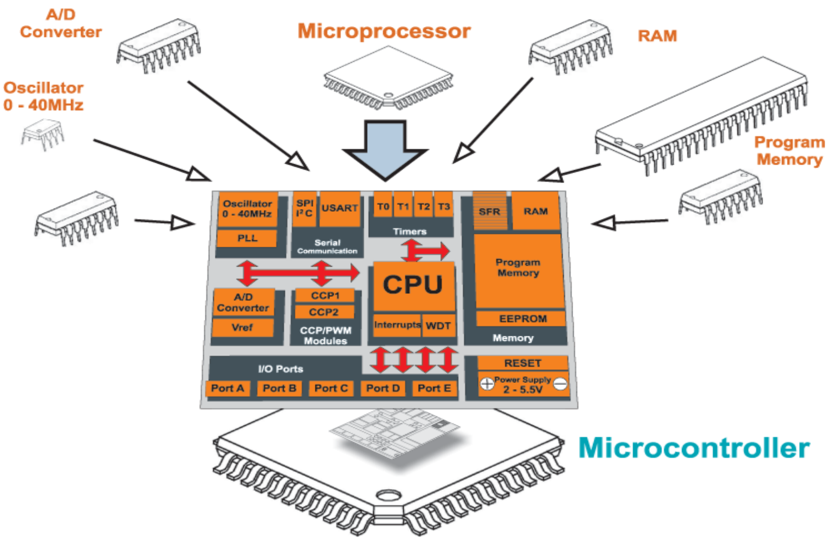


Microprocessors and Microcontrollers

Prepared

By

Papa Rao N, Asst. Professor

What is Microprocessor?

- 1960s' CPU – designed with logic gates
- LSI – Large Scale Integration
- SSI to LSI – called Microprocessor
- Microcomputer
- Intel – 4 bit microprocessor 4004 in 1971
- 8 bit microprocessor 8080
- 8-bit 8085 (8 bit data bus + 16 bit address bus)
- 16-bit 8086 (16 bit data bus + 20 bit address bus)
- 16 bit processors – 8088,80186,80188, 80286
- 32 bit processors – 80386 , 80486, 80586 (P)

What is Microprocessor?

- The word comes from the combination micro and processor.
- Processor means a device that processes whatever. In this context processor means a device that processes numbers, specifically binary numbers, 0’s and 1’s.
- To process means to manipulate. It is a general term that describes all manipulation. Again in this content, it means to perform certain operations on the numbers that depend on the microprocessor’s design.
- The microprocessor is a programmable device that “takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers as a result”.

What is Microcontroller?

- LSI to VLSI – called Microcontroller
- To build Microprocessor, memory and I/O devices on a single chip
- Components
    - Microprocessor
    - A/D Converter
    - D/A Converter
    - Parallel I/O Interface
    - Serial I/O Interface
    - Timers and Counters

8085 Microprocessor

The salient features of 8085 µp are:

- It is a 8 bit microprocessor.
- It is manufactured with N-MOS technology.
- It has 16-bit address bus and hence can address up to 216 = 65536 bytes (64KB)
- memory locations through A_0-A_{15}.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed $AD_0 – AD_7$.
- Data bus is a group of 8 lines $D_0 – D_7$.
- It supports external interrupt request.
- A 16 bit program counter (PC)
- A 16 bit stack pointer (SP)
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock. It is enclosed with 40 pins DIP (Dual in line package).

8085 Architecture

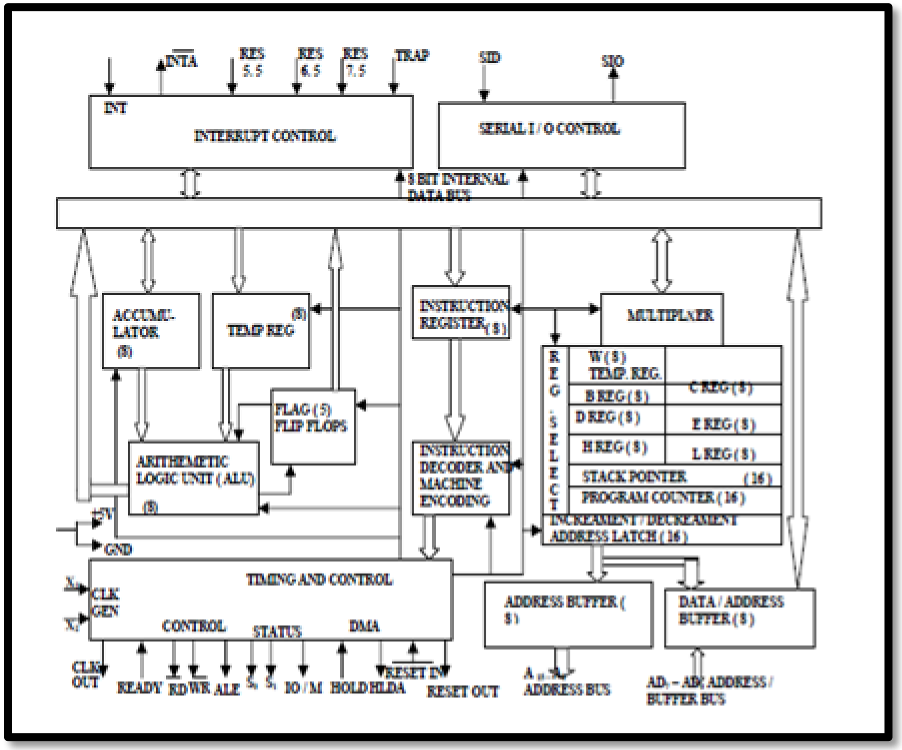


Figure: 8085 Micro Processor Architecture

Flag register and GPR of 8085

Flag register

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	X	AC	X	P	X	CY

General Purpose registers:

Individual	**B, C, D, H and L**
Combinations	**BC, DE and HL**

Instruction Set

- 8085 instruction set consists of the following instructions:
- Data moving instructions.
- Arithmetic - add, subtract, increment and decrement.
- Logic - AND, OR, XOR and rotate.
- Control transfer - conditional, unconditional, call subroutine, return from subroutine and restarts.
- Input / Output instructions.
- Other - setting/clearing flag bits, enabling/disabling interrupts, stack operations, etc.

Addressing modes

- **Register:**

  references the data in a register or in a register pair.

- **Register indirect:**

  instruction specifies register pair containing address, where the data is located.

- **Direct, Immediate:**

  8 or 16-bit data.

8086 microprocessor

- It is a 16-bit μp.
- 8086 has a 20 bit address bus can access up to 220 memory locations (1 MB).
- It can support up to 64K I/O ports.
- It provides 14, 16 -bit registers.
- It has multiplexed address and data bus AD0- AD15 and A16 – A19.
- It requires single phase clock with 33% duty cycle to provide internal timing.
- 8086 is designed to operate in two modes, Minimum and Maximum.
- It can prefetches upto 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply.
- A 40 pin dual in line package.

- **The 8086 architecture has two parts:**
  - Bus Interface Unit(BIU)
  - Execution Unit(EU)

8086 block diagram

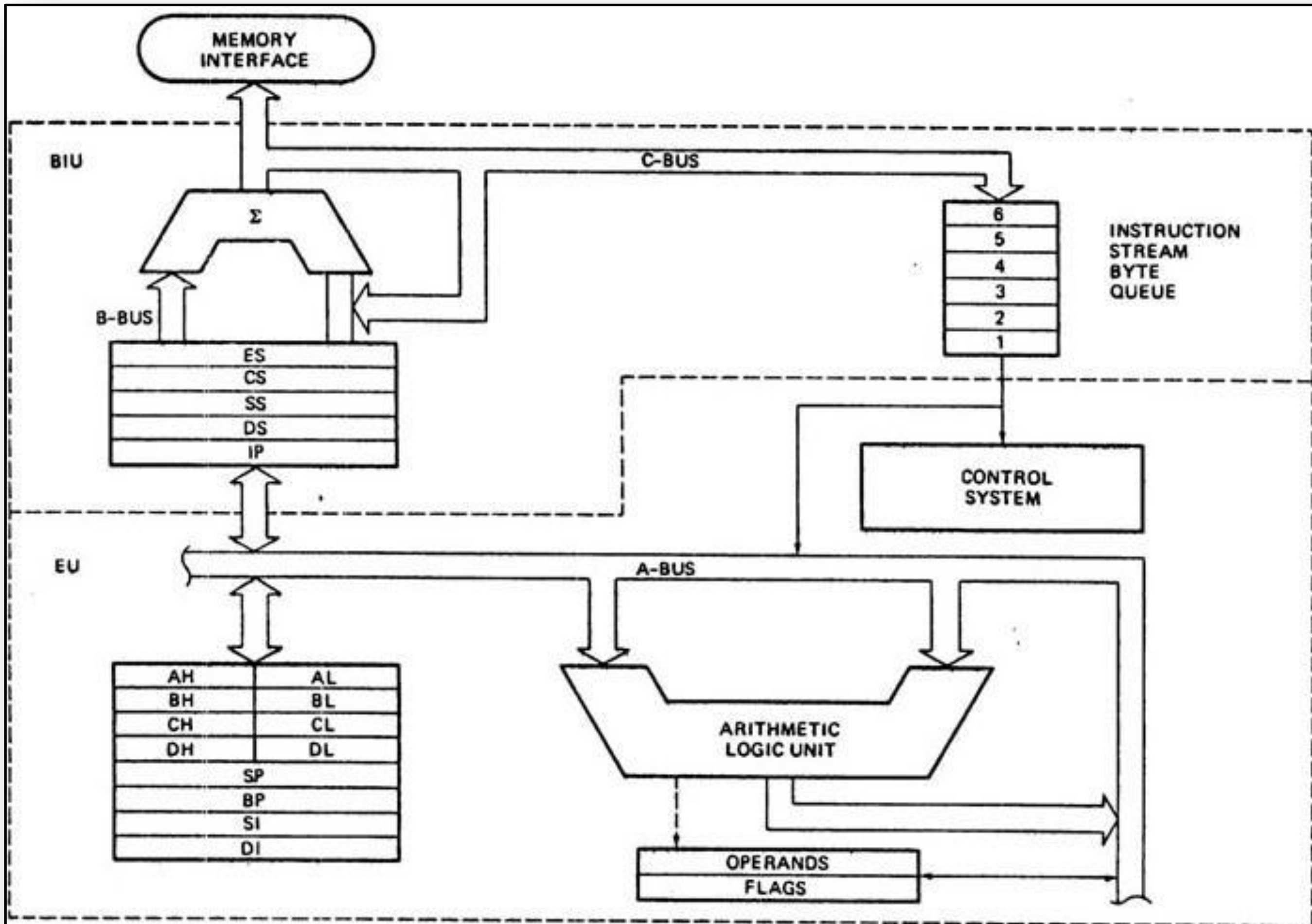


Figure: 8086 Microprocessor Architecture

- Bus Interface Unit contains
  - Instruction queue,
  - Segment registers,
  - Instruction pointer, and
  - Address adder.

- Execution Unit contains
  - Control circuitry,
  - Instruction decoder,
  - ALU,
  - Pointer and Index register,
  - Flag register

Bus interface unit functions

Responsible for performing external bus operations

- The functions of BIU are:
  - Instruction Fetch
  - Instruction Queuing
  - Operand Fetch & storage
  - Address Relocation
  - Bus control
- Idle state
- Address adder – fetching of physical address of next instruction(CS+IP

Execution Unit Functions

- Decoding of Instructions
- Execution of instructions
- **Steps**
  - EU extracts instructions from top of queue in BIU
  - Decode the instructions
  - Generates operands if necessary
  - Passes operands to BIU & requests it to perform read or write bus cycles to memory or I/o
  - Perform the operation specified by the instruction on operands
  - Branch or jump instruction

Register Organization

- The types of registers are:

1. General Data Registers(AX, BX, CX, DX)
2. Segment Registers(CS, DS, ES, SS)
3. Pointers and Index Registers(IP, BP, SP)
4. Flag Registers(S,Z,P,C,T,I,D,AC,O)

General Data Registers

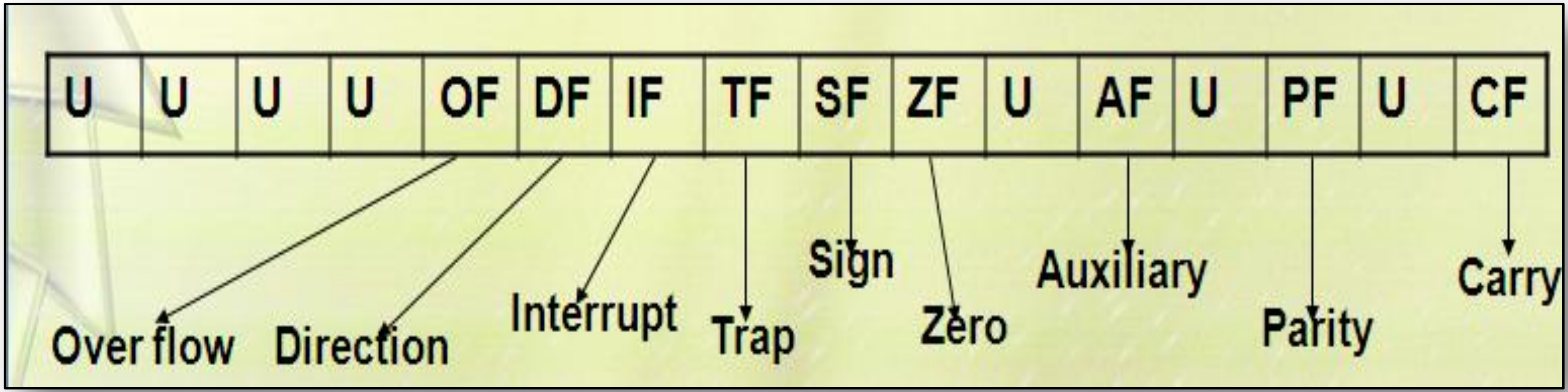
- AX—16 bit accumulator(AH+AL)
- BX-offset storage(BH+BL)
- CX-default counter in case of string and loop instructions(CH+CL)
- DX-General purpose register (DH+DL)

Segment Registers

- Code Segment Register(CS)
- Data Segment Register(DS)
- Extra Segment Register(ES)
- Stack Segment Register(SS)

Flag Registers(S,Z,P,C,T,I,D,AC,O)

- A flag is a flip flop which indicates some conditions produced by the execution of an instruction or controls certain operations of the EU .
- In 8086 The EU contains
  - a 16 bit flag register
  - 9 of the 16 are active flags and remaining 7 are undefined.
  - 6 flags indicates some conditions- status flags
  - 3 flags –control Flags



