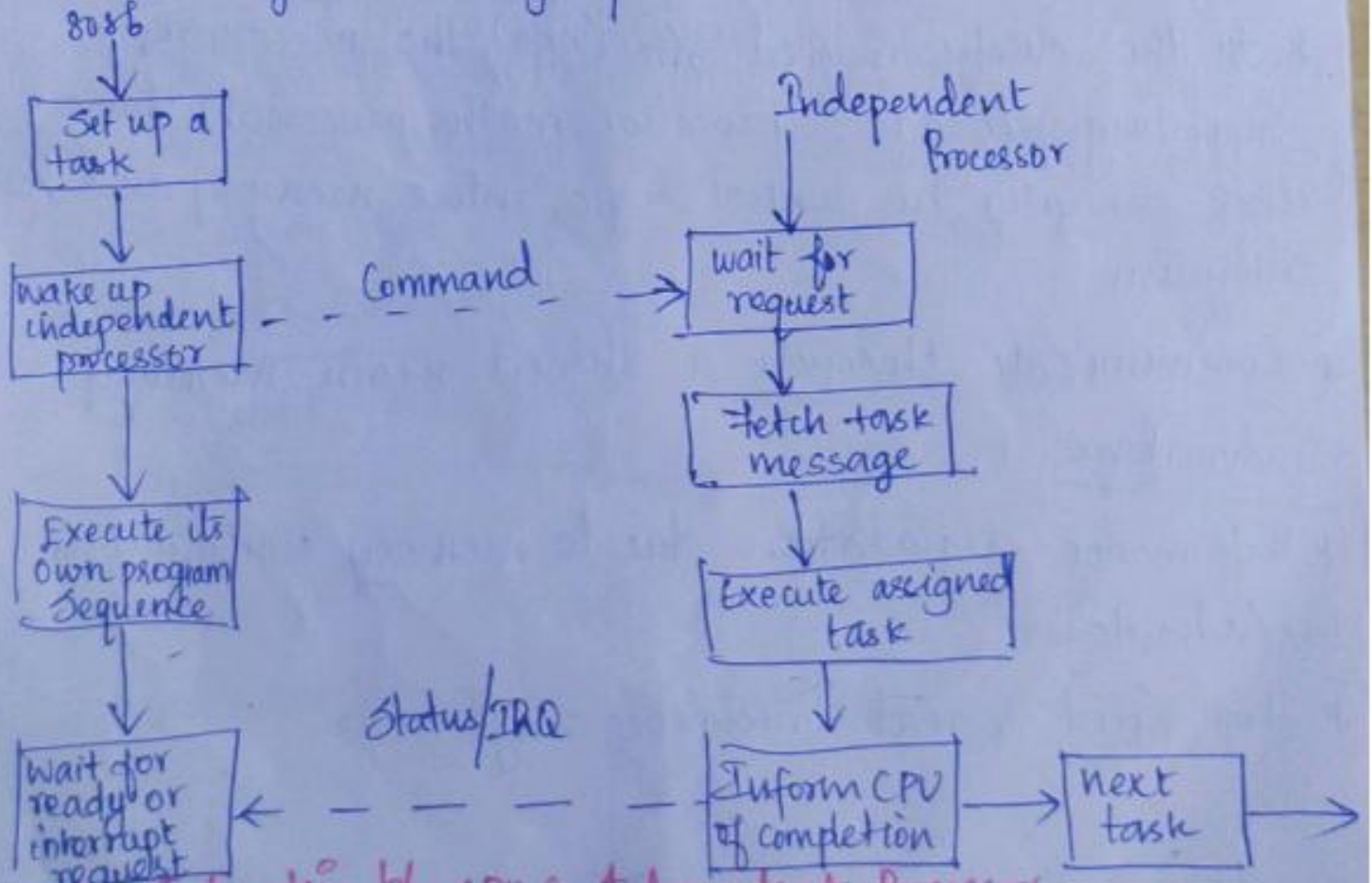


Fig: Interaction b/w CPU & Independent Processor

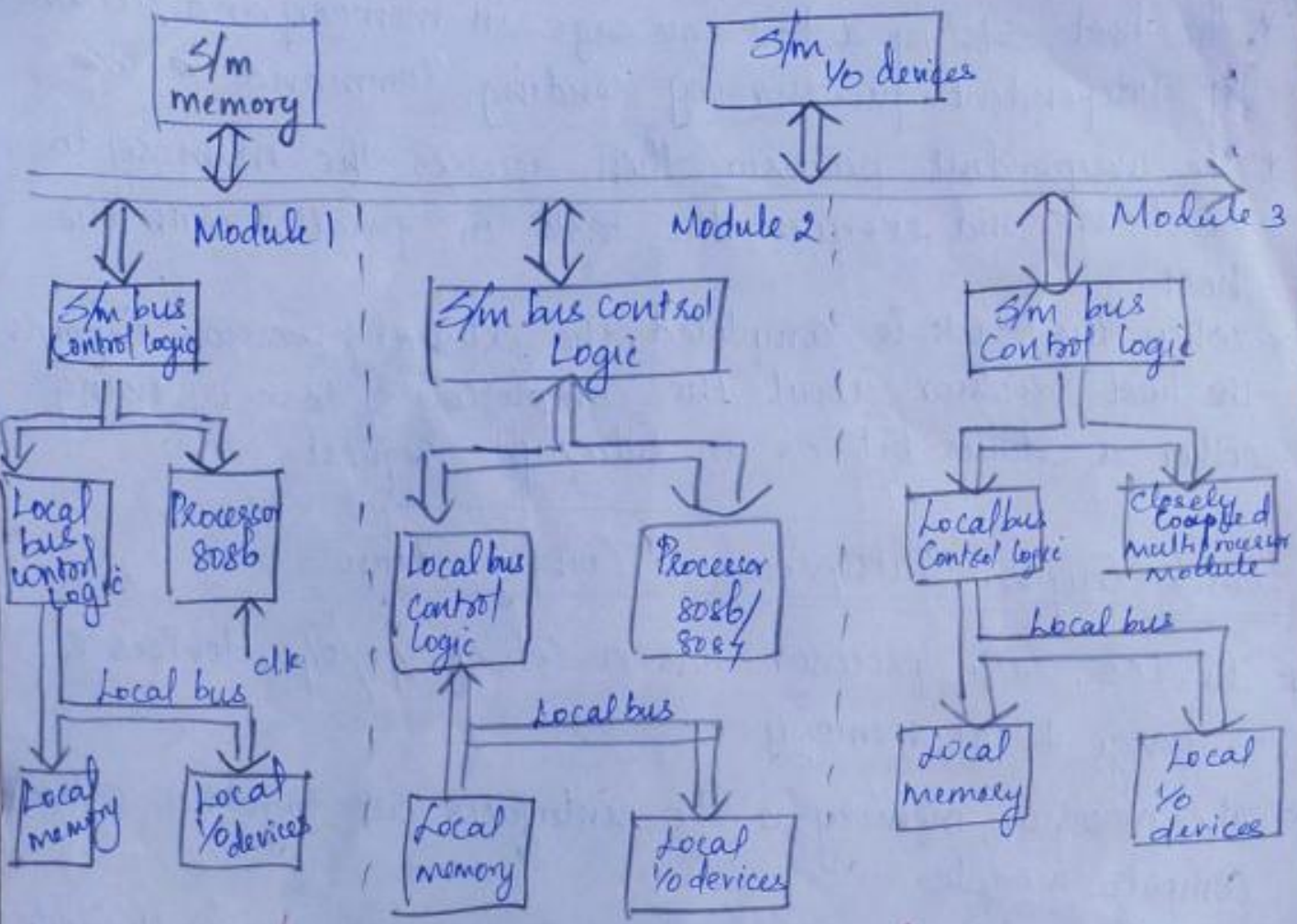
* The host sets up a task message in memory and wakes up independent processor by sending command to one
* The independent processor then accesses the memory to read task and executes the task in parallel with the host.
* When the task is completed, the external processor informs the host processor about the completion of task by using either a status bit or an interrupt request.

Loosely Coupled Multiprocessor Configuration

* In LCS, each processor has a set of I/p/o/p devices & a Large Local memory.
* The processor, memory, & I/O interfaces are together called Computer module.
* Computer modules communicate by exchange messages through a Message Transfer System (MTS)
* This S/m is also referred as distributed systems.
* Efficient, when the interactions b/w tasks are minimal.

Loosely Coupled S/m using 8086

* It has different modules. Each module may consists of an 8086, an another processor or a coprocessor or closely coupled configuration
* Each processor has its own memory and I/O devices
* They can share only the S/m resources.



Loosely Coupled Configuration

Advantages

1) Better S/m throughput
2) parallel processing
3) More flexible
4) Failure in one module does not cause a breakdown of the entire S/m.

	Closely Coupled S/m	Loosely coupled System
1	A multiprocessor S/m with common shared memory	1) A multiprocessor S/m in which each processor has its own private local memory.
2.	Information can be shared among the CPU's by placing it in the common global memory.	2) Information is transfered from one processor to other by message passing system
3	Parallelism can be implemented less efficiently.	3) Parallelism can be implemented more efficiently
4)	System structure is less flexible	4) S/m structure is more flexible

Loosely Coupled Configuration :-

* In a multiprocessor system, two 8086s or 8088 Cannot be tied directly together,

In a loosely Coupled Configuration each cpu has its Own bus Control logic and bus arbitration is resolve by extending this logic and adding external logic that is Common to all master modules

* A loosely Coupled Configuration provides the following advantage

1. High system throughput can be achieved by having more one cpu
2. The system can be expanded in a Modular form. Each bus master module is an independent Unit and normally resides on a seperate pc Board
3. A failure in one module normally does not cause a breakdown of the entire system & the faulty module can easily detected & replaced.
4. Each bus master may have a local bus to access dedicated memory or I/o devices so that a greater degree of Parallel processing can be achieved.

* Since each master module running independently the access to resource (or) shared system bus can be used by all masters. but resolve Bus arbitration problem an extra bus Control logic must provided.
* There are three schemes for establishing Priority :-
  1. Daisy chaining
  2. Polling
  3. Independent requesting

* Daisy chaining :-
  * The daisy chain method is characterized by its simplicity & low cost
  * All master use the same line for making bus requests. To respond to a bus request s/l, the Controller sends out a bus grant signal if the bus busy signal is inactive
  * The priority is determined by the physical location of the Modules
  The one located closest to the Controller has the highest priority.
  * Compare to the other methods, the daisy chain scheme requires the least no of ctrl lines & this number is independent of the no of Modules in the system. The arbitration time is slow due to the propagation delay of the bus grant signal.

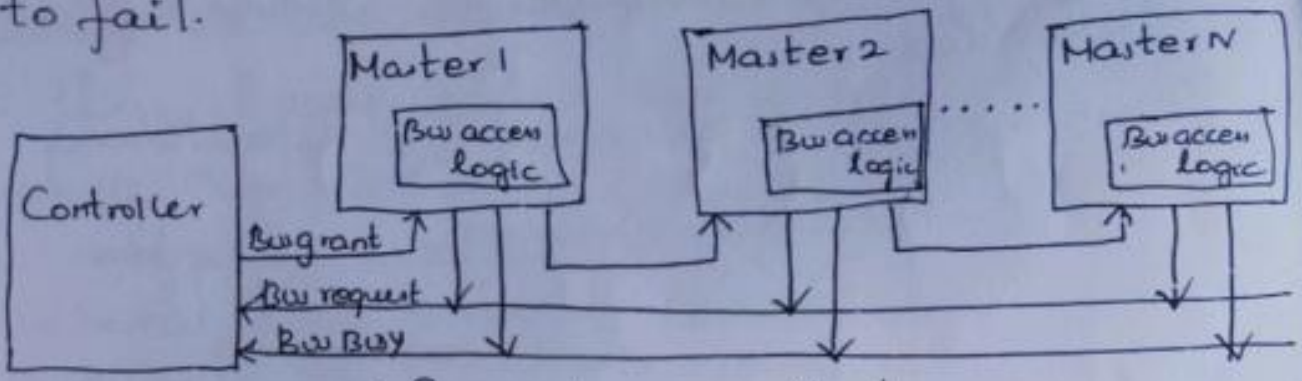
* Since each master module running independently the access to resource (or) shared system bus can be used by all masters. but resolve Bus arbitration problem an extra bus Control logic must provided.

* There are three schemes for establishing Priority :-
  1. Daisy chaining
  2. Polling
  3. Independent requesting

* Daisy chaining :-

  * The daisy chain method is characterized by its simplicity & low cost
  * All master use the same line for making bus requests. To respond to a bus request s/l, the Controller sends out a bus grant signal if the bus busy signal is inactive
  * The priority is determined by the physical location of the Modules
  The one located closest to the Controller has the highest priority.
  * Compare to the other methods, the daisy chain scheme requires the least no of ctrl lines & this number is independent of the no of Modules in the system. The arbitration time is slow due to the propagation delay of the bus grant signal.

* It has limited Modules. The priority of each module is fixed by its physical location and the failure of a module causes the whole system to fail.



a) Daisy chain method.

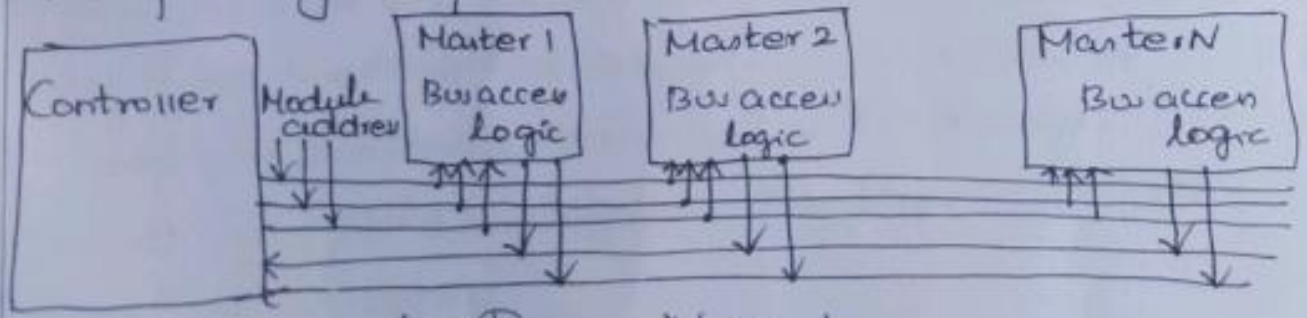
ii) Polling Method:-

* The polling scheme uses a set of lines sufficient to address each module.

* In response to a bus request, the Controller generates a sequence of Module addresses.

* When a requesting module recognizes its address, it activates the busy line and begins to use the bus.

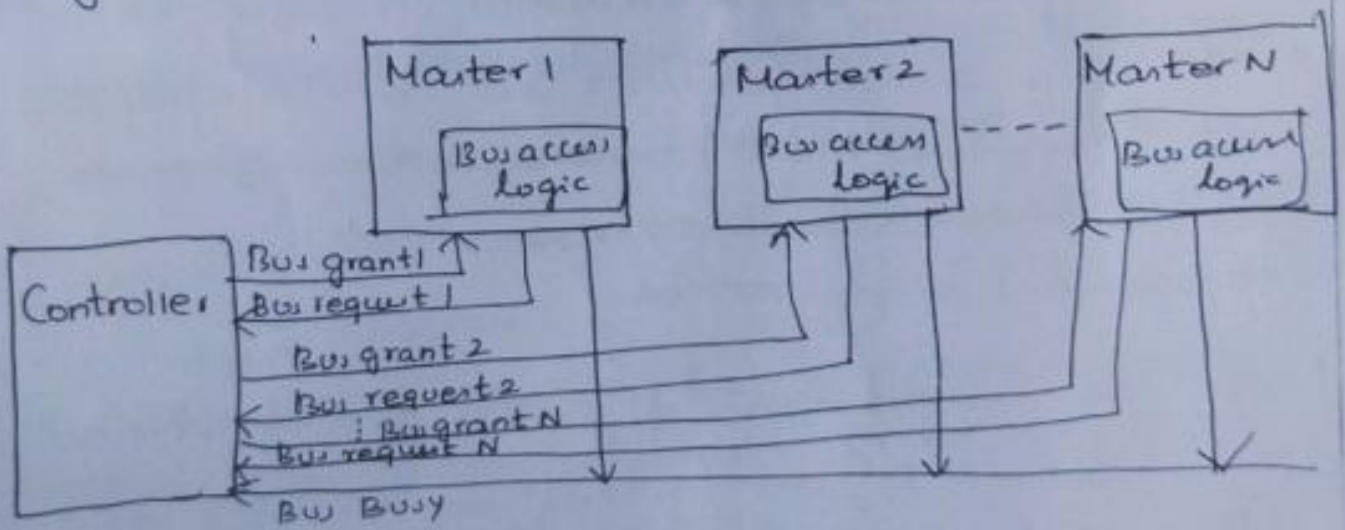
* The major advantage of polling is that the Priority can be dynamically changed by altering the polling sequence stored in the Controller.



b) Polling Method.

c) Independent Request Method :-

* The independent request scheme resolves the priority in parallel fashion
* Each module has a seperate pair of bus request & bus grant line & each pair has a priority assigned to it.
* The Controller includes a priority decoder which select the request with the highest Priority & returns the Corresponding bus grant signal.
* Arbitration is fast & is independent of no of modules in the system. Compared to the other two methods, the independent request design is the fastest
* However, it requires more bus request & bus grant lines (2n lines for n modules)



c) Independent Request Method.

Introduction to advanced processors:-

The 8087 Numeric Data processor:-

* The 8087 Numeric data processor (NDP) is specially designed to perform arithmetic operations efficiently.
* It can operate on data of the Integer, decimal and real types with lengths ranging from 2 to 10 bytes
* The Instruction set not only includes various forms of addition, subtraction, Multiplication and division, but also provides many useful functions, such as taking the Square root, exponentiation, taking the tangent and soon.
* The Computing power of 8087 NDP is high eg two 64 bit no's can be added in about 27 µs, Calculation of Square root in about 36 µs
* The NDP cannot fetch its own Instructions and therefore must operate with either an 8086 or 8088, which acts as host in a Coprocessor Configuration.

NOTE: The Instructions of 8087 are identified with a letter 'F', an Instruction starting with a letter 'F' in 8087's Instruction.

The 8089 Ilo processor:-

* The 8089 Ilo processor (Iop) is designed to handle the details involved in Iloprocessing.
* 8089 Iop can fetch and execute its own Instructions
* The Instructions specifically tailored for Ilo operations can also perform arithmetic and logical operations, branches, searching, and translation.
* The cpu Communicates with 8089 through memory based Control blocks where the information about Various tasks will be present There tasks will be dispatched to Iop through Interrupt like signal.
* Iop reads there Instructions through a program sequences called a Channel program.
* The 8089 performs the assigned task by fetching and executing Instructions from the Channel program.
* When it has finished, the Iop notifies the cpu either through an interrupt or by updating a status location in memory.

The 80286 / 80287 :-

* The enhancement to the 8086 that have been included in the 80286 are for improving the multiprogramming or Multitasking, capability of the 8086 family.
* This was primarily done by adding the necessary memory management and task switching logic in such a way there is adequate protection.

1) Between the application task and the system and other more privileged tasks
2) For seperating the tasks from each other
3) Between Code and data modules
4) Against unwanted accessing or Changing of either Code or data.

* The primary key to multitasking is the ability to break the various entities to be processed into Parts.
* The second key to multitasking is the abiliny to Change quickly and smoothly from one task to another.

Comparison Between different Processors

Name	Date	Internal Reg:	Clock Speed	Data Width	Address Lines	Max: Memory Space
8086	1974	16 Bit	2 MHZ	16 bits	20 Bit	1MB
80286	1982	16 Bit	6 MHZ	16 bits	24 Bit	16 MB
80386	1985	32 Bit	16 MHZ	32 bits	32 Bit	4 GB
80486	1989	32 Bit	25 MHZ	32 bits	32 Bit	4 GB
Pentium	1993	32 Bit	60 MHZ	32 bits. 64 bit bus	32 Bit	4 GB
Pentium II	1997	32 Bit	233 MHZ	32 bits. 64 bit bus	32 Bit	64 GB
Penium III	1999	32 Bit	450 MHZ	32 bits . 64 bit bus	32 Bit	64 GB
Pentium IV	2000	32 Bit	1.5 MHZ	32 bits. 64 bit bus	32 Bit	64 GB

Unit 7 - Peripheral Interfacing

External device (input, output, Memory) ⇓ Connector

Ic 8255 - Programmable Peripheral Interface (PPI)
↳ notes

Ic 8279 - Keyboard and Display Controller
↳ notes

Ic 8253/8254 - Programmable Interval timers
↳ Krishna kant Pg no 313-318

Ic 8259 - Programmable Interrupt Controller (PIC)
↳ notes

Ic 8251 - Universal Synchronous asynchronous Receiver Transmitter (USART)
↳ notes (pdf)

Ic 8237 - DMA Controller - notes

A to D, D to A Converter → Pg no 335, Pg no 345 (Krishnakant)

Programmable peripheral Interface (PPI) Ic 8255 (16)

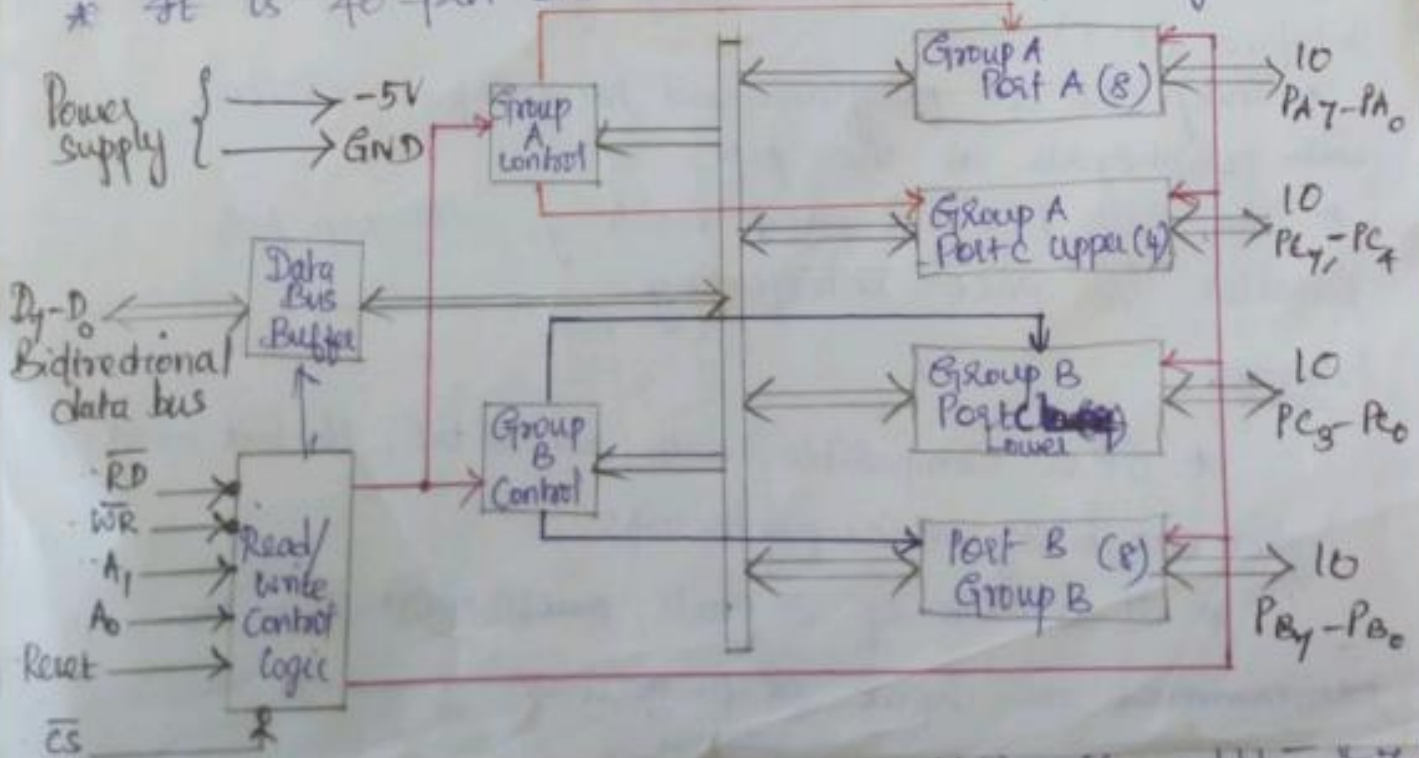
* Ic 8255 chips are called programmable Peripheral Interface (PPI).
* They can be programmed to interface various peripherals to the µp
* The 8255 PPI is used widely with µp for parallel i/o port interfacing

Features :-

* It is Compatible with all 8-bit, 16-bit and higher capability microprocessors
* It consists of 3 ports, each port can be programmed as input or output

- Port A (has 8 I/O lines)
- Port C (upper) has 4 I/O lines)
- Port B has 8 I/O lines
- Port C (lower) has 4 I/O lines

* These ports are divided into two groups.
  * Group A
    - Port A and Port C (upper)
  * Group B
    - Port B and Port C (Lower)
* It Consists of 8-bit data bus ($D_0 - D_7$)
* $\overline{RD}$ and $\overline{WR}$ for read write control signals
  $\overline{CS}$ for chip select from address decoder
  A_0, A_1 for port address
* It is 40-pin IC dual-in-line package



(000 - PC_0, 001 - PC_1, ... 110 - PC_6, 111 - PC_7)

Modes of Operation

1. Bit Set - reset mode (BSR)
2. Control I/O mode (I/O).

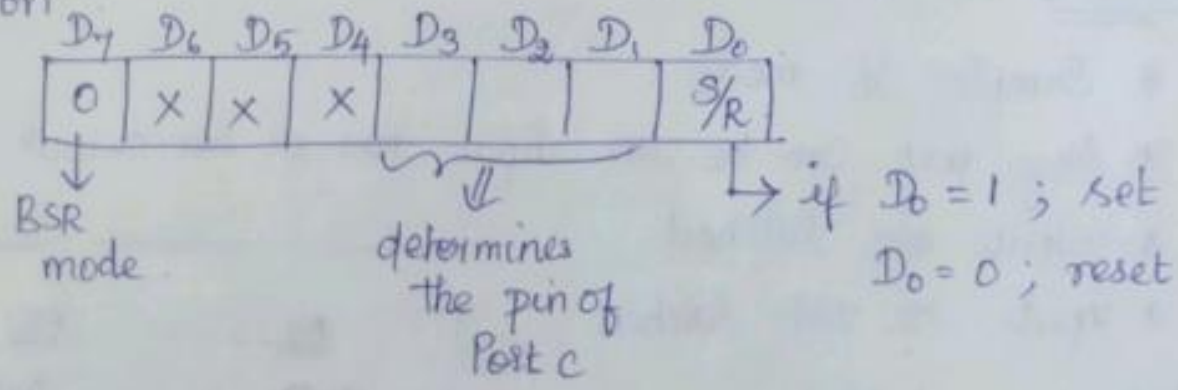
* Based on the D_7 bit in data bus, the modes are differentiated.

$D_7 = 0$; BSR mode

$D_7 = 1$; I/O mode

Bit set-reset mode :-

* It is used to set/reset the individual bits of Port C, when it is used for control/status operation



* for example, to set the Port C '5'th pin, the control word should be

$0\,x\,x\,x\,1\,0\,1\,1$ → set

BSR mode — PC_5

* D_3, D_2, D_1 - used to define the bit no. of the Port C (000 - PC_0, 001 - PC_1, ... 110 - PC_6, 111 - PC_7)

2. Control I/O mode :-

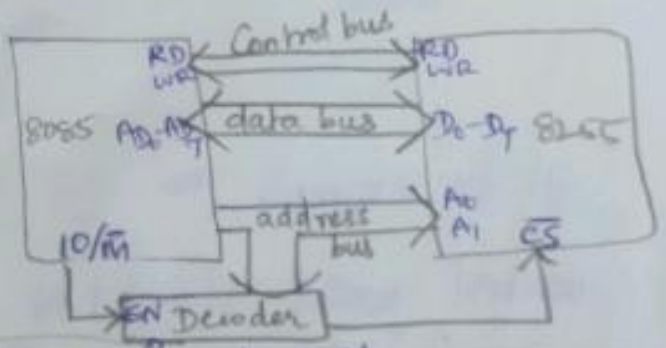
* It consists of three modes
  1. mode 0 - simple I/O mode
  2. mode 1 - strobed I/O mode
  3. mode 2 - strobed bidirectional I/O mode.

* Port A - 8 bit I/O port used in all three modes
* Port B - 8 bit I/O port used in mode 0 and mode 1.
* Port C - 8 bit I/O port acts as a control signals (handshake s/l) for port A and Port B in the handshake mode.

Mode 0 :-

* Simple I/O mode
* Any port can be an input port or an output port
* outputs are latched
* inputs are not latched.

Interfacing 8255 PPI to 8085 µp

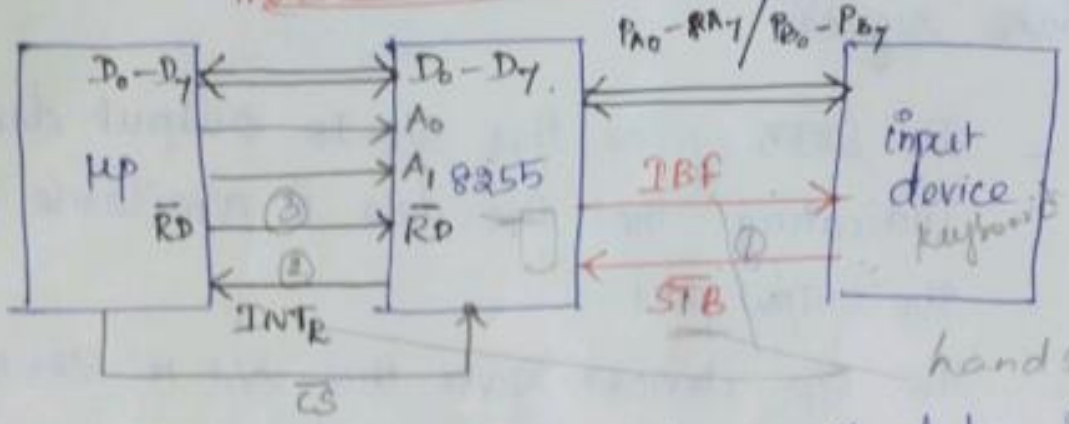


Mode 1 - strobed I/O mode

input port

* Port A (or) Port B act as a input port
* Port C is used as control s/l
* The input device (peripheral) transmits the data to the µp through 8255.

mode 1 - input port



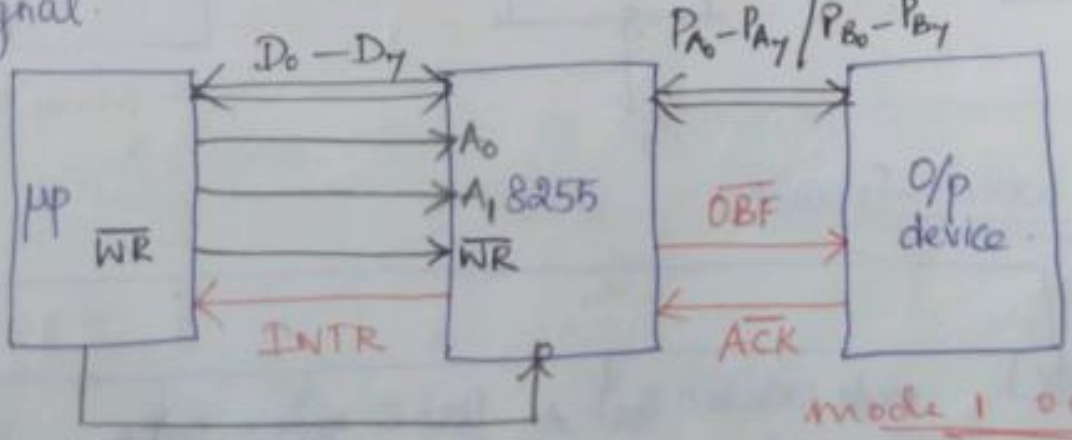
* when i/p devices wants to transmit data, it gives $\overline{STB}$ (strobe input) — to load data from the input Port lines to input buffer.

* If the input buffer is full in 8255, it gives IBF (Input Buffer full) signal.

* Using $\overline{RD}$ signal the CPU reads the data from 8255 when it gives INTR signal to the µp

Output Port :-

* Either Port A or Port B can be used as output port Port C (upper) or Port C (lower) can be used as handshake signal.



mode 1 output port

* The µp transferring data to the O/p device via (through) 8255

* handshake signals

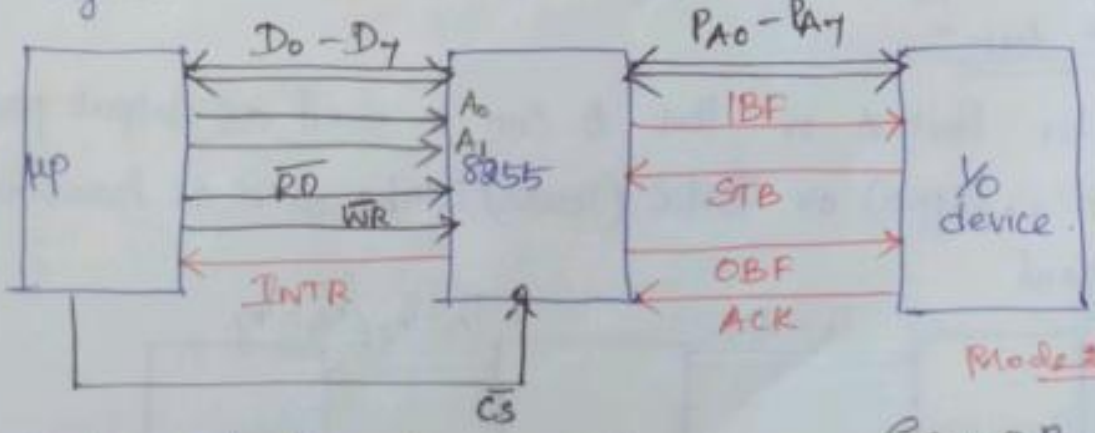
$\overline{OBF}$ – Ic 8255 gives this s/l to output devices indicating that the data is available on the output port.

$\overline{Ack}$ – The o/p devices gives this s/l to 8255 that the data is accepted

$\overline{INTR}$ – used to interrupt CPU when the o/p device accepted the data.

Mode 2 - Bidirectional Port

* Port A can be used ~~for~~ as bidirectional (I/O) port.
* Port c ($PC_3 - PC_7$) can be used as handshaking signals.



Mode 2 I/O port

I/O control word format

Group A | Group B

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
D_7 = 1 - I/O mode	mode selection 00 – mode 0 01 – mode 1 1x – mode 2		Port A 1 – I/P 0 – O/P	Port C (upper) 1 – I/P 0 – O/P	mode selection 0 – mode 0 1 – mode 1	Port B 0 – O/P 1 – I/P	Port C (lower) 1 – input 0 – output

Direct Memory Access ①

DMA - Direct Memory access ~~is one of the ways~~ provides to accomplish high-speed data transfer directly between the memory and peripheral devices, without the intervention of the microprocessor. It is often used when a large block of data is to be transferred.

Programmed data transfer involve moving data from the memory into the accumulator, and then from the accumulator to the output ports A program has to be written to transfer data from a device to the memory Thus, programmed data transfer is a slow process.

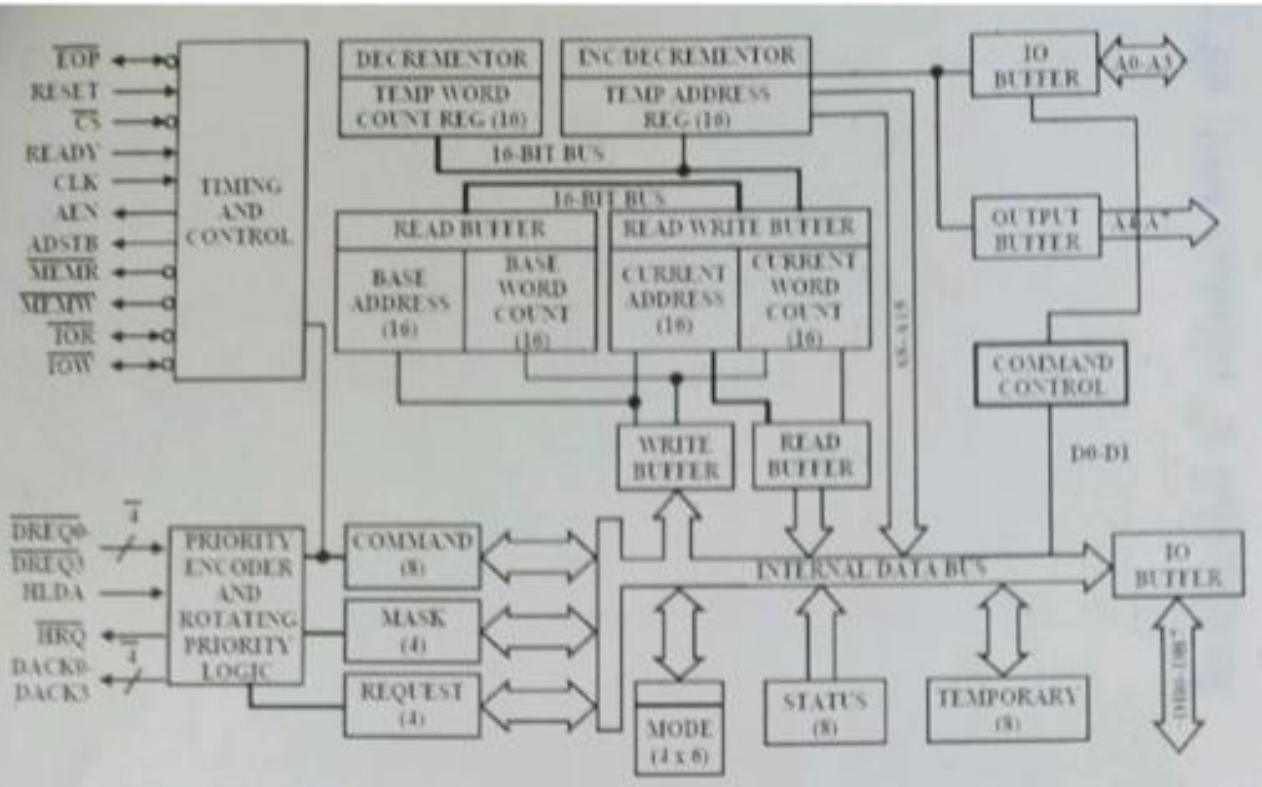
DMA Controller: Intel IC 8237

DMA data transfer is controlled using a separate DMA controller.