Programming model

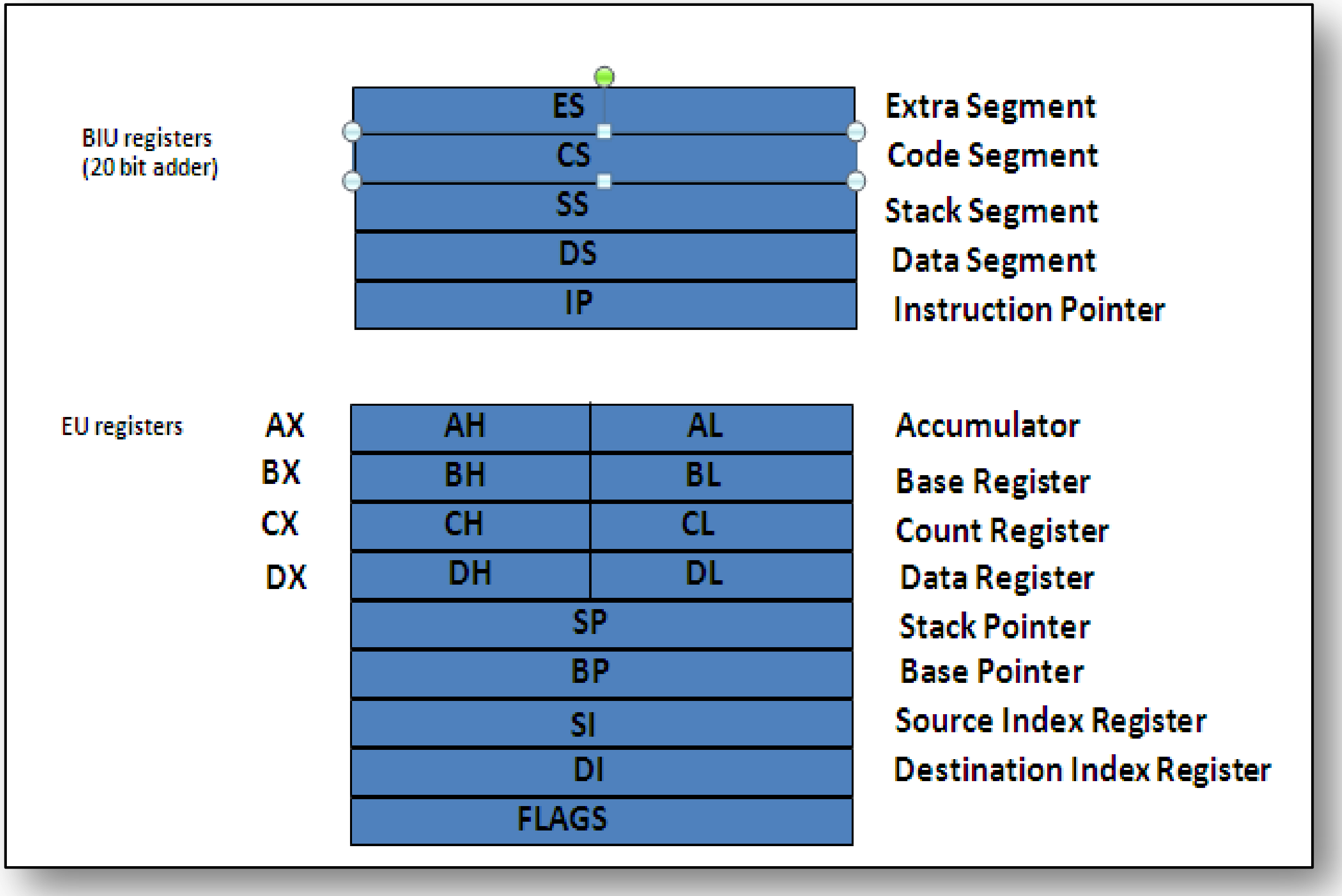
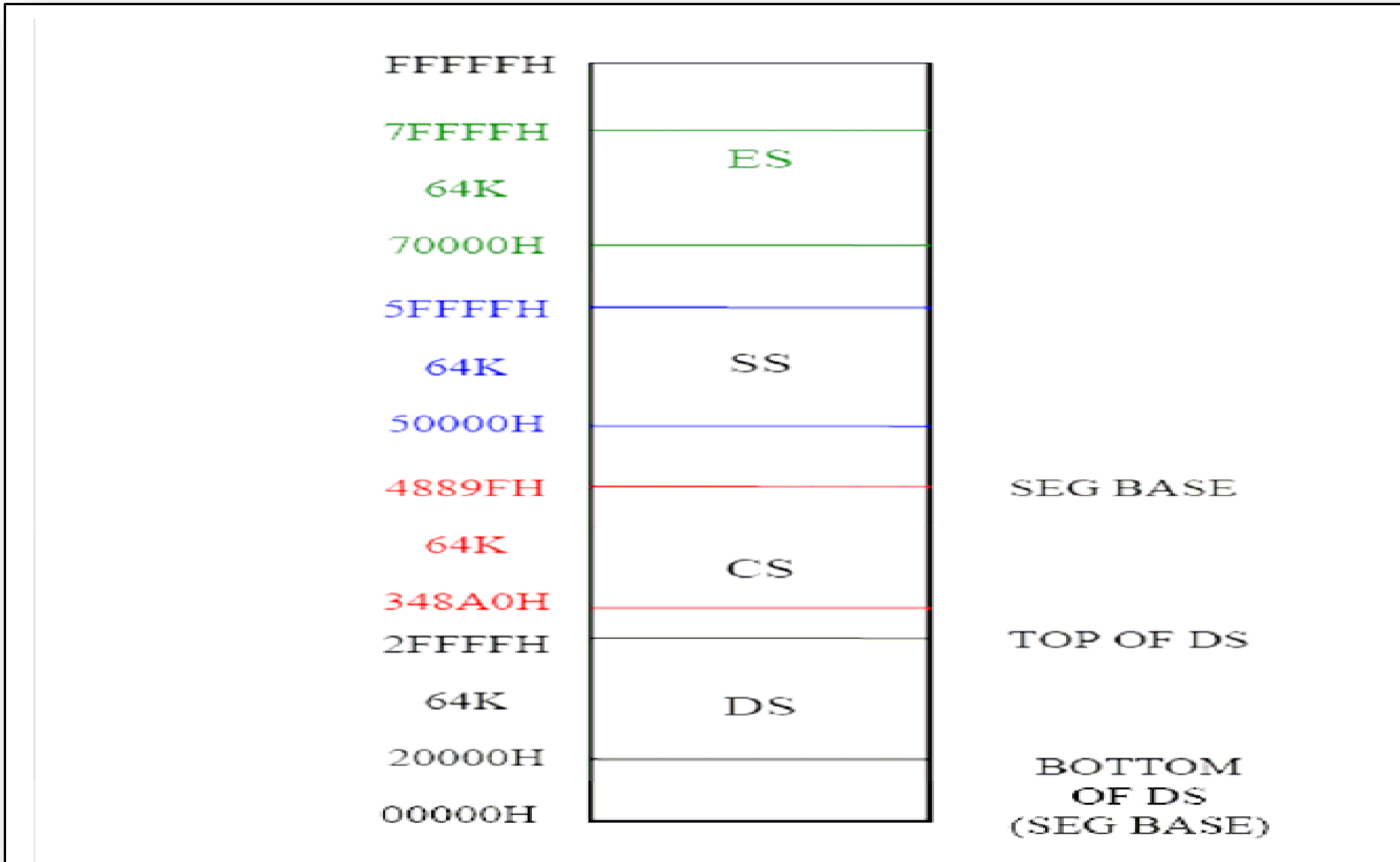
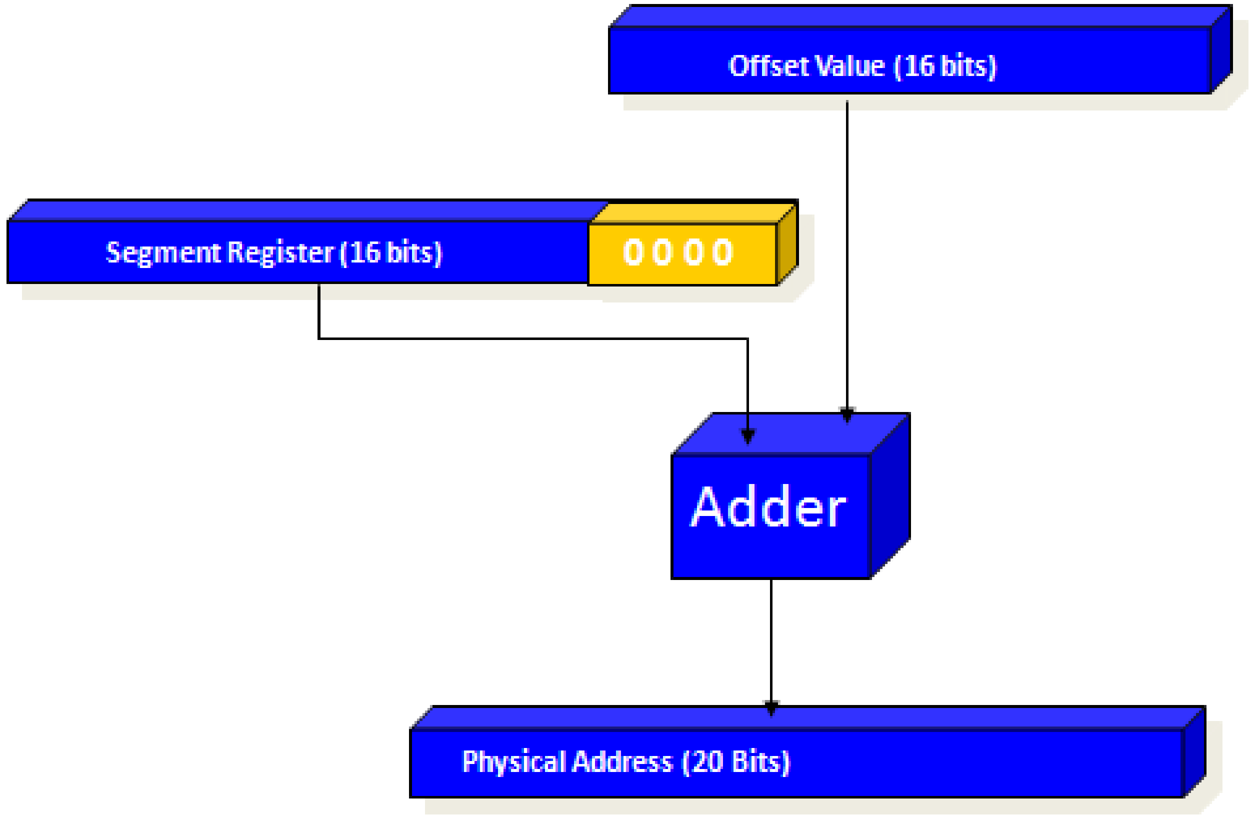


Figure: 8086 Micro Processor Programming Model

Memory Segmentation

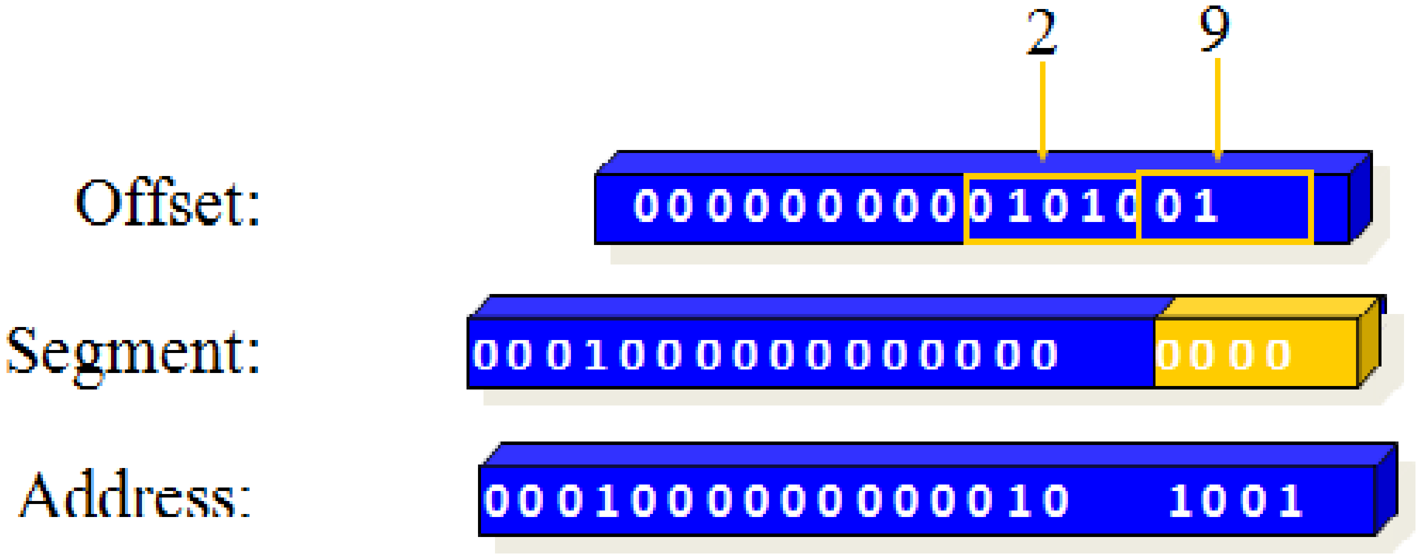


Memory address, physical memory organization



Address calculation

If the data segment starts at location 1000h and a data reference contains the address 29h where is the actual data?



Generation of 20 bit physical address

The 20-bit Physical address is often represented as:

Segment Base : Offset OR CS : IP

CS 3 4 8 0 0 →Implied Zero (from shft Left)

+IP 1 2 3 4

----

3 5 A3 4 H

Signal Description of 8086

- The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.
- The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).
- The 8086 signals can be categorized in three groups. The first are the signal having common functions in minimum as well as maximum mode.
- The second are the signals which have special functions for minimum mode and third are the signals having special functions for maximum mode.

signal descriptions are common for both modes

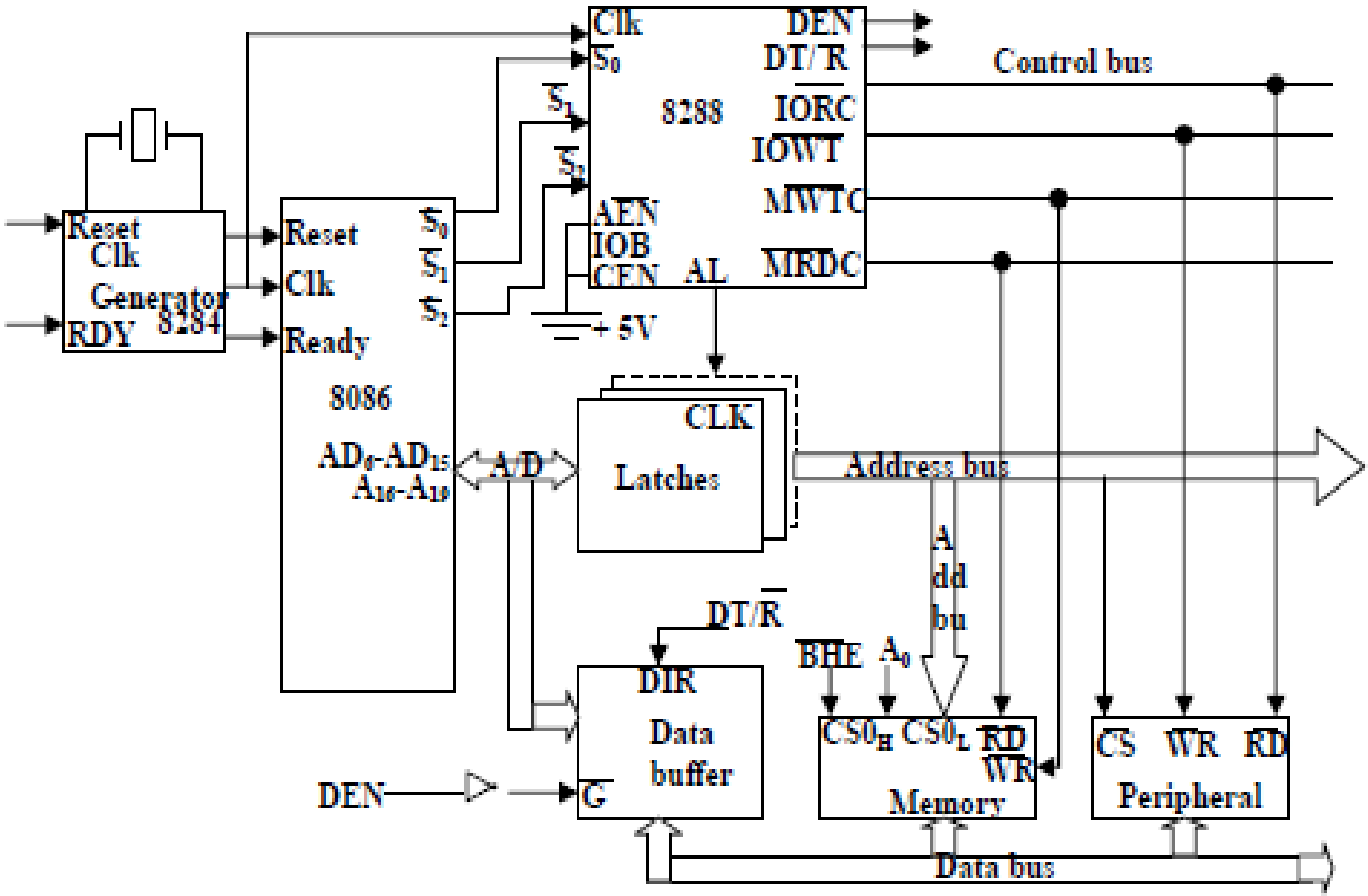
- AD15-AD0: These are the time multiplexed memory I/O address and data lines.
- Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4.
- These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.
- A19/S6,A18/S5,A17/S4,A16/S3: These are the time multiplexed address and status lines.
- During T1 these are the most significant address lines for memory operations.
- During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2,T3,Tw and T4.
- The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.
- The S4 and S3 combined to indicate which segment register is presently being used for memory accesses as in below fig.

S_4	S_3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

8086 OPERATION's

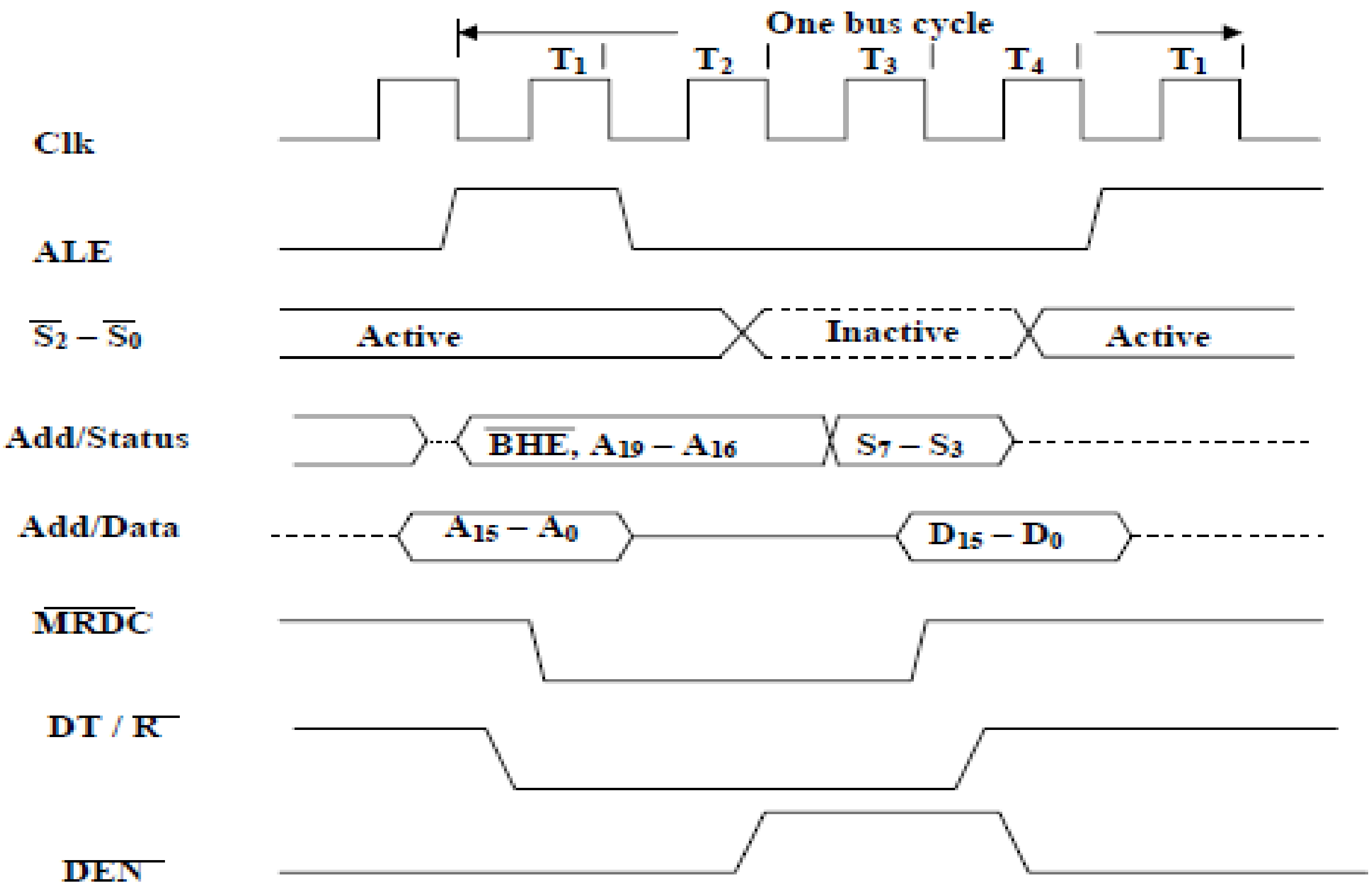
- It contains two modes of operation
  - i) Maximum mode of operation
  - ii) Minimum mode of operation

Maximum mode operation of 8086



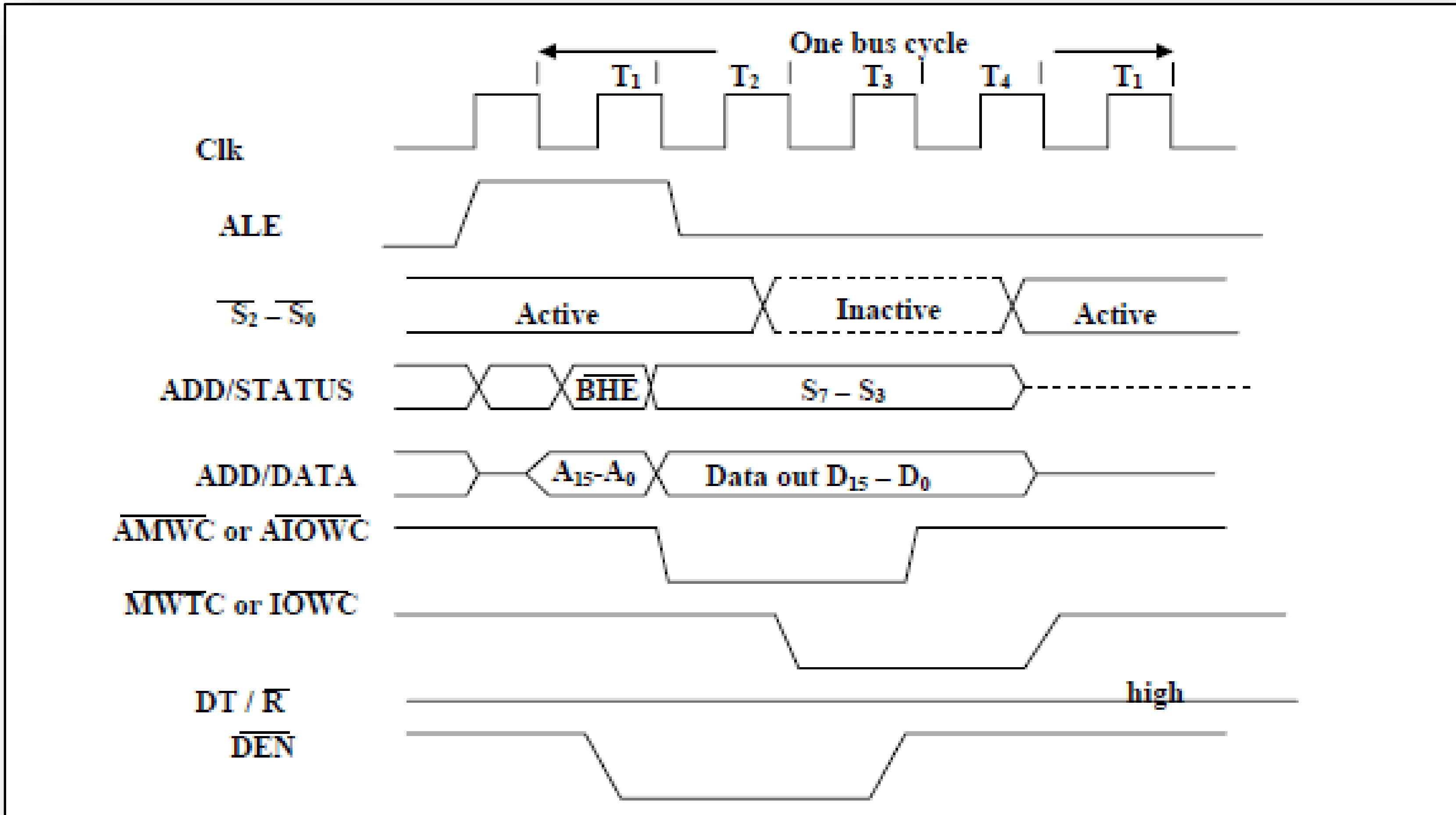
Maximum Mode 8086 System.

Memory read timing in maximum mode



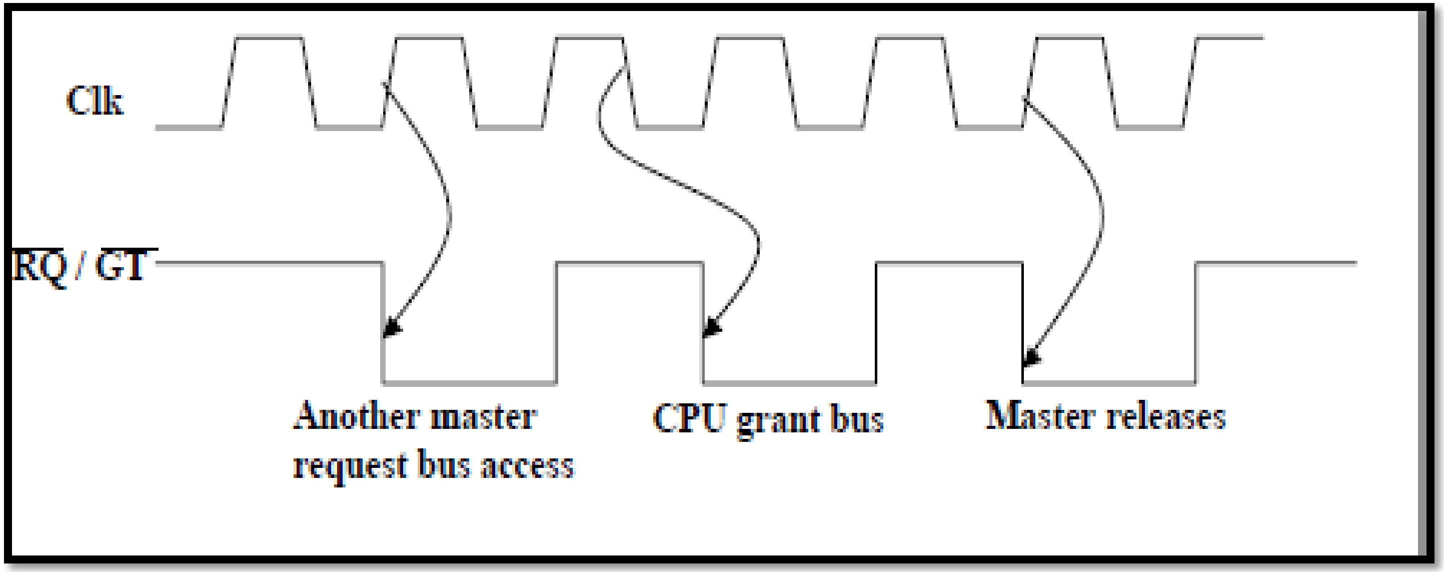
Memory Read Timing in Maximum Mode

Memory write timing in maximum mode

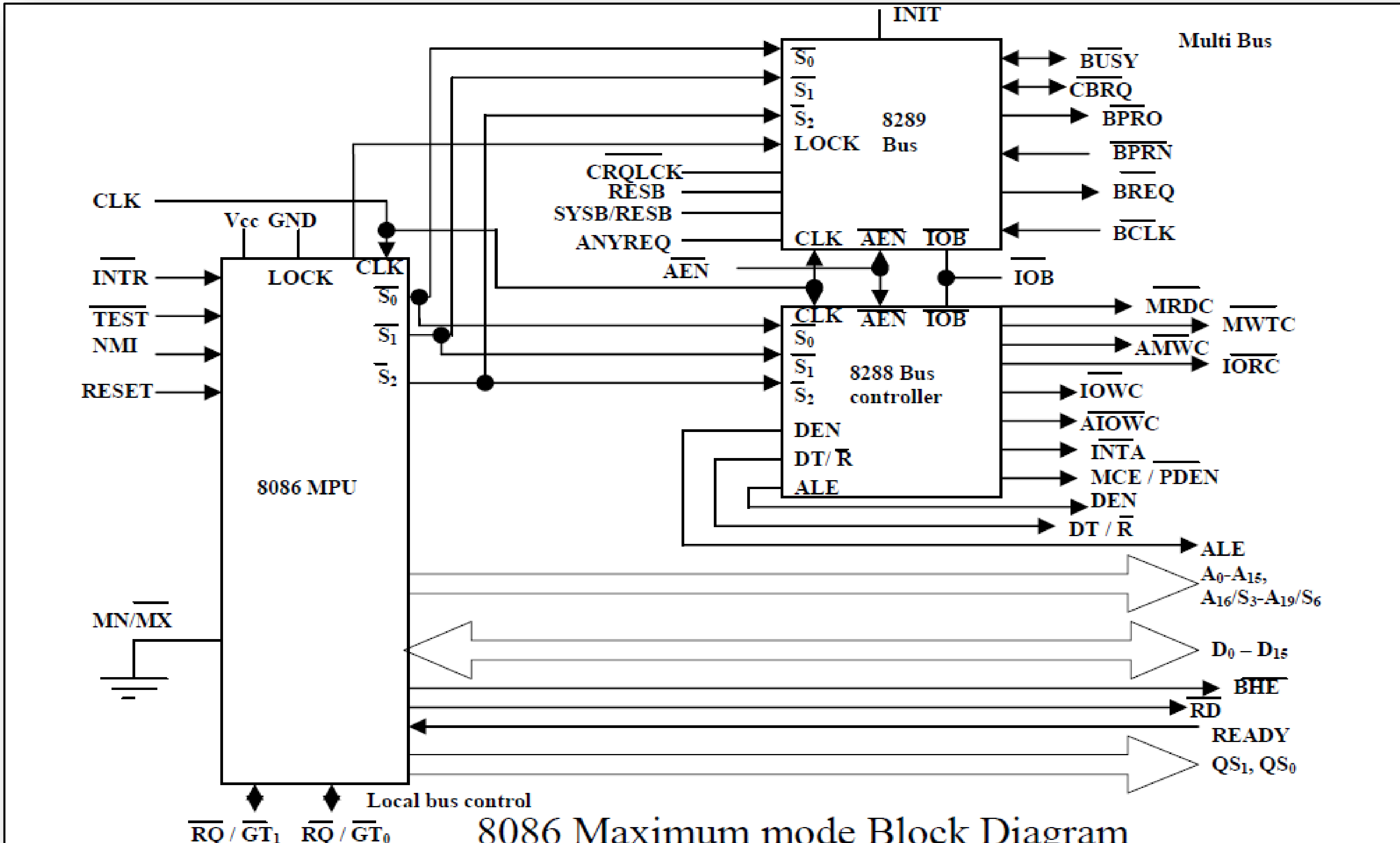


memory write timing in maximum mode

$\overline{RQ}/\overline{GT}$ Timings in Maximum Mode.