The DMA controller temporarily borrows the address bus, data bus and control bus from the µp and transfers the data bytes directly from the external peripheral devices to memory locations

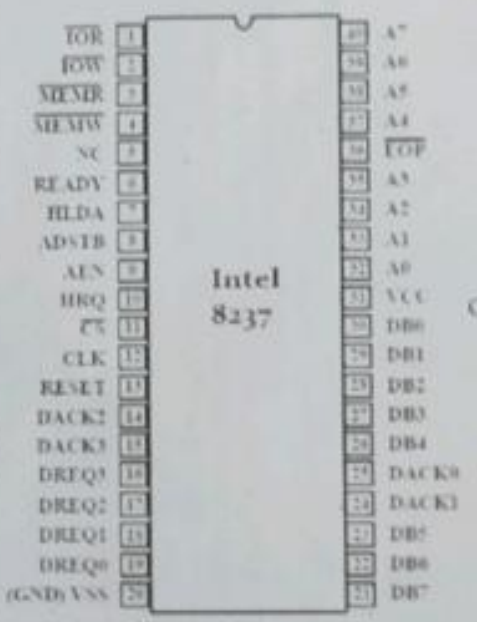
Features of 8237

* 4 different DMA channels
* Enable and disable control of individual requests.
* Possibility of memory-to-memory transfer.
* Address increment or decrement.
* Cascading to expand to any number of DMA channels.



Signal Description

- **DREQ0-DREQ3 (DMA request)**: Used to request a DMA transfer for a particular DMA channel.
- **DACK0-DACK3 (DMA channel acknowledge):** Acknowledges a channel DMA request from a device.
- **HRQ (Hold request):** Requests a DMA transfer.
- **HLDA (Hold acknowledge) signals** the 8237 that the microprocessor has relinquished control the address, data anD control buses.
- **AEN (Address enable):** Enables the DMA address latch connected to the 8237 and disable any buffers in the system connected to the microprocessor. (Use to take the control of the address bus from the microprocessor)
- **ADSTB (Address strobe):** Functions as ALE to latch address during the DMA transfer.



of

- **EOP (End of process):** Signals the end of the DMA process.
- **IOR (I/O read):** Used as an input strobe to read data from the 8237 during programming and used as an output strobe to read data from the port during a DMA write cycle.
- **IOW (I/O write):** Used as an input strobe to write data to the 8237 during programming and used as an output strobe to write data to the port during a DMA read cycle.
- **MEMW (Memory write):** Used as an output to cause memory to write data during a DMA write cycle.
- **MEMR (Memory read):** Used as an output to cause memory to read data during a DMA read cycle.

The internal registers of 8237 are. Each channel has these reg

1. Current address Register :- (16 bit register)
   * Holds the value of the addr used during DMA transfer.

Current word Count register :- (16-bit)
* This reg determines the number of transfers to be performed.

Base address and base word count registers :-
* The 16 bit registers store the original value of their associated current registers.
* Cannot be read by the µp.

Command register :-
* 8 bit reg controls the operation of 8237
* It is programmed by the µp.

Mode register :- (8-bit).
* Each channel has mode reg associated with it. to determine the mode of transmission.

Request Register :-
* Each channel has a request bit associated with it in the 4-bit request register. These are non-maskable

Mask Register :-
* Each channel has mask bit associated with it that can be set to disable an incoming DREQ.

Status Register
* It contains information about the status of the devices, to be read by the processor at any time

Temporary reg :-
* It is used to hold data during memory to memory transfers

The DMA Controller operates in two major cycles

1. Active
2. Idle

Idle cycle :- When no channel is requesting service, the 8237 enters the idle cycle

Active cycle :- when a device requests a DMA service, the Controller issues Hold request (HRQ) to the µp and enters the active cycle

Modes of DMA Controller

Single transfer mode :- * A single byte (a word) is transferred at a time * The peripheral will request the DMA each time it is ready for another transfer

eg :- Floppy disk

Block transfer mode :-

* Once the System bus is acquired by the DMA controller, an entire block of data is transferred
* Transfer continues until (terminal count) TC or external end of Process (EOP) is encountered.

Demand transfer mode :-

* The device continues making transfers until a terminal count TC or an external EOP is encountered or until DREQ goes inactive.
* The data transfer continues until the I/O device has exhausted its capacity
* Higher priority may intervene in the demand process

Cascade mode :-

* More than one 8237 are connected for simple s/m expansion

IC 8237 (DMA controller):

Operation: → Idle cycle
→ Active cycle

Idle cycle:

→ When the system is turned ON switch position A.
→ So, buses connect µP with peripheral device/ disc controller.
→ µP, executes the pgm until it needs a data from disc
→ µP, gives command to disk controller telling it to read & search data from disc

Active cycle

→ when DMA gets control of the bus it sends the addr where the 1st byte of data from disc is to be written.
→ Send DMA ack. DACK to disc controller telling it to be ready
→ It assert both IOR & MEMW signal.
IOR ↓ enable disk controller to o/p byte from data → disc
MEMW ↓ enable add. memory to accept data from data bus
→ when data transfer is completed, DMA controller unasserts the HOLD request to µP and switch position from B to A. After getting control of all buses the µP executes remaining program.

Block: - 3 Blocks:

Timing control Block: → It genrates internal timing & external control signal

Program command control Block: → It decodes command before servicing DMA request → decodes mode control word used to select type of DMA during servicing.

Priority Encoder Block: It resolve the priority blw DMA channels.
→ has 4 bit internal memory in form of register.

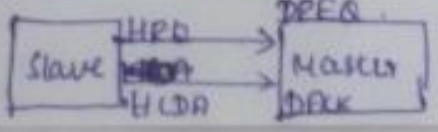
DMA Operation: ✱ mode * Single TRANSFER MODE
* BLOCK TRANFER MODE * Demand or BURST TRANSFER MODE * CASCADE MODE.

1) Single Transfer mode:
→ In this mode, device make only one transfer.
→ After each transfer DMA gives control of buses to process

2) Block Transfer mode:
→ device can make no. of transfer as pgmed in word count reg. After each transfer word count decremented by '1'. and addr also dec/incre by '1'.
→ used when controller needs to transfer block of data.

3) Demand or Burst transfer mode:
→ device continue make transfer until Terminal count (TC) or $\overline{EOP}$ (End of Process) is encontred or until DREQ goes inactive (data mem request)

4) Cascade mode: → DMA channels are expanded using this mode
→ 2 additional devices are cascaded to master device. This is 2 level DMA system ⇒ Allows DMA request of additional devices to communicate through the priority of device →



HRQ → Hold request

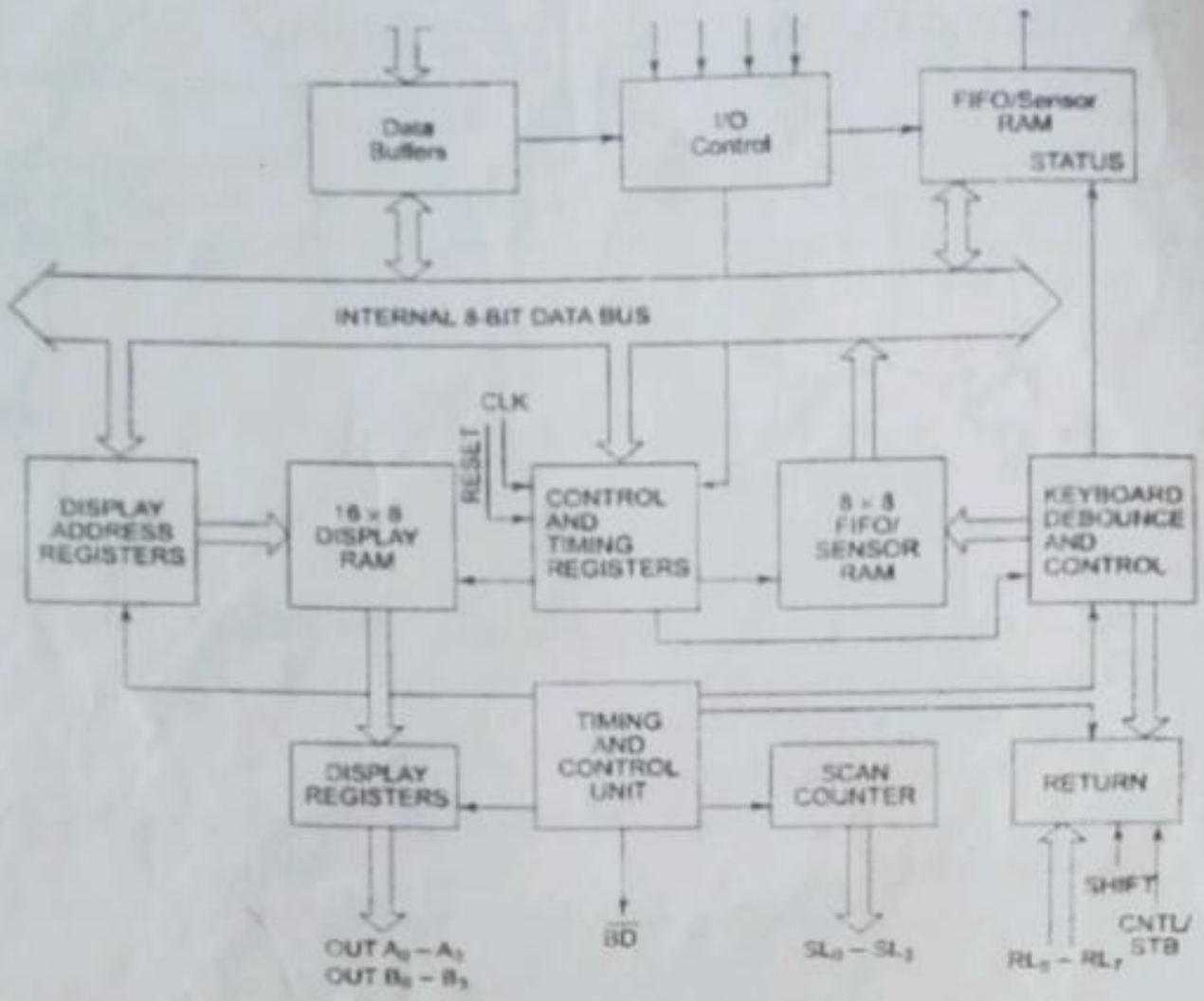
IC8279 KEYBOARD AND DISPLAY INTERFACING

➢ Intel's 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interfaces a keyboard with the CPU.

➢ IC8255 can be used in interfacing keyboards and displays. The disadvantages of this method of interfacing keyboard and display is that the processor has to refresh the display and check the status of the keyboard periodically.Thus a considerable amount of CPU time is wasted, reducing the system operating speed

Features:

- It is designed by Intel
- It is support 64 contact key matrix with two more keys "CONTROL" and "SHIFT"
- It provides 3 operating modes
  1.Scanned keyboard mode 2.Scanned sensor matrix mode 3.Strobed Input mode.
- It has inbuilt debounce key .
- It provides 16 byte display RAM to display 16 digits and interfacing 16 digits.
- It provides two output modes:
  1.Left entry (Typewriter type).
  2.Right entry (Calculator type).
- Simultaneous keyboard and display operation facility allows to interleave keyboard and display software.
- The interrupt output of 8279 can be used to tell CPU that the key press is detected.



8279 Internal Architecture

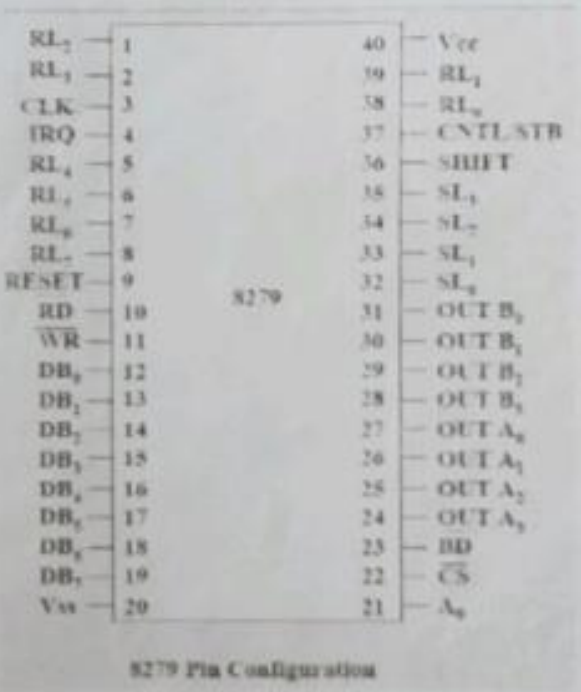
It consists 4 main section.

1. CPU interface and control section.
2. Scan section
3. Keyboard Section
4. Display section.

CPU INTERFACE AND CONTROL SECTION:

It consists of

- Data buffers
- I/O control
- Control and timing registers.
- Timing and control logic.



8279 Pin Configuration

Data Buffers:

- 8-bit bidirectional buffer.
- Used to connect the internal data bus and external data bus.

I/O control:

- I/O control section uses the A0,CS,RD and WR signals to controls the data flow.
- The data flow is enabled by CS=0 otherwise it is the high impedance state.
- A0=0 means the data is transferred.

- A0=1 means status or command word is transferred.

I/O control signals listed below

A0	$\overline{RD}$	$\overline{WR}$	Interpretation
0	1	0	Data from CPU to 8279
0	0	1	Data from 8279 to CPU
1	1	0	Command word from CPU to 8279
1	0	1	Status word from 8279 to CPU

TIMING AND CONTROL REGISTERS:

- Store the keyboard and display modes and others operating condition programmed by the CPU.The modes are programmed by sending proper command A0=1.

TIMING AND CONTROL:

- It consist timing counter.
- It provides proper key scan, display scan time.

Ic 8279 is two types,

- Input modes.
- Display modes.

INPUT MODES (key board):

It is basically 3 types,

Scanned keyboard.

Scanned sensor matrix.

Strobed mode.

SCANNED KEYBOARD:

Key board can be scanned in two ways.

1.Encoded Scan

2.Decoded Scan.

ENCODED MODE:

It has a scan counter and four scan lines(SC3-SC0).A 3:8 decoder is needed to send signals to the individual rows.This enables the 8x8 keyboard interfacing

DECODED MODE:

It is provide four row activation signals SC0-SC3) which can be directly interfaced to 4 rows of keyboard. Thus, it is used to directly to interface8*4 matrix keyboard

KEY LOCKOUT:

In this mode, the two key depression is not allowed. When any key is depressed, the debounce logic is set and 8279 checks for any key depress next two scans.

N-KEY ROLLOVER:

Each key is depression is treated as independently from all others.

SCANNED SENSOR MATRIX MODE

Scan and return lines can be used to interface a matrix of switches (sensors).The return lines contain the data regarding the switch position (ON/OFF) in the scanned row of the sensor matrix.The image of sensor matrix is kept in the sensor RAM. If there is a change in sensor value, the IRQ goes high.

STROBED INPUT MODE

Data will enter to the FIFO RAM from the return lines. The data is entering at the rising edge of the CNTL/STB signal.

OUTPUT (DISPLAY) MODES :

8279 provides two output modes for selecting the display options.

1. Display Scan:

In this mode, 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4-bit or single 8-bit display units.

2. Display Entry:

The Display data is entered for display either from the right side or from the left side.

DISPLAY MODES

Signals used

- Uses 4 scan lines
- 2 four output ports (A0-A3,B0-B3)
- BD- to blank the display

8259 provides two basic output modes.

- Left Entry (Typewritter Type)
- Right Entry (Calculator Type)

LEFT ENTRY:

In this mode , 8279 display characters from left to right. Like a typewriter.

AUTOINCREMENT IN LEFT ENTRY:

In left entry mode , Autoincrement flag is set after each operation display RAM address is incremented.

RIGHT ENTRY:

In this mode , 8279 display characters from Right to left.Like a Calculator.

AUTOINCREMENT IN RIGHT ENTRY:

In right entry mode , Auto increment flag is set after each operation display RAM address is incremented.

Command Words of 8279

- All the Command words or status words are written or read with Ao = 1 and CS = 0 to or from 8279.

a. Keyboard Display mode set

RIGHT ENTRY:

In this mode , 8279 display characters from Right to left.Like a Calculator.

AUTOINCREMENT IN RIGHT ENTRY:

In right entry mode , Auto increment flag is set after each operation display RAM address is incremented.

Command Words of 8279

- All the Command words or status words are written or read with Ao = 1 and CS = 0 to or from 8279.

a. Keyboard Display mode set

The format of the command word to select different modes of operation of 8279 is given below with its bit definitions.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	0	D	D	K	K	K	1

D	D	*Display modes*
0	0	Eight 8-bit character Left entry
0	1	Sixteen 8-bit character Left entry (Default after reset)
1	0	Eight 8-bit character Right entry
1	1	Sixteen 8-bit character Right entry

K	K	K	*Keyboard modes*
0	0	0	Encoded scan, 2 key lockout (Default after reset)
0	0	1	Decoded scan, 2 key lockout
0	1	0	Encoded scan N-Key roll over
0	1	1	Decoded scan N-Key roll over
1	0	0	Encoded scan sensor matrix
1	0	1	Decoded scan sensor matrix
1	1	0	Strobed Input Encoded scan
1	1	1	Strobed Input Decoded scan

b. Programmable Clock

The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	1	P	P	P	P	P	1

PPPPP is a 5-bit binary constant. The input frequency is divided by a decimal constant ranging from 2 to 31, decided by the bits of an internal prescalar, PPPPP.

c. Read FIFO/Sensor RAM (keyboard)

The format of this command is given as shown below

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	1	0	AI	X	A	A	A	1

SCANNED KEYBOARD:

Key board can be scanned in two ways.

1.Encoded Scan

2.Decoded Scan.

ENCODED MODE:

It has a scan counter and four scan lines(SC3-SC0).A 3:8 decoder is needed to send signals to the individual rows.This enables the 8x8 keyboard interfacing

DECODED MODE:

It is provide four row activation signals SC0-SC3) which can be directly interfaced to 4 rows of keyboard. Thus, it is used to directly to interface8*4 matrix keyboard

KEY LOCKOUT:

In this mode, the two key depression is not allowed. When any key is depressed, the debounce logic is set and 8279 checks for any key depress next two scans.

N-KEY ROLLOVER:

Each key is depression is treated as independently from all others.

SCANNED SENSOR MATRIX MODE

Scan and return lines can be used to interface a matrix of switches (sensors).The return lines contain the data regarding the switch position (ON/OFF) in the scanned row of the sensor matrix.The image of sensor matrix is kept in the sensor RAM. If there is a change in sensor value, the IRQ goes high.

STROBED INPUT MODE

Data will enter to the FIFO RAM from the return lines. The data is entering at the rising edge of the CNTL/STB signal.

OUTPUT (DISPLAY) MODES :

8279 provides two output modes for selecting the display options.

1. Display Scan:

In this mode, 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4-bit or single 8-bit display units.

2. Display Entry:

The Display data is entered for display either from the right side or from the left side.

DISPLAY MODES

Signals used

- Uses 4 scan lines
- 2 four output ports (A0-A3,B0-B3)
- BD- to blank the display

8259 provides two basic output modes.

- Left Entry (Typewritter Type)
- Right Entry (Calculator Type)

LEFT ENTRY:

In this mode , 8279 display characters from left to right. Like a typewriter.

AUTOINCREMENT IN LEFT ENTRY:

In left entry mode , Autoincrement flag is set after each operation display RAM address is incremented.

X - don't care
AI - Auto increment flag
AAA - Address pointer to 8 bit FIFO RAM
This word is written to set up 8279 for reading FIFO/Sensor RAM.

d. Read Display RAM

This command enables a programmer to read the display RAM data

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	1	1	AI	A	A	A	A	1

The CPU writes this command word to 8279 to prepare it for display RAM read operation. AI is auto incremented flag and AAAA, the 4-bit address, points to the 16-byte display RAM that is to be read. If AI = 1, the address will be automatically, incremented after each read or write to the display RAM.

e. Write Display RAM

The format of this command is given as shown below

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	0	0	AI	A	A	A	A	1

AI - Auto increment flag
AAAA - 4-bit address for 16-bit display RAM to be written

f. Display Write Inhibit/Blanking

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	0	1	X	IW	IW	BL	BL	1
Output nibbles		→		A	B	A	B	

The IW (Inhibit write flag) bits are used to mask the individual nibble Here Do and D2 corresponds to OUTBo – OUTB3 while D1 and D3 corresponds to OUTAo-OUTA3 for blanking and masking respectively.

g. Clear Display RAM

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	1	0	CD_2	CD_1	CD_0	CF	CA	1

The CD2, CD1, CDo is a selectable blanking code to clear all the rows of the display RAM as given below. The characters A and B represents the output nibbles.

CD	CD1	CDo	
1	0	x	All Zeros (x don't care) AB = 00
1	1	0	A3-Ao = 2(0010) and B3-Bo = 00(0000)
1	1	1	All ones (AB = FF), i.e. clear RAM

Here, CA represents clear All and CF represents Clear FIFO RAM

End Interrupt/Error Mode Set

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	1	1	E	x	x	x	x	1

x—do not care

Memory Interfacing:-

* Memory is a integral part of a microprocessor system

* The memory interfacing circuit is used to access memory quite frequently to read Instruction Codes and data stored in memory

* This read/write operations are monitored by Control signals

* The Microprocessor activates these signals when it wants to read from and write into memory

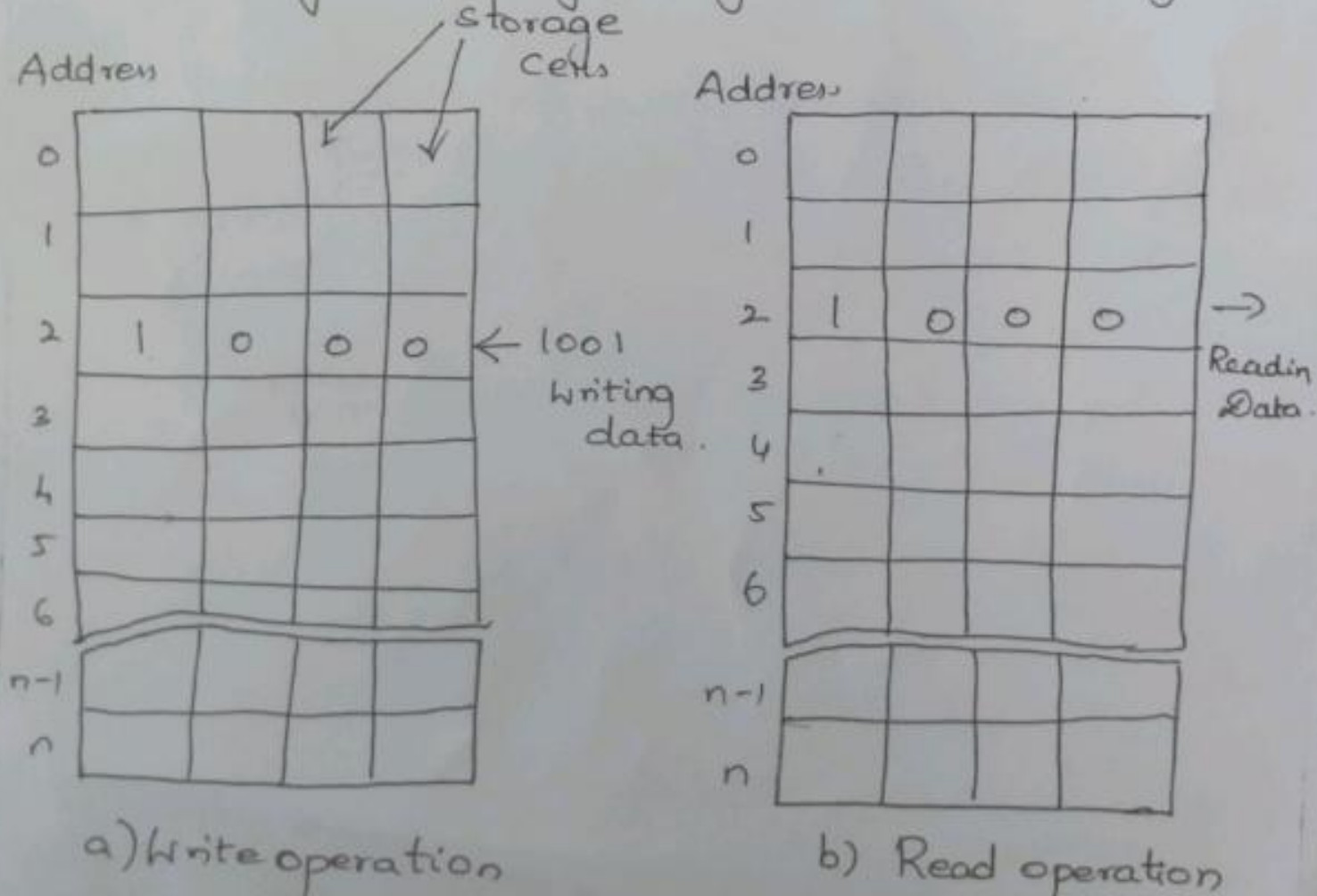
Terminology and operations:-

* Memories are made up of registers. Each register in the memory is one storage location

* Each location is identified by an address

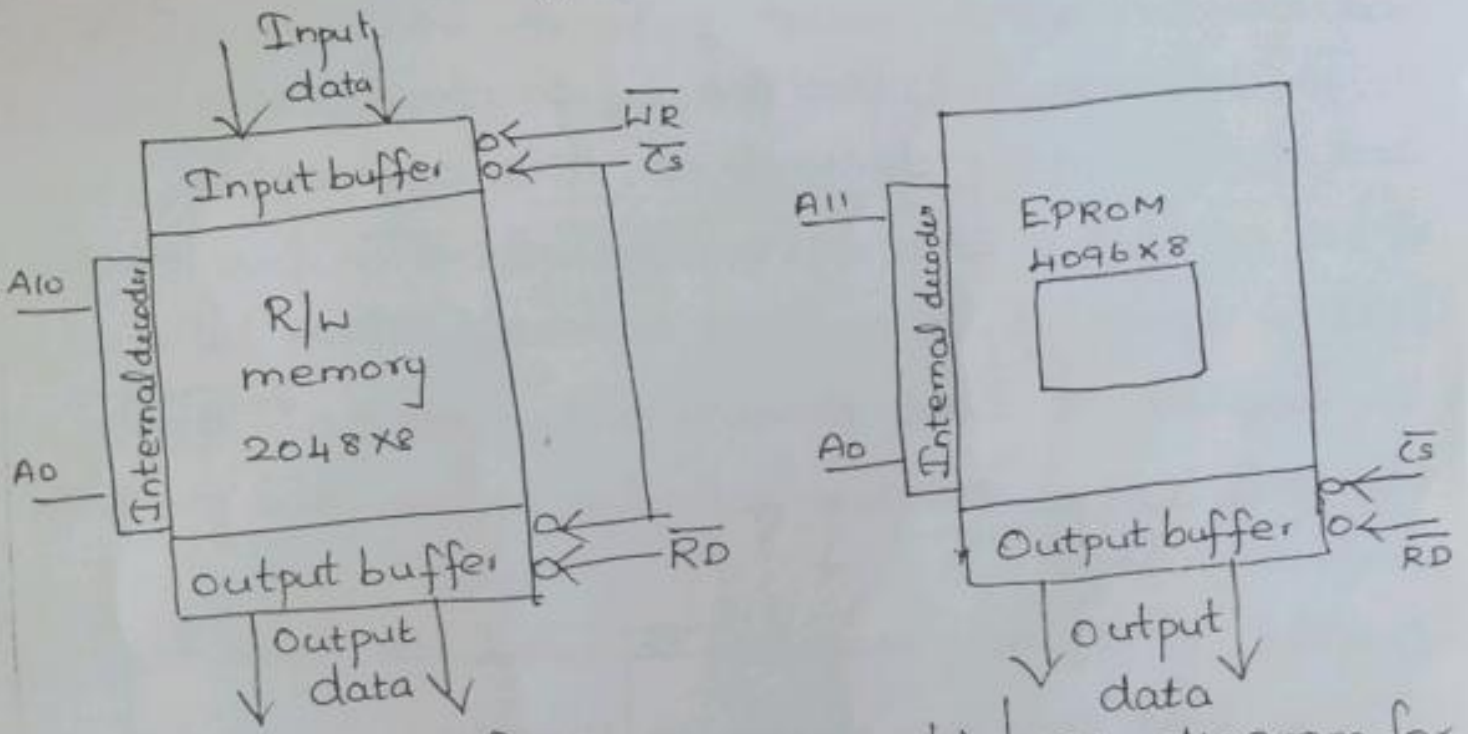
* Each location can accommodate one or more bits.
* Generally, the total number of bits that a memory can store is its Capacity.
* Each register Consists of storage elements (flipflops or Capacitor in SemiConductor and magnetic domain in magnetic storage), each of which stores one bit of data, A storage element is Called a cell.
* The data stored in a memory by a process called Writing and are retrieved from the memory by a Process called reading.

Figure shows the Concept of write, read, address and storage capacity for generalized memory.



Memory structore :-

* Read / write memories Cartsist of an array of registers, in which each register has Unique address.
* The size of the memory is N x M as shocon in fig. where N is the number of register & M is the word length, in number of bits.



a) Logic diagram for RAM

b) Logic diagram for EPROM

* As shown in the fig a) Memory chip has 11 address lines ($A_0 - A_{10}$), one chip select (CS) and two Control lines, Read ($\overline{RD}$) to enable output buffer and Write ($\overline{WR}$) to enable the input buffer. The Internal decoder is used to decode the address lines.
* As shown in fig (b) EPROM (Erasable programmable read-only memory) with 4k registers. It has 12 Address lines ($A_0 - A_{11}$), one chip select ($\overline{CS}$), one Read ctrl sig. Since EPROM is a read only memory. It does not require $\overline{WR}$ signal.

Address decoding:-

For interfacing memory devices to Microprocessor 8086 following points as follows:

1) Microprocessor 8086 can access 1 Mbyte memory since address bus is 20-bit

2) EPROM is used as a program memory and RAM is used as a data memory. When both, EPROM and RAM are used

3) The individual capacities of program memory and data memory depends on the application

4) We can place EPROM/RAM anywhere in full 1Mbyte address space. But program memory (EPROM) should be located at last memory page so that the starting address FFFF0H will lie within the program memory range

5) While interfacing memory to 8086 we have to Provide odd and even bank of memory.

- Even bank is selected When $A_0 = 0$ and
- Odd bank is selected When $\overline{BHE} = 0$

Odd and Even banks are not required for interfacing memory to 8088

* The Memory interfacing requires to

a) Select the chip

b) Identify the register

c) Enable the appropriate buffer.

Address Decoding Techniques :-

* There are three address decoding techniques:

1) Absolute decoding
2) Linear decoding
3) Block decoding

Absolute decoding:-

* In absolute decoding technique the memory chip is selected only for the specified logic level on the address lines.

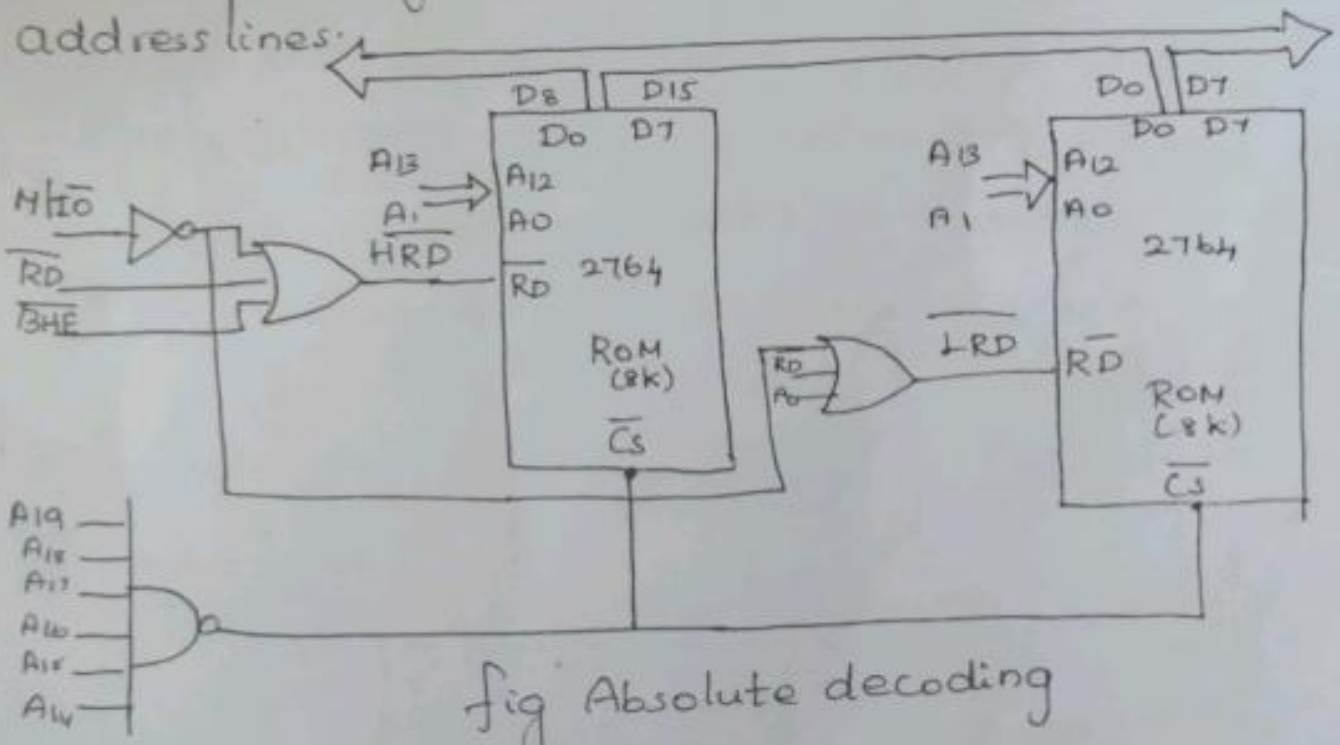


fig Absolute decoding

* fig shows the memory interface with absolute decoding.
* Two 8k EPROMs (2764) are used to provide even & odd memory banks
* Control signals $\overline{BHE}$ & A_0 are used to enable outputs of odd and even memory banks respectively.
* Each memory chip has 8k memory locations.

thirteen address lines are required to address each locations, independently.

* All remaining address lines are used to generate an unique chip select signal.
* When address lines $A_{19} - A_{14}$ are logic 1, the output of NAND gate is low and memory ICs are selected.

Linear decoding:-

* In small systems, Hardware for the decoding logic can be eliminated by using only required number of address lines. other lines are simply ignored This technique is referred as Linear decoding or Partial decoding.

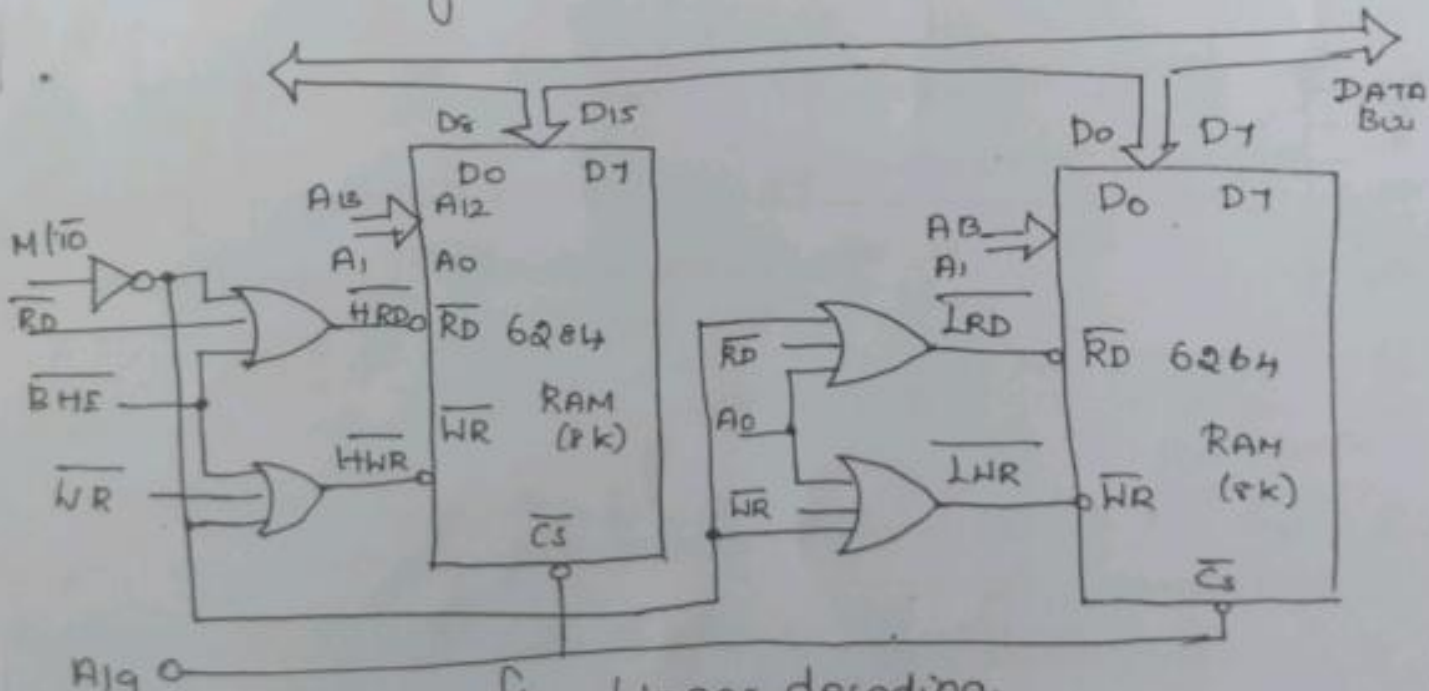


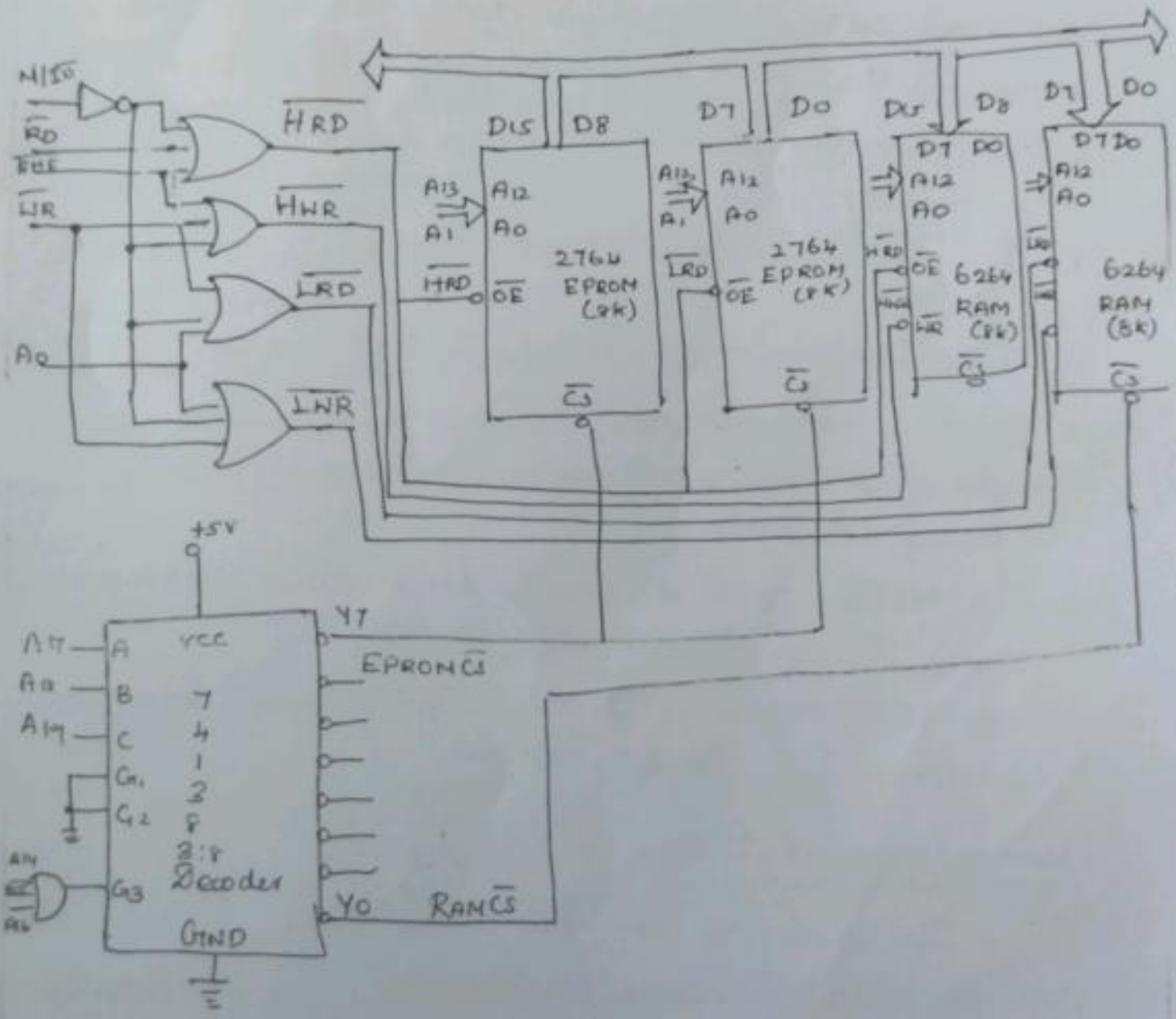
fig Linear decoding.

* Control signals $\overline{BHE}$ and A_0 are used to enable odd and even memory banks, respectively

Block Decoding:-

* In a microComputer system the memory array is consists of several blocks of memory chips. Each block of memory requires decoding circuit.
* To avoid seperate decoding for each memory block special decoder IC is used to generate chip select signal for each block

figure shaw the block decoding technique using 74138 , 3:8 decoder.



I/o Interfacing :-

I/o interfacing Technique in 8086 :

Input / output devices can be interfaced with Microprocessor systems in two ways :

1. I/o mapped I/o
2. Memory mapped I/o

I/o mapped I/o :

* The 8086 has four special Instructions IN, INS, OUT and OUTS to transfer data through the Input / output ports in I/o mapped I/o system.
* The IN Instruction Copies data from a port to the accumulator.
* If an 8-bit port is read, the data will go to AL and
  If an 16-bit port is read, the data will go to AX
* On the other hand, The OUT Instruction Copies a byte from AL or a word from AX to the Specified port.
* The M/$\overline{IO}$ signal is always low when 8086 is executing these Instruction
* so M/$\overline{IO}$ signal is used to generate seperate address for Input / output.

Memory mapped I/o :-

* In this type of I/o interfacing, the 8086 uses 20 address lines to identify an I/o device.

* An I/O device is Connected as if it is a mem register.

* The 8086 use same Control signals and instructio to access I/O. Here $\overline{RD}$ and $\overline{WR}$ signals are activated when M/$\overline{IO}$ signal is high, indicating memory bus Cycle.

I/O Device selection:

To select an appropriate I/O device its shown in the following

1. Decode the address bus to generate Unique signal Corresponding to the device address on the address bus
2. When device address signal and Control signal ($\overline{IORC}$ or $\overline{IOWC}$) both are low, generate device select signal.
3. Use device select signal to activate the interfacing device. (I/O port)

Figure shows the absolute decoding circuit for the I/O device

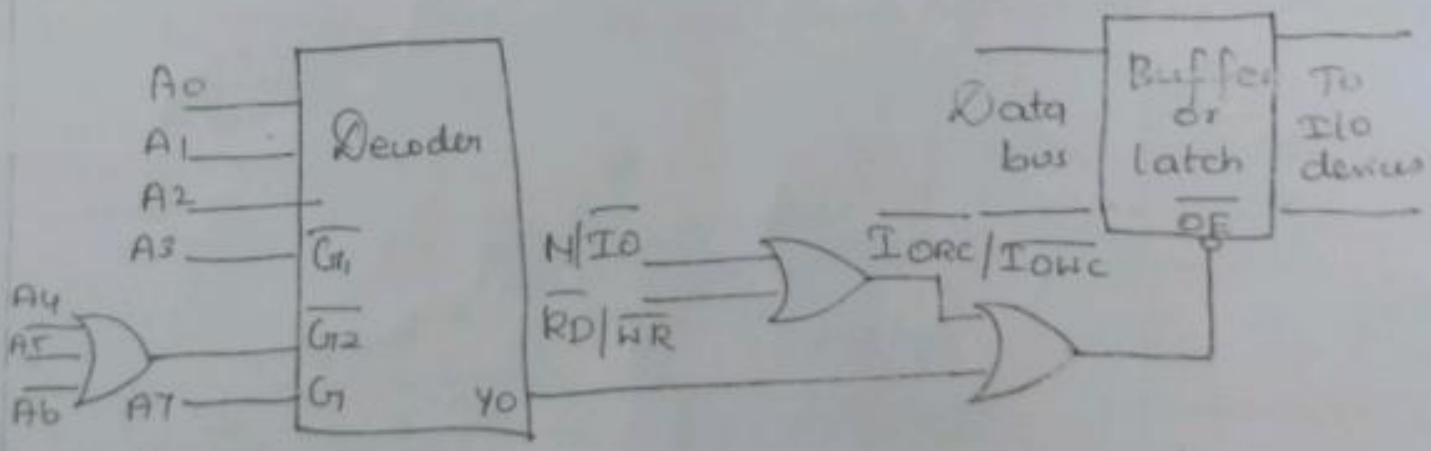


fig: Absolute decoding Circuit for the I/O devices.

* The address for this input device is 80H as device select signal goes low when address is 80H.

* When the switch is in the released position, the status of line is high otherwise status is low. With this information microprocessor can check a particular key is pressed or not.

The following program checks whether the switch 2 is pressed or not

Program:-

```
IN    AL, 80H     ; Read status of all switches
AND   AL, 02H     ; Mask bit position for other
                    Switches
JZ    NEXT        ; If Z=0, i.e switch 2 is pressed
                    then
                  ; Program control is transferred
                    to label NEXT.
```

Interfacing 16-bit Input port :-

* The circuit diagram to interface 16-bit input Port (buffer) which is used to read the status of 16 switches
* The address for this input port is 80H as device select signal goes low when address is 80H

The following program checks whether the switch is is pressed or not

PROGRAM:

```
IN    AX, 80H
AND   AX, 8000H
JZ    NEXT
```

Fig: Circuit diagram to interface 16-bit input p

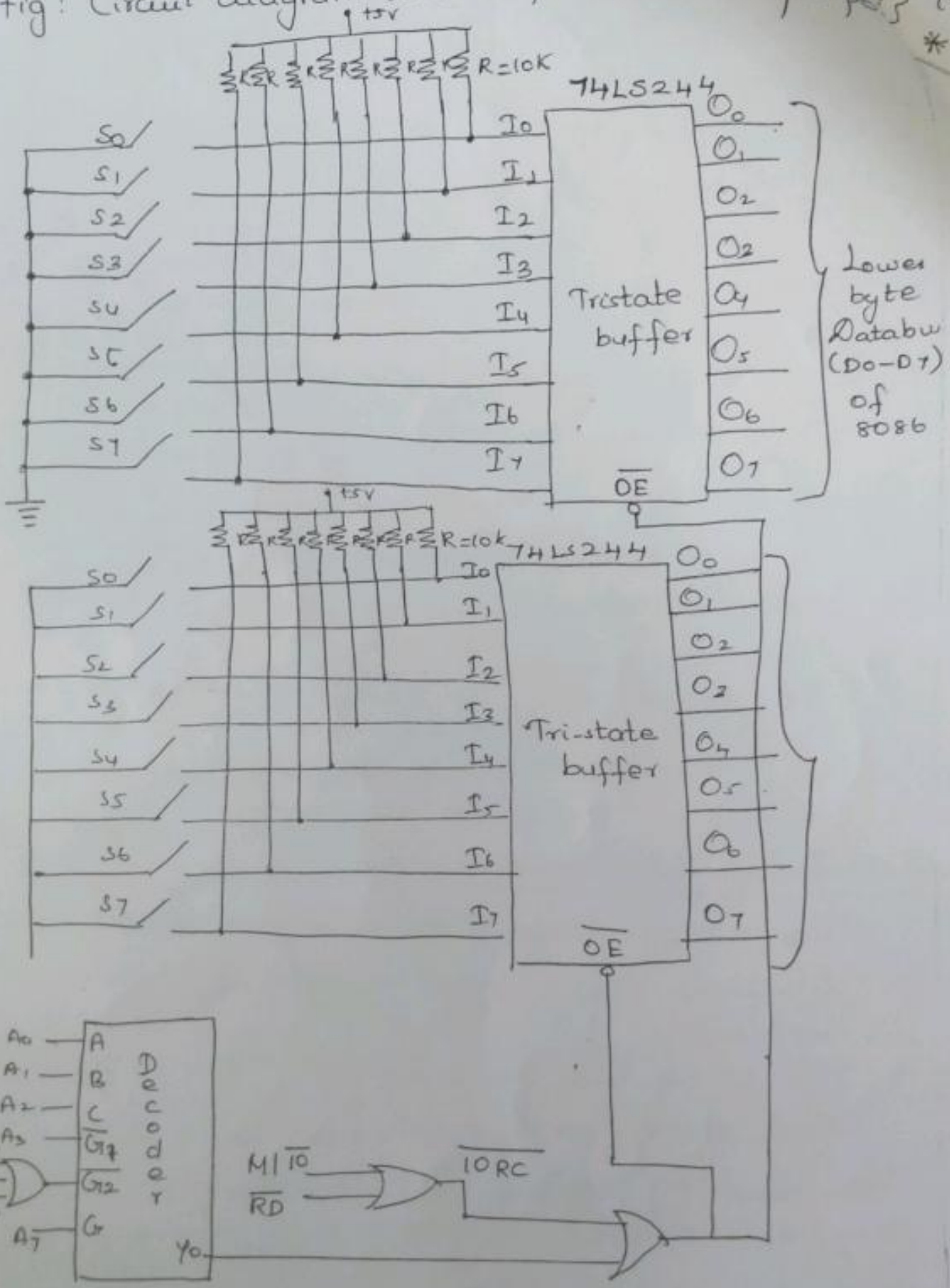


Fig: Circuit diagram to interface 16-bit inputport

Timer :-

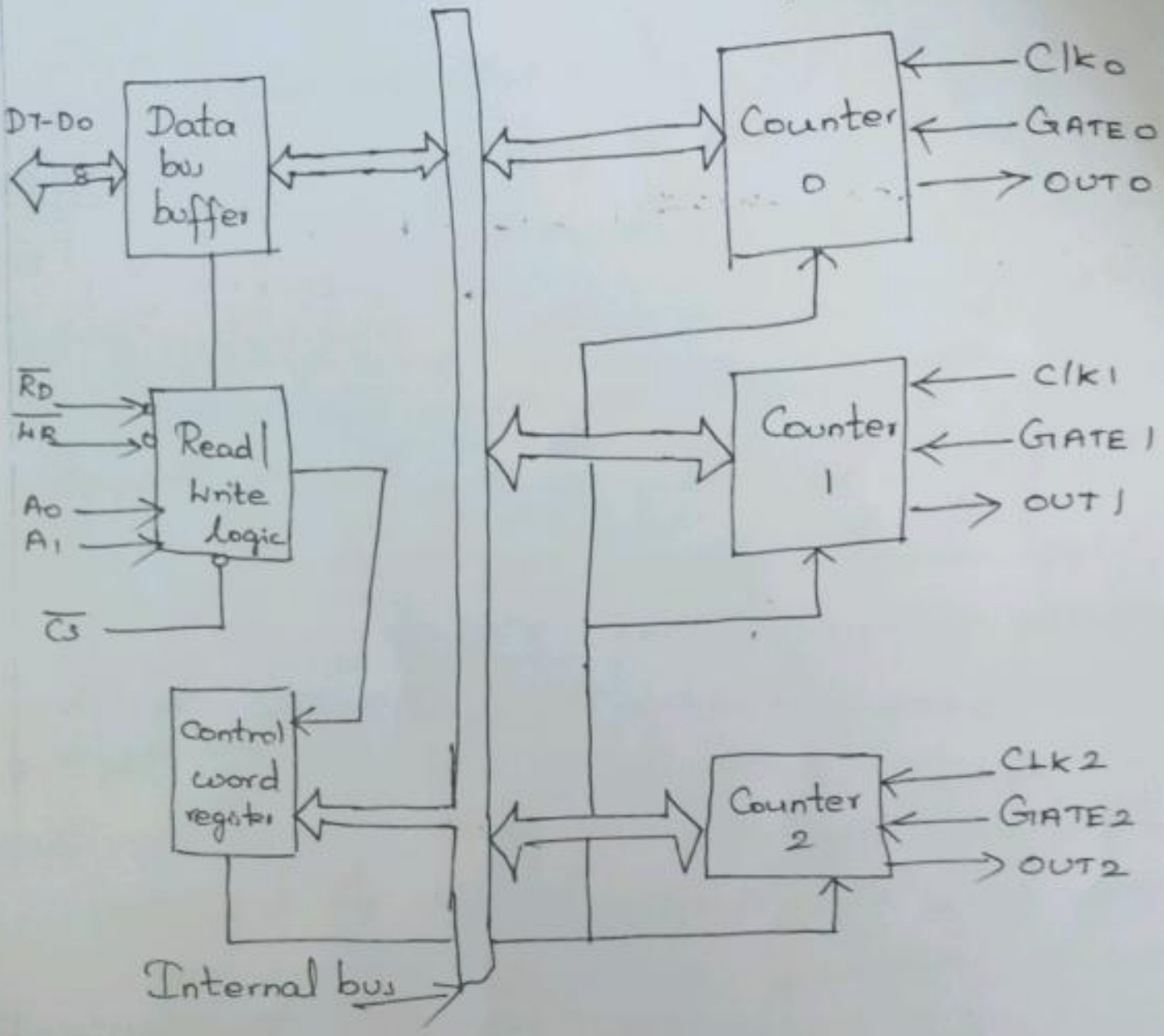
Programmable interval Timer INTEL 8253 and INTEL 8254

* One of the most Common problems in any microprocessor System is the generation of accurate time delays under Software Control.
* The 8253 solves this problem by facilitating three 16-bit programmable Counters on the same chip.
* The programmer Configures the 8253 to match his requirements and initializes one of the Counters with the desired quantity.
* The 8253, on receiving the Command from the Cpu, Count out the delay and interrupt the cpu at the end, thus reducing the Software Overhead. other function which can be acheived by 8253, are as follows.

- Even Counter
- Programmable Rate Generator
- Binary Rate multiplier
- Real Time clock
- Digital one-shot
- Complex Motor Controller.

* The functional Block diagram of the 8253 is shown in figure.

Functional block diagram of 8253 chip



Operating Modes:-

Mode 0: — Interrupt on terminal Count:

* This mode is Used for event Counting
* The gate input is used to enable or disable Counting
* When the GATE input is high, the Counting is enabled

When the GATE input is low the Counting is disabled.

* After the Counting is loaded into the Selected Count register, the OUT remains low and the Counter [illegible] the Counter is loaded.
* The OUT remains high until the Count register is reloaded or the Control word is written.
* The OUT pin can be Connected to any interrupt request pin of the microprocevor.

Mode1:— Programmable one-shot:

* The GATE Input is used to trigger the OUT.
* The OUT will be initally high and go low on the Count following the rising edge at the Gate.
* When the terminal Count is reached. The OUT goes high & remain high until the clock pulse after the next trigger.
* Thus the duration of [illegible] can be programmed through the Count.

Mode 2:— Rate Grenerator:-

* The Counter in this mode act as divide-by-N Counter.
* It is used to generate a real-time clock interrupt

* The OUT will intially be high.
* When the Count has decremented to 1, The ou
goes low for one clock pulse.
* Then the OUT goes high again, the Counter
reloads the initial Count and the process is repeated
* The GATE input, when low, disable the Counting
* When the GATE input goes high, the Counter
starts from the initial Count. The OUT gives one
Pulse after every N clockpulse

Mode 3 :- Square wave Generator :

* It is similar to mode 2 except that the
OUT remains high for the first half of the Count
& goes low for the Second half.
* Thus generating a programmable Square wave
shape at OUT.
* suppose n is the number loaded into the
Counter, then the OUT will be
  - High for $N/2$ Counts & low for $n/2$ Counts if n = even
  - High for $(n+1)/2$ Counts & low for $(n-1)/2$ Counts if n = odd.
* The GATE input is same as that in Mode 2

Mode 4: — Software triggered strobe

* The OUT is initially high and goes low for one clock period on the terminal Count.
* The GATE input when low, disables the Counting and when high, enables the Counting.
* The difference between Mode 4 and Mode 2 is that in Mode 2.
* The OUT pulses are generated Continuously after every N clock pulses
* But in mode 4, The OUT pulse is generated only once after N clock pulses

Mode 5: — Hardware triggered strobe

* The OUT is initially high.
* The Counter will start Counting after the rising edge of the GATE input
* The OUT will go low for one clock period when the terminal Count is reached.
* The Hardware Circuit should trigger the GATE input to initiate the Counting operation thereby generating OUT pulse.

Keyboard and Display Controller (INTEL 8279)

* In the keyboard Interface,

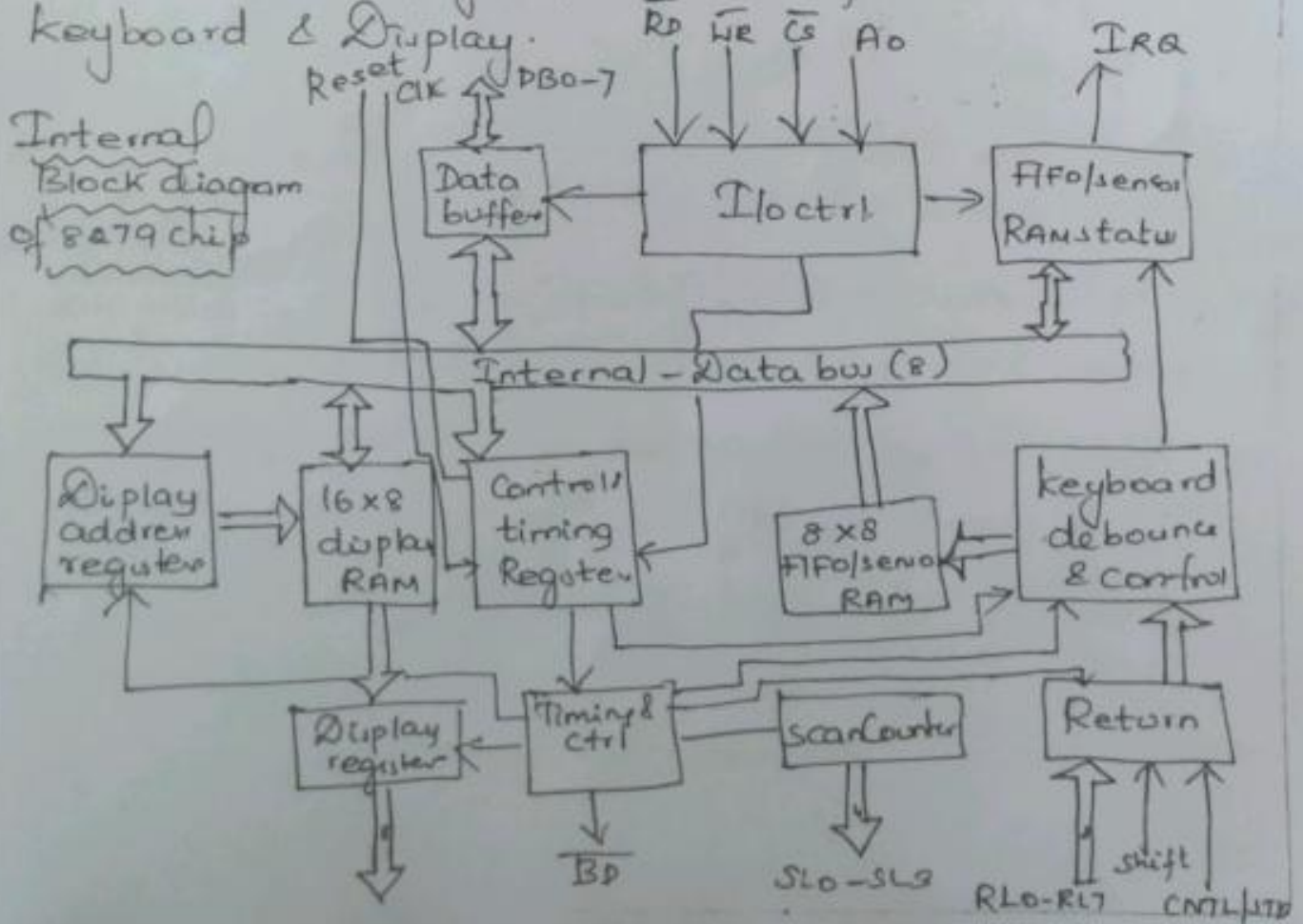
The microprocessor scan the Various keys, Perform software debouncing and finally finds out the Code of the depressed key.

* In the Display Interface,

The microprocessor is busy in sending bit-by-bit information to the shift registers regarding the display of Various Segments.

* The 8279 (figure) Consists of two sections - keyboard & Display.

Internal Block diagram of 8279 chip



* Block diagram Consist of 4-main section :

i) Cpu Interface & Control
ii) Scan section
iii) keyboard section
iv) Display section

i) cpu Interface & Control :-

* Data buffer : Data buffer is 8-bit bidirectional buffers that Connect the internal data bus to the external data bus.

* I/o Control : The I/o Control section uses the A_0, $\overline{CS}$, $\overline{RD}$, $\overline{WR}$ signal to Control dataflow to and from Various Registers and buffers. The dataflow from 8279 is enabled only when $\overline{CS}=0$.

A_0	$\overline{RD}$	$\overline{WR}$	Interpretation
0	1	0	Data from cpu to 8279
0	0	1	Data to cpu from 8279
1	1	0	Command word from cpu to 8279
1	0	1	status word to cpu from 8279

* Timing & Control registers :

The Timing & Control registers store the keyboard and display mode and other operating Conditions programmed by the cpu.

The modes are programmed by sending proper Command with $A_0 = 1$

*Timing Control:
The timing control Consist of basic timing Counter chain. The first Counter is divided by N-prescalar to give frequency of 100 kHz.

ii) Scan section:-
Scan section has a Scan Counter which has two modes. Those are Encoded mode & decoded mode.

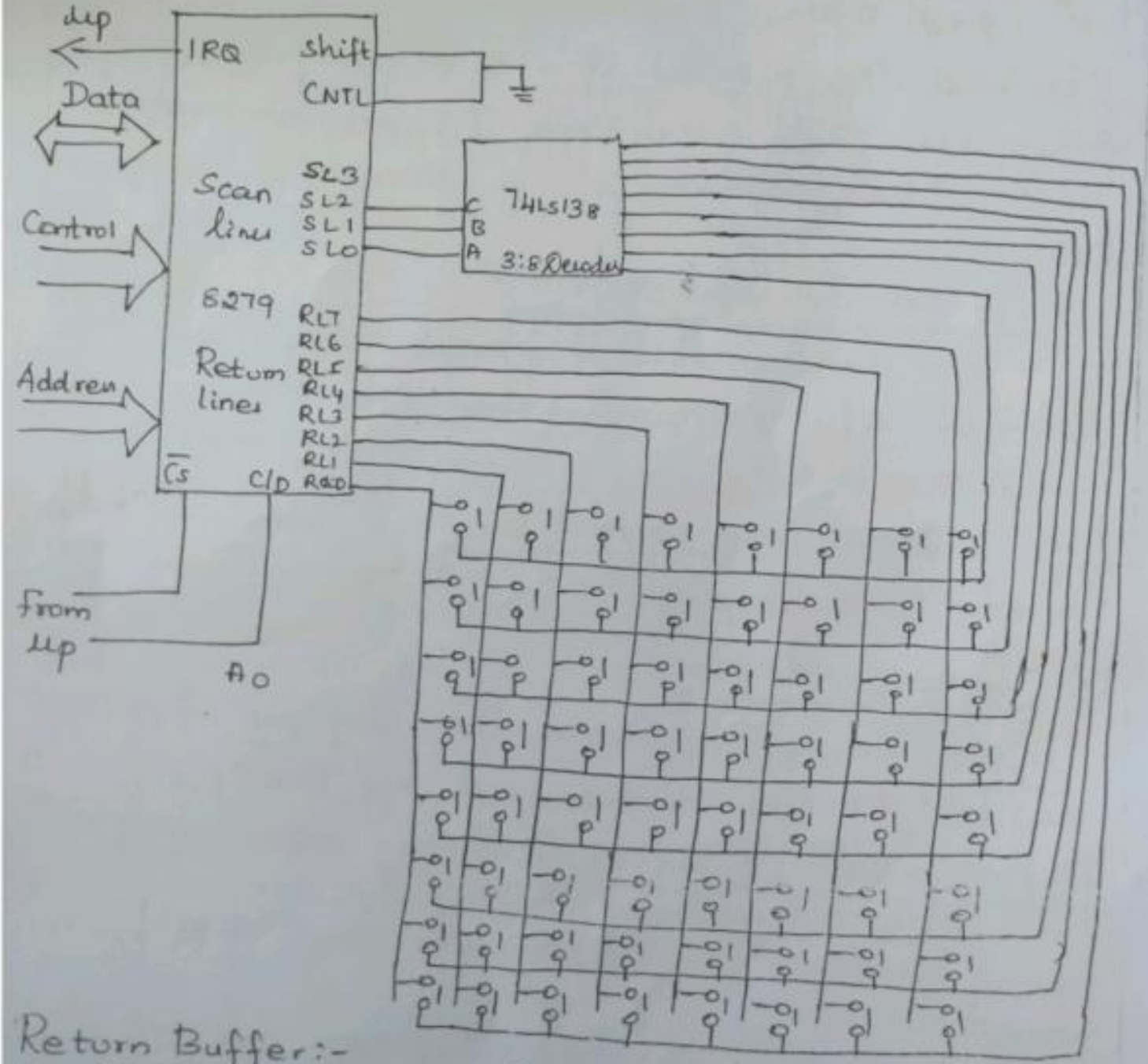
* Encoded mode: In Encoded mode the Scan-Counter provides a binary Count from 0000 to 1111 on the four Scan lines ($S_{L3} - S_{L0}$) with high outputs. The Count is externally decoded to provide 16-scan lines.

* Decoded mode: In decoded mode the decoders decode the least significant 2-bit binary count and provides four possible Combinations (i.e) 1110, 1101, 1011, 0111.
Thus the output of decoded scan is low

Keyboard Section:-

This section Consists of return buffer, keyboard debounce & Control / FIFO sensor RAM.

Fig : Interfacing 8x8 keyboard to the 8279 chip



Return Buffer:-

8-return lines (RL7 – RL0) are buffered and latched by the return buffer during each row Scan in scanned keyboard. It is used to interface 8x8 matrix keyboard.

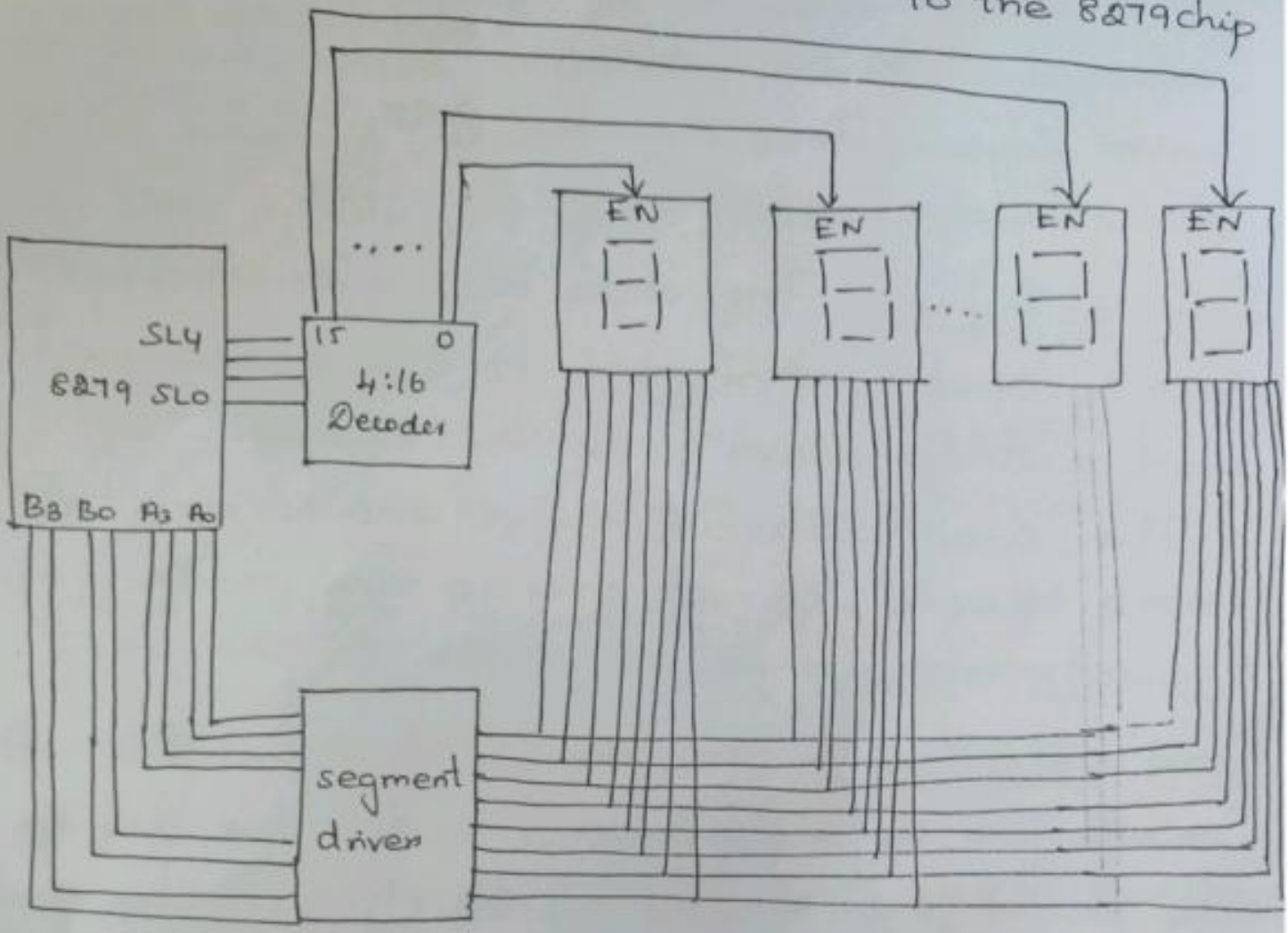
* keyboard debounce & Control:
keyboard debounce and Control is enabled only when Scanned keyboard mode is selected.

* FIFO Sensor RAM:-
8x8 RAM, In scanned keyboard and strobed i/p mode it is a FIFO. Each new entry is written into RAM. In sensor matrix mode the memory is referred as a sensor RAM.

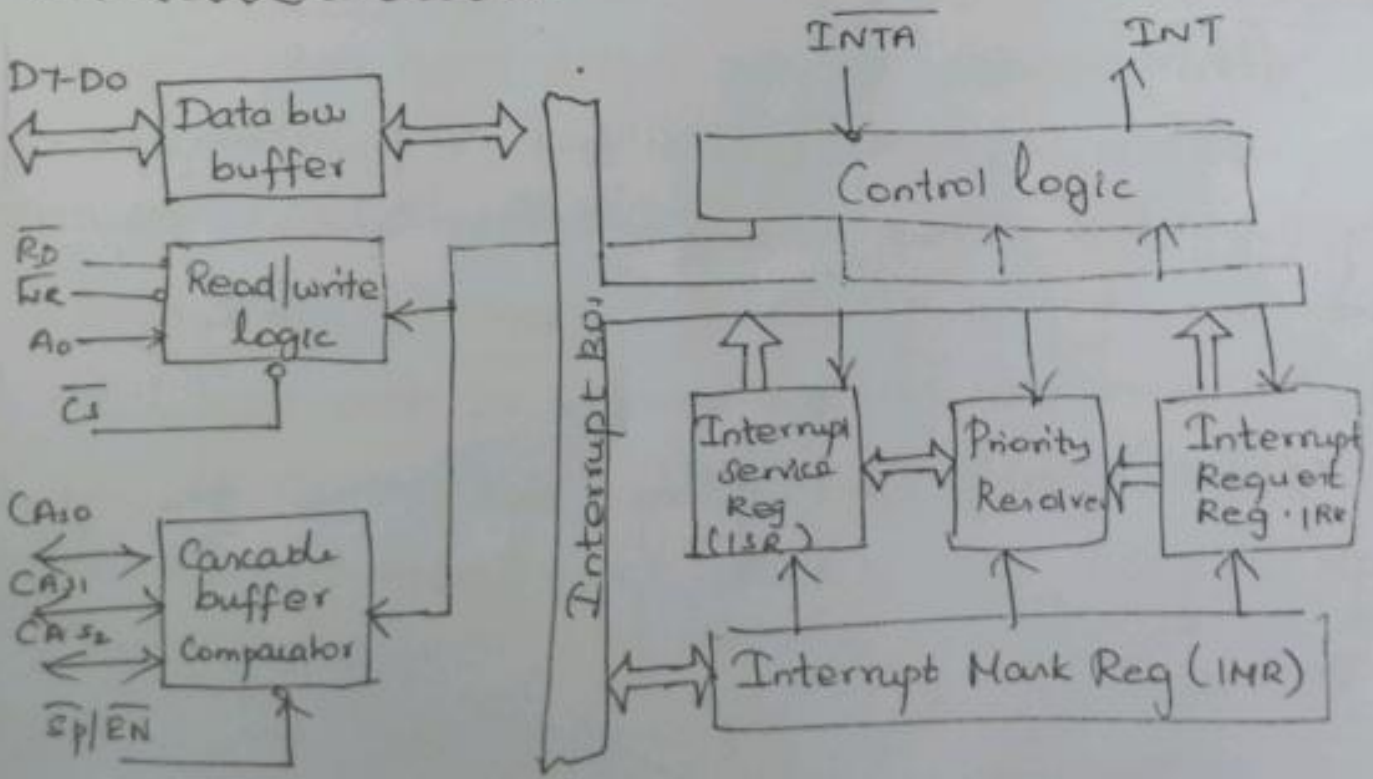
* FIFO sensor RAM status:-
FIFO sensor status keeps track of the number of character in the FIFO and whether it is full or empty. IRQ is high, when FIFO is not empty. IRQ is low when FIFO is empty.