Minimum mode of operation



write cycle timing diagram for minimum mode

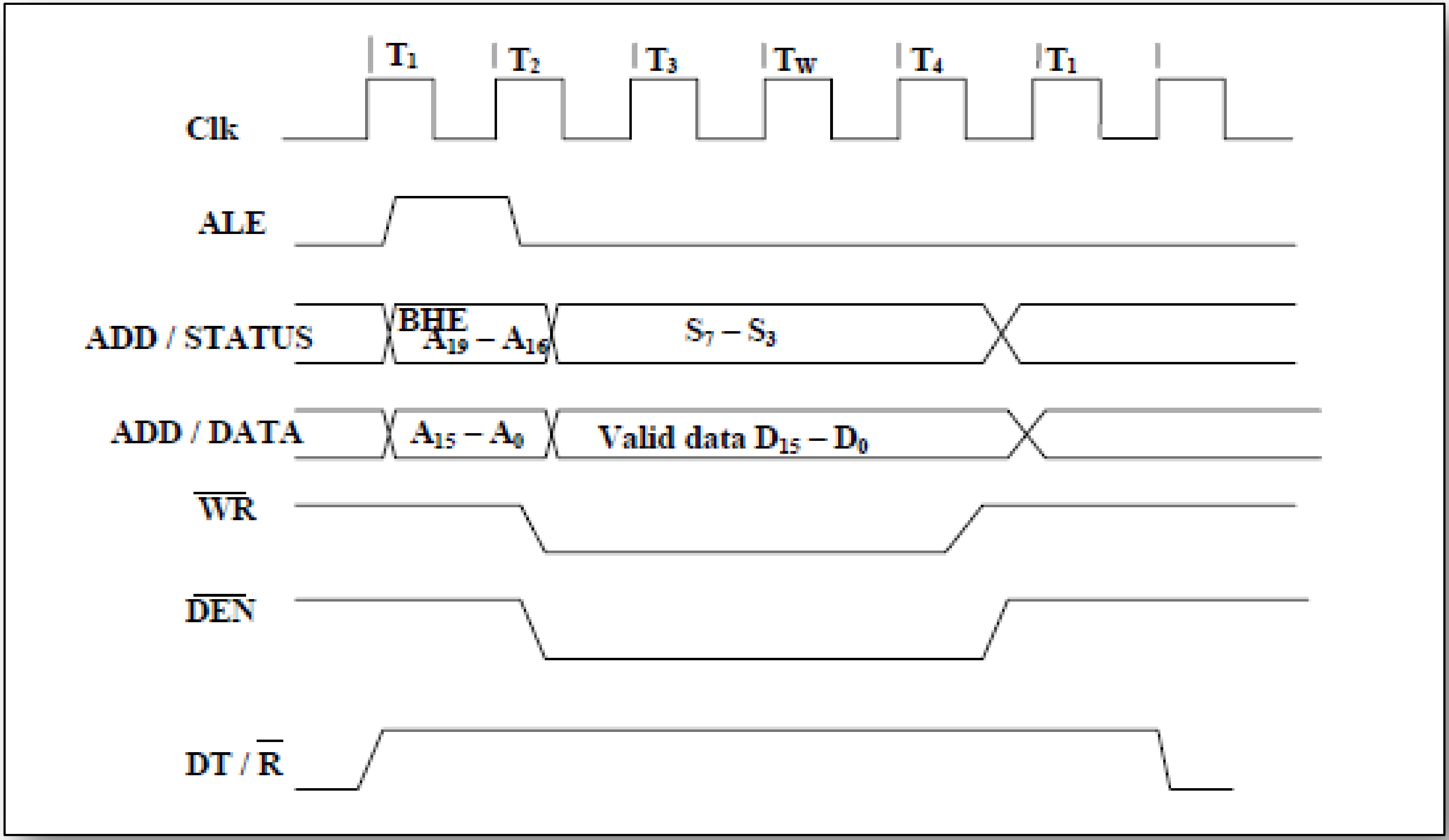
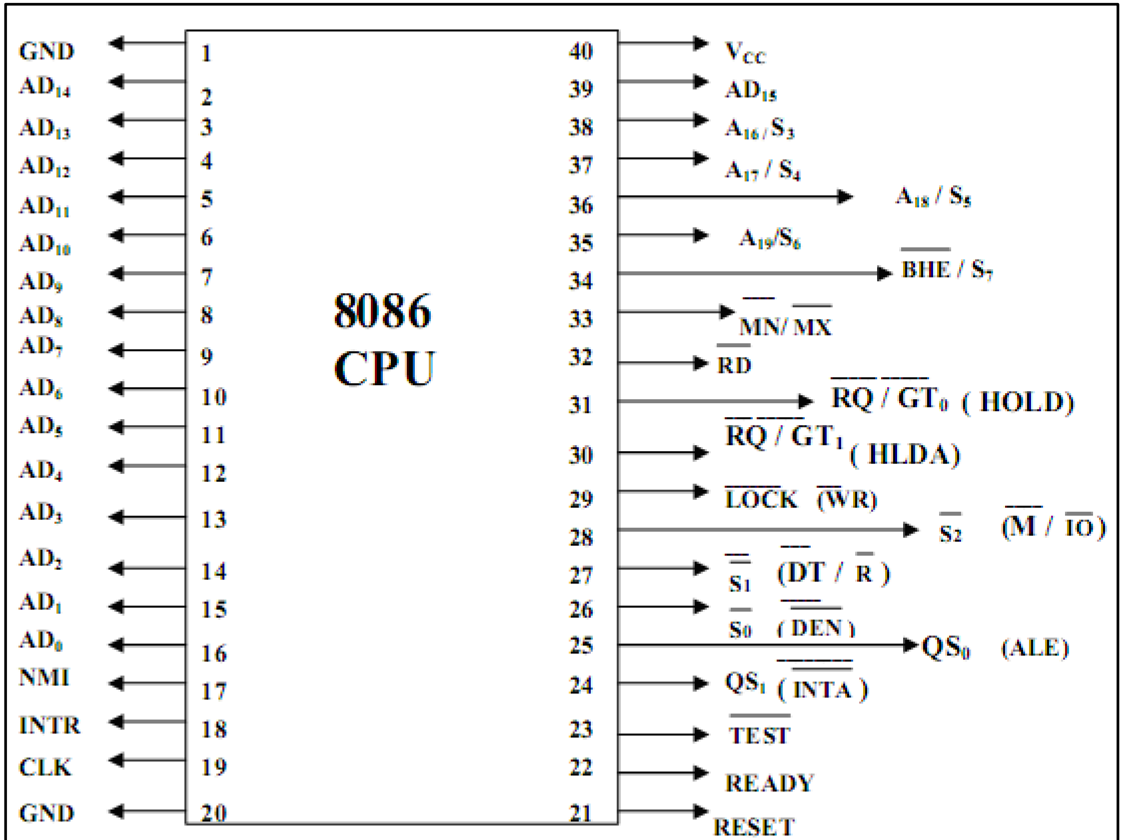


Figure: write cycle timing diagram for minimum mode

8086 Pin diagram



Pin Diagram of 8086

Interrupts of 8086

- The processor has the following interrupts:

- **INTR** is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.

- When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.

- **NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority then the maskable interrupt.
- **Software interrupts** can be caused by:
- INT instruction - breakpoint interrupt. This is a type 3 interrupt.
- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow
- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- **Processor exceptions**: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7).
- Software interrupt processing is the same as for the hardware interrupts.

Assembly Language Programming Fundamentals

Introduction To Programming Languages

- Machine Languages -“natural language” of a computer

- Low Level Languages-In low level language, instructions are coded using mnemonics

- High Level Languages

Format of Assembly Language Instructions

[Label] Operation [Operands] [; Comment]

- **Example: Examples of instructions with varying numbers of fields.**
- [Label] Operation [Operands] [; Comment]

  L1: cmp bx, cx ; Compare bx with cx ***all fields present***

  add ax, 25 ***operation and 2 operands***

  inc bx ***operation and 1 operand***

  ret ***operation field only***

  ; Comment: whatever you wish !! ***comment field only***

Program syntax

Type 1(MASM)	TYPE 2(MASM)	Kit

.model small
.data
Mes db 'HAI $'
N1 db 20h
N2 db 30h
.code
Start:
Mov ax,@data
Mov ds,ax
Mov ax,N1
Mov bx,N2
Add ax,bx
Int 3
End start
	Assume CS:code segment, DS:Data segment
	DATA SEGMENT
	Mes db 'HAI$'
	N1 db 20h
	N2 db 30h
	DATA ENDS
	
	CODE SEGMENT
	Start:
	Mov ax,data
	Mov ds,ax
	Mov ax,N1
	Mov bx,N2
	Add ax,bx
	Int 3
	CODE ENDS
	End start
		Mov ax,20
		Mov bx,30
		Add ax,bx
		Int 3

Addressing Modes of 8086

- The addressing mode describes the types of operands and the way they are accessed for executing an instruction. According to the flow of instruction execution, the instructions may be categorized as

    1.Sequential control flow instructions and

    2. Control transfer instructions.

Addressing Modes of 8086(Contd…)

- Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example the arithmetic, logic, data transfer and processor control instructions are Sequential control flow instructions.

- The control transfer instructions on the other hand transfer control to some predefined address or the address somehow specified in the instruction, after their execution. For example INT, CALL, RET & JUMP instructions fall under this category.

- The addressing modes for Sequential and control flow instructions are explained as follows.

1. Immediate addressing mode:

In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

Example: MOV AX, 0005H.

In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

2. Direct addressing mode:

In the direct addressing mode, a 16-bit memory address (offset) directly specified in the instruction as a part of it.

Example: MOV AX, [5000H].

3. Register addressing mode:

In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

Example: MOV BX, AX

4. Register indirect addressing mode:

Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode.

In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES.

Example: MOV AX, [BX].

5. Indexed addressing mode:

In this addressing mode, offset of the operand is stored one of the index registers. DS & ES are the default segments for index registers SI & DI respectively.

Example: MOV AX, [SI]

Here, data is available at an offset address stored in SI in DS.

6. Register relative addressing mode:

In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the register BX, BP, SI & DI in the default (either in DS & ES) segment.

Example: MOV AX, 50H [BX]

7. Based indexed addressing mode:

The effective address of data is formed in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Example: MOV AX, [BX][SI]

8. Relative based indexed:

The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any of the base registers (BX or BP) and any one of the index registers, in a default segment.

Example: MOV AX, 50H [BX] [SI]

Addressing Modes for control transfer instructions:

1. Intersegment
    - Intersegment direct
    - Intersegment indirect
2. Intrasegment
    - Intrasegment direct
    - Intrasegment indirect

1. Intersegment direct:

In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.

Example: JMP 5000H, 2000H;

Jump to effective address 2000H in segment 5000H.

2. Intersegment indirect:

In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes, i.e. IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

Example: JMP [2000H].

Jump to an address in the other segment specified at effective address 2000H in DS.

3. Intrasegment direct mode:

In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfers instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer.

Example: JMP SHORT LABEL.

4. Intrasegment indirect mode:

In this mode, the displacement to which the control is to be transferred is in the same segment in which the control transfer instruction lies, but it is passed to the instruction directly. Here, the branch address is found as the content of a register or a memory location.

This addressing mode may be used in unconditional branch instructions.

Example: JMP [BX]; Jump to effective address stored in BX

INSTRUCTION SET OF 8086

Classified into 10 categories:

1] Data Transfer

2] Arithmetic

3] Bit manipulation instructions

4] string

5]Iteration Control Instructions

6] program execution transfer instructions

7] Interrupt Control

8] high level language interface instructions

9] processor control instructions

10] External hardware instructions

Data Transfer instructions

- These instructions are used to transfer the data from source operand to destination operand. All the store, move, load, exchange, input and output instructions belong to this group.

- Note : Data Transfer Instructions do not affect any flags

Data Transfer Instructions

1] **MOV destination, source**

Note : source and destination cannot be memory location. Also source and destination must be same type.

2] **PUSH source : *Copies word on stack.***

3] **POP destination: *Copies word from stack into dest. Reg.***

4] **IN acc, port : *Copies 8 or 16 bit data from port to accumulator.***

a) Fixed Port

b) Variable Port

5] **OUT port, acc**

Data Transfer Instructions Cont…

6] **LES Reg, Mem:** ***Load register and extra segment*** *register with words from memory.*

7] **LDS Reg,Mem:** ***Load register and data segment*** *register with words from memory.*

8] **LEA Reg,Src:** ***load Effective address.*** (Offset is loaded in specified register)

9] **LAHF:** ***Copy lower byte of flag register into AH*** *register.*

10] **SAHF:** ***Copy AH register to lower byte of flag***

Data Transfer Instructions Cont …

11] **XCHG destination, source:** *Exchange contents of source and destination.*

12] **XLAT:** ***Translate a byte in AL.***

This instruction replaces the byte in AL with byte pointed by BX. To point desired byte in look up table instruction adds contains of BX with AL (BX+ AL). Goes to this location and loads into AL.

Arithmetic Instructions: ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	ADD D,S	(S)+(D) → (D) carry → (CF)	ALL
ADC	Add with carry	ADC D,S	(S)+(D)+(CF) → (D) carry → (CF)	ALL
INC	Increment by one	INC D	(D)+1 → (D)	ALL but CY
AAA	ASCII adjust for addition	AAA	If the sum is >9, AH is incremented by 1	AF,CF
DAA	Decimal adjust for addition	DAA	Adjust AL for decimal Packed BCD	ALL

Arithmetic Instructions – SUB, SBB, DEC, AAS, DAS, NEG

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	(D) - (S) → (D) Borrow → (CF)	All
SBB	Subtract with borrow	SBB D,S	(D) - (S) - (CF) → (D)	All
DEC	Decrement by one	DEC D	(D) - 1 → (D)	All but CF
NEG	Negate	NEG D		All
DAS	Decimal adjust for subtraction	DAS	Convert the result in AL to packed decimal format	All
AAS	ASCII adjust for subtraction	AAS	(AL) difference (AH) dec by 1 if borrow	CY,AC

Multiplication and Division

Mnemonic	Meaning	Format	Operation	Flags Affected
MUL	Multiply (unsigned)	MUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
DIV	Division (unsigned)	DIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is FF_{16} in case (1) or $FFFF_{16}$ in case (2), then type 0 interrupt occurs	OF, SF, ZF, AF, PF, CF undefined
IMUL	Integer multiply (signed)	IMUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
IDIV	Integer divide (signed)	IDIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is positive and exceeds $7FFF_{16}$ or if Q is negative and becomes less than 8001_{16}, then type 0 interupt occurs	OF, SF, ZF, AF, PF, CF undefined
AAM	Adjust AL for multiplication	AAM	Q((AL)/10) → (AH) R((AL)/10) → (AL)	SF, ZF, PF OF, AF,CF undefined
AAD	Adjust AX for division	AAD	(AH) · 10 + (AL) → (AL) 00 → (AH)	SF, ZF, PF OF, AF, CF undefined
CBW	Convert byte to word	CBW	(MSB of AL) → (All bits of AH)	None
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)	None

(a)

Source
Reg8
Reg16
Mem8
Mem16

(b)

Multiplication and Division

Multiplication (MUL or IMUL)	Multiplicand	Operand (Multiplier)	Result
Byte*Byte	AL	Register or memory	AX
Word*Word	AX	Register or memory	DX :AX
Dword*Dword	EAX	Register or memory	EAX :EDX

Division (DIV or IDIV)	Dividend	Operand (Divisor)	Quotient: Remainder
Word/Byte	AX	Register or Memory	AL : AH
Dword/Word	DX:AX	Register or Memory	AX : DX
Qword/Dword	EDX: EAX	Register or Memory	EAX : EDX

Bit manipulation instructions

i) Logical Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
AND	Logical AND	AND D,S	$(S) \cdot (D) \rightarrow (D)$	OF, SF, ZF, PF, CF AF undefined
OR	Logical Inclusive OR	OR D,S	$(S)+(D) \rightarrow (D)$	OF, SF, ZF, PF, CF AF undefined
XOR	Logical Exclusive OR	XOR D,S	$(S) \oplus (D) \rightarrow (D)$	OF, SF, ZF, PF, CF AF undefined
NOT	LOGICAL NOT	NOT D	$(\overline{D}) \rightarrow (D)$	None

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

Destination
Register
Memory

Logical Instructions Cont…

CMP dest, source

- CF, ZF and SF are used

Ex. CMP CX,BX

	CF	ZF	SF
➢ CX = BX	0	1	0
➢ CX> BX	0	0	0
➢ CX<BX	1	0	1

ii) Shift and Rotate Instructions

- **SHR/SAL: shift logical left/shift arithmetic left**
- **SHR: shift logical right**
- **SAR: shift arithmetic right**
- **ROL: rotate left**
- **ROR: rotate right**
- **RCL: rotate left through carry**
- **RCR: rotate right through carry**

Rotate Instructions

Mnem -onic	Meaning	Format	Operation	Flags Affected
ROL	Rotate Left	ROL D,Count	Rotate the (D) left by the number of bit positions equal to Count. Each bit shifted out from the left most bit goes back into the rightmost bit position.	CF OF undefined if count $\neq$ 1
ROR	Rotate Right	ROR D,Count	Rotate the (D) right by the number of bit positions equal to Count. Each bit shifted out from the rightmost bit goes back into the leftmost bit position.	CF OF undefined if count $\neq$ 1
RCL	Rotate Left through Carry	RCL D,Count	Same as ROL except carry is attached to (D) for rotation.	CF OF undefined if count $\neq$ 1
RCR	Rotate right through Carry	RCR D,Count	Same as ROR except carry is attached to (D) for rotation.	CF OF undefined if count $\neq$ 1

String?

- An array of bytes or words located in memory

- Supported String Operations
  - Copy (move, load)
  - Search (scan)
  - Store
  - Compare

String Instruction Basics

- Source DS:SI, Destination ES:DI
  - You must ensure DS and ES are correct
  - You must ensure SI and DI are offsets into DS and ES respectively
- Direction Flag (0 = Up, 1 = Down)
  - CLD - Increment addresses (left to right)
  - STD - Decrement addresses (right to left)

STRING CONTROL

1) **MOVS/ MOVSB/ MOVSW**

Dest string name,src string name

This instn moves data byte or word from location in DS to location in ES.

2) **REP / REPE / REPZ / REPNE / REPNZ**

Repeat string instructions until specified conditions exist.

This is prefix a instruction.

STRING CONTROL Contd…

3)**CMPS / CMPSB / CMPSW**
Compare string bytes or string words.

4) **SCAS / SCASB / SCASW**
Scan a string byte or string word.
Compares byte in AL or word in AX. String address is to be loaded in DI.

5) **STOS / STOSB / STOSW**
Store byte or word in a string.
Copies a byte or word in AL or AX to memory location pointed by DI.

6) **LODS / LODSB /LODSW**
Load a byte or word in AL or AX

- Copies byte or word from memory location pointed by SI into AL or AX register.

Iteration control instructions

- These instructions are used to execute a series of instructions for certain number of times.
- LOOP :Loop through a sequence of instructions until CX=0
- LOOPE/LOOPZ : Loop through a sequence of instructions while ZF=1 and CX = 0
- LOOPNE/LOOPNZ : Loop through a sequence of instructions while ZF=0 and CX =0
- JCXZ : jump to specified address if CX=0

Program Execution Transfer instructions

- These instructions are similar to branching or looping instructions. These instructions include conditional & unconditional jump or loop instructions.

- **Unconditional transfer instructions**
- CALL : Call a procedure, save return address on stack
- RET : Return from procedure to the main program.
- JMP : Goto specified address to get next instruction

Conditional transfer instructions

- **JA/JNBE** :Jump if above / jump if not below or equal
- **JAE/JNB** : Jump if above /jump if not below
- **JBE/JNA** : Jump if below or equal/ Jump if not above
- **JC** : jump if carry flag CF=1
- **JE/JZ** **:** jump if equal/jump if zero flag ZF=1
- **JG/JNLE** : Jump if greater/ jump if not less than or equal
- **JGE/JNL** : jump if greater than or equal/ jump if not less than
- **JL/JNGE** : jump if less than/ jump if not greater than or equal
- **JLE/JNG** : jump if less than or equal/ jump if not greater than
- **JNC** : jump if no carry (CF=0)
- **JNE/JNZ** : jump if not equal/ jump if not zero(ZF=0)
- **JNO** : jump if no overflow(OF=0)
- **JNP/JPO** : jump if not parity/ jump if parity odd(PF=0)
- **JNS** : jump if not sign(SF=0)
- **JO** : jump if overflow flag(OF=1)
- **JP/JPE** : jump if parity/jump if parity even(PF=1)
- **JS** : jump if sign(SF=1)

Interrupt instructions

- INT : Interrupt program execution, call service procedure
- INTO : Interrupt program execution if OF=1
- IRET : Return from interrupt service procedure to main program

High level language interface instructions

- ENTER : enter procedure
- LEAVE :Leave procedure
- BOUND :Check if effective address within specified array bounds

Processor control instructions

- Flag set/clear instructions
- STC : Set carry flag CF to 1
- CLC : Clear carry flag CF to 0
- CMC : Complement the state of the carry flag CF
- STD : Set direction flag DF to 1 (decrement string pointers)
- CLD : Clear direction flag DF to 0
- STI : Set interrupt enable flag to 1(enable INTR input)
- CLI : Clear interrupt enable Flag to 0 (disable INTR input)

External Hardware synchronization instructions

- HLT : Halt (do nothing) until interrupt or reset
- WAIT : Wait (Do nothing) until signal on the test pin is low.
- ESC : Escape to external coprocessor such as 8087 or 8089.
- LOCK : An instruction prefix. Prevents another processor from taking the bus while the adjacent instruction executes.

Assembler Directives

- Assembler Directives are directions to the assembler.
- Assembler directives are the commands to the assembler that direct the assembly process.
- They indicate how an operand is treated by the assembler and how assembler handles the program.
- They also direct the assembler how program and data should be arranged in the memory.

List of Assembler Directives

ASSUME	DB	DW	DD	DQ
DT	END	ENDP	ENDS	EQU
EVEN	EXTRN	GLOBAL	GROUP	INCLUDE
LABEL	LENGTH	NAME	OFFSET	ORG
PROC	PTR	SEGMENT	SHORT	TYPE

MACROS

- A macro is a group of repetitive instructions in a program which are codified only once and can be used as many times as necessary.

- Macro with in a macro is a nested MACRO

- A macro can be defined anywhere in program using the directives MACRO and ENDM

- Syntax of macro:

```
Read  MACRO
    mov ah,02h
    int 21h
ENDM

Display MACRO
      mov dl,al
    Mov ah,01h
    int 21h
ENDM
```

Passing parameters to a macro

- Display MACRO INF

```
Display MACRO INF
        mov dx, offset  inf
        mov ah,09h
        int 21h
  ENDM
```

The parameter MSG can be replaced by inf1 or inf2 while calling…

Calling macro:

```
        DISPLAY INF1
        DISPLAY INF2
INF1 db “hai$”
 INF2 db “Hello, How are you..? $”
```

Here parameter is INF

Procedures Vs Macros

Procedures	Macros
Accessed by CALL and RET mechanism during program execution	Accessed by name given to macro when defined during assembly
Machine code for instructions only put in memory once	Machine code generated for instructions each time called
Parameters are passed in registers, memory locations or stack	Parameters passed as part of statement which calls macro
Procedures uses stack	Macro does not utilize stack
A procedure can be defined anywhere in program using the directives PROC and ENDP	A macro can be defined anywhere in program using the directives MACRO and ENDM
Procedures takes huge memory for CALL(3 bytes each time CALL is used) instruction	Length of code is very huge if macro’s are called for more number of times

UINT III
I/O INTERFACE

8255-PROGRAMMABLE PERIPHERAL INTERFACE

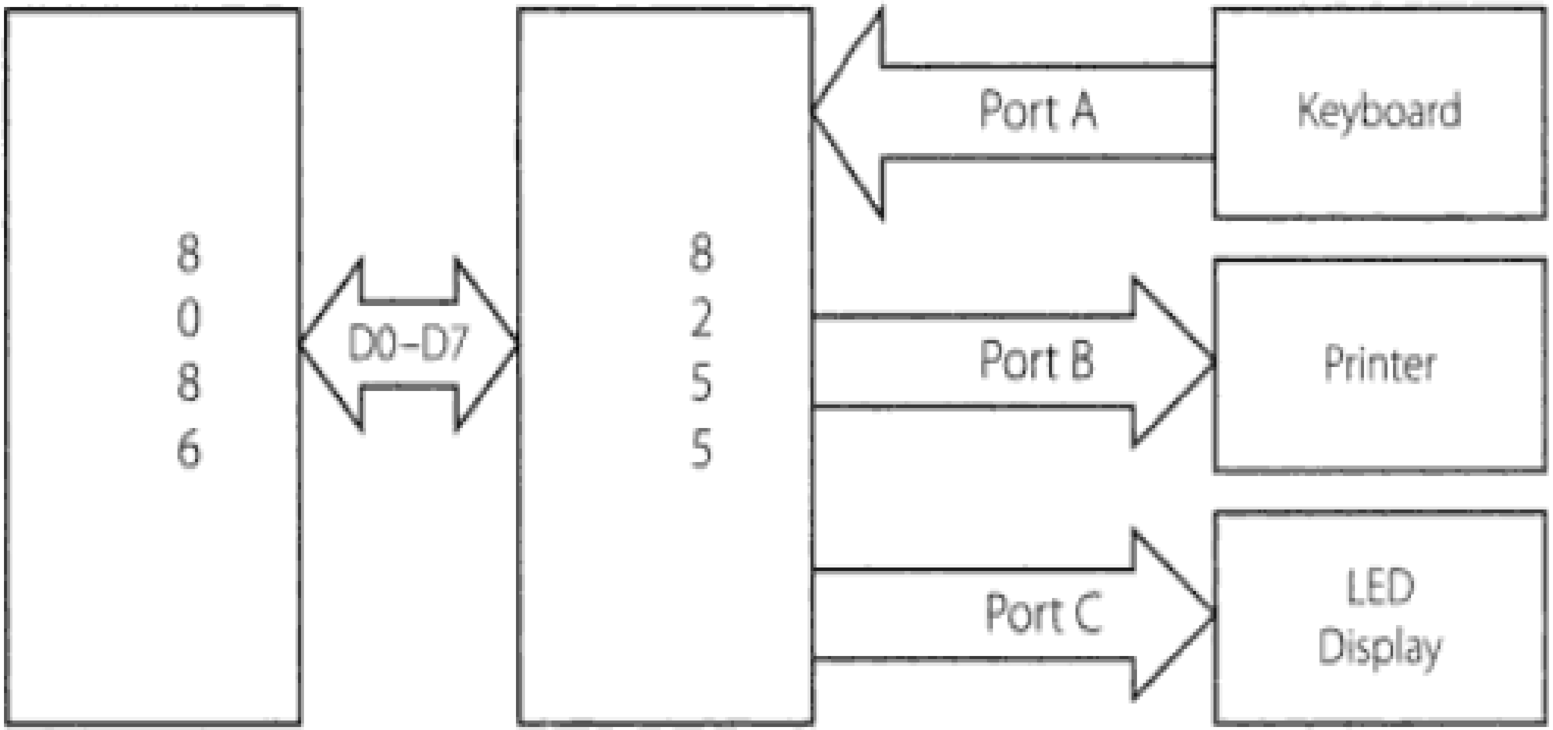
Need of 8255 for I/O interfacing

There are two reasons for using 8255 between 8086 and I/O devices.

1) To achieve Speed compatibility between high speed microprocessor and slow I/O devices.

2) Reducing hardware complexity by interfacing the I/O devices through program.

Purpose of 8255



The connections between an 8086, 8255 and three peripherals

8255

- It has 24 input/output lines

- 24 lines divided into 3 ports
  - Port A(8 bit)
  - Port B(8 bit)
  - Port C upper(4 bit), Port C Lower (4 bit)

  - All the above 3 ports can act as input or output ports

Block Diagram

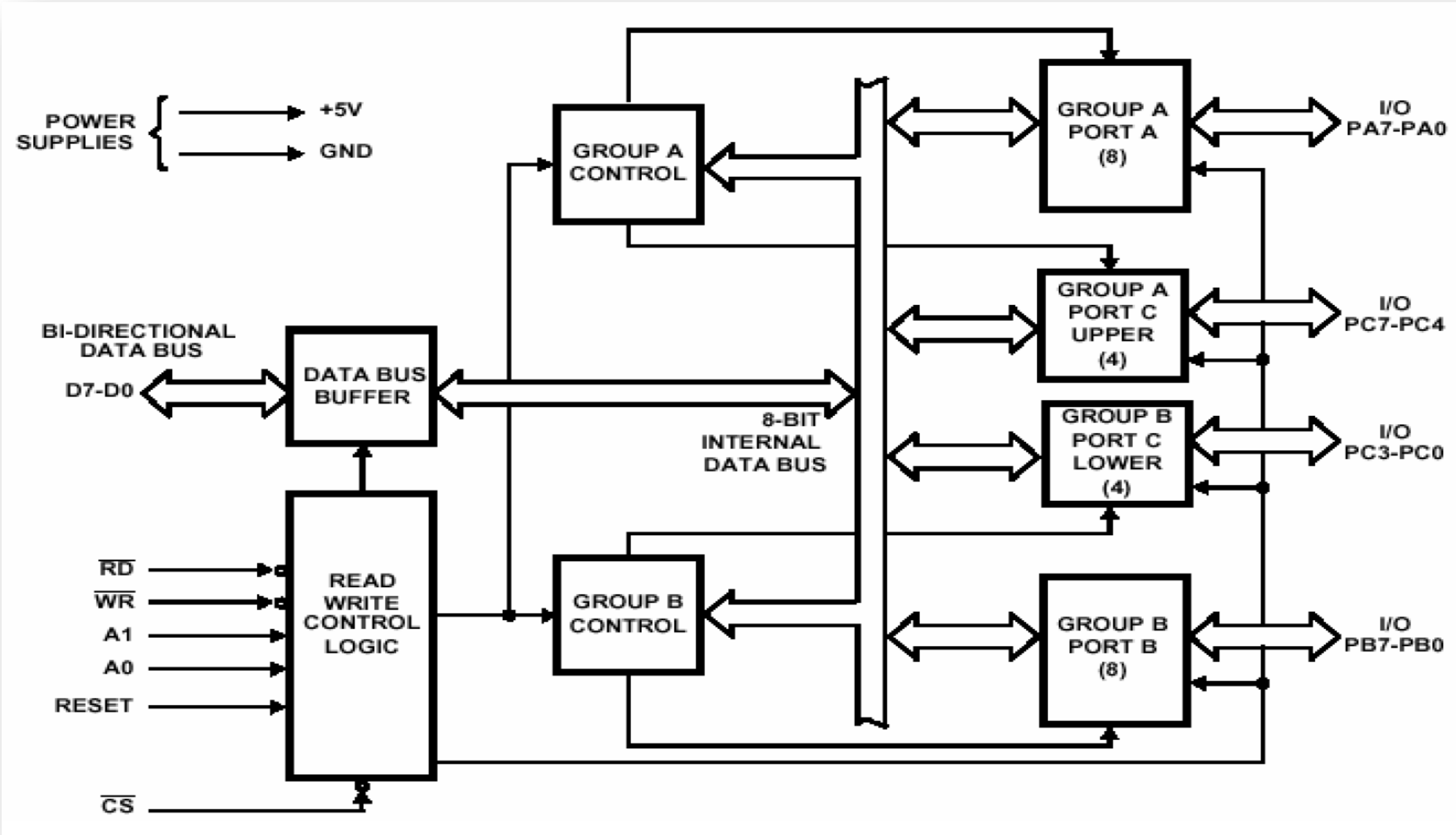


Figure: Block Diagram of 8255(PIC)

Data Bus buffer

- It is a 8-bit bidirectional Data bus.
- Used to interface between 8255 data bus with system bus.
- The internal data bus and Outer pins D_0-D_7 pins are connected in internally.
- The direction of data buffer is decided by Read/Control Logic.