IV) Display section:-
Display RAM: It stores the display for 16-digits. It can be accessed by the cpu
Display address Register: Display address register hold the address of the byte Currently being written or read by the cpu & the scan Count Value.
Display registers: Display reguters are of two-4 bit reguters namely A & B. They hold the bit pattern to be displayed.

fig: Interfacing the seven segment LED display to the 8279 chip



Interrupt Controller:-



* The Internal functional Blocks of interrupt Controller data bus buffer, Read/write logic, Control logic, three registers (IRR, ISR & IMR), Priority resolver and Cascade buffer.

Data Bus Buffer:- The data bus buffer allows the 8085 to send Control words to the 8259A and read a status word from the 8259A.

The 8-bit data bus buffer also allows the 8259A to send interrupt opcode and address of the interrupt service subroutine to the 8085.

Read/Write logic:- The Read ($\overline{RD}$) and $\overline{WR}$ inputs Control the data flow on the data bus, when the device is selected by asserting its chip select ($\overline{CS}$) input low.

Control logic:

This block has an input and an output If the 8259A is properly enabled, the interrupt request will cause the 8259A to assert its INT output pin high. If this pin is Connected to the INTR pin of an 8085 & if the 8085 Interrupt enable (IE) flag is set, then this high signal will cause the 8085 to respond INTR.

Interrupt Request Register (IRR):-

The IRR is used to store all the interrupt levels which are requesting the service. The eight interrupt inputs set corresponding bits of the interrupt request Register upon service request.

Interrupt serice Register (ISR):-

The Interrupt Service Register (ISR) stores all the level that are Currently being serviced.

Interrupt Mask Register (IMR):-

Interrupt mask register (IMR) stores the masking bits of the interrupt lines to masked An interrupt which is masked by software will be recognized and serviced even if it sets the Corresponding bits in the IRR.

Priority Resolver:- The priority resolver determines the priorities of the bits set in the IRR. The bit Corresponding to the highest priority interrupt input is set in the ISR during the $\overline{\text{INTA}}$ Input.

Cascade Buffer Comparator:-

It generates Control signals necessary for Cascade operations. It also generate Buffer-Enable signals.

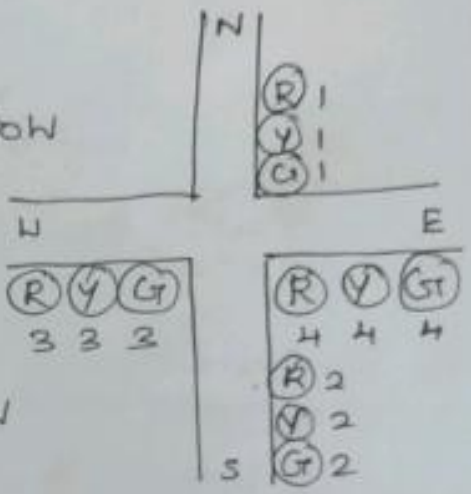
The 8259 can be set up as a master or a slave by the $\overline{\text{SP}}$ / $\overline{\text{EN}}$ pin.

Programming and applications Case studies:

Traffic Light Control:-

Design an microprocessor System to Control traffic lights. The traffic light arrangement is as shown in fig. The Traffic should be Controlled in the following manner.

1) Allow traffic from N to E and E to N transition for 20 seconds
2) Give Transition period of 5 seconds (Yellow bulbs ON)
3) Allow traffic from N to S and S to N for 20 seconds
4) Give Transition period of 5 seconds (Yellow bulbs ON)
5) Repeat the process.



Figure

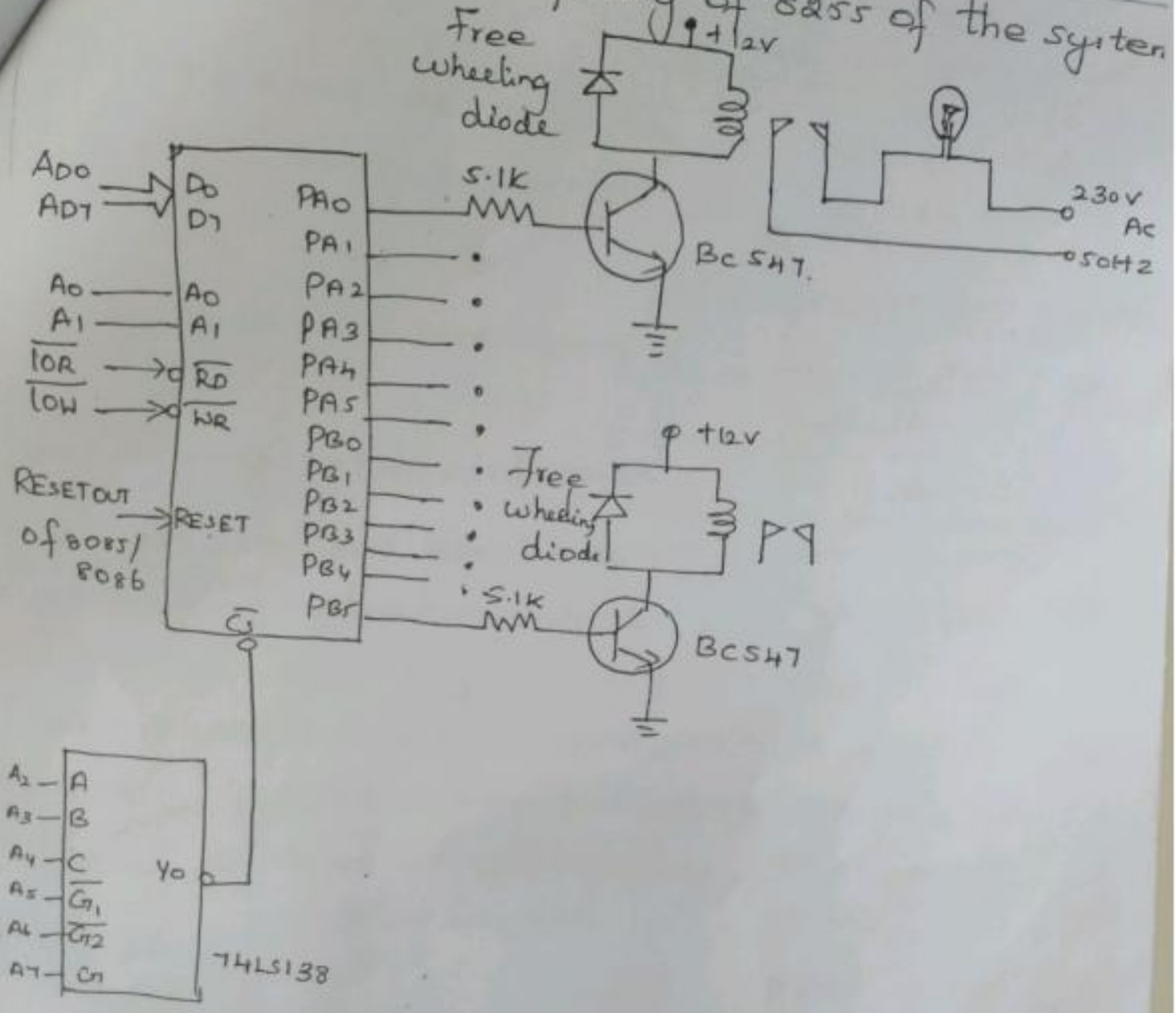
Solution:

Hardware: Table shows the interfacing diagram to control 12 electric bulbs

PORTA ⇒ Control lights on N-S road
PORTB ⇒ Control lights on W-E road

Pins	Light	Pins	Light
PA0	R1	PB0	R3
PA1	Y1	PB1	Y3
PA2	G1	PB2	G3
PA3	R2	PB3	R4
PA4	Y2	PB4	Y4
PA5	G2	PB5	G4

g shows the Interfacing of 8255 of the syster



IO Map:

Ports / control register	Address lines A7 A6 A5 A4 A3 A2 A1 A0	Address
PORTA	1 0 0 0 0 0 0 0	80H
PORTB	1 0 0 0 0 0 0 1	81H
PORTC	1 0 0 0 0 0 1 0	82H
Control Register	1 0 0 0 0 0 1 1	83H

Software: Command word : for Initalization of 8255

BSR/IO	MODE A		PA	PCH	MODE B	PB	PCL
1	0	0	0	X	0	0	X

Source program for 8086:

```
        MOV AL, 80H     ; Initalize 8255 portA & portB
        MOV 83H, AL     ;  in output mode

START:  MOV AL, 09H
        OUT 80H, AL     ; Send data on PA to glow R1 & R2
        MOV AL, 24H
        OUT 81H, AL     ; Send data on PB to glow G3 & G4
        MOV CL, 28H     ; Load multiplier Count (40)10 for delay
        CALL DELAY      ; Call delay subroutine
        MOV AL, 12H
        OUT 80H, AL     ; Send data on port A to glow Y1 & Y2
        OUT 81H, AL     ; Send data on Port B to glow Y3 & Y4
        MOV CL, 0AH     ; Load multiplier Count (10)10 for delay
        CALL DELAY      ; Call delay subroutine
        MOV AL, 24H
        OUT 80H, AL     ; Send data on port A to glow G1 & G2
        MOV AL, 09H
        OUT 81H, AL     ; Send data on port B to glow R3 & R4
        MOV CL, 28H     ; Load Multiplier Count (40)10 for delay
        CALL delay      ; Call delay subroutine
        MOV AL, 12H     ;
        OUT, 80H, AL    ; Send data on port A to glow Y1 & Y2
        OUT 81H, AL     ; Send data on Port B to glow Y3 & Y4
        MOV CL, 0AH     ; Load multiplier Count (10)10 for delay
        CALL DELAY      ; Call delay subroutine
        JMP START       ; Repeat the process
```

DELAY Subroutine:

```
DELAY :   MOV BX, COUNT ; Load Count to give 0.5se delay
BACK :    DEC BX        ; Decrement Count
          JNZ BACK      ; if not zero, repeat
          DEC CL        ; decrement multipler Count
          JNZ DELAY     ; if not zero, repeat
          RET           ; Return to main program
```

LED display

Interface an 8-digit 7 segment LED display using 8255 to the 8086 microprocessor System and write an 8086 assembly language routine to display message on the display.

Solution:

Hardware:

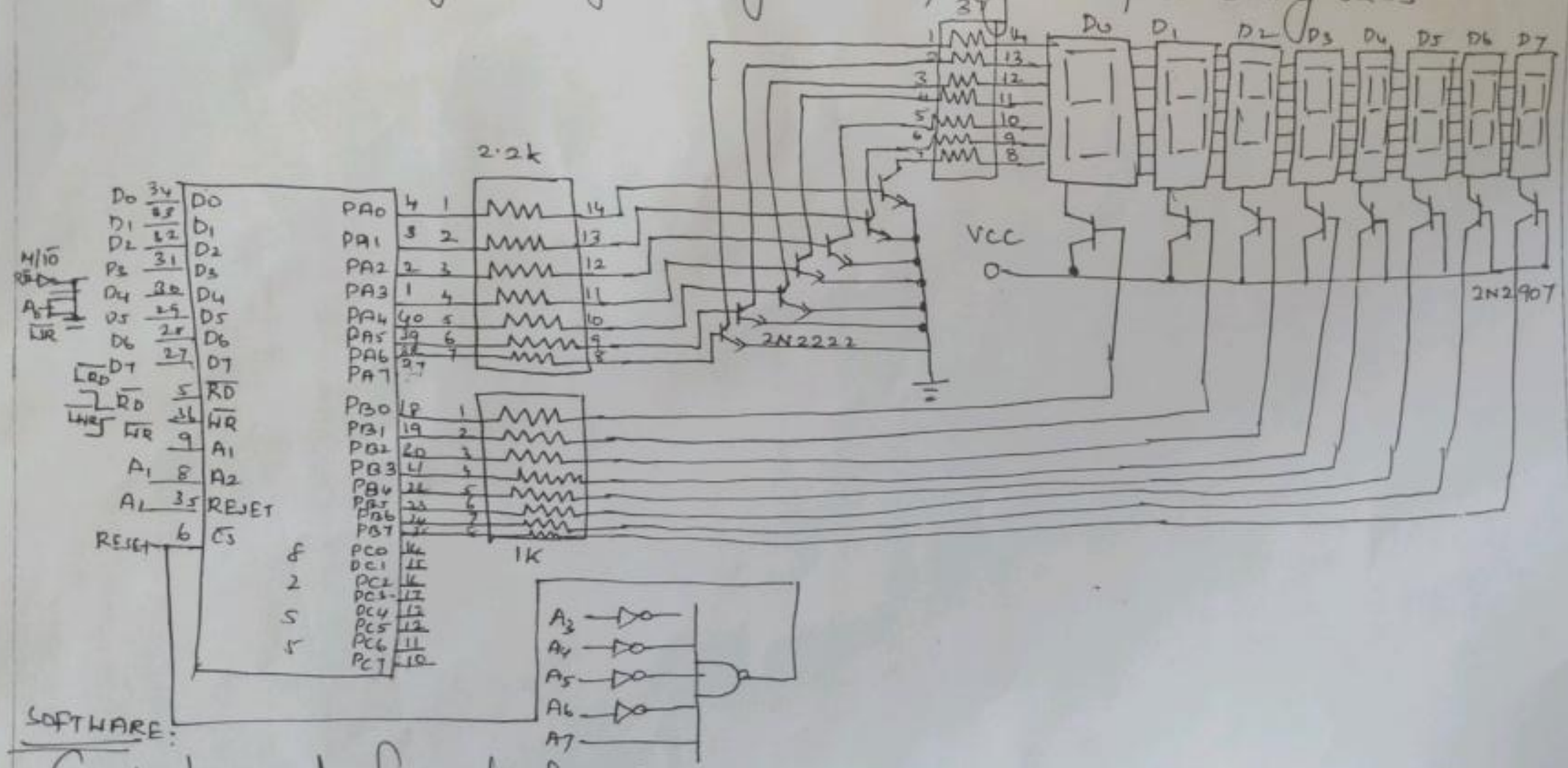
The figure shows the multiplexed eigit digit 7 segment display Connected in the 8086 system using 8255.

For this Circuit different addresses are

PA = 00H PC = 04H

PB = 02H CR = 06H

Fig 8 digit 7Segment display interface Using 8255



Control word format for 8255

BSR	Mode A		PA	PC U	Mode B	FB	PCL
1	0	0	0	X	0	0	X

gram:

```
.MODEL SMALL
.DATA
    PA   EQU  80H
    PB   EQU  82H
    CR   EQU  86H
    MES DB 41H,42H,43H,44H,45H,46H,47H,48H
.CODE
; Procedure to display message on multiplexed
                                  LED display
DISP    PROC NEAR
        MOV AX, @DATA      ;[Initialise
        MOV DS, AX         ; data segment]
        MOV AL, 80H        ;Load ctrl word in AL
        OUT CR, AL         ; Load Ctrl word in CR
        PUSH F             ; save register.
        PUSH AX
        PUSH BX
        PUSH DX
        PUSH SI
; setup register for display
        MOV BX, 08H        ; Load Count
        MOV AH, 7FH        ;Load select pattern
        LEA SI, MES        ;starting addr of msg
; display message
DISP1:  MOV AL, AH         ; select digit
        OUT PB, AL
```

```
MOV AL,[BX+SI]      ; get data
OUT PA,AL           ; display data
CALL DELAY          ; wait for some time
ROR AH,01H          ; adjust selection pattern
DEC BX              ; adjust Count
JN2 DISP1           ; Repeat 8 times
POP SI              ; restore register.
POP DX
POP BX
POP AX
POP F
RET
DISP ENDP
```

LCD Display:

LCD interface with 8086:

Liquid Crystal Display (LCD) Module:

* LCD is a thin, flat display device made up of pixcels arranged in front of a light source or reflector. We Very small amounts of electric Power, and is therefore suitable for use in battery – powered electronic devices

Characteristics:

* Each module Contains a CMOS Controller and all necessary drivers which have low power

sumption. The Controller is equipped with an internal character generator ROM, RAM and RAM for display data

* All display functions are Controllable by Instructions making interfacing.

```
CODE SEGMENT
        ASSUME    CS:CODE,DS:CODE,ES:CODE,SS:CODE
        ;
        ;
START:  ORG  0H   ; Use 1000H for MDA series
        MOV AX,CS
        MOV DS,AX ; Making the DS and CS
                  ; Value same
        ;
        MOV SS,AX ; making the ss also same with
                    CS ; DS
        MOV SP,STACK
        ;
        CALL ALLCLR
        ;
        CALL LN11
        MOV SI,OFFSET LINE1
        CALL STRING1
        ;
        CALL LN21
        MOV SI,OFFSET LINE2
        CALL STRING1
```

```
;Blinks the whole display

BLINK: CALL DISPOFF
       CALL TIMER
       CALL DISPON
       CALL TIMER
       JUMP BLINK
;
LINE1 DB 'Hi BUET students!',00H,00H
LINE2 DB 'Make me friend!',00H,00H
;
;LCD Instruction
ALLCLR: MOV AH,00000001B ;clear entire display
        JMP OUT1
;
DISP OFF:
        MOV AH,00001000B; Display off, Cursor off,
                                    not blink
        JMP OUT1
;
DISPON: MOV AH,00001111B; Display on, Cursor on,
                                    Cursor blink

        JMP OUT1
;
LN11: MOV AH 10000010B ; Return to home
                                    position
      JMP OUT1
;
```

```
21:  MOV  AH,11000000B; Set RAM address so
                        that the Cursor is positioned
                     ; at the head of the 2nd line
     JMP  OUT1
     ;
     ; To write to Instruction register
OUT1 : PUSH   AX
       PUSH   DX
       CALL   BUSY
       MOV    AL,AH
       MOV    DX,IR_WR
       OUT    DX,AL
       POP    DX
       POP    AX
       RET
     ; busy flag check must be done before
                        any write operation
BUSY : PUSH   DX
       MOV    DX,ST_RD
BUSY1 : IN    AL,DX
       AND    AL,10000000B
       JNZ    BUSY1
       POP    AX
       POP    DX
       RET
```

```
; To send a single character
CHAR OUT:
        PUSH DX
        PUSH    AX
        CALL    BUSY
        MOV     AL,AH
        MOV     DX,DR_WR
        OUT     DX,AL
        POP     AX
        POP     DX
        RET
;To out a string line from address Cs:[SI]
STRING: MOV     AH,BYTE PTR CS:[SI]
        CMP     AH,00H
        JE      STRING1
        ;
        CALL    BUSY
        CALL    CHAROUT
        INC     SI
        JMP     STRING
STRING1:
        RET
        ; Timer make delay
TIMER:  PUSH CX
        MOV   CX,0FFFFH
TIMER1: DEC     CX
        JNZ     TIMER1
        POP     CX
        RET
        ;
CODE    ENDS
        END START
```

keyboard interfacing:

Simple keyboard interfacing:

fig shows the Simple keyboard interface

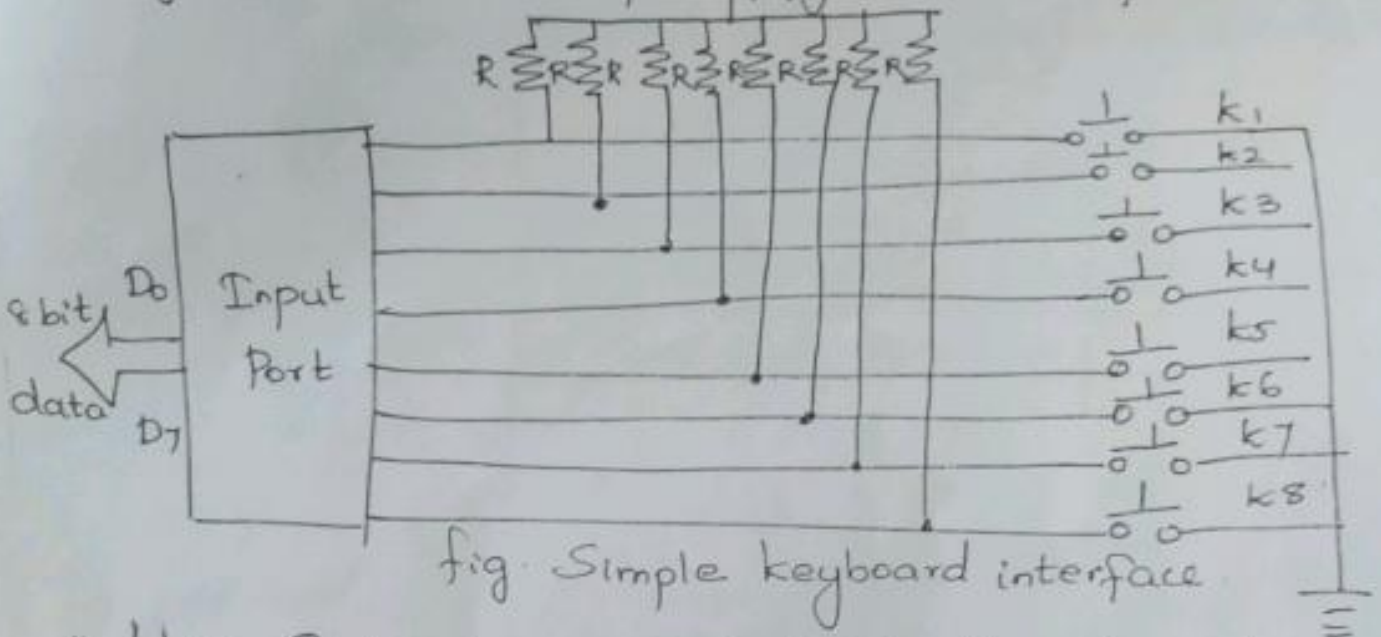


fig. Simple keyboard interface

* Here Eight keys are individually Connected to specified pin of input port.
* Each port pin gives the status of key Connected to that pin.
* When port pin is logic, key is open ow key is closed.

Software routine to get keycode with key debounce:

```
START:   IN  AL,IN-PORT          ; Read key status
         CMP AL,FFH              ;Check if keys are open
         JNZ START               ; if no, goto start otherwise
                                 ; Continue
         CALL DEBOUNCE-DELAY     ; call debounce delay
AGAIN:   IN AL, IN-PORT          ; Read key status
```

```
CMP  AL, FFH        ; Check if any key is pressed
JZ   AGAIN          ; if no, goto AGAIN ; otherwise
                    ; Continue
CALL DEBOUNCE_DELAY ; Call debounce delay
IN   AL, INPORT     ; Get key code
RET                 ; Return from subroutine
```

key	keycode							
	D7	D6	D5	D4	D3	D2	D1	D0
k1	1	1	1	1	1	1	1	0
k2	1	1	1	1	1	1	0	1
k3	1	1	1	1	1	0	1	1
k4	1	1	1	1	0	1	1	1
k5	1	1	1	0	1	1	1	1
k6	1	1	0	1	1	1	1	1
k7	1	0	1	1	1	1	1	1
k8	0	1	1	1	1	1	1	1

Matrix keyboard Interfacing :-

* In Matrix keyboard interfacing, To reduce number of Connection keys are arranged in the matrix form

* When key are open, row & Column donot have any Connection

When a key is pressed, it shorts Corresponding one row & one Column

* Figure shows the interfacing of Matrix keyboard.

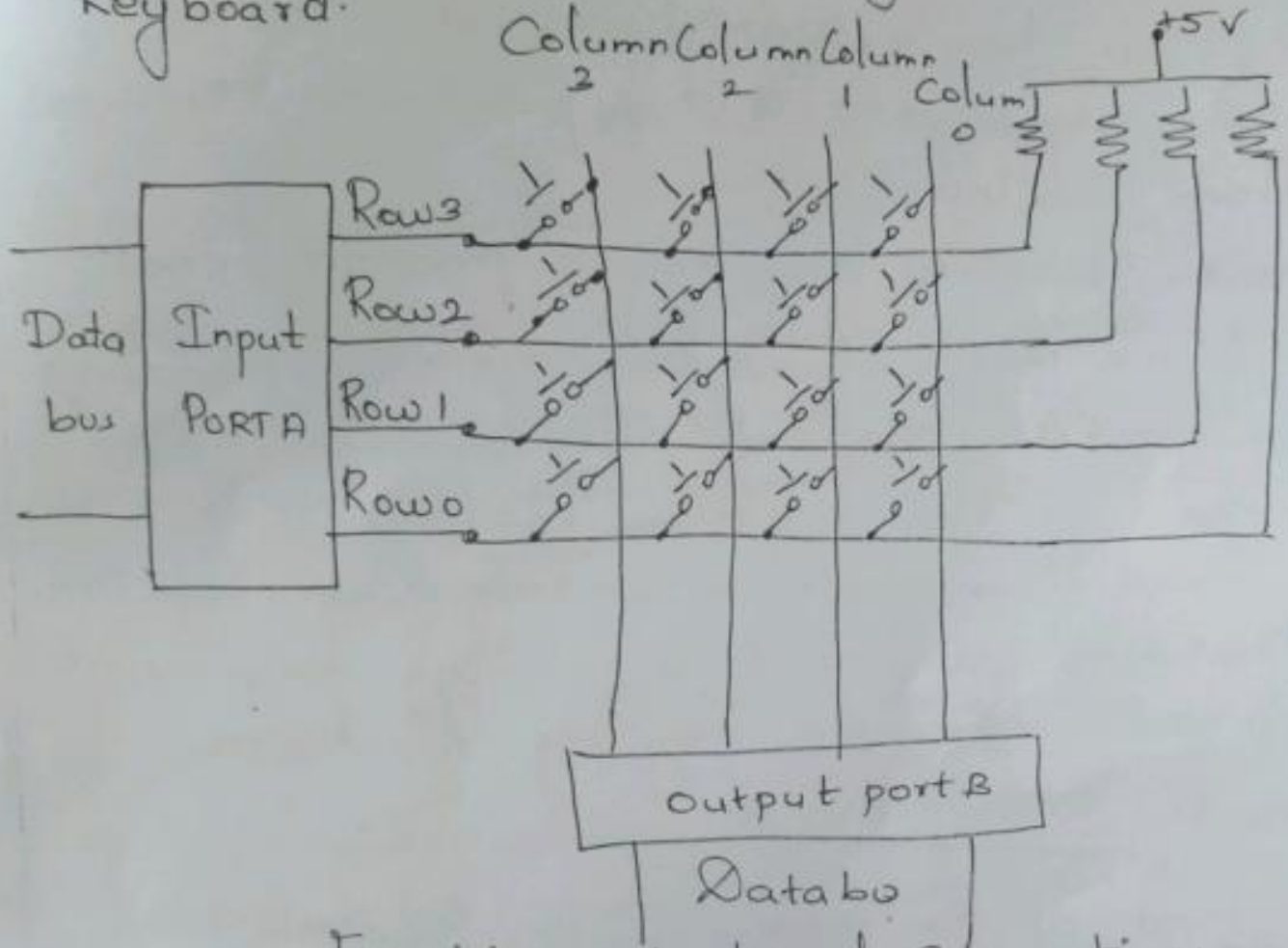


Fig: Matrix keyboard Connection

* Rows are Connected to the input port referred to as returned lines & Column are Connected to the output port referred to as Scan lines.

* When all keys are open, row & Column donot have any Connection.

When any key is pressed it shorts Corresponding row & Column.

* If the output line of this column is low it makes Corresponding row line low, o.w the Status of row line is high.
* The key is identified by data sent on the output port and input Code received from the input port.

Check: Whether any key is pressed or not?

1. Make all Column lines Zero by sending low on all o/p lines. This activates all keys is in the keyboard matrix.
2. Read the status of return lines. If the Status of all lines is logic high, key is not Pressed, otherwise key is pressed.

Display interface

* Most of the microprocessor-Controlled Instruments and machines need to display letters of the alphabet and numbers to give directions to user
* This information can be displayed Using CRT, LED or LCD displays.
* CRT displays are Used when a large amount of data is to be displayed.
* In systems small amount of data is to be displayed, simple LED & LCD displays are Used.

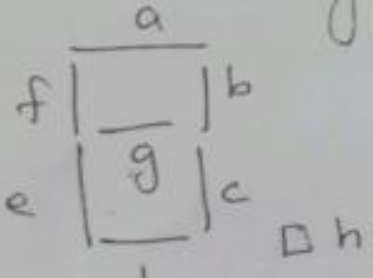
LED Displays:

LED displays are available in two very common formats they are

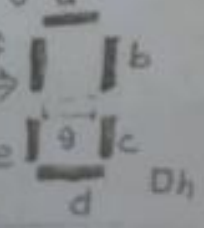
i) 7 segment displays
ii) 5 by 7 dot matrix displays

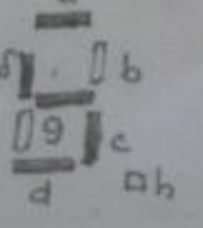
Seven segment display:

* Seven Segment displays are generally used as indicator and consists of no of LEDs arrangement in Seven Segment shown in fig



* Any number between 0 and 9 can be indicated by lighting the appropriate segments
* The Seven Segments are labeled a to g and dot is labelled as h
* By forward biasing different LED segments, we can display the digits 0 through 9.
* For instance, to display 0, we need to light up a, b, c, d, e & f. To light up 5, we need to segments a, f, g, c & d

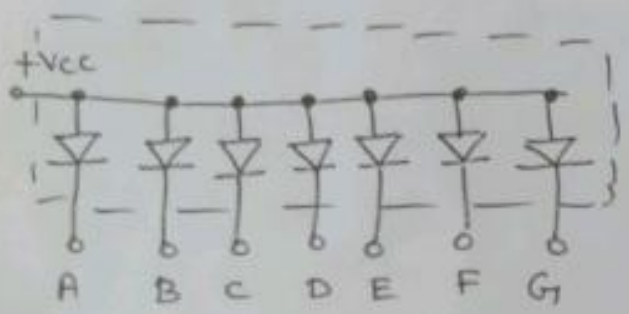
eg) 0 ⇒ 

5 ⇒ 

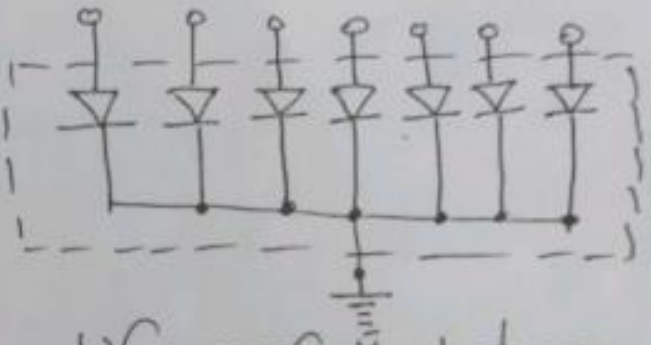
* These 7 segments displays are of two types
  - Common anode type
  - Common Cathode type

* In Common anode, all anodes of LEs are Connected together as shown in fig a)

In Common Cathode, all Cathodes are Connected together as shown in fig b)



a) Common anode type



b) Common Cathode type

5By 7 Dot Matrix LED

* Fig a & b show the 5by 7 dot matrix LED display and its Circuit Connections.

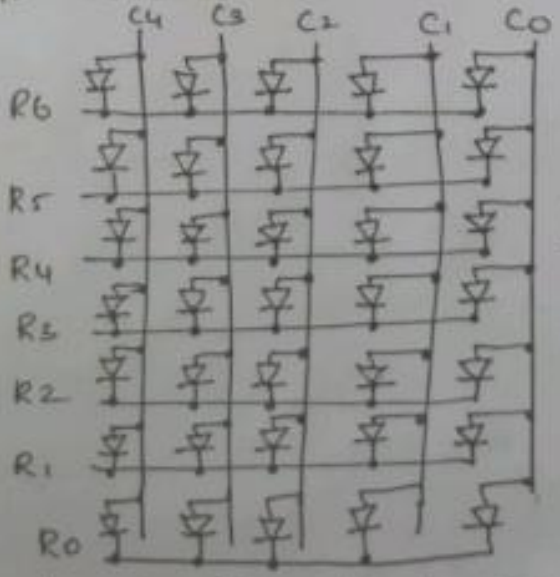


Fig: 5x7 dot matrix Circuit Connections

This display can be used to display number as well as alphabets.

Alarm Controller :-

Design a presettable alarm system using 8253/54 timer. Use thumb wheel switches to accept 4 digit value in seconds. Alarm should last for 5 seconds. Donot use interrupt.

Solution:

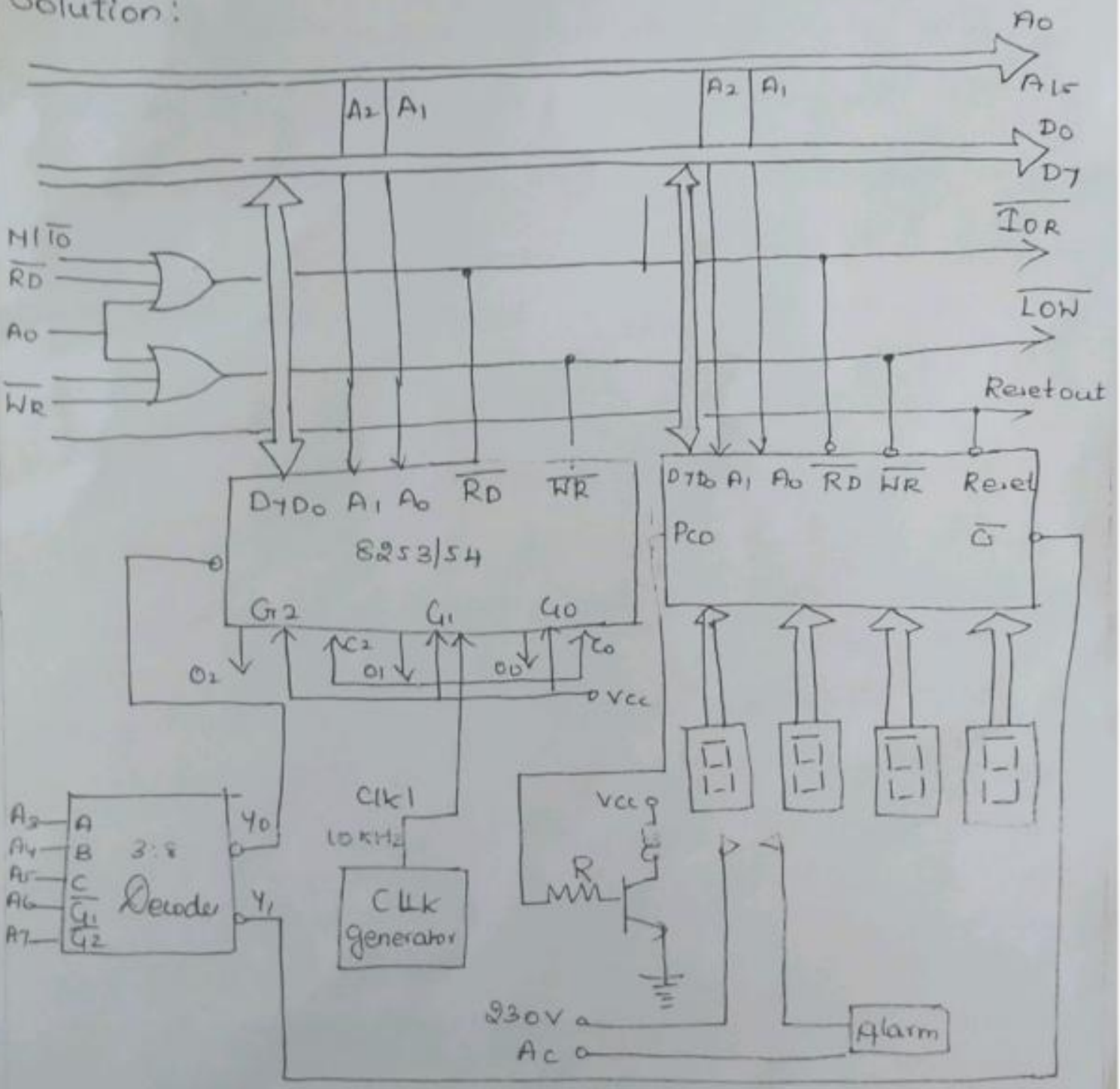


Fig: Presettable Alarm system

Program:

```
        MOV AL,92H  ; Load control word in accumulator
        OUT 0EH,AL  ; Initialise 8255 by sending Control
                    ; word at the addres of ctrl reg

START:  IN  AL,08H  ; Get the lower two digit of the Count
        MOV BL,AL   ; Store the lower two digit of the Count
        IN  AL,0AH  ; Get the higher two digit of the Count
        MOV BH,AL   ; Store the higher two digit of the Count
        MOV AL,31H
        OUT 06,AL   ; Loads Control word (31H) in the ctrl
                    ; register to
                    ; Load 16-bit Count in the Count reg
                    ; of Counter 0
        MOV AL,BL
        OUT 00,AL   ; Loads lower byte of the Count
        MOV AL,BH
        OUT 00,AL   ; Loads higher byte of the Count
BACK:   MOV AL,01H
        OUT 06,AL   ; Loads Control word (01H) in the
                    Control
                    ; register to
                    ; Latch 16-bit Counter in the Count
                    ; register of Counter 0
        IN  AL,00H  ; Get the lower two digits of the Count
        CMP AL,00H  ; Compare with zero
```

```
JNZ  BACK  ; Repeat
IN   AL,00H ; Get the higher two digits of the Count
CMP  AL,00H ; Compare with zero
JNZ  BACK  ; Repeat
MOV  AL,01H ; Load bit pattern to run Alarm
OUT  0CH,AL ; Send it to portc
CALL DELAY ; Wait for 5 seconds
MOV  AL,00H ; Load bit pattern to stop Alarm
OUT  0CH,AL ; Send it to portc
JMP  START ; Repeat
```

Delay routine:

This delay routine gives delay of 5 seconds.

```
MOV  AL,77H
OUT  06,AL      ; Loads Control word (77H) in the
                ; Control register
MOV  AL,    H
OUT  08,AL      ; Loads lower byte of the Count
MOV  AL,27H     ; Loads higher byte of the Count
OUT  02H
MOV  AL,B H
OUT  06H,AL
MOV  AL,05H
OUT  04H,AL     ; Loads lowerbyte of the Count
MOV  AL,00H     ; Loads higher byte of the Count
OUT  04,AL
BACK: MOV AL,81H
```

```
OUT 06,AL       ; Loads Control word (81H) in the
                ; Control register
                ; to latch 16-bit Count in the Count
                ; register of Counter2
IN  AL,04H      ; Get the lower two digits of the Count
CMP AL,00       ; Compare with 00H
JNZ BACK        ; If not zero, Repeat
IN  AL,04H      ; Get the higher two digits of the Count
CMP AL,00       ; Compare with 00H
JNZ BACK        ; If not zero, Repeat
RET             ; Return to main program
```

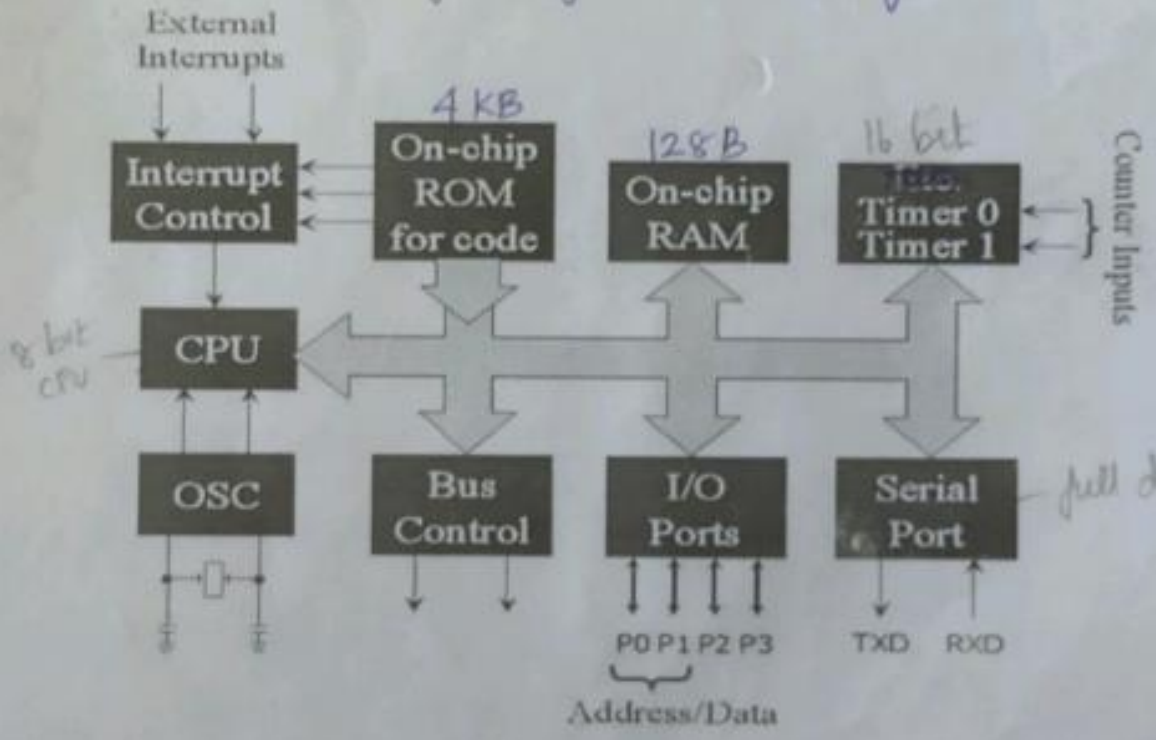
UNIT - 1

8051 - Hardware Architecture

* 8051 is a standalone high performance μc
* Intended for use in sophisticated real-time applications, such as instrumentation, industrial control automobiles and computer peripherals
* It provides extra features like interrupts, bit addressability and an enhanced set of instr which make the chip very powerful and cost effective

Features:-

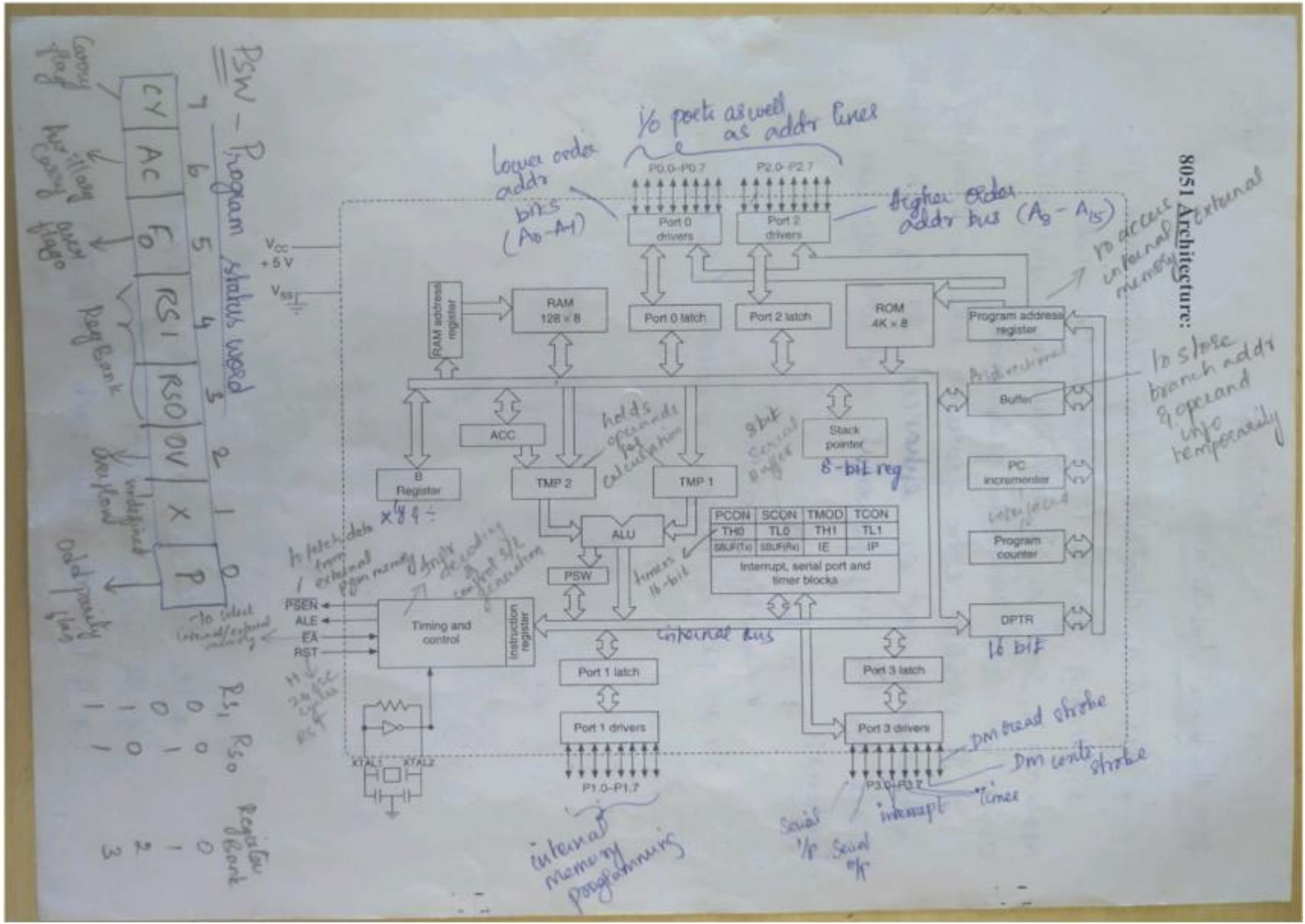
* 8-bit CPU
* On-chip oscillator
* 4KB of Program memory (ROM)



Block Diagram of 8051

* 128 bytes of Data memory (RAM)
* 21 Special function Registers
* 4 Ports of 8 bits each (P_0 to P_3)

8051 Architecture:



* 64 kB addr space for external data memory.
* 64 kB addr space for program memory
* 2 16 bit timer/counter
* 5 source Interrupt Structure
* Full duplex Serial port
* Bit addressability
* On-chip clock Oscillator
* Power bit processing Capability

Bus :- * Bus is a collection of wires which transfers data.

Address Bus :- * 16 bit address bus
* used to transfer the addr from CPU to memory

Data Bus :-
* 8 bit data bus
* used to carry data

Special Function Registers (21 SFRs)

* All the resources in the 8051 can be accessed through SFRs maintained in internal data memory

Accumulator (Acc): 'A' reg

* In all arithmetic operations, Acc is used as one of the operands and after the operation, the result is also stored in the Acc

'B' register: It is mainly used for multiply and divide operation in which it acts as one operand & after the operation a part of the result is stored in this reg. It is also called

Scratch pad i.e a temporary reg

Stack pointer :-

* 8-bit reg.
* This pointer can point to any location in the internal data RAM. (location 0 to 127)
* when the chip is reset, this reg is initialized to 07_H.
* During PUSH & CALL instr, the SP is first incremented and then data is stored in the stack.

Data pointer :- (DPTR)

* 16 bit register. It normally contains 16-bit addr
* Consisting of a high byte (DPH) and a low byte (DPL)

Ports (P_0 to P_3) :-

* 4 bidirectional 8 bit each I/O ports * These are directly represented as pins of the 8051 chip & are bit addressable
* P_0 and P_2 can be used as IO ports or addr lines for external memory. * The lower order bits ($A_0 - A_7$) are sent through P_0 & the higher order address bits are sent through P_2 ($P_{2.0} - P_{2.7}$)
* P_1 acts as IO ports and it plays an important role in programming of internal memory of 8051.
* P_3 acts as IO ports and its pins have other functions like Serial input line ($P_{3.0}$), Serial Output line ($P_{3.1}$), External Interrupt lines ($P_{3.2}$, $P_{3.3}$) External Timer I/P lines ($P_{3.4}$, $P_{3.5}$), External Data memory write strobe ($P_{3.6}$) & External Data memory Read Strobe ($P_{3.7}$)

* Timer 0 (TH0, TL0) and Timer 1 (TH1, TL1) are two 16 bit registers that can be used in timer/counter operations.

Serial Data Buffer (SBUF) (temporary reg)

* This reg holds the data that has to be transmitted through the serial port and also holds the data that is received.
* This reg is interconnected to 2 shift reg (8-bits each)

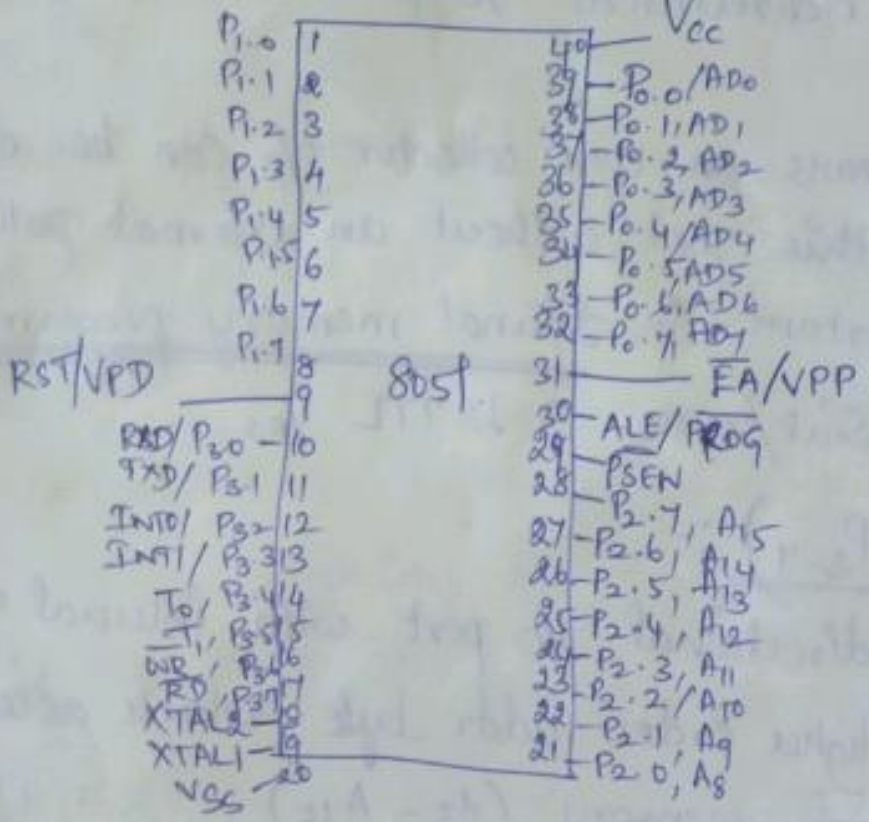
Control and status registers :-

* All the SFRs (like IP, IE, TMOD, TCON, PCON, SCON) that are used for controlling the internal resources.
* The instr decoding and controlling Signal generation are performed using the Timing and control circuit block
* The bidirectional ports P_0 to P_3 have the basic structure containing drivers & latches
* The port drivers are connected to I/O ports, whereas the latches are connected to the internal Bus.
* To access the internal program memory (ROM) as well as external pgm memory, the pgm addr reg is interfaced to pgm counter which is connected to the incrementer circuit for incrementing PC.
* A Bidirectional buffer has been provided to temporarily store the branch addr & operand addr information.

* Two temporary reg TMP_1 and TMP_2 are connected to ALU. These reg hold the operands for calculation
* The PSW reg is directly interfaced to ALU for updation on the basis of the last operation.

PCON - Power control reg
SCON - Serial control reg
TMOD - Timer mode control reg
TCON - Timer control reg
IE - Interrupt enable reg
IP - Interrupt Priority reg

Pin Configuration of 8051 µC



Vcc and Vss : Power supply pin

Vcc : connected to +5V

Vss : ground pin

Port 0 (P0.0 to P0.7) :-

* 8 bit bidirectional I/O pins
* Bit addressable
* They can sink upto 8 LS TTL logic gates.
* while accessing external memory, these pins acts as LSB addr bus/data bus. ($AD_0 - AD_7$).

Port 1 ($P_{1.0} - P_{1.7}$) :-

* 8 bit bidirectional I/O port with internal pull-ups
* That means an open collector o/p can be directly connected to this port without an external pull up
* Plays important in internal memory programming
* It can Sink/source 3 LS TTL P/p's.

Port 2 ($P_{2.0} - P_{2.7}$) :-

* 8 bit bidirectional I/O port with internal pull ups
* Acts as higher order addr byte, which accessing the external memory ($A_8 - A_{15}$).
* plays a role during the programming of the internal ROM.

Port 3 ($Port_{3.0}$ - $Port_{3.7}$) :-

* 8 bit didirectional I/O port with internal pull-ups.
* Alternate fn's of Port 3 pins are.

$P_{3.0}$ - RxD (Serial I/p)
$P_{3.1}$ - TxD (Serial O/p)

$P_{3.1}$ - $\overline{INT0}$ } External
$P_{3.2}$ - $\overline{INT1}$ } Interrupts

$P_{3.4}$ – T_0 (Timer 0 external i/p)

$P_{3.5}$ – T_1 (Timer 1 external i/p)

$P_{3.6}$ – $\overline{WR}$ (External data memory write strobe)

$P_{3.7}$ – $\overline{RD}$ (External Data memory read strobe)

ALE/$\overline{PROG}$

ALE – Address Latch enable (o/p pin)
- used for latching the lower-order lines (A_0-A_7) of the addr bus.

$\overline{PROG}$ – i/p pin
- act as an i/p pin for the programming pulses during the programming of the internal EPROM.

XTAL 1 and XTAL 2 :-

* The 8051 has an internal clk circuit
* a crystal oscillator is connected directly to these pins
* freq range is 3.5 – 12 MHz
* XTAL 1 is grounded and the Oscillator s/l is gn to XTAL 2

$\overline{PSEN}$:- (Active low o/p pin)

* Used for fetching the data from external pgm memory

$\overline{EA}$/VPP :-

$\overline{EA}$ - If connected to 5V; indicate the processor that the 4KB of the pgm memory within the chip should be used.

low : external memory.

VPP -

* while programming the 8751, this pin used as programming supply voltage (VPP) [= 21 V]

RST/VPD :-

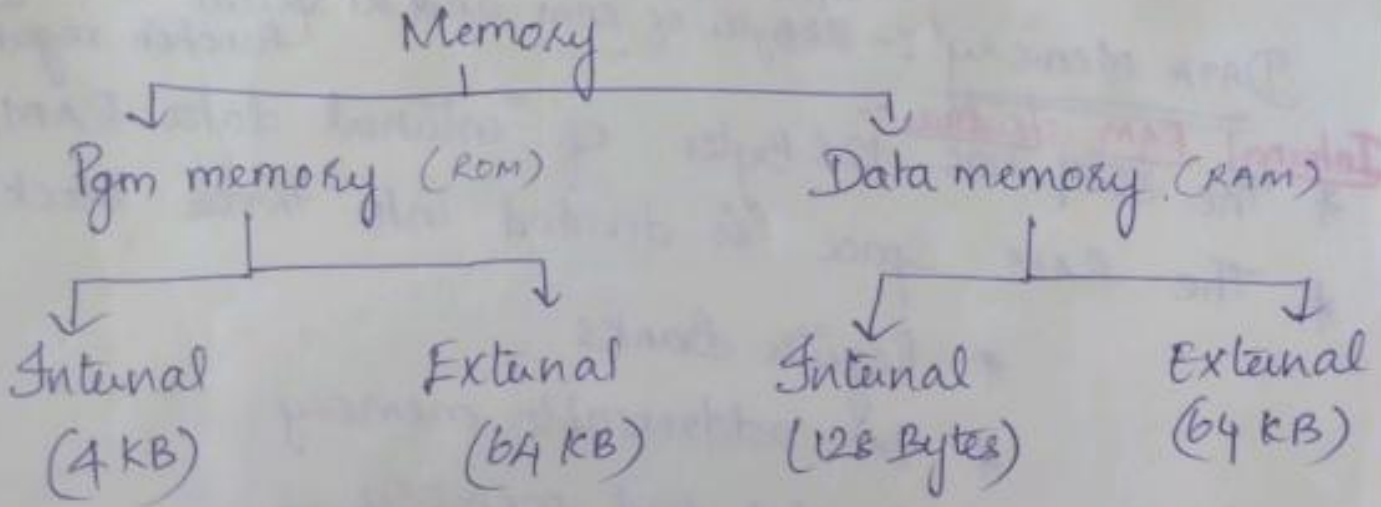
RST :- High on this pin, for more than 24 osc cycles will reset the chip.

VPD :-

* used to supply power to the internal RAM during power failure or power down modes.

Memory Organisation

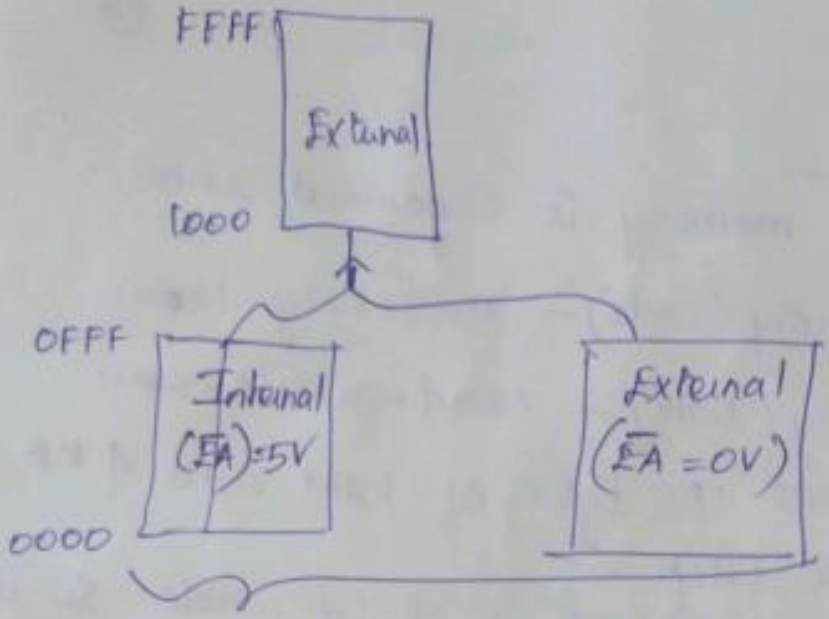
* In the 8051, The memory is Organised into
  Program memory (code) - read-only (ROM)
  Data memory (Data) - read-write (RAM)
* The Intel 8051 has 128 Bytes of RAM and 4 KB of ROM within the chip.
* The addr bus of the 8051 is 16-bit wide so it can access 64 KB of external memory. (Pgm & Data).



* The internal DM is accessed with 8-bit addresses and the external data memory with 16 bit addresses.

Program memory :-

* Based on the status of $\overline{EA}$ pin the program memory is accessed.



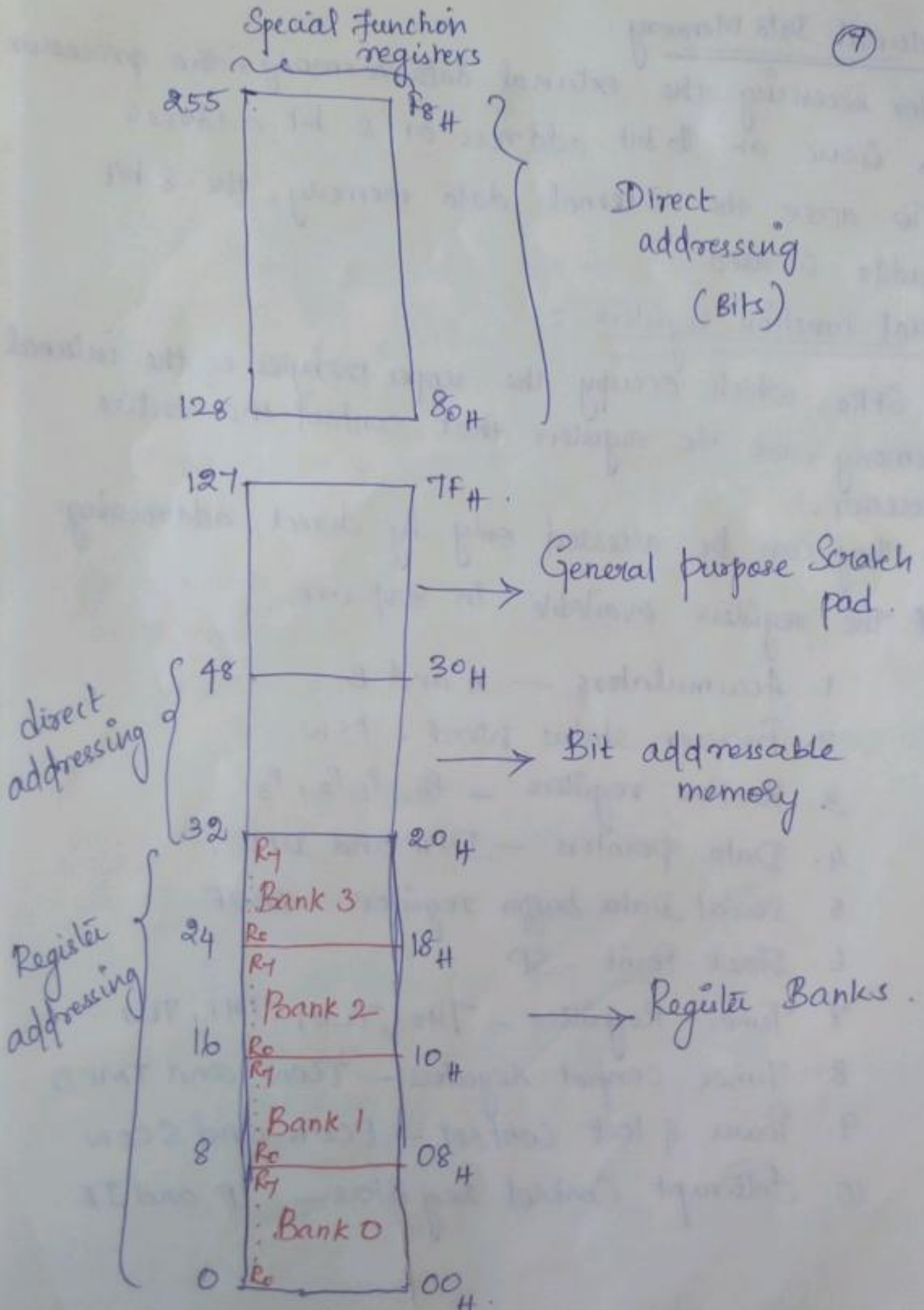
If $\overline{EA} = 0V$;
* External pgm memory = 0000_H - $FFFF_H$ (64K)

If $\overline{EA} = 5V$
* Internal pgm mem (ROM) = 0000_H - $0FFF_H$ (4K)

Program memory
* External program memory = 1000_H - $FFFF_H$ (60K)

DATA Memory :- The 8051 can access 64KB - external memory
- 128 Bytes of RAM and 21 Special function registers.

Internal RAM structure:-
* The 8051 has 128 Bytes of internal data RAM.
* The RAM Space is divided into three blocks.
  * Register Banks
  * Bit addressable memory
  * Scratch pad memory.
* The 8051 has 4 banks of 8 registers, each R_0-R_7
* The register Banks are identified with two bits in the processor status word (PSW).
* Bit addressable memory is both bit addressable and byte addressable.
* Using General purpose Scratch pad memory, programmers can read & write data at any time for any purpose.



Internal RAM Partitioning

External Data Memory :-

* For accessing the external data memory, the processor can issue an 16 bit address or 8 bit address.
* To access the internal data memory, the 8 bit addr is used.

Special Function Registers :-

* SFRs, which occupy the upper 128 bytes of the internal memory, are the registers that control the entire processor.
* They can be accessed only by direct addressing
* The registers available in 8051 are.

1. Accumulators — A and B.
2. Processor status Word — PSW.
3. I/O Port registers — P_0, P_1, P_2, P_3
4. Data pointers — DPH and DPL.
5. Serial Data buffer registers — SBUF
6. Stack Point — SP
7. Timer Registers — TH0, TL0, TH1, TL1
8. Timer control Registers — TCON and TMOD
9. Power & Port control — PCON and SCON
10. Interrupt Control registers — IP and IE.

Port Operation

* The 8051 has 4 bidirectional ports P_0, P_1, P_2, P_3 each port consists of 8 pins (8 bits)
* P_1, P_2, P_3 have internal pull-ups
  P_0 has open collector o/p's.
* Each port line consist of a latch, a driver and an i/p buffer.
* Whenever an i/p operation is required, a '1' is stored in the latch.

Port-0

* It performs two simple operations
  1. Simple I/O operation
  2. External memory interface for lower order address bus & data bus.

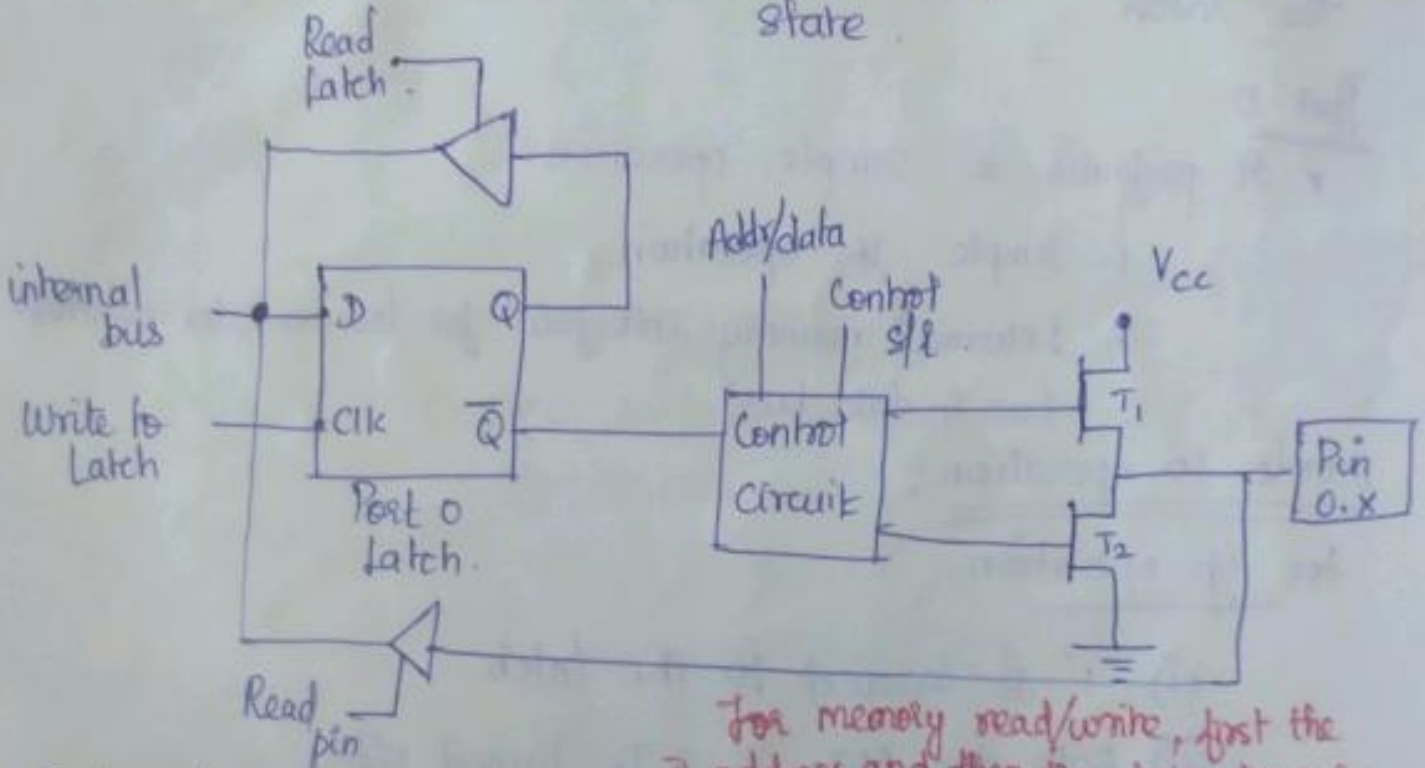
Simple IO operation :-

For i/p operation

(i) '1' is loaded to the latch.
(ii) Both the FETs T_1 & T_2 turned off
(iii) This cause port pin to float to high impedance state (z)
(iv) and it gets connected to the Pin Read Buffer.

Output Operation

* data directly loaded to latch.
* If latch o/p is 0; FET T_2 is turn on and o/p pin will be grounded.
* If latch o/p is 1; Both FET ON, o/p pin will float to high impedance & external pull-up registers used to o/p 1 as the data state



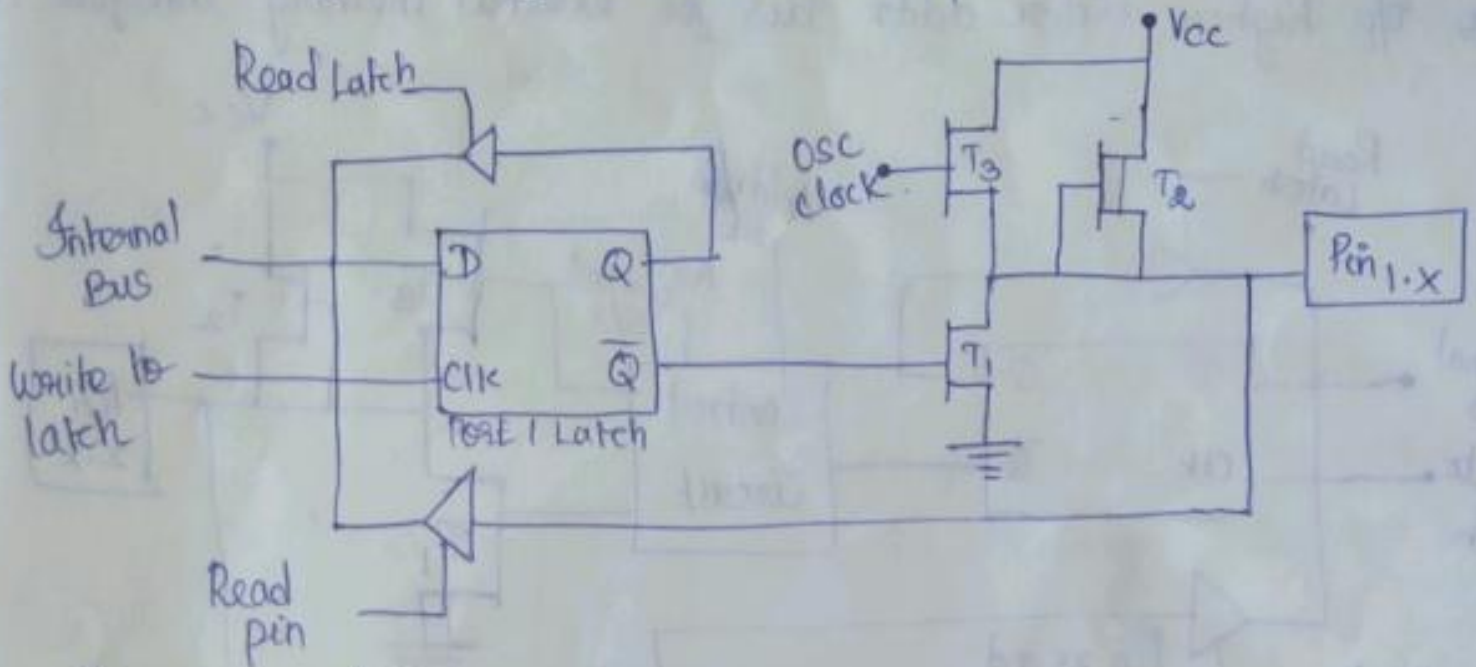
External memory Interface

For memory read/write, first the address and then the data transfer will be done

* when the addr is transferred, the control s/l's will connect the addr information directly to the pin through FET's T_1 and T_2, port latch is disconnected.
* If address bit = 1, T_1 ON, T_2 OFF, logic 1 s/l on Port Pin
  address bit = 0, T_1 OFF, T_2 ON, logic 0 s/l on Port Pin

Port 1 Operation

* Port 1 is a bidirectional port with internal pull-up's. FET T_2 and T_3 act as internal pull up to T_1.



input operation

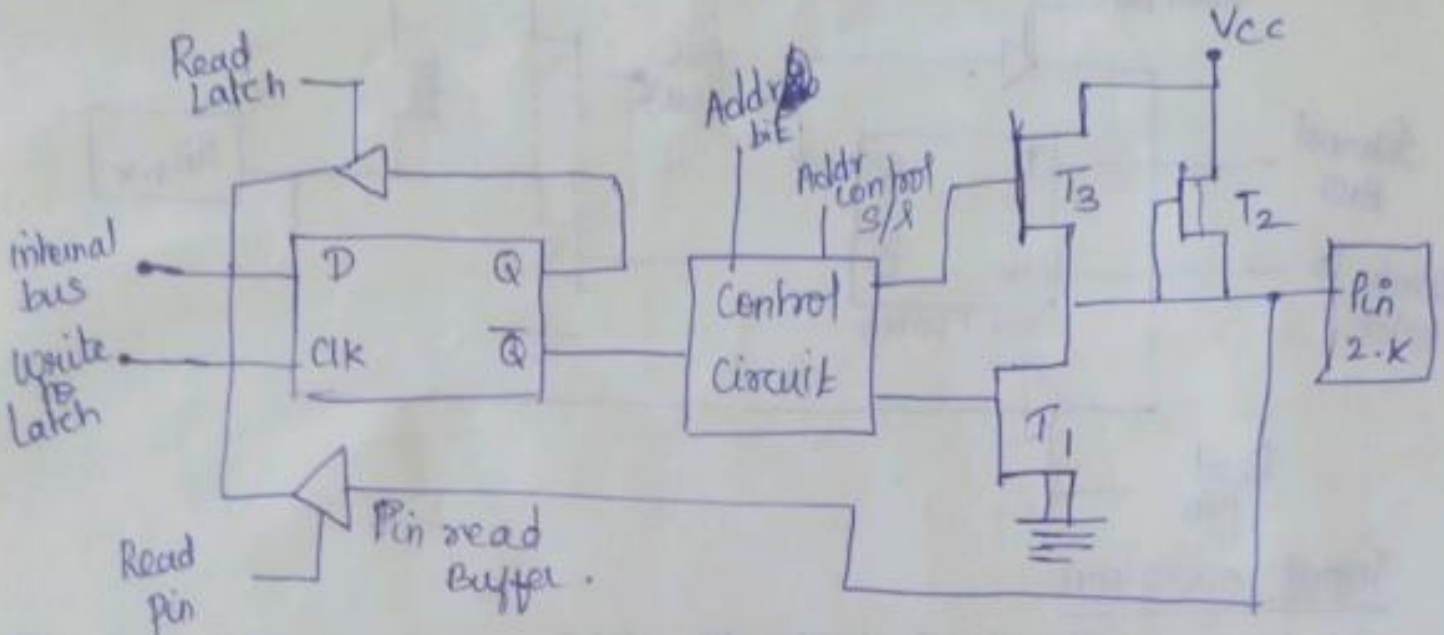
(i) Logic '1' is loaded to the latch
(ii) This turn off T_1.
(iii) Port pin will float to high impedance state and will be connected to the Pin read buffer.
(iv) A external device may place a '0' by driving the pin to the ground.