Read/Write Control Logic

- This is getting the input signals from control bus and Address bus

- Control signal are RD and $\overline{WR}$.

- Address signals are A0, A1, and $\overline{CS}$.

- 8255 operation is enabled or disabled by $\overline{CS}$.

Group A and Group B control:

- Group A and B get the Control Signal from CPU and send the command to the individual control blocks.
- Group A send the control signal to port A and Port C (Upper) PC7-PC4.
- Group B send the control signal to port B and Port C (Lower) PC3-PC0.

PORT A:

- This is a 8-bit buffered I/O latch.
- It can be programmed by mode 0 , mode 1, mode 2 .

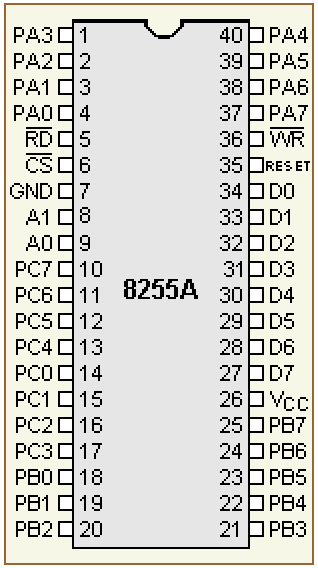
PORT B:

- This is a 8-bit buffer I/O latch.
- It can be programmed by mode 0 and mode 1.

PORT C:

- This is a 8-bit Unlatched buffer Input and an Output latch.
- It is spitted into two parts.
- It can be programmed by bit set/reset operation.

8255A pins



8255A
PA3 1
PA2 2
PA1 3
PA0 4
RD 5
CS 6
GND 7
A1 8
A0 9
PC7 10
PC6 11
PC5 12
PC4 13
PC0 14
PC1 15
PC2 16
PC3 17
PB0 18
PB1 19
PB2 20
40 PA4
39 PA5
38 PA6
37 PA7
36 WR
35 RESET
34 D0
33 D1
32 D2
31 D3
30 D4
29 D5
28 D6
27 D7
26 VCC
25 PB7
24 PB6
23 PB5
22 PB4
21 PB3

Pin Description

- **PA7-PA0**: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

- **PC7-PC4:** Upper nibble of port C lines. They may act as either output latches or input buffers lines.

  This port also can be used for generation of handshake lines in mode 1 or mode 2.

- **PC3-PC0**: These are the lower port C lines, other details are the same as PC7-PC4 lines.

- **PB0-PB7**: These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

Pin Description(Contd…)

- **RD** : This is the input line driven by the microprocessor and should be low to indicate read operation to8255.

- **WR** : This is an input line driven by the microprocessor. A low on this line indicates write operation.

- **CS** : This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal are neglected.

- **A1-A0** : These are the address input lines and are driven by the microprocessor.

- **RESET** : The 8255 is placed into its reset state if this input line is a logical 1. All peripheral ports are set to the input mode.

Operating Modes

BIT SET/RESET MODE

- The PORT C can be Set or Reset by sending OUT instruction to the CONTROL registers.

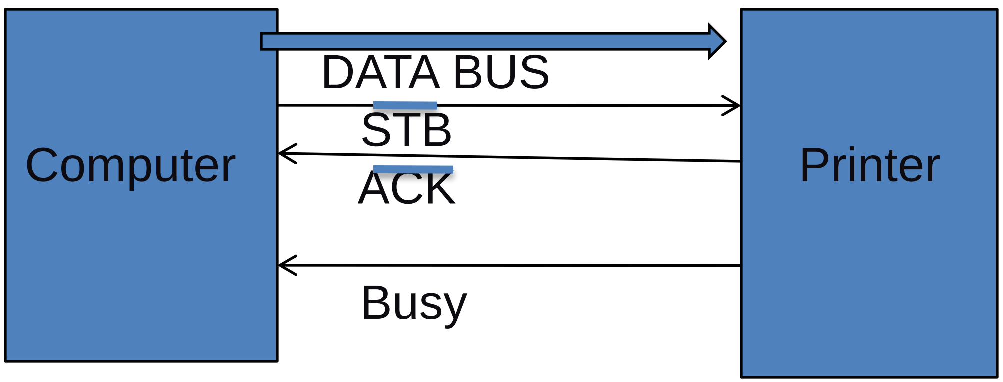
I/O MODES:

MODE 0(Simple input / Output):

- In this mode , port A, port B and port C is used as individually (Simply).
- Ports do not have Handshake or interrupt capability.

MODE 1 :(Input/output with Hand shake)

- In this mode, input or output is transferred by hand shaking Signals.



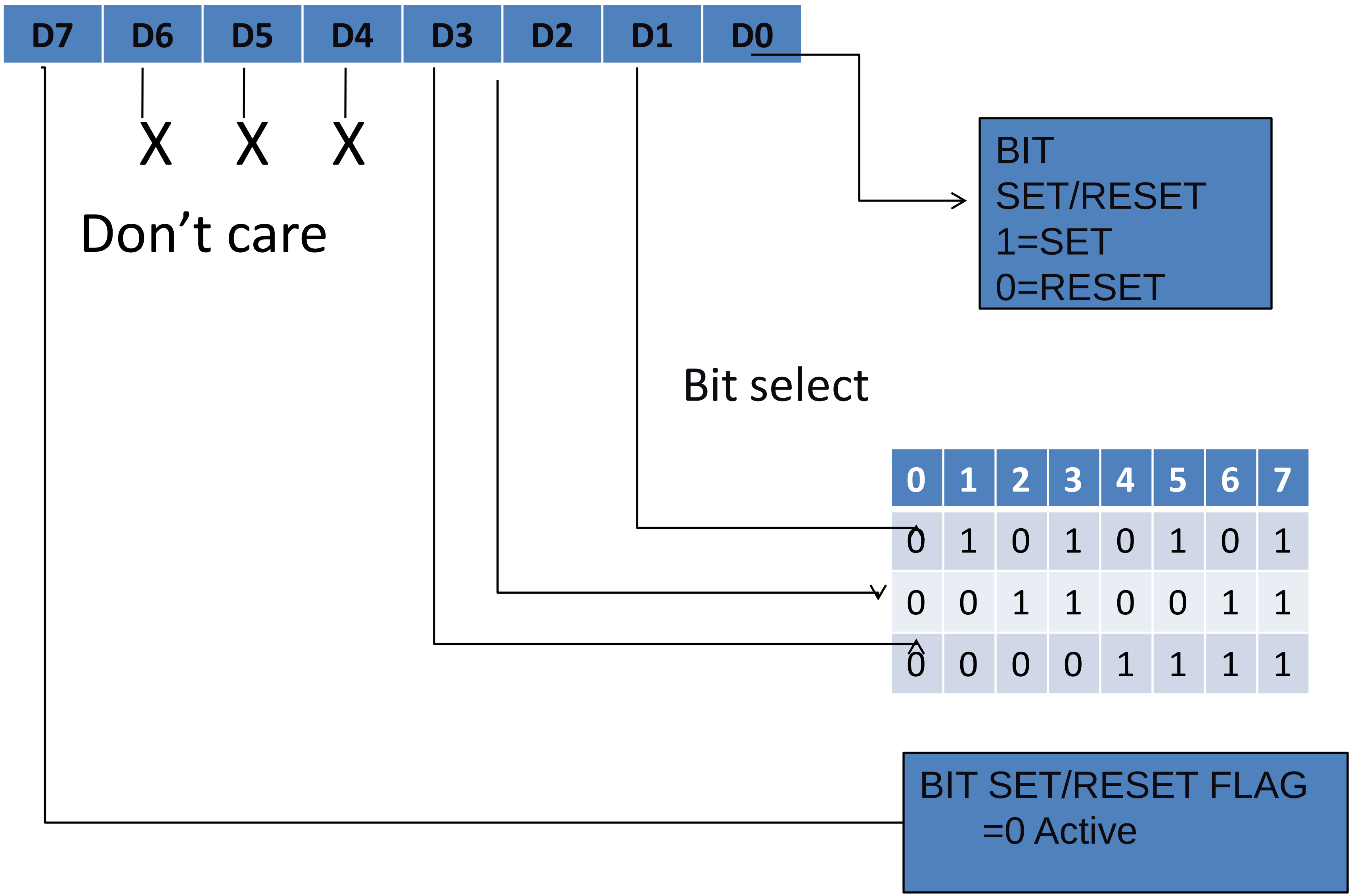
- Handshaking signals is used to transfer data between whose data transfer is not same.

MODE 2:bi-directional I/O data transfer:

- This mode allows bidirectional data transfer over a single 8-bit data bus using handshake signals.
- This feature is possible only Group A
- Port A is working as 8-biy bidirectional.
- PC3-PC7 is used for handshaking purpose.
- The data is sent by CPU through this port , when the peripheral request it.

FOR BIT SET/RESET MODE:

- This is bit set/reset control word format.



- PC0-PC7 is set or reset as per the status of D0.

- A BSR word is written for each bit

Example:

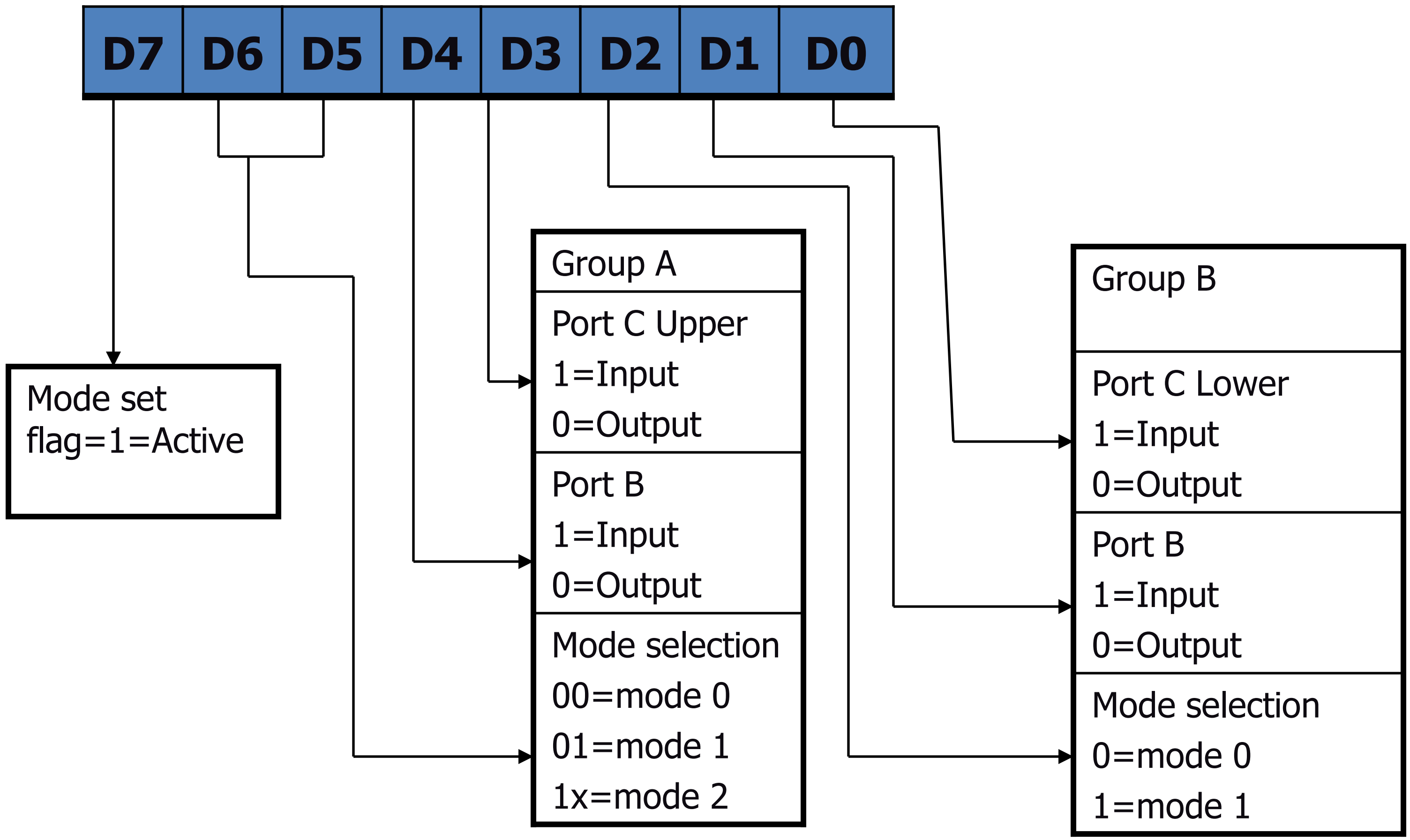
- PC3 is Set then control register will be 0XXX0111.

- PC4 is Reset then control register will be 0XXX01000.

- X is a don’t care.

FOR I/O MODE

The mode format for I/O as shown in figure



- The control word for both modes is same.
- Bit D7 is used for specifying whether word loaded in to Bit set/reset mode or Mode definition word.
- D7=1=Mode definition mode.
- D7=0=Bit set/Reset mode.

8255 Operations

- lines A1-A0 with RD, WR and CS form the following operations for 8255.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Input (Read) cycle
0	1	0	0	0	Port A to Data bus
0	1	0	0	1	Port B to Data bus
0	1	0	1	0	Port C to Data bus
0	1	0	1	1	CWR to Data bus

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Output (Write) cycle
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Function
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

Control Word Register

Programming 8255

- 8255 has three operation modes: *mode 0, mode 1, and mode 2*

Mode 0 - Simple Input or Output mode

Mode 1 - Input or Output with Handshake mode

Mode 2 - Bidirectional Data Transfer mode

Mode 0 - Simple Input or Output

- In this mode, ports **A**, **B** are used as **two simple 8-bit I/O** ports & port **C** as **two independent 4-bit ports.**

- **Each port** can be programmed to function as simply an input port or an output port.

- The **input/output features** in Mode 0 are as follows.

1. Outputs are latched.

2. Inputs are not latched.

3. Ports don't have handshake or interrupt capability.

Handshaking

- Many I/O devices accept or release information slower than the microprocessor.
- A method of I/O control called **handshaking** or **polling**, synchronizes the I/O device with the microprocessor.
- An example is a parallel printer that prints a few hundred characters per second (CPS).

Mode 1 - Input or Output with Handshake

- In this mode, **handshake signals are exchanged** between the MPU and peripherals prior to data transfer.

- The **features** of the mode include the following:

  1. Two ports (**A** and **B**) function as 8-bit I/O ports.
     They can be configured as either as input or output ports.

  2. Each port uses **three lines from port C as handshake signals**.
     The remaining two lines of Port C can be used for simple I/O
     operations.

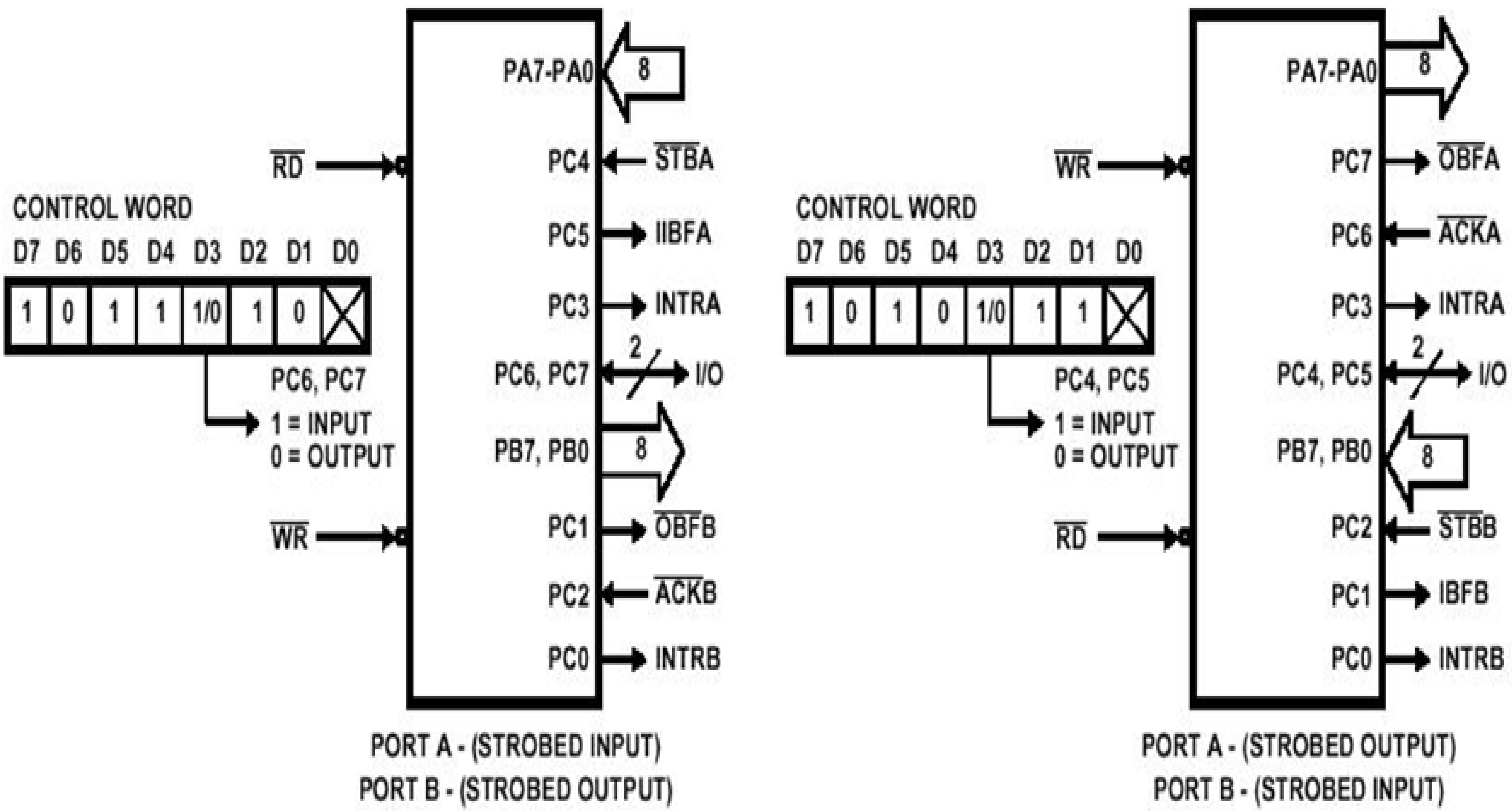
  3. Input and Output data are latched.

  4. Interrupt logic is supported.

Example:

- The computer send the data to the printer large speed compared to the printer.
- When computer send the data according to the printer speed at the time only, printer can accept.
- If printer is not ready to accept the data then after sending the data bus , computer uses another handshaking signal to tell printer that valid data is available on the data bus.
- Each port uses three lines from port C as handshake signals

Mode 1 - Input or Output with Handshake



Combinations of Mode 1: Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

82C55: Mode 1 Strobed Input

- **STB** : The strobe input loads data into the port latch on a 0-to-1 transition.

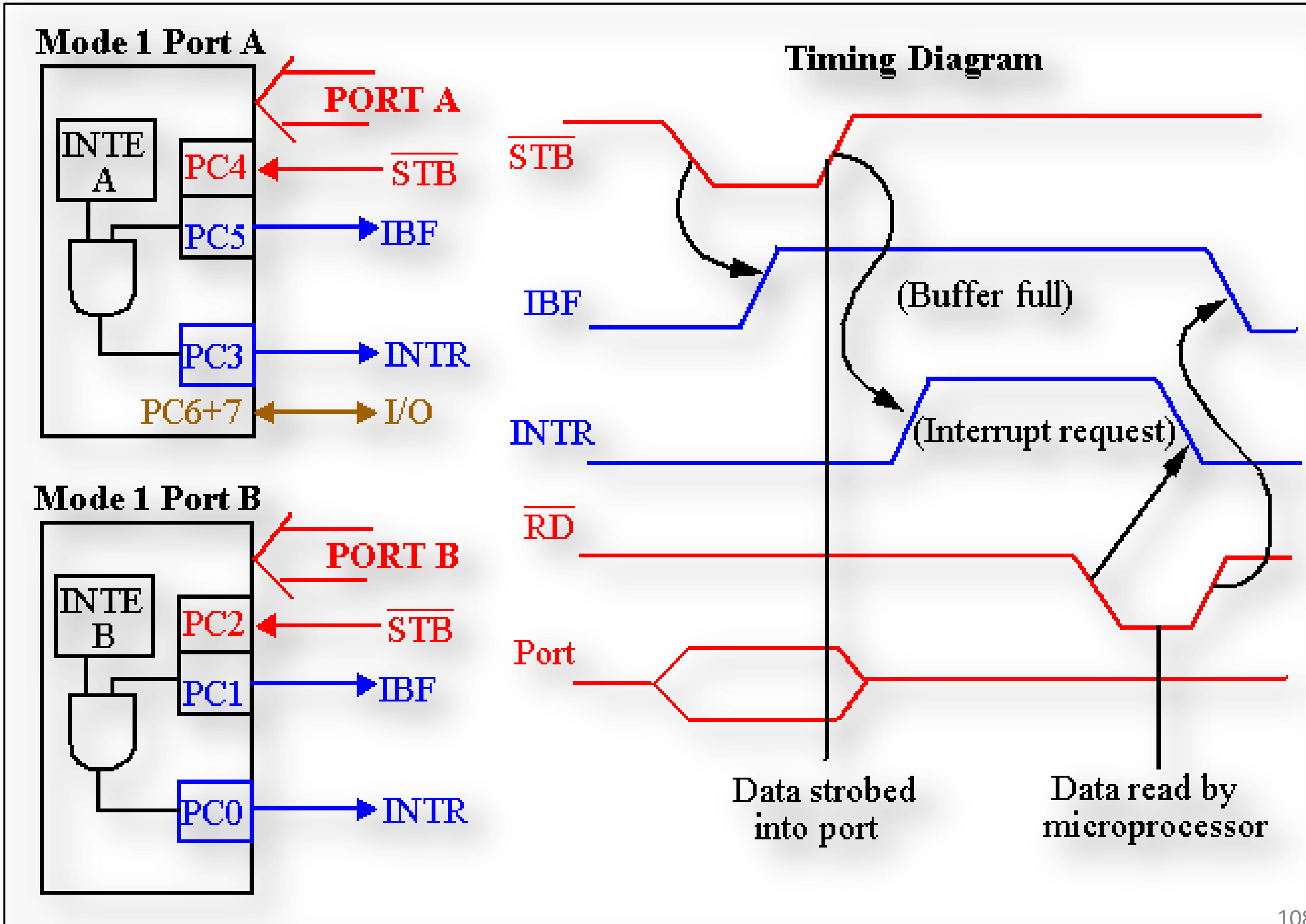
- **IBF** : Input buffer full is an output indicating that the input latch contain information.

- **INTR** : Interrupt request is an output that requests an interrupts.

- **INTE** : The interrupt enable signal is neither an input nor an output; it is an internal bit programmed via the PC4 (port A) or PC2 (port B) bits.

- **PC7,PC6** : The port C pins 7 and 6 are general purpose I/O pings that are available for any purpose.

8255: Mode 1 Strobed Input



82C55 : Mode 1 Output

- **OBF** : Output buffer full is an output that goes low when data is latched in either port A or port B. Goes low on ~ACK.

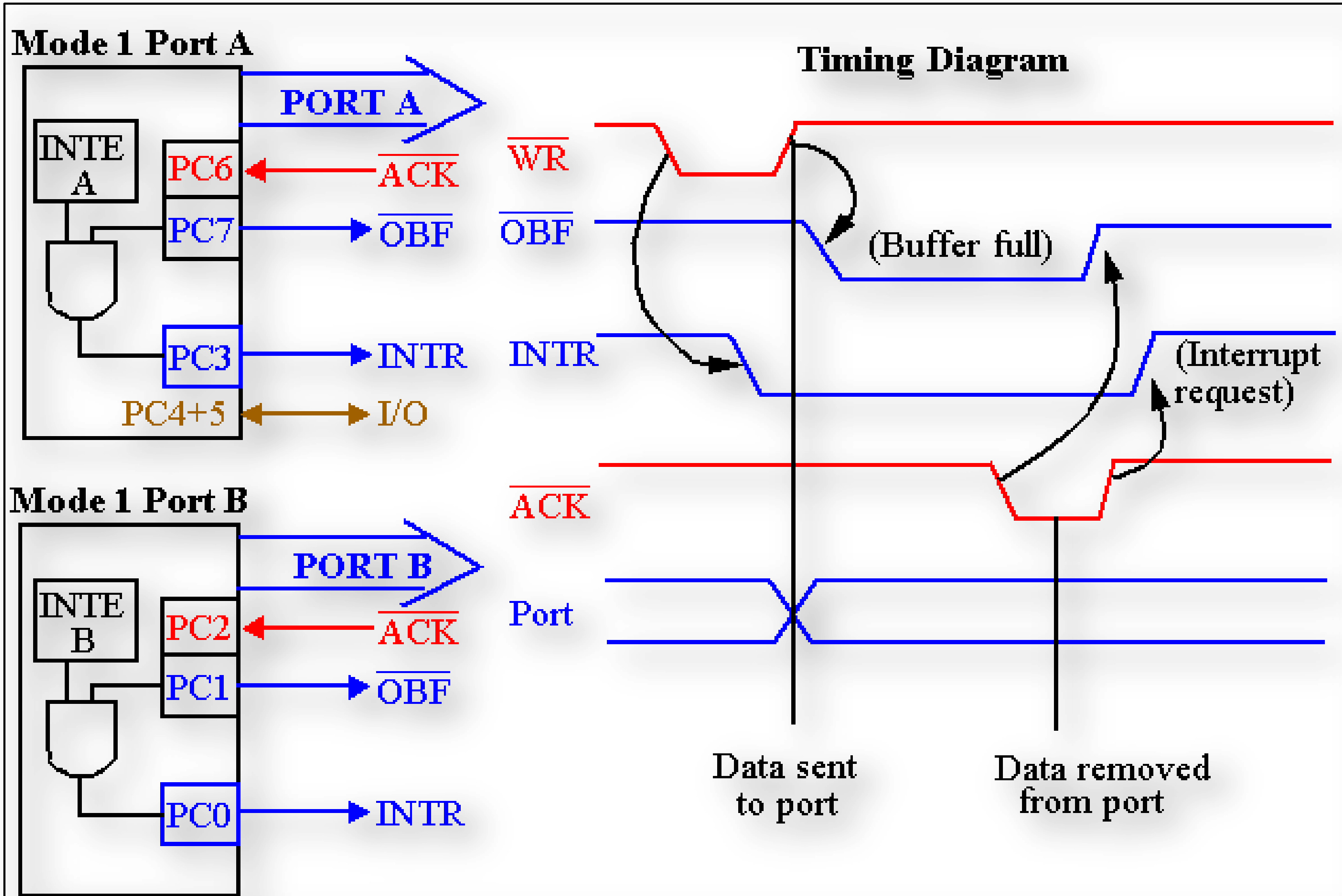
- **ACK** : The acknowledge signal causes the ~OBF pin return to 0. This is a response from an external device.

- **INTR** : Interrupt request is an output that requests an interrupt.

- **INTE** : The interrupt enable signal is neither an input nor an output; it is an internal bit programmed via the PC6(Port A) or PC2(port B) bits.

- **PC5,PC4** : The port C pins 5 and 4 are general-purpose I/O pins that are available for any purpose.

8255 : Mode 1 Output



Mode 2 - Bidirectional Data Transfer

- This mode is used primarily in applications such as **data transfer between two computers.**

- In this mode, **Port A** can be configured as the bidirectional port, **Port B** either in Mode 0 or Mode 1.

- **Port A uses five signals from Port C as handshake signals for data transfer.**

- The remaining three signals from **Port C** can be used either as simple I/O or as handshake for port B.