Operation :-

(i) The value will be loaded to the ~~port~~ latch.
(ii) If it is '1', FET T_1 will turned OFF. and this drives pin high through pull-up's (T_2 and T_3).
(iii) If the latch value is 0, FET T_1 is ON, This will ground the port pin and it show '0'.

Port 2 Operation :-

* Used as bidirectional I/O port or it can be also used to o/p higher order addr bus for external memory interface.



Input operation

- Logic '1' is stored in Latch. The FET T_1 is turned off and floats in high impedance state.
- The pin directly connected to Pin read Buffer. external device can place 0 or 1 on the pin.

Output port

- Latch containing 0, will turn ON FET T_1, thus grounding the pin
- Latch containing 1, will turn off FET T_1 and the pin driven to logic level 1.

Ex:-

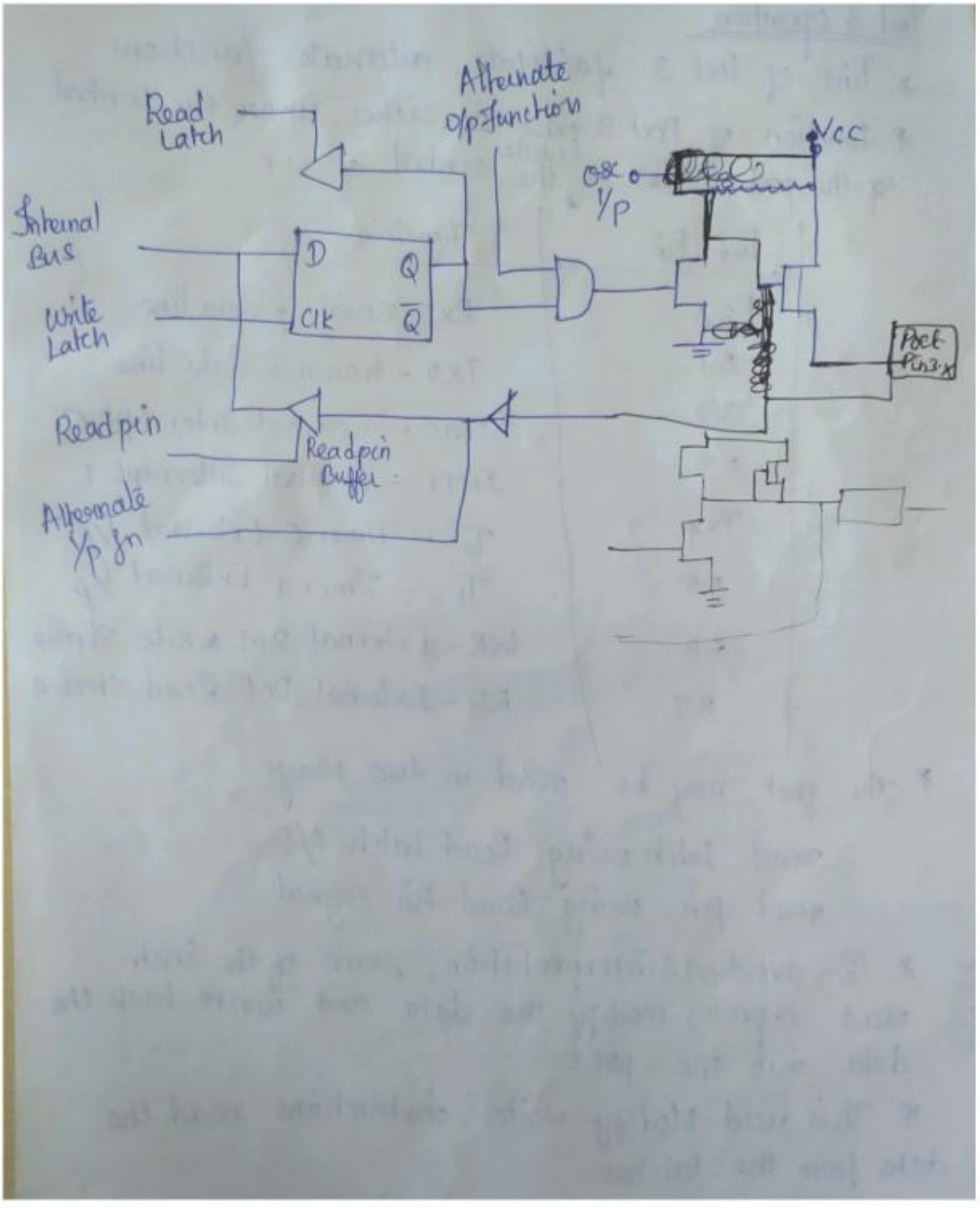
- When Port 2 is used to o/p higher order addr bits for external memory access, the control S/A will bypass the latch and the addr bits are directly connected to the pin through FETs.

Port 3 operation

* Pins of Port 3 facilitate alternate functions
* Function of Port 3 pins is either under the control of the port latches or under the control of SFR.

Port Pin	Function
P3.0	RxD Serial i/p data line
3.1	TXD - transmit data line
3.2	INT0 - External Interrupt 0
3.3	INT1 - External Interrupt 1
P3.4	T_0 - Timer 0 External i/p
3.5	T_1 - Timer 1 External i/p
3.6	WR - External DM write Strobe
3.7	RD - External DM Read Strobe

* The port may be read in two ways
  - read latch using Read Latch S/l
  - read pin using Read Pin Signal
* To avoid misinterpretation, some of the instr read a port, modify the data and write back the data into the port.
* This read-Modify write instructions read the data from the latches
* All other instructions read the data from the port pins only.


Read Latch
Alternate o/p function
Vcc
Internal Bus
D
Q
Write Latch
CLK
Q̄
Port Pin3.x
Read pin
Read pin Buffer
Alternate I/p fn

Addressing modes of 8051

* The way in which data is specified in an instruction is called addressing mode

① Immediate addressing mode:-

* data is given in the instruction itself.
* The data is preceded by the '#' symbol.

`ADD A, #08H` ; Add the data #08H to the accumulator & store the result in accumulator itself.

② Register Direct addressing

* The register contains the data to be manipulated is specified in the instruction.

`ADD A, R0` ; Adds the content of reg R0 to Accumulator contents and store result in accumulator

③ Memory Direct Addressing:-

* The memory address that contains the data is specified in the instruction

`ADD A, [74]H` ; Add the accumulator with content in the memory address 74H.

④ Memory Indirect addressing mode:-

* The register that contains the memory address of the data is specified in instr.
* The register specified is preceded by the '@' symbol.

`ADD A, @R0` ; Add the accumulator content with the content of the address given in the register R0.

⑤ Indexed Addressing (or) Base register plus index register indirect addressing

* The DPTR or PC register act as the base register & register A acts as the index register.
* Thus the base register (DPTR or PC) and the index register A are added to get the memory location.

```
MOVC A, @A+DPTR
```

Instruction Set:-

DATA TRANSFER INSTRUCTIONS:-

* It is divided into 3 classes
  1. General purpose transfers
  2. Accumulator-Specific transfers
  3. Address-Object transfers

General Purpose Transfers:-

* MOV
* PUSH
* POP

MOV — performs data (bit or byte) transfer from the source to destination

```
Mov destination, source
```

```
Mov A, R0
```

to move bit data

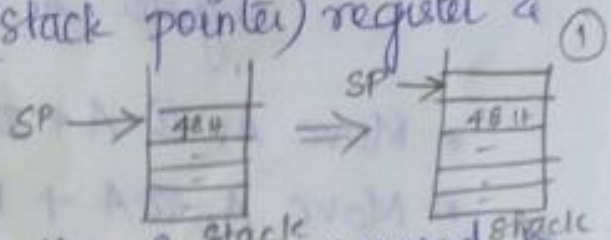
```
Mov C, bit
```

PUSH

PUSH <Source>

- Push increments the SP (stack pointer) register &

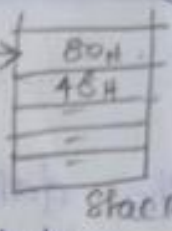
(SP) ← (SP) + 1



- then transfers a byte from the source operand to the stack element, currently addressed by SP

((SP)) ← direct

PUSH 80H



POP :- * POP transfers a byte operand from the stack element addressed by the SP register to the destination operand. (direct) ← ((SP))

* then decrements SP (SP) ← (SP) - 1

POP <Destination>

Accumulator - Specific Transfers :- (process on accumulator)

- XCH - Exchange accumulator with byte variable

XCH A, <Byte>

exa: XCH A, Rn

- XCHD - Exchange lower-order (nibble - 4 bits) byte with lower order nibble of register A.

Exchange digits

XCHD A, @Rn

- MOVX - Move byte from External Data memory (specified by DPTR) and register A

MOVX A, @DPTR (A) ← ((DPTR))

- Movc – Move a byte from the program memory to register A

* Movc A, @A+DPTR
* Movc A, @A+PC

(A) ← ((A)) + (DPTR)
PC ← (PC) + 1
A ← ((A)) + ((PC))

Address-Object transfer:

Mov DPTR, #data 16 — (DPTR) ← #data 16

* the data pointer (DPTR) is loaded with 16-bit constant

Arithmetic Instructions

- ADD

ADD A, <Source Byte> ⇒ A = A + <byte>

exa:-
ADD A, Rn
ADD A, direct

- ADDC – Add register A with Source byte and carry

ADDC A, <byte> ⇒ A = A + <Byte> + <carry>

exa:- ADDC A, Rn ; ADDC A, #data

- SUBB (Subtract with borrow)

SUBB A, <Source byte> ⇒ A = A – <byte> – Carry.

exa:- SUBB A, Rn
SUBB A, #data

Increment / Decrement :-

* Increment / decrement the indicated variable by 1.

<Byte> = <Byte> + 1

<Byte> = <Byte> − 1

Eka :-

INC A ; (A) ← (A) + 1.

DEC A ; (A) ← (A) − 1.

Multiplication / Division :-

MUL – performs an unsigned multiplication of register A by register B

MUL AB ⟹ B : A = B × A

bit (15-8) bit (7-0)

DIV – divides the unsigned 8-bit integer in accumulator by the unsigned 8 bit integer in register B

DIV AB → A = Int [A/B] – A saves quotient
B = Mod [A/B] – B has remainder

Decial adjust accumulator for addition :-

* The DAA instruction is used to convert the binary sum obtained after adding two BCD numbers into a BCD number

DA A

Logical Instructions

- ANL <destination byte>, <Source byte> – Logical AND for byte variable.

  * ANL performs the bit-wise logical AND operation between the variables indicated and stores result in destination variable.

exa :-
```
ANL A, Rn          ; (A) ← (A) ∧ (Rn)
ANL direct, #data  (direct) ← (direct) ∧ #data.
```

- ORL <dest. byte>, <Src_ byte> – Logical OR for byte variable.

  * performs Logical OR operation & stores the results in destination byte.

exa :-
```
ORL A, #data ;  (A) ← (A) ∨ #data
ORL direct, A ;  (direct) ← (direct) ∨ A.
```

- XRL <dest. byte>, <src_ byte> – Logical XOR for byte Variable.

  * performs bitwise Logical XOR operation b/w the indicated variables, & stores result in destination.

exa :-
```
XRL A, Rn ;   (A) ← (A) ∀ (Rn)
XRL direct, #data ;  (A) ← (Direct) ∀ #data
```

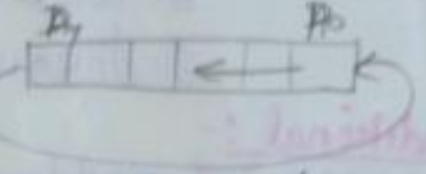
Clear accumulator.

CLR A ; (A) ← 0

- Complement accumulator.

CPL A ; (A) ← ($\bar{A}$)

- Rotate accumulator left.



RL A ; (A_0) ← (A_7)

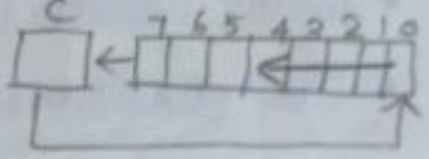
(A_{n+1}) ← (A_n) n = 0 — 6.

- Rotate accumulator left through the carry flag.

RLC A ; (A_{n+1}) ← (A_n)

(A_0) ← (C)

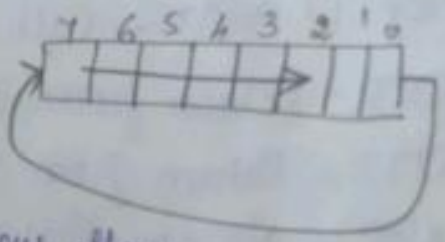
(C) ← (A_7)



- Rotate accumulator right

RRA ; (A_n) ← (A_{n+1})

(A_7) ← A_0

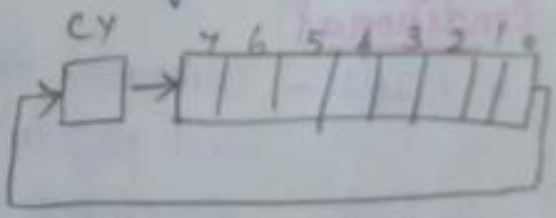


- Rotate accumulator right through carry flag

RRC A ; (A_n) ← (A_{n+1})

(A_7) ← (C)

(C) ← (A_0)



- SWAP A – interchanges the low and high-order nibbles of the accumulator

$A_{(3-0)} \longleftrightarrow A_{(7-4)}$

Control transfer Instructions :-

* 3 classes

1. unconditional calls, returns & jumps
2. Conditional jumps
3. Interrupts

Unconditional :-

- SJMP rel - address (Short jump) - jump to (PC) + 8 bit rel_addr
- AJMP 11-bit address (Absolute jump) - jump to PC:addr
- LJMP address (Long jump) - jump to addr
- JMP @A+DPTR - jump to A+DPTR
- ACALL 11 bit address - Call subroutine at PC:addr.
- LCALL addr - Call subroutine at addr.
- RET - Return from subroutine
- RETI - Return from Interrupt
- NOP - No operation

Conditional

- CJNE - compares the first operand to the second operand and performs a jump if they are not equal.
- DJNZ - decrements the Source operand and returns the result to the operant. A jump is performed if the result is not zero
- JN rel - jump if A = 0
- JNZ rel - jump if A ≠ 0

Bit Manipulation Instruction

* A special feature of 8051 µc is that it can handle bit data as well as byte data

ANL C, bit → C AND bit

ORL C, bit → C OR bit

MOV C, bit → C = bit

CLR C → C = 0

SETB C → C = 1

CPL bit → bit = Not bit

JC rel → jump if c = 1

JNC rel → jump if c = 0

Relative (short range) ⇒ +127 to −128 ($+7F_H$ to -80_H) (1 KB)

Absolute range : 0000_H to $07FF_H$ (2KB)

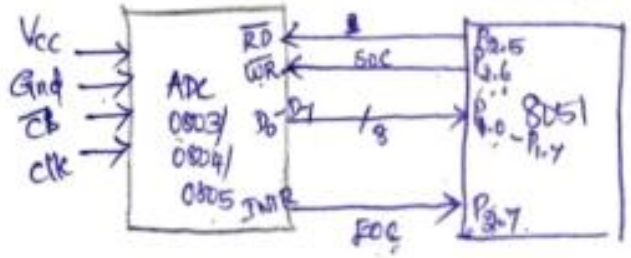
Long range : 0000_H to $FFFF_H$

Important 8051 Interfacing Programs

1) Interfacing ADC Using 8051 (Analog to digital converter)

0803/0804/0805

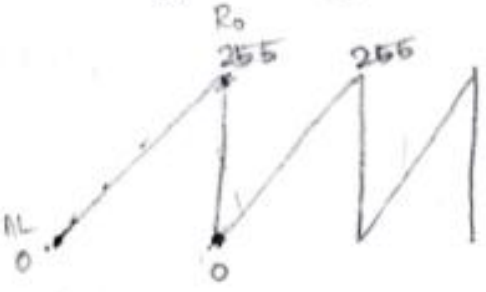
```
        MOV P1, #0FF_H ; Configure Port 1 as i/p
Back :  CLR P2.6 ;  } To generate start of conversion, first
        SETB P2.6 ; } make signal Low then high
AGAIN : JB P2.7, AGAIN ; check for End of conversion signal
        CLR P2.5 ; } Read converted data through Port 1
        MOV A, P1 ; }
        SETB P2.5 ; Disable read after reading data
        SJMP BACK ; go for next conversion
```



Interfacing DAC (Digital to Analog Converter)

To generate Sawtooth waveform.

```
        MOV R0, #255_H ;
        MOV AL, 00_H ;
L1 :    MOV Port, A ; Any port
        INC A ;
        DJNZ R0, L1 ;
        RET ;
```



1A Square Wave Generation

```
        MOV SP, #08H
Repeat: MOV P1, #OFFH
        LCALL Delay
        MOV P1, #00H
        LCALL Delay
        LJMP Repeat
Delay:  MOV R0, #OFFH
BACK:   DJNZ R0, BACK
        RET
```

} delay

1B- Square wave generation using timers of 8051.

* Assume that XTAL = 11.0592 MHz. Write ALP to generate a square wave of 2 kHz freq on Pin P1.5 (Timer 1 is used)

Soln

Freq of Square wave = 2 kHz

Time period of Square wave = $\frac{1}{f} = \frac{1}{2\times10^3} = 500\mu s$

Time period (T) = $T_{ON} + T_{OFF} = 500\mu s$

$T_{ON} = 250\mu s$; $T_{OFF} = 250\mu s$

$$\text{Count} = \frac{T_{ON}}{\text{Timer clock period}}$$

$$= \frac{250\mu s}{1.085\mu s}$$

Count = 230

$$\text{Timer clock period} = \frac{12}{\text{Crystal freq}}$$

$$= \frac{12}{11.0592\times10^6}$$

$T_{clk} = 1.085\mu s$

Count to be loaded in TH_1 & TL_1

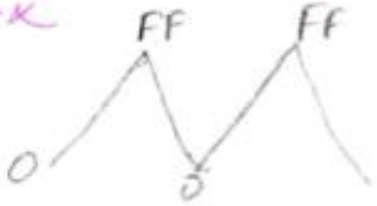
$= 65536 - 230$

$= 65306 = FF1A_H$

$TH_1 = FF$ and $TL_1 = 1A_H$

```
        MOV TMOD, #10H ; timer 1, mode 1
                       (16 bit mode)
AGAIN : MOV TL1, #1AH ;
        MOV TH1, #FFH : } load timer 1
        SETB TR1; — Start Timer 1
BACK : JNB TF1, BACK ; wait for timer rollover
        CLR TR1 ; Stop timer 1
        CPL P1.5 ; complement P1.5
        CLR TF1 ; clear timer flag 1
        SJMP AGAIN ; Reload timer 1 & continue.
```

Triangular Wave




```
        MOV SP, #08H
        MOV R0, #00H ; Send output digital data.
Repeat : MOV Port, R0 ; data is go through ports.
        INC R0
        CJNZ R0, #FFH, Repeat
Repeat1 : MOV Port, R0
        DJNZ R0, Repeat 1 ; decrement digital data to least
        LJMP Repeat.
```

Write ALP to find Square of a number using 8051

```
MOV  DPTR, #5000H .  Input memory location
MOVX A,@DPTR ; Move ip data to A
MOV  B, A ; A ⟶ B
MUL  AB ; 24 x 24 = 0576   05-B  76-A
INC  DPTR; 5000+1 = 5001
MOVX @DPTR, A ; 76 ⟶ 5001
INC  DPTR; 5002
MOV  A, B ; B = 05 ⟶ A
MOVX @DPTR, A ; A = 05 ⟶ 5002.
HLT.
```

Input

Address	Data
5000	24

Output

Addr	data
5001	76
5002	05

Duty cycle $= \frac{T_{ON}}{T_{ON} + T_{OFF}} = 63\%$.

Kavya
Nivedhitha
Jennifer Mary
Shyamala
Sathathi
Revathi
Jegatheeswari
Ramasamy.
Kingslin

Unit - V

Interfacing MicroControllers

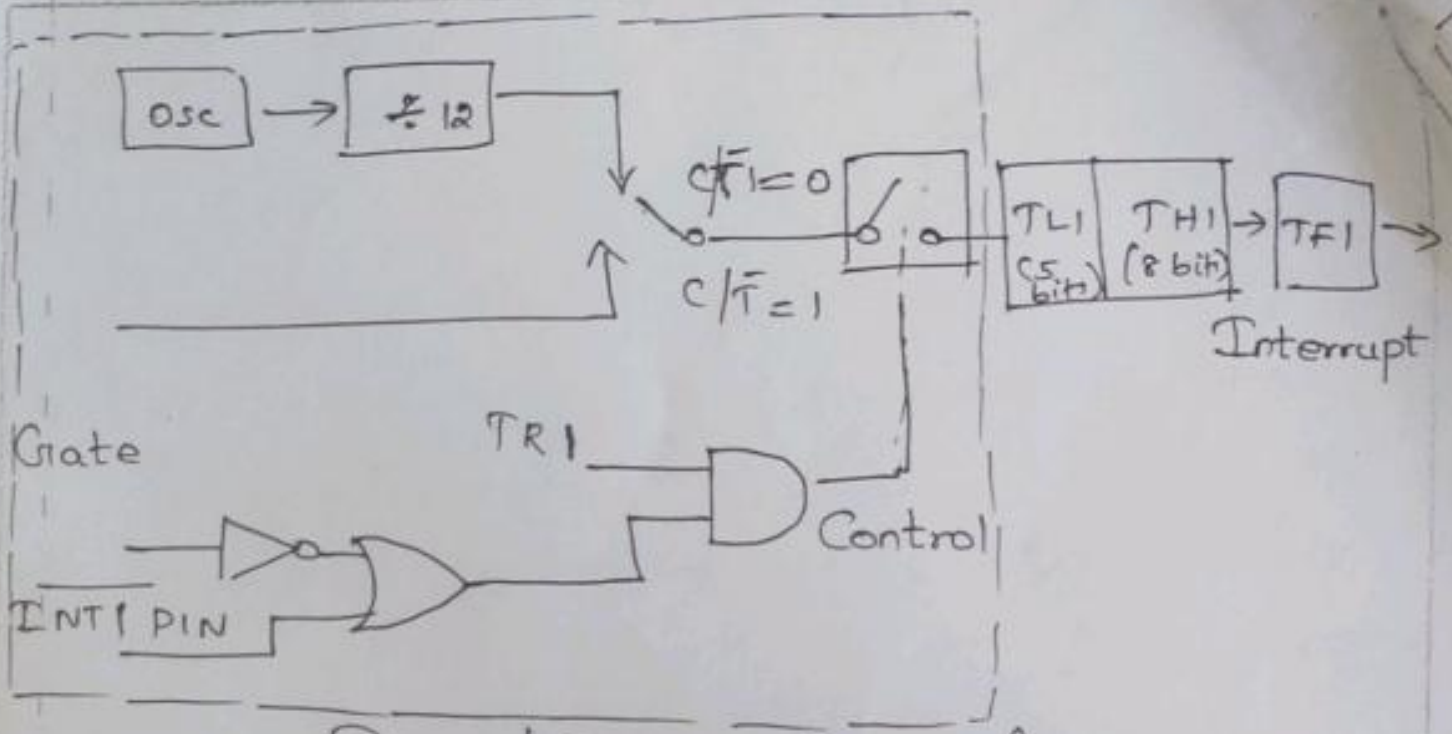
Programming 8051 Timers – Serial port programming - Interrupts programming – LCD and keyboard interfacing ADC, DAC and Sensor Interfacing - External Memory interface - stepper motor & Waveform generator.

Programming 8051 Timers :-

There are four modes of timer, mode 0, mode 1, mode 2 and mode 3

Mode 0 :- Both Timers in Mode 0 are 8-bit Counter with a divide-by-32 prescaler.

- In this mode, The Timer register is Configured as a 13-bit register. As the Count rolls over from all 1s to all 0s. & It sets the Timer interrupt flag TF1.
- The Counted i/p is enabled to the Timer When TR1 = 1 & either GATE = 0 or $\overline{INT1}$ = 1
- TR1 is a Control bit in the Special function Register TCON GATE is in TMOD



Timer / Counter Control logic

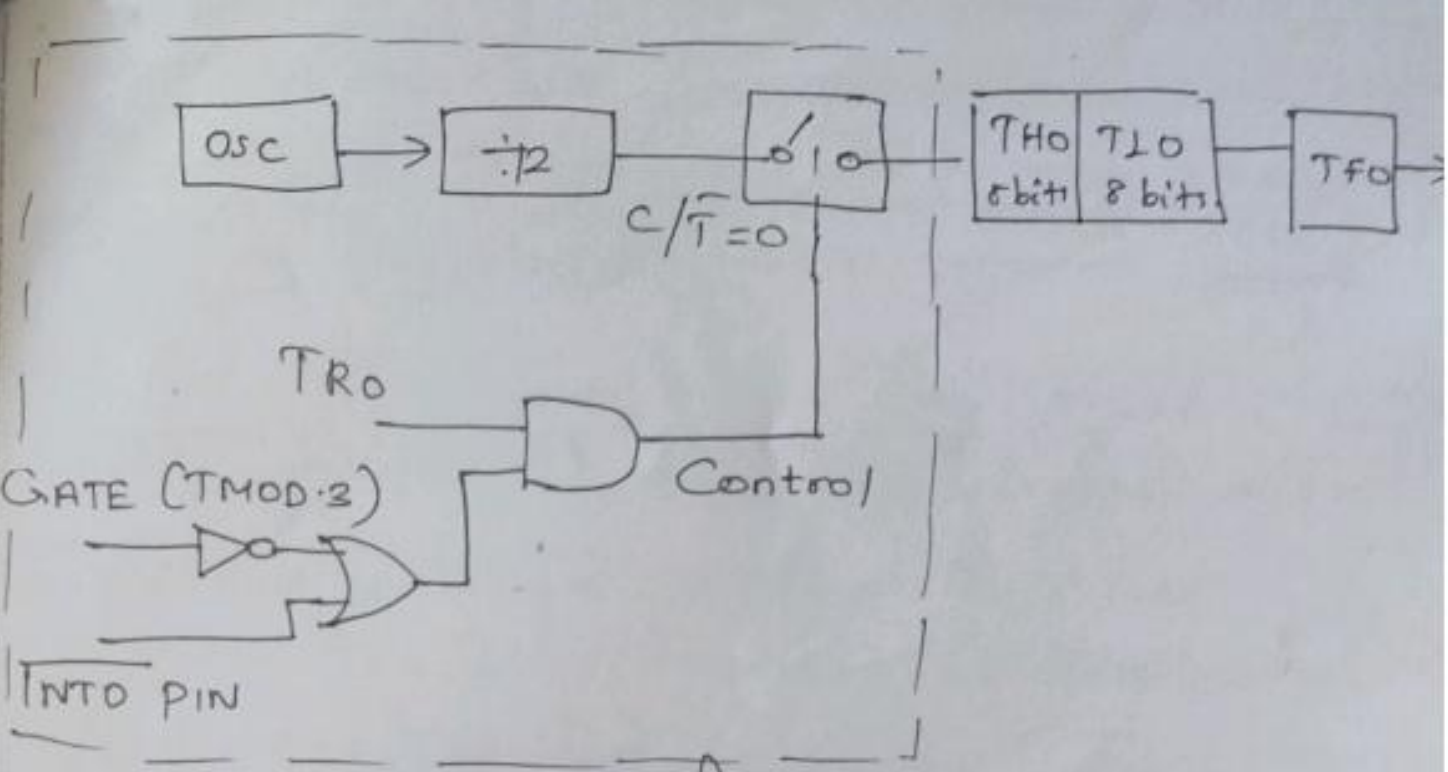
Fig: Timer / Counter 1 mode 0 : 13-bit Counter

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate & should be ignored Setting the run flag (TR1) does not clear the register.

Mode 1: Both Timer in Mode 1 are 16-bit Counter As the Count rolls over from all 1s to all 0s, it sets the Timer interrupt flag TF. The Counter input is Enabled to the Timer When TR=1 & either GATE=0 or $\overline{INT1}$ = 1

Timer 0 Mode 1 programming :-

The fig shows the Timer Control logic for Timer 0 in mode 1



Timer 0 Control logic

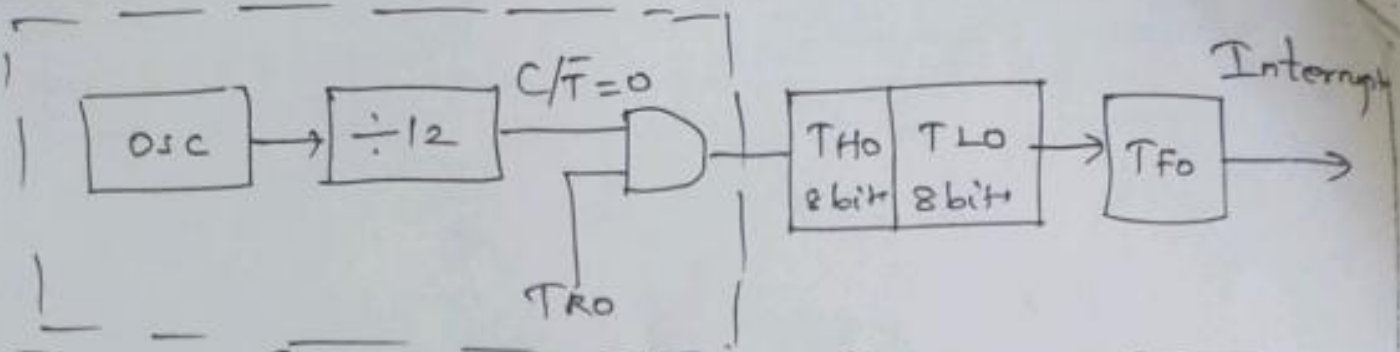
A Time delay can be generated using mode 1 of the Timer 0 using following steps.

1. Load TMOD register indicating Timer 0 is used & Mode 1 is selected

	7	6	5	4	3	2	1	0	
TMOD	x	x	x	x	0	0	0	1	= 01

2. Load TL0 and TH0 register with Count Value
3. Start the Timer by setting TR0 bit = 1
4. Monitor the Timer flag (TF0) with the JNB TF0, Target address instruction.
5. stop the Timer by clearing TR0 bit = 0 with CLR TR0 Instruction
6. Clear TF0 flag with CLR TF0 Instruction.

When start & stop of timer is done using software, no External hardware is needed.



Timer 0 Control logic When GATE=0 & $\overline{INT0}$=1

Fig Timer 0 in mode 1, No External hardware is used to start & stop

Timer 1 mode 1 programming:-

The fig shows the timer Control logic for Timer 1 in mode 1

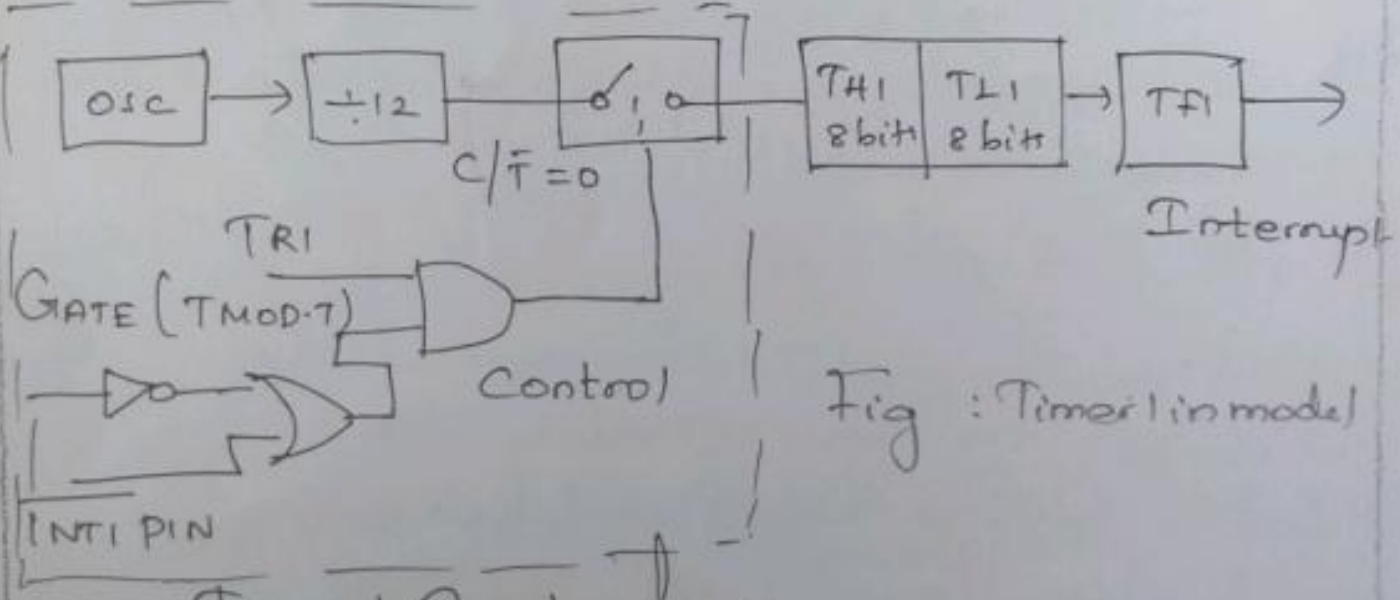


Fig : Timer 1 in mode 1

Timer 1 Counter logic

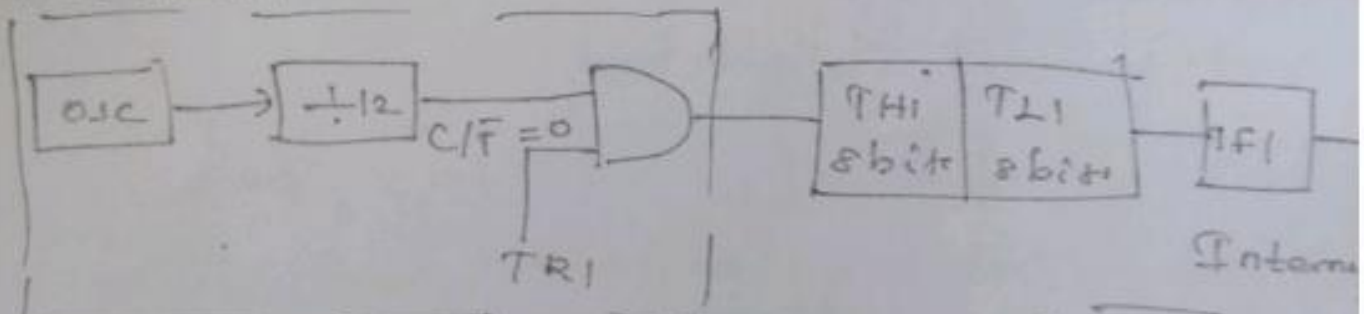
A Time delay can be generated using mode 1 of the timer 1 Using following steps

1. Load TMOD register indicating time 1 is Used & Mode 1 is selected.

	7	6	5	4	3	2	1	0	
TMOD	0	0	0	1	x	x	x	x	= 10H

2. Load TH1 and TH1 register with Count values
3. Start the timer by setting TR1 bit = 1
4. Monitor the timer flag (TF1) with the JNB TF1, target address Instruction.
5. Stop the Timer by clearing TR1 bit = 0 with CLR TR1 Instruction.
6. CLEAR TF1 flag with CLR TF1 Instruction

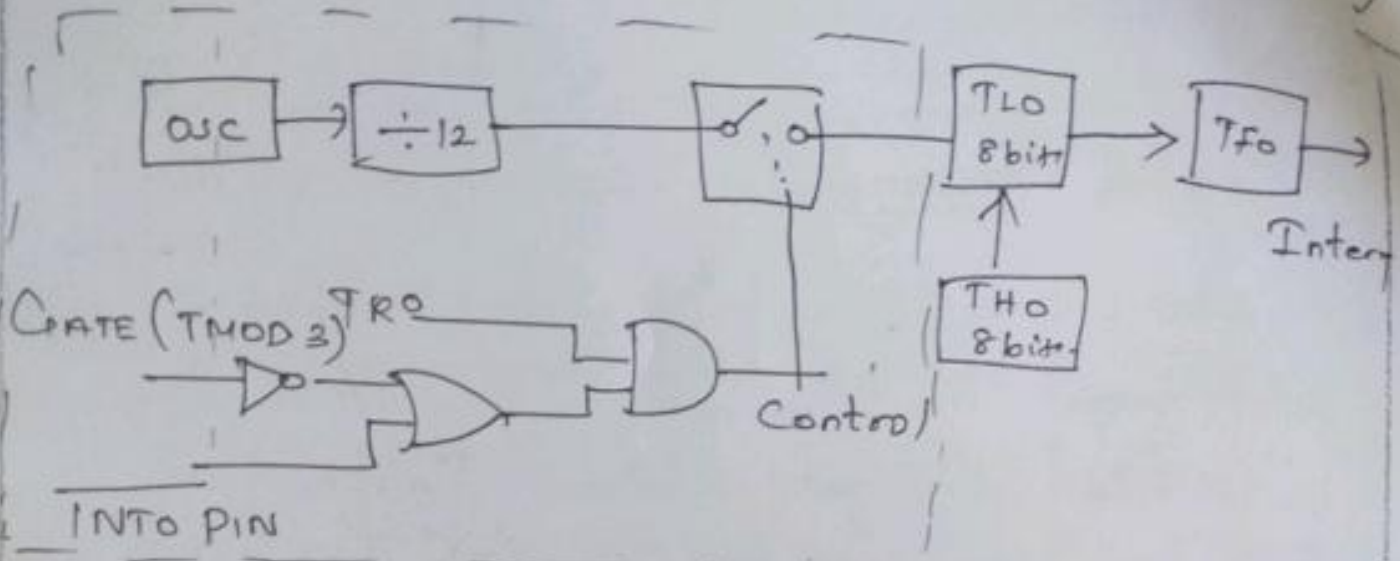
When the start and stop of timer is done Using software, no External Hardware is needed.



Timer1 Control logic When GATE=0 & INT1 = 1

Mode 2: Mode 2 Configures the Timer register as an 8-bit Counter (TL) with automatic reload.

Timer 0 Mode 2 programming: The figure shows the timer Control logic for timer 0 in mode 2.



Timer 0 Control logic.

Fig: Timer 0 in mode 2.

A Time delay can be generated using mode 2 of the timer 0 using following steps

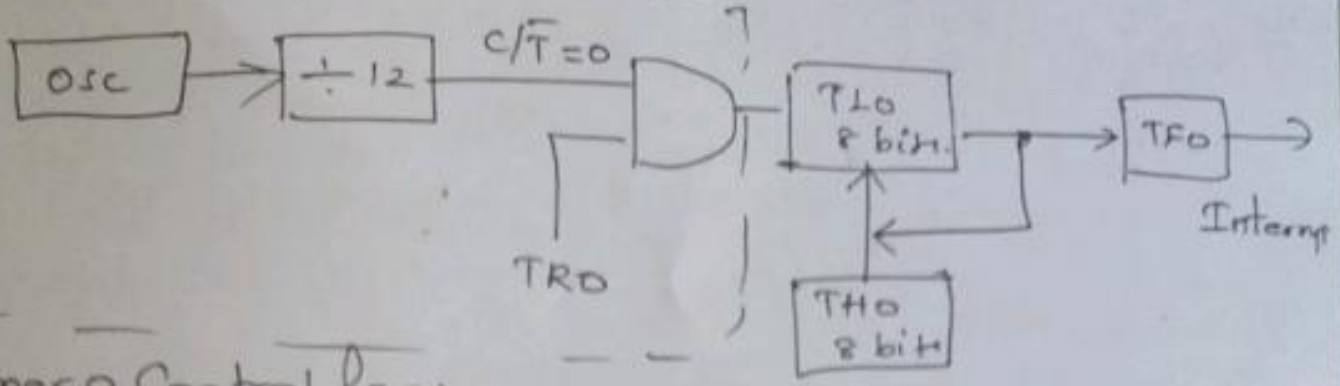
1. Load TMOD register indicating timer 0 is used and mode 2 is selected.

	7	6	5	4	3	2	1	0	
TMOD	x	x	x	x	0	0	1	0	= 02 H

2. Load THO register with Count Value
3. Start the timer by setting TRO bit = 1.
4. Monitor the timer flag (TFO) with the JNB TFO, target address Instruction
5. Clear the TFO flag, with CLR TFO Instruction
6. Go back to step 4. There is no need to load THO register.

When start & stop of timer is done using

Software, no External hardware is needed.



Timer 0 Control logic

When GATE = 0 & $\overline{INT0}$ = 1

Timer 1 Mode 2 programming :-

The figure shows the timer Control logic for timer 1 in mode 2.

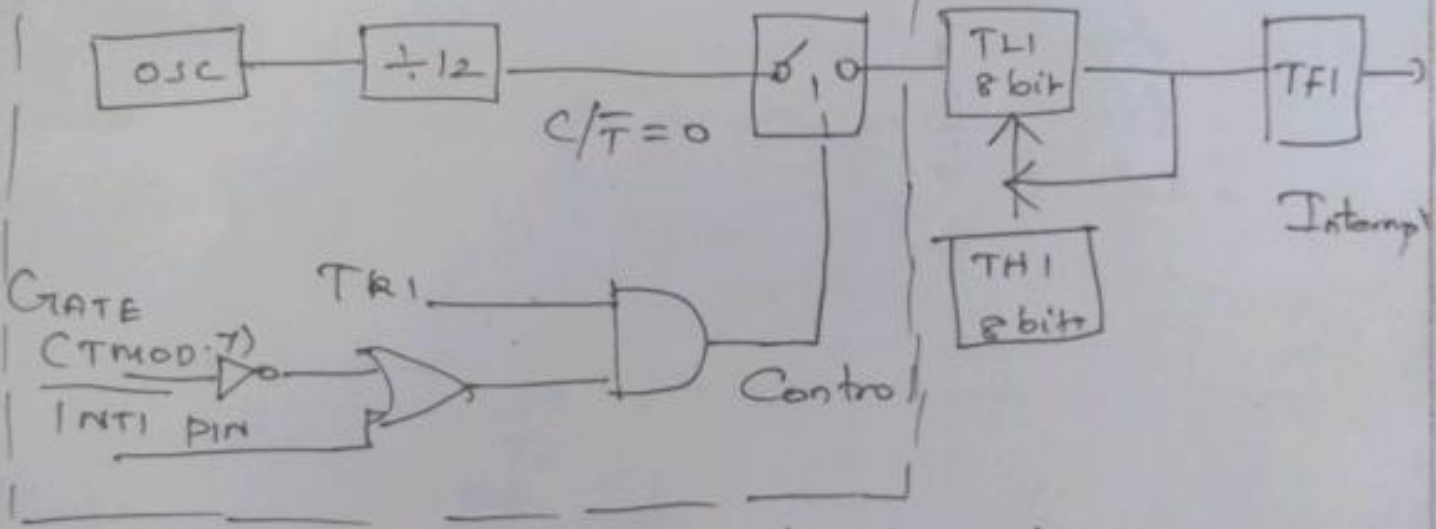


Fig Timer 1 in mode 2.

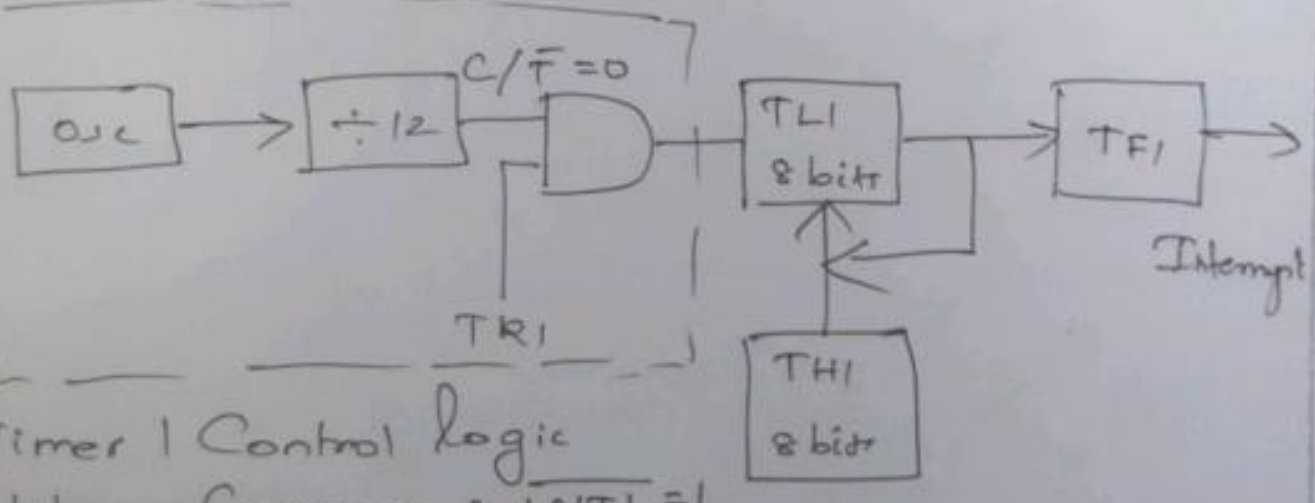
A Timer delay can be generated using mode 2 of the timer 1 using following steps:

1) Load TMOD register indicating timer 1 is used and mode 2 is selected.

	7	6	5	4	3	2	1	0	
TMOD	0	0	1	0	x	x	x	x	= 20H

2. Load TH1 register with Count Value
3. Start the Timer by setting TR1 bit = 1
4. Monitor the Timer flag (TF1) with the JNB TF1, target address Instruction.
5. Clear the TF1 flag, with CLR TF1 Instruction
6. Go back to step 4. There is no need to load TH1 register again since mode 2 is auto loaded.

When start & stop of Timer is done using Software, no External Hardware is needed.



Timer 1 Control logic
When GATE = 0 & $\overline{INT1}$ = 1

Mode 3:- Timer 1 in Mode 3 simply holds its Count. The Effect is the

Same as setting Timer 0. Timer 0 in mode 3 Establishes TL0 & TH0 as two seperate Counters.

The logic for Mode 3 on Timer 0 is shown in fig. TL0 uses the Timer 0 Control bits: C/$\bar{T}$, GATE, TR0, $\overline{INT0}$ & TF0. TH0 is locked into a Timer mode and takes over the use of TR1 & TF1 from Timer 1. Thus TH0 now Controls the Timer 1 Interrupt.

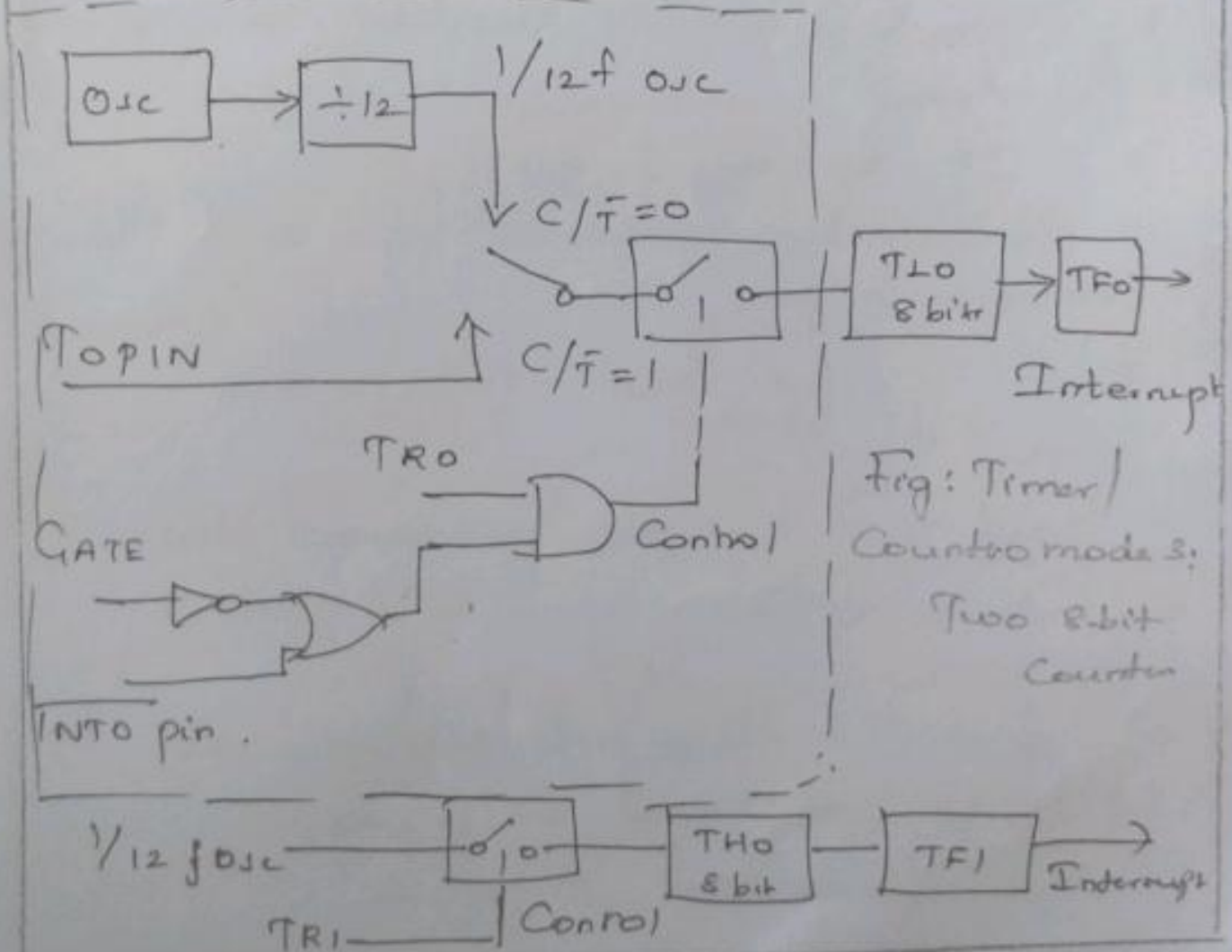


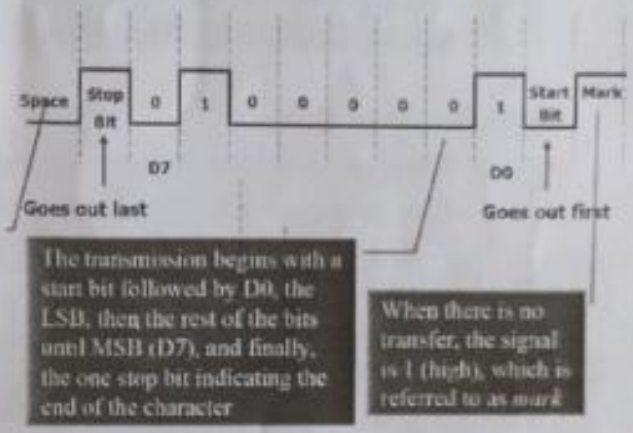
Fig: Timer/Counter mode 3: Two 8-bit Counter

SERIAL PORT PROGRAMMING:

- *Serial data communication uses two methods*
  - *Synchronous method transfers a block of data at a time*
  - *Asynchronous method transfers a single byte at atime*
- *There are special IC's made by many manufacturers for serial communications.*
  - *UART (universal asynchronous Receiver transmitter)*
  - *USART (universal synchronous-asynchronous Receiver-transmitter)*

Asynchronous – Start & Stop Bit

- *Asynchronous serial data communication is widely used for character-oriented transmission.*
- *Each character is placed in between start and stop bits, this is called framing*
- *Block-oriented data transfers use the synchronous method.*
- *The start bit is always one bit, but the stop bit can be one or two bits*
- *The start bit is always a 0 (low) and the stop bit(s) is 1 (high*



Data Transfer Rate

- *The rate of data transfer in serial data communication is stated in* ***bps (bits per second).***
- *Another widely used terminology for bps is* ***baud rate.*** *It is modem terminology and is defined as the number of signal changes per second*

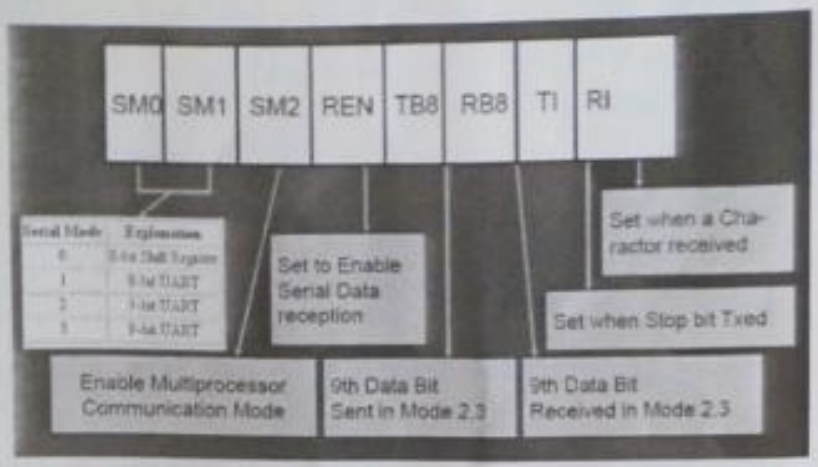
Registers related to Serial Communication

1.SBUF Register
2.SCON Register
3.PCON Register

SBUF Register

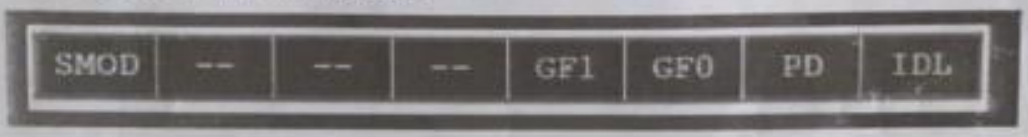
- *SBUF is an 8-bit register used solely for serial communication.*
- *For a byte data to be transferred via the TxD line, it must be placed in the SBUF register. The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line.*
- *SBUF holds the byte of data when it is received by 8051 RxD line. When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF.*

SCON Register



Doubling Baud Rate

- *There are two ways to increase the baud rate of data transfer*
  1. By using a higher frequency crystal
  2. By changing a bit in the ***PCON register***
- *PCON register is an 8-bit register.*



- *When 8051 is powered up, SMOD is zero*
- *We can set it to high by software and thereby double the baud rate.*

8051 Serial Communication

- *Synchronous and Asynchronous*
- *SCON Register is used to Control*
- *Data Transfer through TXd & RXd pins*
- *Some time - Clock through TXd Pin*

Four Modes of Operation:

- *Mode 0 :Synchronous Serial Communication*
- *Mode 1 :8-Bit UART with Timer Data Rate*
- *Mode 2 :9-Bit UART with Set Data Rate*
- *Mode 3 :9-Bit UART with Timer Data Rate*

8051 Serial Port – Mode 0

- *The Serial Port in Mode-0 has the following features:*
  - *Serial data enters and exits through RXD*
  - *TXD outputs the clock*
  - *8 bits are transmitted / received*
  - *The baud rate is fixed at (1/12) of the oscillator frequency*

8051 Serial Port – Mode 1

- *10 bits are transmitted / received*
  - *Start bit (0)*
  - *Data bits (8)*
  - *Stop Bit (1)*
- *Serial data enters through RXD*

- *Serial data exits through TXD*
- *On reception is completed , the stop bit goes into RB8 bit in SCON register.*
- *Baud rate is determined by the Timer 1 over flow rate.*

8051 Serial Port – Mode 3

- *11 bits are transmitted / received*
  - *Start bit (0)*
  - *Data bits (8)*
  - *1 bit can be programmed*
  - *Stop Bit (1)*
- *9th data bit is the TB8 in SCON register, used to send the parity of the byte to be transmitted.*
- *On reception of frame, the 9th data bit goes into RB8 in SCON register*
- *Baud rate is is determined by Timer 1 overflow rate.*

Programming Serial Data Transmission

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. The TH1 is loaded with one of the values to set baud rate for serial data transfer.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8- bit data is framed with start and stop bits.
4. TR1 is set to 1 to start timer 1
5. TI is cleared by CLR TI instruction
6. The character byte to be transferred serially is written into SBUF register.
7. The TI flag bit is monitored with the use of instruction JNB TI, xx to see if the character has been transferred completely.
8. To transfer the next byte, go to step 5

Programming Serial Data Reception

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. TH1 is loaded to set baud rate
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8- bit data is framed with start and stop bits.
4. TR1 is set to 1 to start timer 1
5. RI is cleared by CLR RI instruction
6. The RI flag bit is monitored with the use of instruction JNB RI, xx to see if an entire character has been received yet
7. When RI is raised, SBUF has the byte, its contents are moved into a safe place.
8. To receive the next character, go to step 5.

Example 1. **Write a program for the 8051 to transfer letter 'A' serially at 4800- baud rate, 8 bit data, 1 stop bit continuously.**

```
        ORG 0000H
        LJMP START
        ORG 0030H
START:  MOV TMOD, #20H ; select timer 1 mode 2
        MOV TH1, #0FAH ; load count to get baud rate of 4800
        MOV SCON, #50H ; initialize UART in mode 2
; 8 bit data and 1 stop bit
        SETB TR1 ; start timer
```

```
AGAIN:  MOV SBUF, #'A' ; load char 'A' in SBUF
BACK:   JNB TI, BACK ; Check for transmit interrupt flag
        CLR TI ; Clear transmit interrupt flag
        SJMP AGAIN
        END
```

Example 2. **Write a program for the 8051 to transfer the message 'EARTH' serially at 9600 baud, 8 bit data, 1 stop bit continuously.**

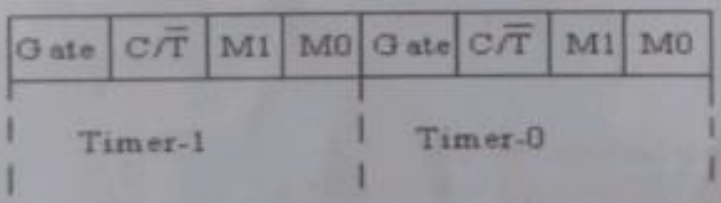
```
        ORG 0000H
        LJMP START
        ORG 0030H
START:  MOV TMOD, #20H ; select timer 1 mode 2
        MOV TH1, #0FDH ; load count to get reqd. baud rate of 9600
        MOV SCON, #50H ; initialise uart in mode 2
; 8 bit data and 1 stop bit
        SETB TR1 ; start timer
        LOOP: MOV A, #'E' ; load 1st letter 'E' in a
        ACALL LOAD ; call load subroutine
        MOV A, #'A' ; load 2nd letter 'A' in a
        ACALL LOAD ; call load subroutine
        MOV A, #'R' ; load 3rd letter 'R' in a
        ACALL LOAD ; call load subroutine
        MOV A, #'T' ; load 4th letter 'T' in a
        ACALL LOAD ; call load subroutine
        MOV A, #'H' ; load 4th letter 'H' in a
        ACALL LOAD ; call load subroutine
        SJMP LOOP ; repeat steps
        LOAD: MOV SBUF, A
HERE:   JNB TI, HERE ; Check for transmit interrupt flag
        CLR TI ; Clear transmit interrupt flag
        RET
        END
```

PROGRAMMING 8051 TIMERS:

- *The 8051 has two timers: timer0 and timer1. They can be used either as timers or as counters. Both timers are 16 bits wide.*

Timer Mode control (TMOD) Special Function Register:

- *TMOD register is not bit addressable.*
- *8 bit register.*
- ***Gate:*** *This is an OR Gate enabled bit which controls the effect of on START/STOP of Timer*
- ***C/T bit*** *and is used to decide whether a timer is used as a time delay generator or an event counter.*



M1	M0	Mode
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

- *If this bit is 0 then it is used as a timer and*
- *if it is 1 then it is used as a counter.*

Timer/ Counter control logic (TCON):

- *TCON is bit addressable.*
- *It is partly related to Timer and partly to interrupt.*

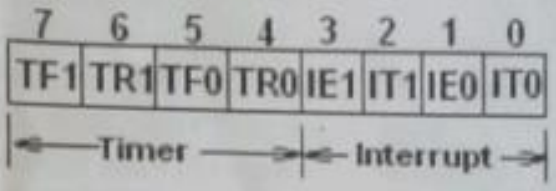
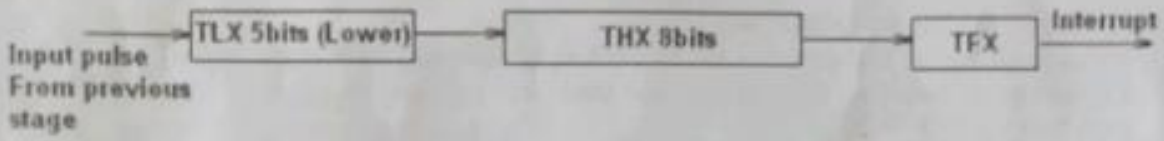


Figure: TCON Register

The various bits of TCON are as follows.

- ***TF1 :*** *Timer1 overflow flag.*
- ***TR1 :*** *Timer1 run control bit.*
- ***TF0 :*** *Timer0 overflow flag.*
- ***TR0 :*** *Timer0 run control bit.*
- ***IE1 :*** *Interrupt1 edge flag.*
- ***IE0 :*** *Interrupt0 edge flag.*
- ***IT1 :*** *Interrupt1 type control bit.*
- ***IT0 :*** *Interrupt0 type control bit.*

Timers can operate in ***four different modes****. They are as follows*



Timer Mode-0: (13-bit UP counter)

- *In this mode, the timer is used as a* ***13-bit UP counter***

Figure: Operation of Timer on Mode-0

- *The lower 5 bits of TLX and 8 bits of THX are used for the 13 bit count. Upper 3 bits of TLX are ignored. When the counter rolls over from all 0's to all 1's, TFX flag is set and an interrupt is generated.*

Timer Mode-1: (16-bit mode)

- *This mode is similar to mode-0 except that the Timer operates in 16-bit mode.*

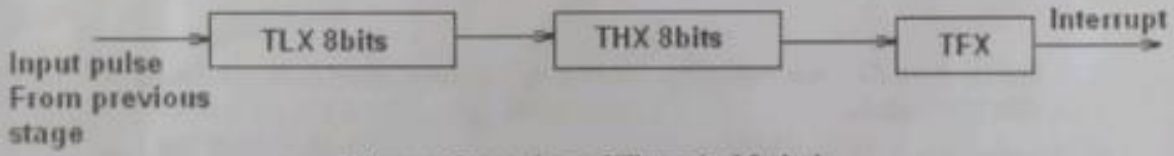


Figure: Operation of Timer in Mode 1

Timer Mode-2: (Auto reload mode)

- *This is a 8 bit counter/timer operation.*
- *Counting is performed in TLX while THX stores a constant value. In this mode when the timer overflows i.e. TLX becomes FFH, it is fed with the value stored in THX.*
- *This mode is useful in applications like fixed time sampling.*

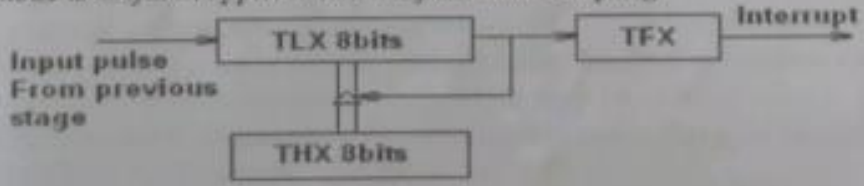
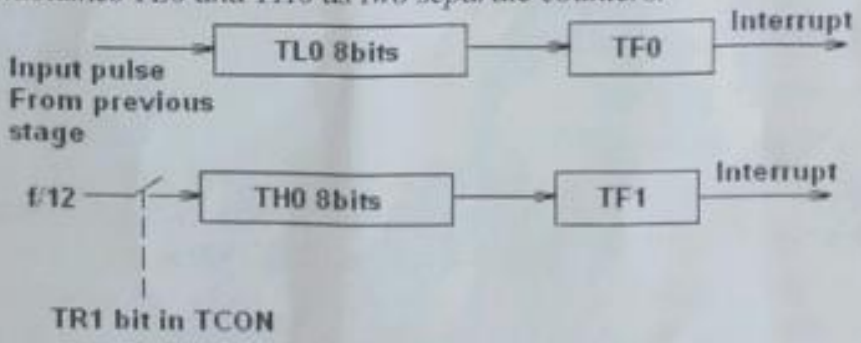


Figure: Operation of Timer in Mode 2

Timer Mode-3: Split Mode (Two 8-bit counters)

- *Timer 1 in mode-3 simply holds its count: The effect is same as setting TR1=0. Timer0 in mode-3 establishes TL0 and TH0 as two separate counters.*



PROGRAMMING 8051 TIMERS IN ASSEMBLY

In order to program 8051 timers, it is important to know the calculation of initial count value to be stored in the timer register. The calculations are as follows.

In any mode, Timer Clock period = 1/Timer Clock Frequency.

= 1/(Master Clock Frequency/12)

a. Mode 1 (16 bit timer/counter)

Value to be loaded in decimal = 65536 – (Delay Required/Timer clock period)

Convert the answer into hexadecimal and load onto THx and TLx register.

(65536D = FFFFH+1)

b. Mode 0 (13 bit timer/counter)

Value to be loaded in decimal = 8192 – (Delay Required/Timer clock period)

Convert the answer into hexadecimal and load onto THx and TLx register.

(8192D = 1FFFH+1)

c. Mode 2 (8 bit auto reload)

Value to be loaded in decimal = 256 – (Delay Required/Timer clock period)

Convert the answer into hexadecimal and load onto THx register. Upon starting the timer this value from THx will be reloaded to TLx register.

(256D = FFH+1)

Steps for programming timers in 8051

Mode 1:

- Load the TMOD value register indicating which timer (0 or 1) is to be used and which timer mode is selected.
- Load registers TL and TH with initial count values.
- start the timer by the instruction "SETB TR0" for timer 0 and "SETB TR1" for timer 1.
- Keep monitoring the timer flag (TF) with the "JNB TFx,target" instruction to see if it is raised. Get out of the loop when TF becomes high.
- stop the timer with the instructions "CLR TR0" or "CLR TR1", for timer 0 and timer 1, respectively.
- Clear the TF flag for the next round with the instruction "CLR TF0" or "CLR TF1", for timer 0 and timer 1, respectively.
- Go back to step 2 to load TH and TL again.

Mode 0:

The programming techniques mentioned here are also applicable to counter/timer mode 0.

The only difference is in the number of bits of the initialization value.

Mode 2:

- Load the TMOD value register indicating which timer (0 or 1) is to be used; select timer mode 2.
- Load TH register with the initial count value. As it is an 8-bit timer, the valid range is from 00 to FFH.
- Start the timer.

Keep monitoring the timer flag (TFx) with the "JNB TFx,target" instruction to see if it is raised. Get out of the loop when TFx goes high.

- Clear the TFx flag.
- Go back to step 4, since mode 2 is auto-reload.

1. Write a program to continuously generate a square wave of 2 kHz frequency on pin P1.5 using timer 1. Assume the crystal oscillator frequency to be 12 MHz.

The period of the square wave is T = 1/(2 kHz) = 500 □s.

Each half pulse = 250 □s.

The value n for 250 □s is: 250 □s /1 □s = 250 65536 - 250 = FF06H.

TL = 06H and TH = 0FFH.

```
        MOV TMOD,#10 ;Timer 1, mode 1
AGAIN:  MOV TL1,#06H ;TL0 = 06H
        MOV TH1,#0FFH ;TH0 = FFH
        SETB TR1 ;Start timer 1
BACK:   JNB TF1,BACK ;Stay until timer rolls over
        CLR TR1 ;Stop timer 1
        CPL P1.5 ;Complement P1.5 to get Hi, Lo
        CLR TF1 ;Clear timer flag 1
        SJMP AGAIN ;Reload timer
```

2. Write a program segment that uses timer 1 in mode 2 to toggle P1.0 once whenever the counter reaches a count of 100. Assume the timer clock is taken from external source P3.5 (T1).

The TMOD value is 60H. The initialization value to be loaded into TH1 is 256 - 100 = 156 = 9CH

```
        MOV TMOD,#60h ;Counter1, mode 2, C/T'= 1
        MOV TH1,#9Ch ;Counting 100 pulses
        SETB P3.5 ;Make T1 input
        SETB TR1 ;Start timer 1
BACK:   JNB TF1,BACK ;Keep doing it if TF = 0
        CPL P1.0 ;Toggle port bit
        CLR TF1 ;Clear timer overflow flag
        SJMP BACK ;Keep doing it
```

INTERRUPTS PROGRAMMING.

During program execution if peripheral devices needs service from microcontroller, device will generate interrupt and gets the service from microcontroller. When peripheral device activate the interrupt signal, the processor branches to a program called interrupt service routine. After executing the interrupt service routine the processor returns to the main program.

Steps taken by processor while processing an interrupt:

1. It completes the execution of the current instruction.

2. PSW is pushed to stack.
3. PC content is pushed to stack.
4. Interrupt flag is reset.
5. PC is loaded with ISR address.

ISR will always ends with RETI instruction. The execution of RETI instruction results in the following.

1. POP the current stack top to the PC.
2. POP the current stack top to PSW.

Classification of interrupts.

1. External and internal interrupts.

External interrupts are those initiated by peripheral devices through the external pins of the microcontroller.

Internal interrupts are those activated by the internal peripherals of the microcontroller like timers, serial controller etc.)

2. Maskable and non-maskable interrupts.

The category of interrupts which can be disabled by the processor using program is called maskable interrupts.

Non-maskable interrupts are those category by which the programmer cannot disable it using program.

3. Vectored and non-vectored interrupt.

Starting address of the ISR is called interrupt vector. In vectored interrupts the starting address is predefined. In non-vectored interrputs, the starting address is provided by the peripheral as follows.

- Microcontroller receives an interrupt request from external device.
- Controller sends an acknowledgement (INTA) after completing the execution of current instruction.
- The peripheral device sends the interrupt vector to the microcontroller.

8051 INTERRUPT STRUCTURE.

8051 has five interrupts. They are maskable and vectored interrupts. Out of these five, two are external interrupt and three are internal interrupts.

Interrupt source	*Type*	*Vector address*	*Priority*
External interrupt 0	External	0003	Highest
Timer 0 interrupt	Internal	000B	
External interrupt 1	External	0013	
Timer 1 interrupt	Internal	001B	
Serial interrupt	Internal	0023	Lowest

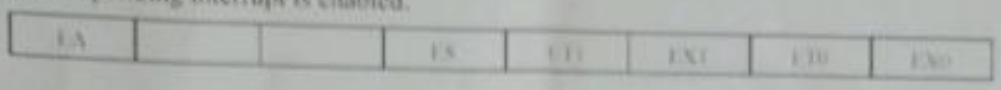
8051 makes use of two registers to deal with interrupts.

1. IE Register

This is an 8 bit register used for enabling or disabling the interrupts. The structure of IE register is shown below.

IE : Interrupt Enable Register (Bit Addressable)

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.



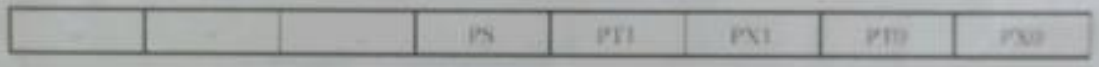
EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
-	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

2. IP Register.

This is an 8 bit register used for setting the priority of the interrupts.

IP : Interrupt Priority Register (Bit Addressable)

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is the corresponding interrupt has a higher priority.



-	IP.7	Not implemented, reserved for future use*.
-	IP.6	Not implemented, reserved for future use*.
-	IP.5	Not implemented, reserved for future use*.
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 Interrupt priority level.
PX1	IP.2	Defines External Interrupt priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level.
PX0	IP.0	Defines the External Interrupt 0 priority level.

Interrupts in 8051

* The 8051 has 5 interrupt sources

Source	Description
$\overline{INT_0}$	External request from $P_{3.2}$ pin
Timer 0	Overflow from Timer 0 activates the Interrupt Request Flag TF0
$\overline{INT_1}$	External request from Pin 3.3
Timer 1	Overflow from Timer 1 activates the Interrupt Request Flag TF_1.
Serial port	Completion of the transmission/ reception of a serial frame activates the flags T_1/R_1.

* Interrupts can be enabled or disabled by (SFR-IE) Interrupt Enable Register
  Setting or clearing a bit in
* Their priorities also can be programmed by ~~using~~ Setting or clearing the bits in Interrupt Priority Register (SFR-IP)
* ~~If two interrup~~ A low priority source can be interrupted by another high priority interrupt
* A high priority interrupt cannot be interrupted by another high/low priority interrupt.

* If two interrupts of same priority arise simultaneously, then the interrupt priority is decided as.

Source	Priority level
$\overline{INT0}$	(Highest)
Timer 0	
INT 1	
Timer 1	
Serial port	(lowest)

An interrupt is blocked, if

* During each machine cycle, all the interrupts are separately scanned.
* all the interrupts are prioritized by S_6 of any machine cycle.
* The processor will go to the interrupt serivee in the S_1 of the next machine cycle, only if is not blocked by any of the following condition.

(1) An interrupt of equal or higher priority is already in process.

(2) No interrupt request will be responded to until the instruction in progress is completed.

(3) The instr in progress is RETI or an access to the special function Reg IE or IP. In this case, the response will come only after executing atleast one more instruction

* Once an interrupt is not blocked, a h/w call is generated, after which the program counter value is stored in stack. the pgm branches to predefined location

Source	Location
$\overline{INT0}$	0003_H
T0	$000B_H$
$\overline{INT1}$	0013_H
T_1	$001B_H$
Serial port	0023_H.

* when RETI (Return from Interrupt) instruction of the Interrupt Service Routine (ISR) is executed, the processor can return to the location where the Interrupt took place.

Interrupt Enable Register (IE)

EA	X	X	ES	ET1	EX1	ET0	EX0

EA
Disable all interrupts

ES = 0; Serial port interrupt is disabled
ES = 1; Enabled

ET_1 – Timer 1 Overflow Interrupt.
ET_1 = 0; disabled
ET_1 = 1; Enabled

ET_0 – Timer 0 overflow Interrupt
ET0 = 0; disabled
ET_0 = 1; Enabled

EX_1 – External Interrupt1
EX_1 = 0; disabled
EX_1 = 1; Enabled

EX_0 – External Interrupt0
EX0 = 0; disabled
EX0 = 1; Enabled

Interrupt Priority Register (IP)

x	x	x	PS	PT1	PX1	PT0	PX0

* If any of this bit is set to 1 : then it is highest priority

Interrupt Control Registers

* External interrupts can be programmed to be either level triggered or edge triggered.

*

Source

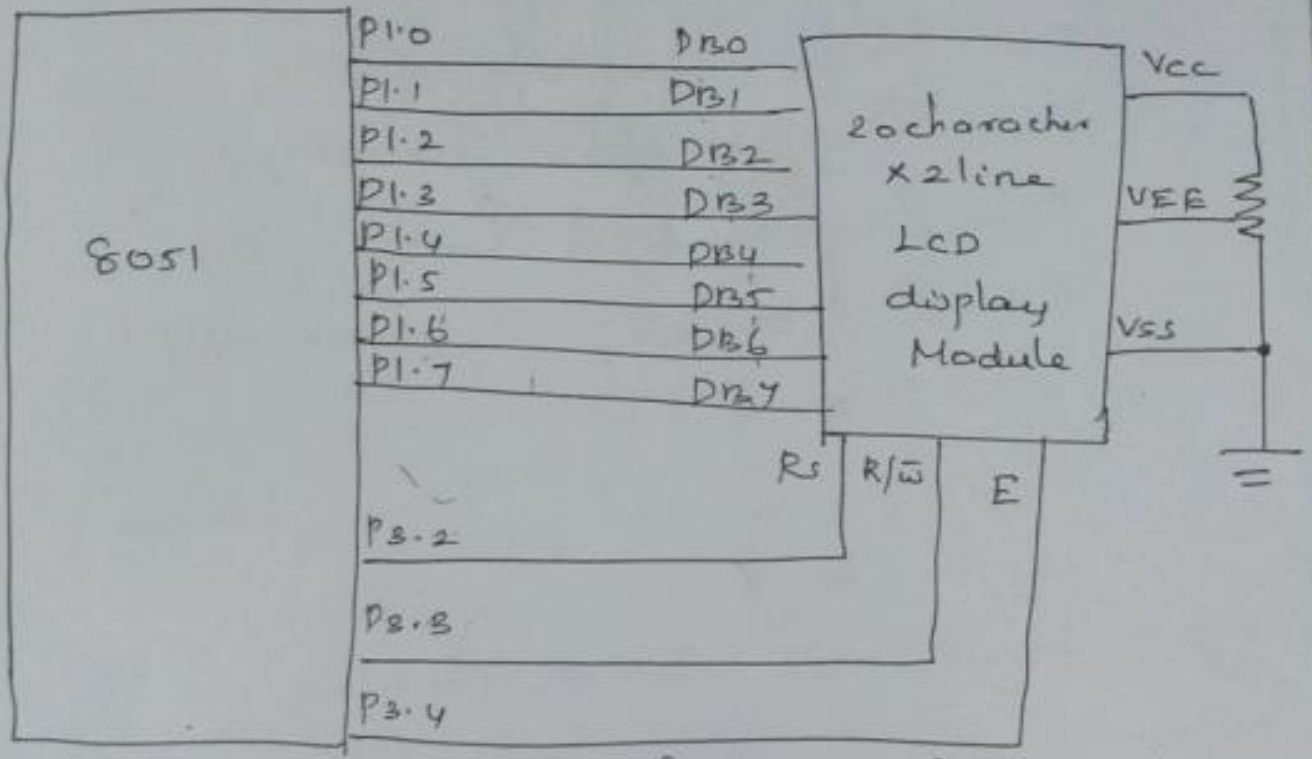
$\overline{INT0}$ – Level triggered, if IT_0 bit TCON reg is 0.

– Edge triggered, If IT_0 bit in TCON reg 1

$\overline{INT1}$ – Level triggered, If IT_1 in TCON = 0

edge triggered, If IT_1 in TCON = 1

Interfacing LCD Module with 8051.



- It shows the Interfacing of a 20character x 20 line LCD module with the 8051.
- The data lines are Connected to the port1 of 8051 and Control lines R_S, R/W & E are driven by 3.2, 3.3 and 3.4 lines of port 3
- The Voltage of VEE pin is adjusted by a potentiometer to adjust the Contrast of the LCD.

Command Code:

1- clear display screen
2- Return home
4- Decrement Cursor

6 – Increment Cursor
5 – shift display right
7 – shift display left
8 – Display off, Cursor off
A – Display off, Cursor ON
C – Display off, Cursor off
E – Display off, Cursor ON
F – Display off, Cursor Linker
10 – shift Cursor right
14 – shift Cursor left

Program:

```
Mov A, Command 1
Lcall cmd
Lcall display
Mov A, Command 2
Lcall cmd
Lcall display
Mov A, #'A'
Lcall data
Lcall delay
Mov A, #'A'
Lcall data
Lcall delay
```

```
CMd :  Mov A1 , A
       Clr b  P3.3
       Clr b  P3.4
       SETb   P3.5
       Clr b  P3.5
       RET
data :  Mov  P1, A
        SETA  P3.3
        SETB  P3.4
        SETB   P3.5
        SETB   P3.5
         RET
```

Keyboard Interfacing:-

i) Simple keyboard Interface:

The Eight keys are individually Connected to Specific pins of port p1. Each port pin gives the status of key Connected to that Pin. When port pin is logic 1, key is open, otherwise key is closed.

The figure shows the Simple keyboard Interface.

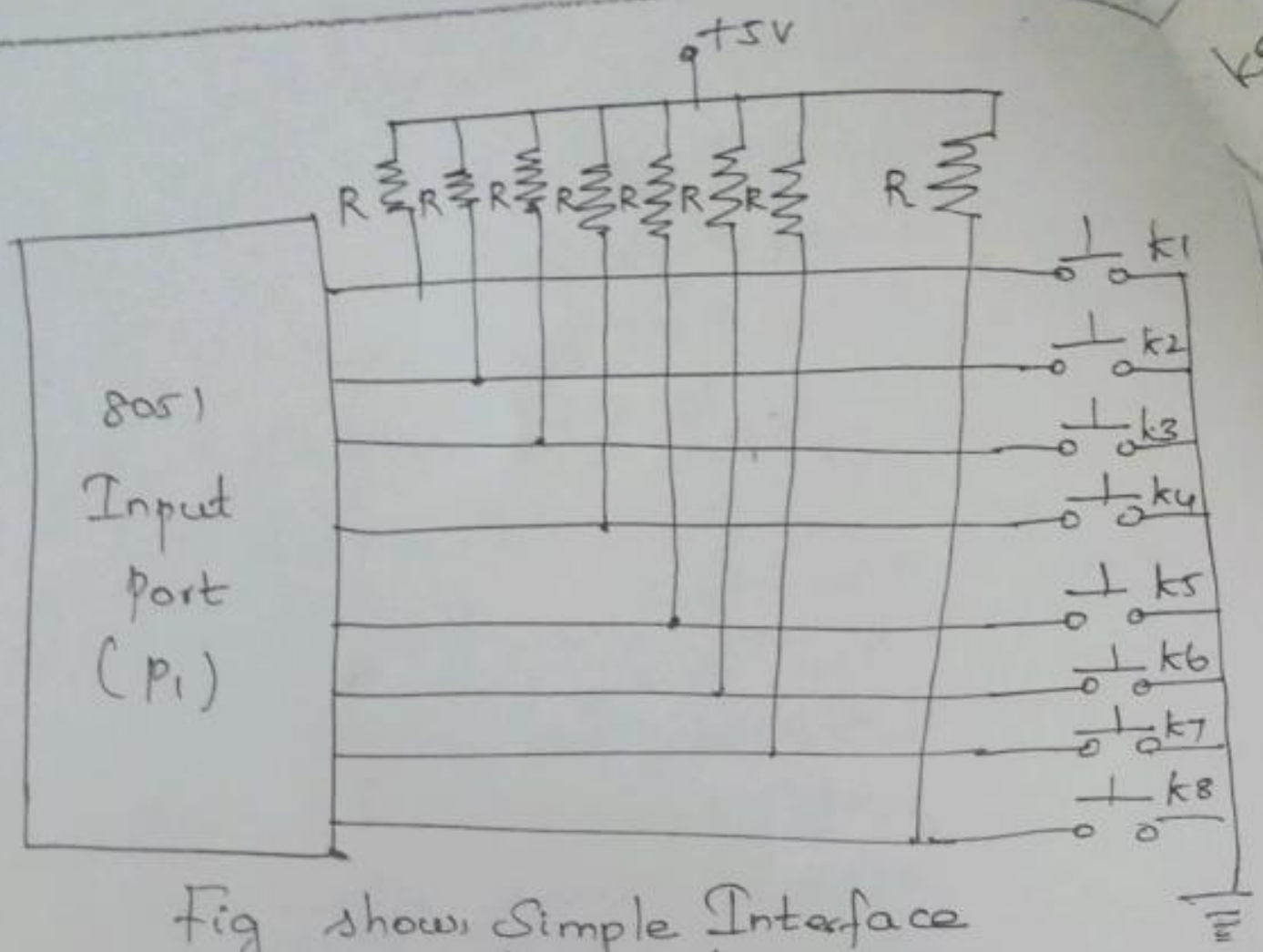


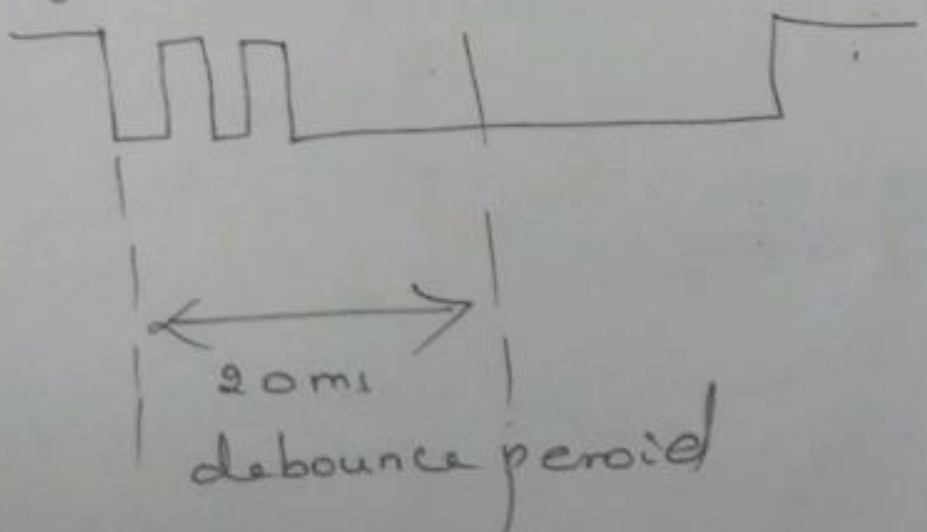
Fig shows Simple Interface keyboard

Two cases:

i) Open ⇒ high, close - low

ii) If press ⏚ it will bounce.

key bouncing



key	key code							
	D7	D6	D5	D4	D3	D2	D1	D0
k1	1	1	1	1	1	1	1	0
k2	1	1	1	1	1	1	0	1
k3	1	1	1	1	1	0	1	1
k4	1	1	1	1	0	1	1	1
k5	1	1	1	0	1	1	1	1
k6	1	1	0	1	1	1	1	1
k7	1	0	1	1	1	1	1	1
k8	0	1	1	1	1	1	1	1

(ii) Matrix keyboard Interface :-

In simple keyboard Interface one input line is required to interface one key and this number will increase with no of keys. Therefore such technique is not suitable when it is necessary to interface large no of keys. To reduce number of Connection key are arranged in matrix form

— The figure shows 16 keys arranged in four rows & four Columns, when keys are open, row & Column donot have any Connection.

- When a key is pressed, it shorts Corresponding one row & one Column.

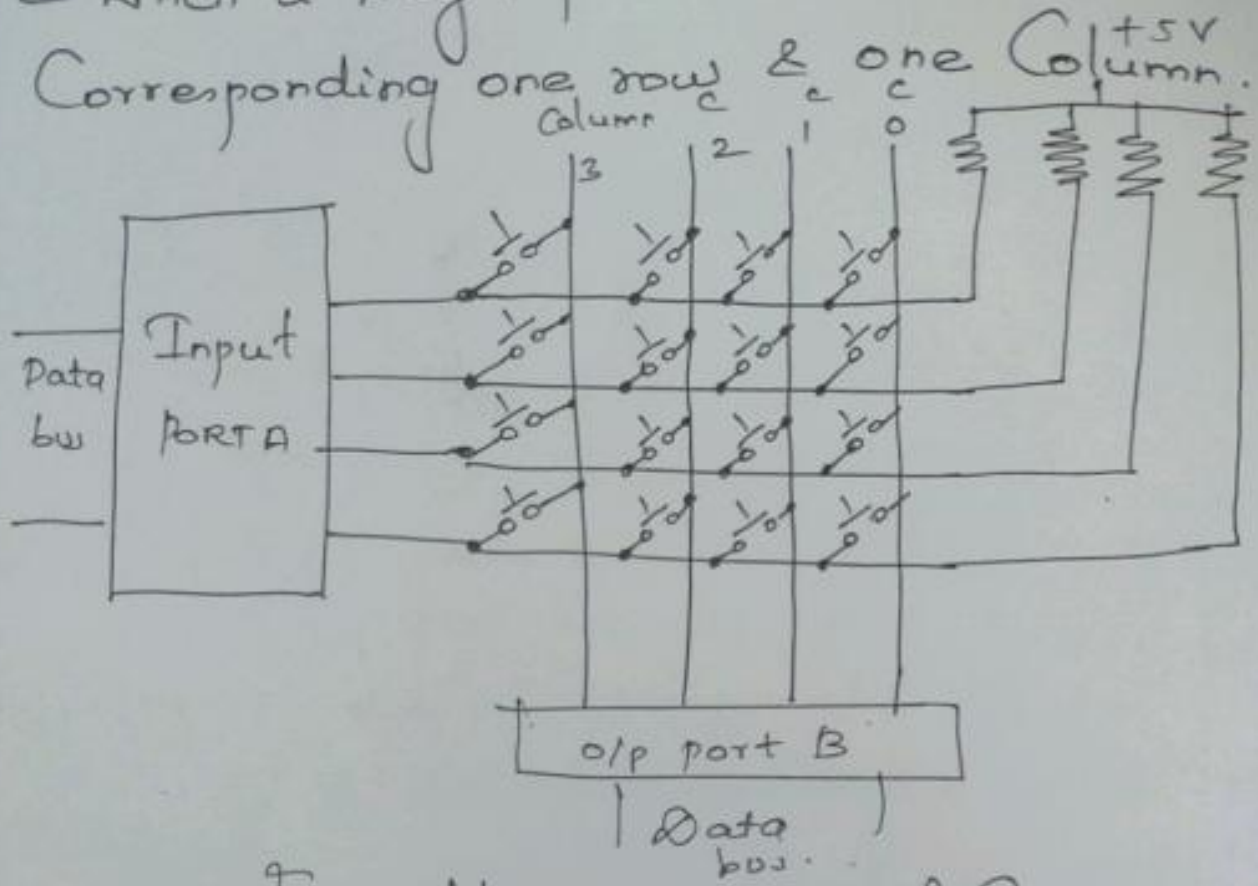


Fig : Matrix keyboard Connection.

- It shows the Interfacing of Matrix keyboard
- It requires two ports
  - Input port
  - output port
- Rows are Connected to the input port referred to a returned lines
- Columns are Connected to the output Port referred to as Scan lines.
- When any key is pressed it shorts Corresponding row & Column.
- If the o/p line of this Column is low,

, it makes Corresponding row line low, otherwise the status of row line is high.

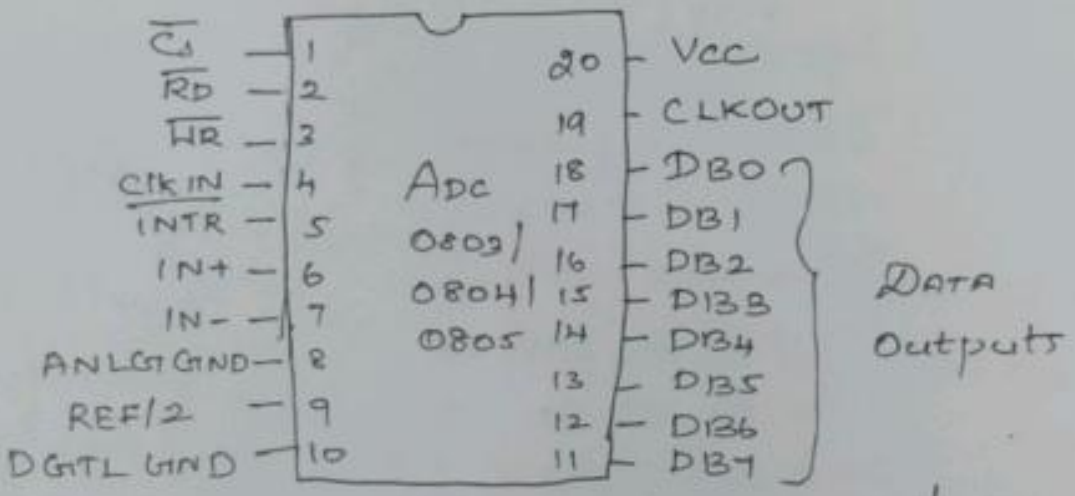
Program:-

```
Scan :  Mov P1,#EF
Scan1: JNB p1.0, db-0
Ccan2: JNB P1.1, db-1
Scan3 : JNB P1.2, db-2
        JNB P1.3, db-3
        Mov P1, # DF
        JNB P1.0, db-0
        JNB P1.1, db-1
        JNB P1.2, db-2
        JNB P1.3, db-3
        Mov P1, # Bf
        JNB P1.0, db-0
        JNB P1.1, db-1
        JNB P1.2, db-2
        JNB P1.3, db-3
        Lcall wait_20 ms
        JB P1.0, Scan1
        Mov A, #00
        LJMP get
        Lcall wait_20 ms
        JB P1.1, Scan 2
        Mov A, #01
        LJMP get
```

```
Mov dptr, #tab
Moc c A@ A+dpt
LJMP scan
End.
```

ADC Interfacing :-

The ADC 0803, ADC0804 and ADC 0805 are CMOS 8-bit successive-approximation analog to digital. These devices are design to operate from Common Microprocessor control buses, with tri-state output latches driving the data bus and are identical except for accuracy.



Pin diagram of ADC 0803/0804/0805

Features :-

- 8 bit successive Approximation ADC
- Conversation time 100μs
- Access time 135ns
- It has an on-chip clock generator
- It doesnot require any zero adjustment
- It operates on single 5V power supply
- Output meet TTL Voltage level specification

Interfacing of ADC 0803/0804/0805 with 8051.

The fig shows the Interfacing of ADC 0803/0804/0805 with 8051 using port1 and port2 Here, port1 is used to read digital data from ADC and port2 is used to provide Control signals to ADC 0803/0804/0805. Potentiometer is used to adjust Vin+ Voltage. The clock signal is Provided using internal clock generator & two external Components, resistor and Capacitor.

The frequency of such clock can be determined by

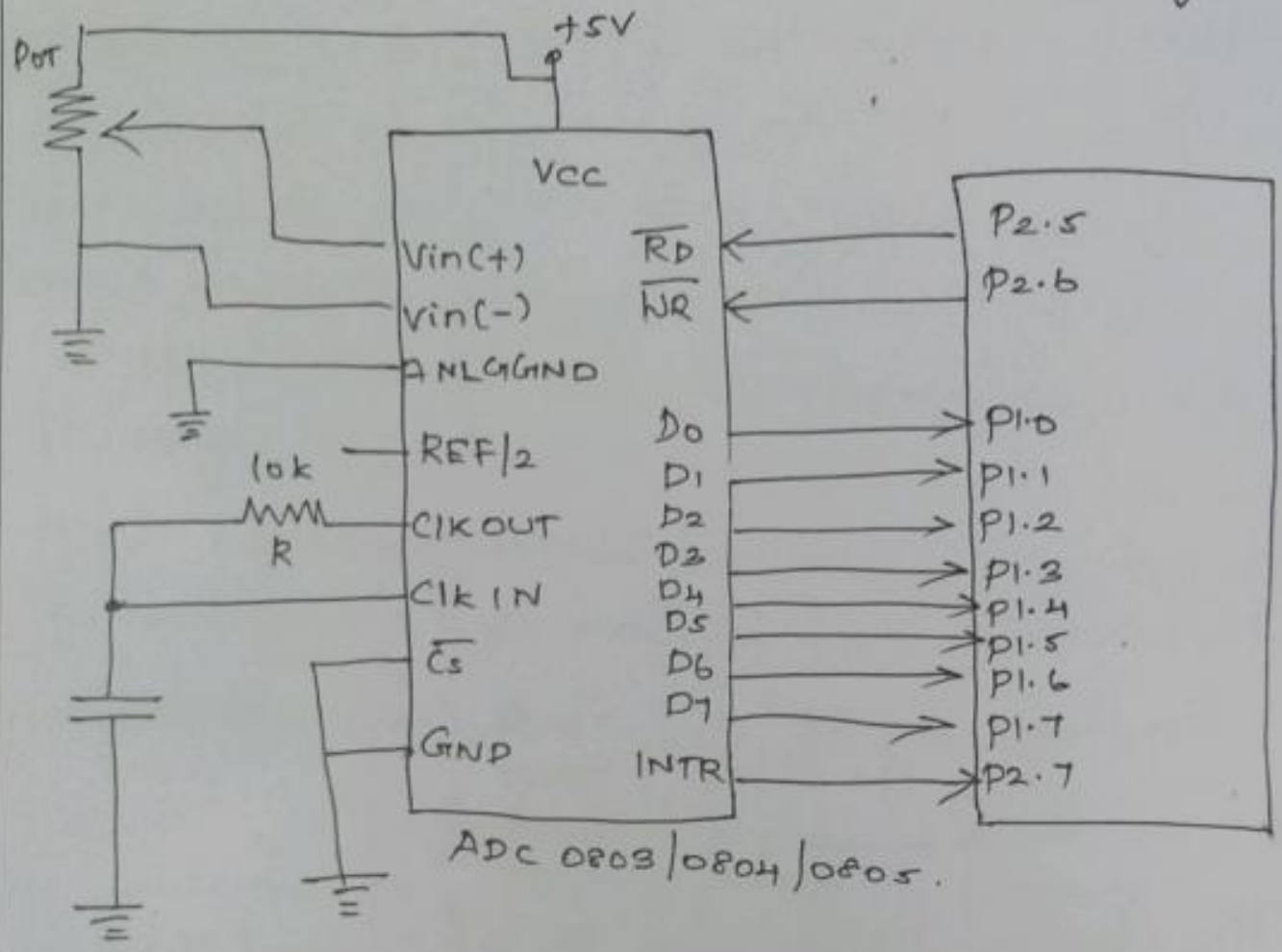
$$f = \frac{1}{1.1RC}$$

The typical Values are R = 10kΩ and C = 150pf with these Values we get approximately 606 kHz Clock frequency. In such case, the Conversion time is around 110 μs.

AD Conversation Program:

```
        Mov P1,#0ffH  ; Configure port1 as input
Back:   CLR P2.6      ; Make WR=0 and
        SETB P2.6     ; Make WR=1 to generate
                      Start of Conversion pulse
AGAIN:  JB p2.7,AGAIN ; Wait for end of Conversation
        CLR P2.5      ; Enable Read
        Mov A,P1      ; Read data through PORT1
        SETB P2.5     ; Disable read after reading data
        SJMP BACK     ; go for next Conversation cycle
```

Posl Connection to ADC 0804 with self-clocking mode



DAC Interfacing :- IC DAC 1408

The 1408 is an 8 bit R/2R ladder type D/A Convertor Compatible with TTL and CMOS logic It is designed to use where the output Current is linear product of an eight-bit digital word.

Fig shows the pin diagram and block diagram for IC 1408 DAC

Square ware:

To generate Square ware first we have to Output FF and then 00 on port 1 of 8051. The port 1 is Connected as an i/p to the DAC 0808. According to frequency requirement delay is provided between two o/ps.

Program:

```
        Mov SP, #08H    ; Initialize stack pointer
Repeat: Mov P1, #0FFH   ; Load all 1's in port 1
        Lcall Delay     ; call delay routine
        Mov P1, #00H    ; Load all 0's in port 1
        Lcall Delay     ; Call delay routine
        LJMP REPEAT     ; Repeat
DELAY : Mov R0, #0FFH   ; Load delay count
BACK :  ~~Mov R0, #0FFH~~ ; Decrement & check
        DJNZ R0, BACK   whether delay count
                        is zero if not repeat
                        the operation
        RET             ; Return to Main program
```

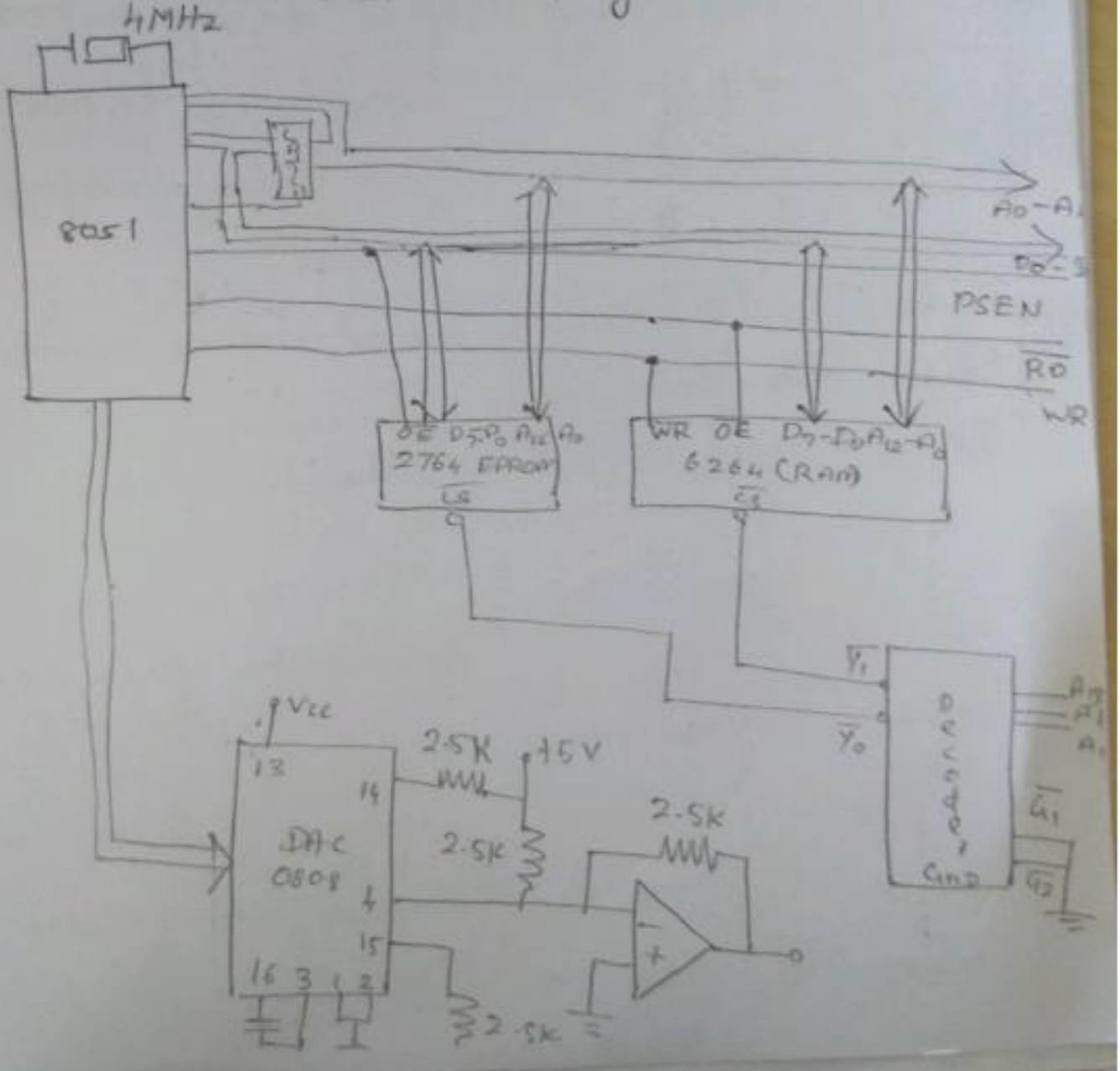
Sensor Interfacing :-

The National LM35 is a temperature sensor. It is a temperature-sensitive Voltage source. Its Output Voltage

Fig: Pin diagram

Interfacing DAC 1408 with 8051

Fig shows the interfacing of DAC 0808 with 8051 microController based system.



increases by 10mV for each °c increase in its temperature.

The fig shows the Circuit Connection for temperature sensor LM35

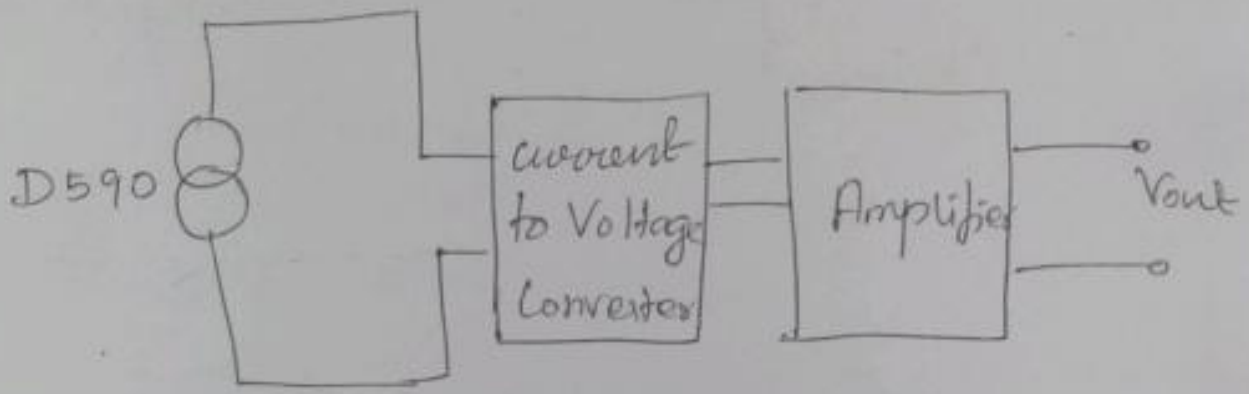


fig: Temperature Sensor Using AD59

Main program:-

```
        Mov 81H, #30H ; Initialize stack
        Mov P3, #0FFH ; Configure port3 as i/p
        Mov P2, #0FFH ; Configure port2 as i/p
START : ACALL R_ADC ; Read data from ADC
        ACALL BAC ; Convert binary to ASCII
        ACALL COMT ; If temperature is greater
                     than 100°C make
                     buzzer ON
                     O.W make buzzer off
        ACALL LCD ; Display it on LCD
        SJMP START ; Repeat.
```

External Memory Interface:-

To interface a memory chip to the microprocessor, the following signals are required to be generated.

i) Address bus signals
ii) Data bus signals
iii) Memory chip select signals
iv) Read control signal
v) Write Control signal. (only in case of RAM)

— Address and data bus signals are provided through alternate function in ports P0 & P2 Pins.

- Pins P2.0 to P2.7 Constitute A_8 to A_{15} (i.e) Higher order address lines
- Pin P0.0 to P0.7 represent both data lines (D_0-D_7) and lower-order address line A_0,

— The Memory chip select signals are obtained by decoding the address information, through a decoder.

The chip select signal to a Memory chip signifies that the address in question is Contained in the chip & thus the read/write operation will be performed on the chip selected.

— The Address space required for the application is decided and then, the same is divided into program & data memory as well as RAM & ROM.

— The Address space for each chip is decided & based on this, the decoder is Connected to different address lines

- For External data Memory, $\overline{RD}$ and $\overline{WR}$ are used as read & write control signals. Fig. External data memory shows the Interfacing of the external data memory to the 8051.

& Access to the external data memory can be either a 16 bit address or 8 bit address.

- Only ROM is allowed as the External Program memory. $\overline{PSEN}$ (program store enable) act as the read ctrl signal to access the Memory. Fig shows the Interfacing of External program memory to the 8051

- Merging the External data Memory & external program memory space into One single 64KB Memory space. Fig shows the interfacing of the Combined external program & data Memories to Intel 8051.

$\overline{WR}$ is used as write control signal.

$\overline{PSEN}$ & $\overline{RD}$ are ANDed to provide read ctrl signal.

Stepper Motor Using 8051

① A Switch is connected to pin P2.7, write a ALP to monitor the states of switch and perform the following.

(i) If sw=0 ; Stepper motor moves clockwise

(ii) If sw=1 ; Stepper motor moves counter clockwise.

Clockwise (condition is sw=0.)

CLR $P_{2.7}$

```
Start : Mov DPTR,#4500;
        Mov R0,#04;
Again : Movx A,@DPTR;
        PUSH DPH;
        PUSH DPL;
        Mov DPTR,#FFC0H;
        MOV R2,#04H;
        MOV R1,#FFH;
        MOV R3,#FFH;
DLY1 : DJNZ R3, DLY;
DLY : DJNZ R1, DLY1;
        MOVX @DPTR,A;
        POP DPH;
        POP DPL;
        INC DPTR;
        DJNZ R0, Again;
```

Lookup table

Clockwise

4500	09
4501	05
4502	06
4503	0A

Anticlockwise

4500	0A
4501	06
4502	05
4503	09

Anticlockwise (sw=1)

SETB $P_{2.7}$.

```
Start : Mov DPTR,#45
        ;
```

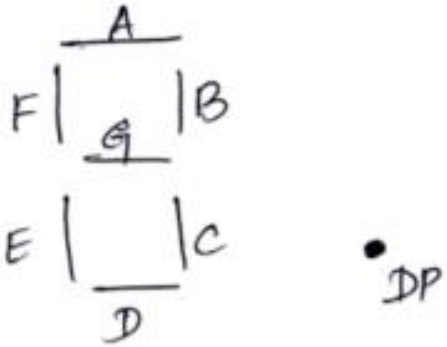
Repeat Same program

DJNZ R0, Again.

2) Develop an 8086 based system to display a message "HELLO" in the Seven Segment LED display.

Execution: HELLO

7 Seg Display



Letter table

D	C	B	A	DP	G	F	E	DATA
1	0	0	1	1	0	0	0	H 98
0	1	1	0	1	0	0	0	E 68
0	1	1	1	1	1	0	0	L 7C
0	1	1	1	1	1	0	0	L 7C
0	0	0	0	1	1	0	0	O 0C

Lookup table

1200	FF	
1201	FF	
1202	FF	
1203	FF	
1204	FF	
1205	FF	
1206	FF	
1207	FF	
1208	98	H
1209	68	E
120A	7C	L
120B	7C	L
120C	0C	O
120D	FF	
120E	FF	
120F	FF	

Program

```
Start : MOV SI,1200
        MOV CX,000F
        MOV AX,10;
        OUT C2,AX
        MOV AX,CC
        OUT C2,AX
        MOV AX,90
        OUT C2,AX
Next  : MOV AX,[SI]
        OUT C0,AX
        CALL DELAY
        INC SI
        LOOP Next
        JMP start

Delay : MOV DX,#count
Loop  : DEC DX
        JNZ loop
        RET
```