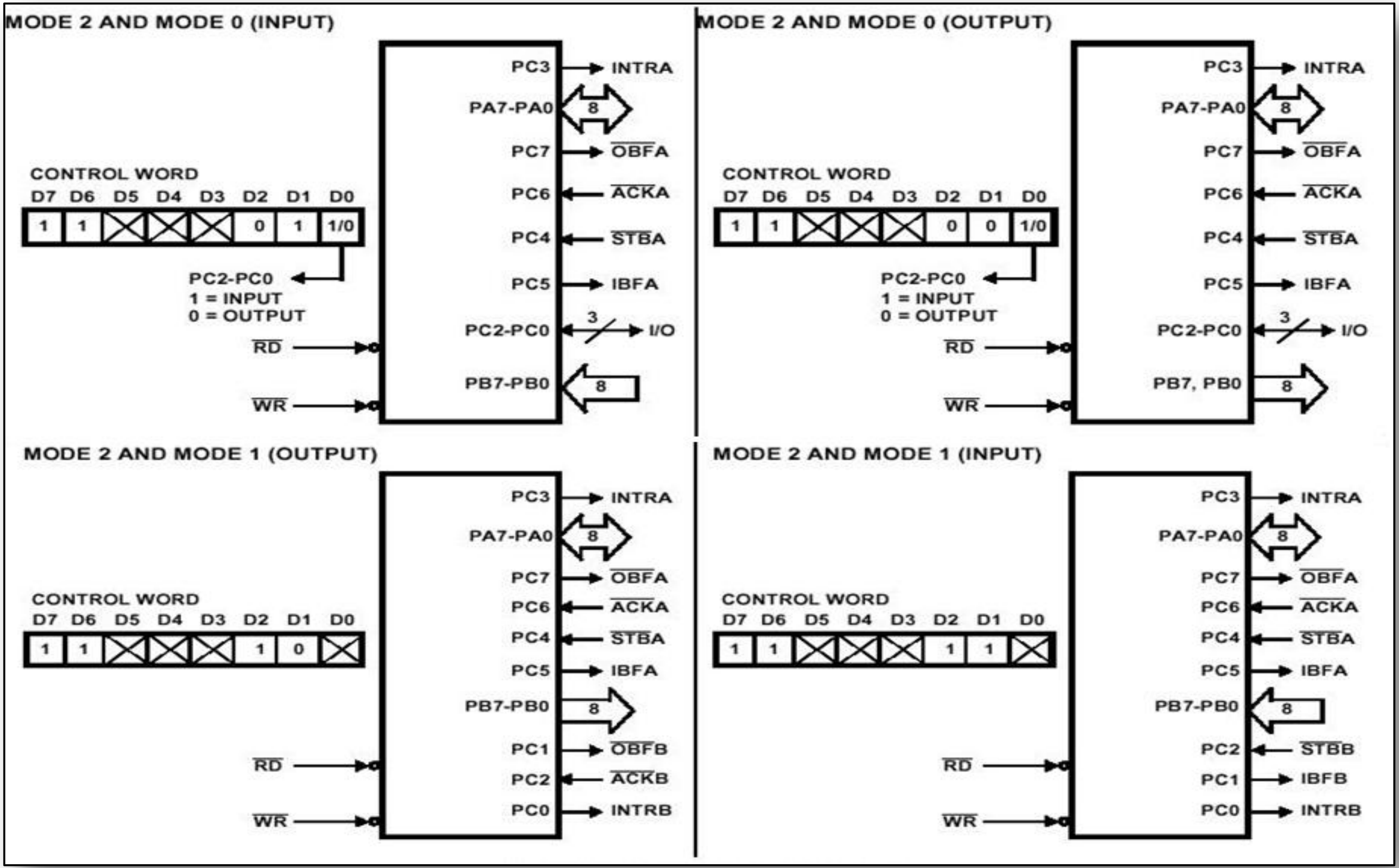
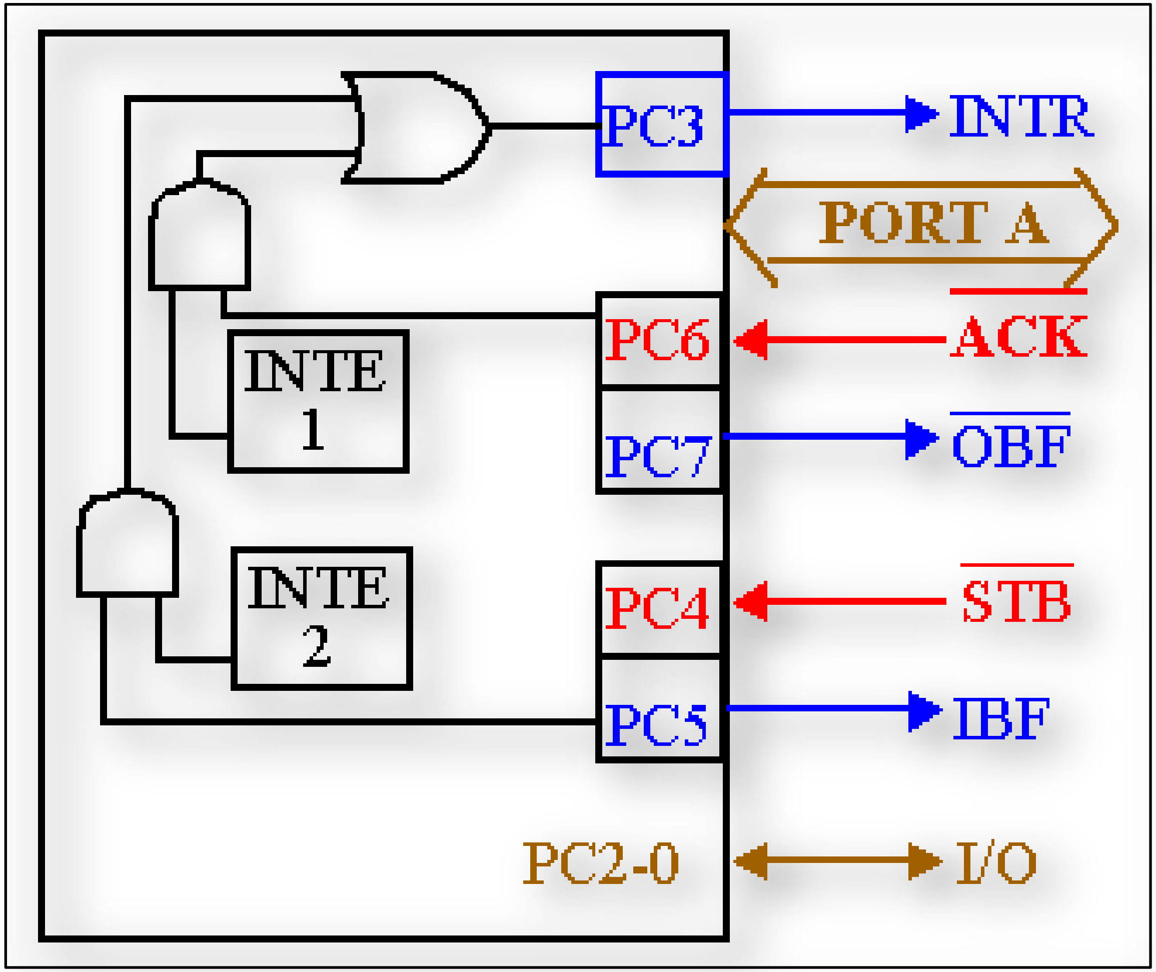


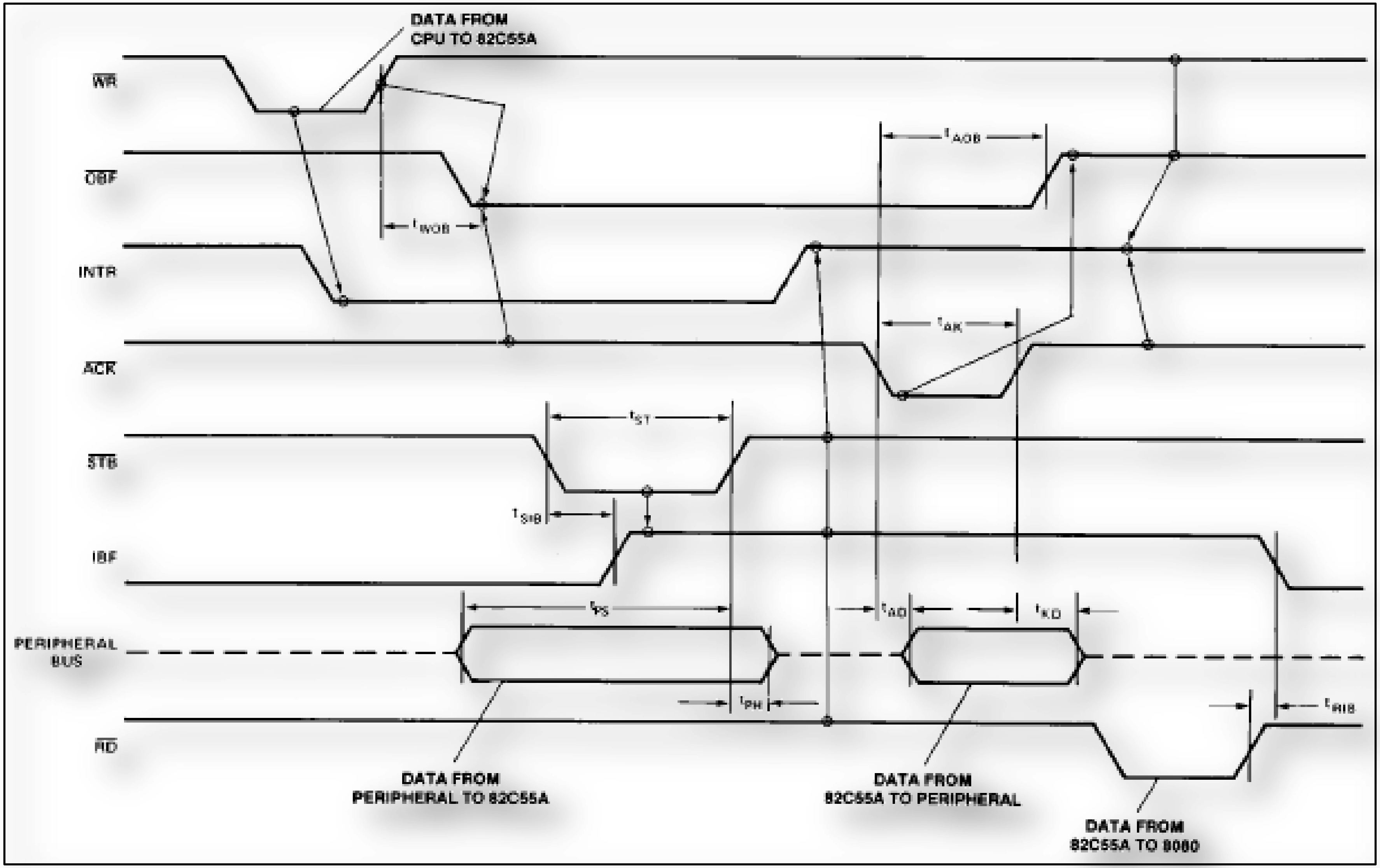
Figure: Mode 2 - Bidirectional Data Transfer

8255: Mode 2 Bi-directional Operation



- Timing diagram is a combination of the Mode 1 Strobed Input and Mode 1 Strobed Output Timing diagrams.

Mode 2 Timing Diagram



8255 Control Words

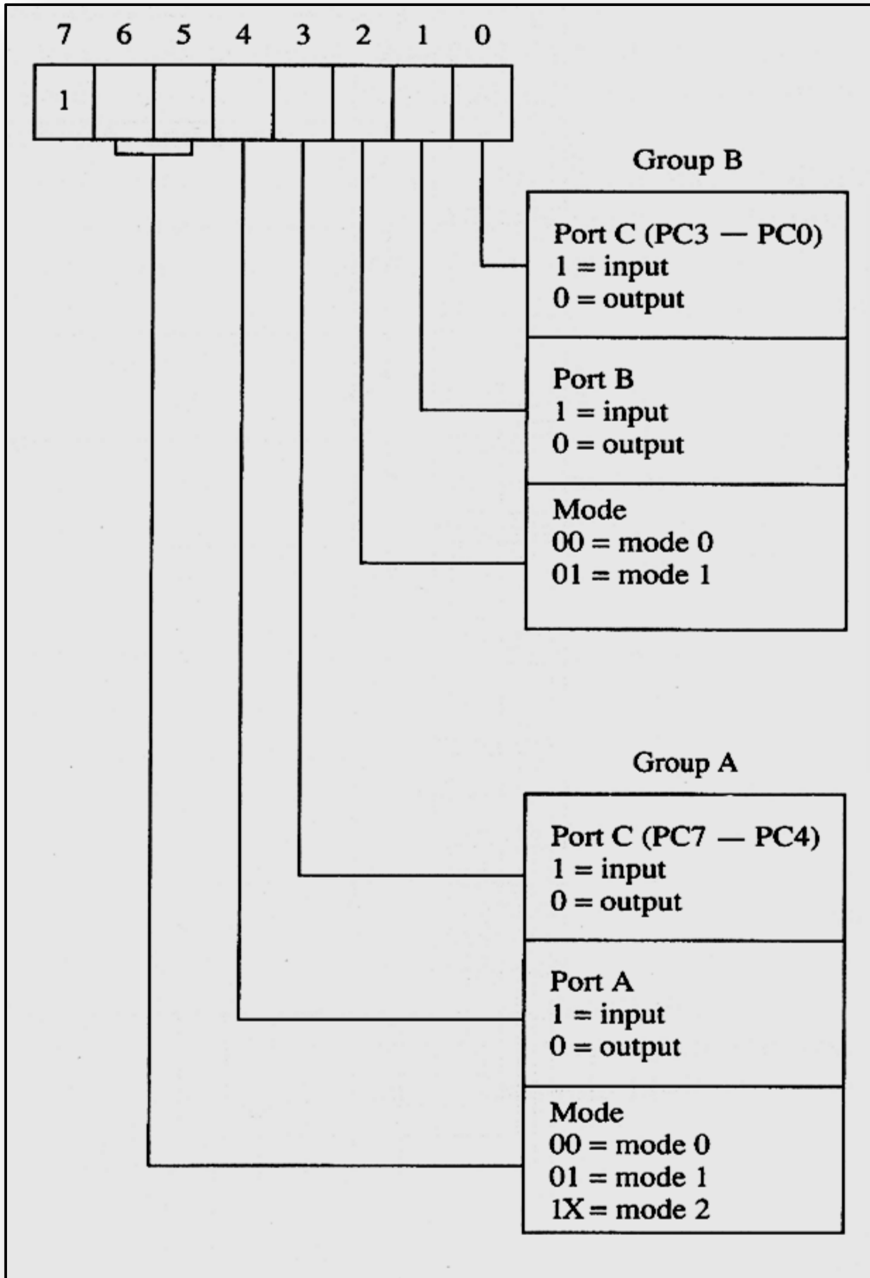
- There are 2 control words in 8255.
  - **Mode Definition (MD) Control word and**
  - **Bit Set / Reset (BSR) Control Word**

- **MD control word** configures the ports of 8255 as input or output in Mode 0, 1, or 2.

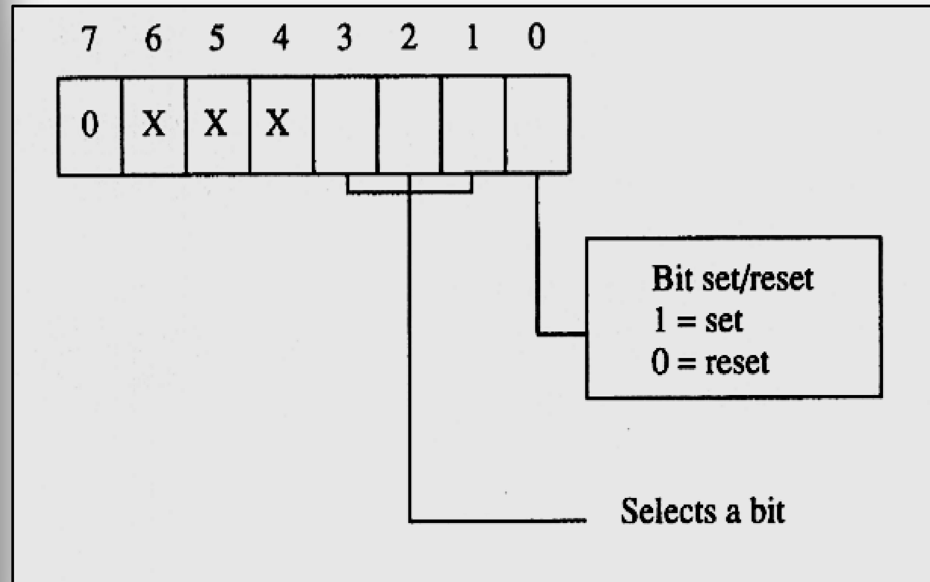
- **PCBSR control word** is used to set to 1 or reset to 0 any one selected bit of Port C

8255 Control words

1.Mode Definition (MD) Control word



2. Bit Set / Reset (BSR) Control Word



A_1	A_0	Function
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Command Register

Displays Interfacings

1. LCD Interfacing
2. LED Interfacing

INTERFACING LCD MODULE TO 8086

Introduction:

LCD or Liquid Crystal Display is an output device used in many processor based applications like calculators, Xerox machines, speedometers etc. The 8086 kit, which you use in the lab, also uses a LCD display to view the data entered into and coming out of the processor

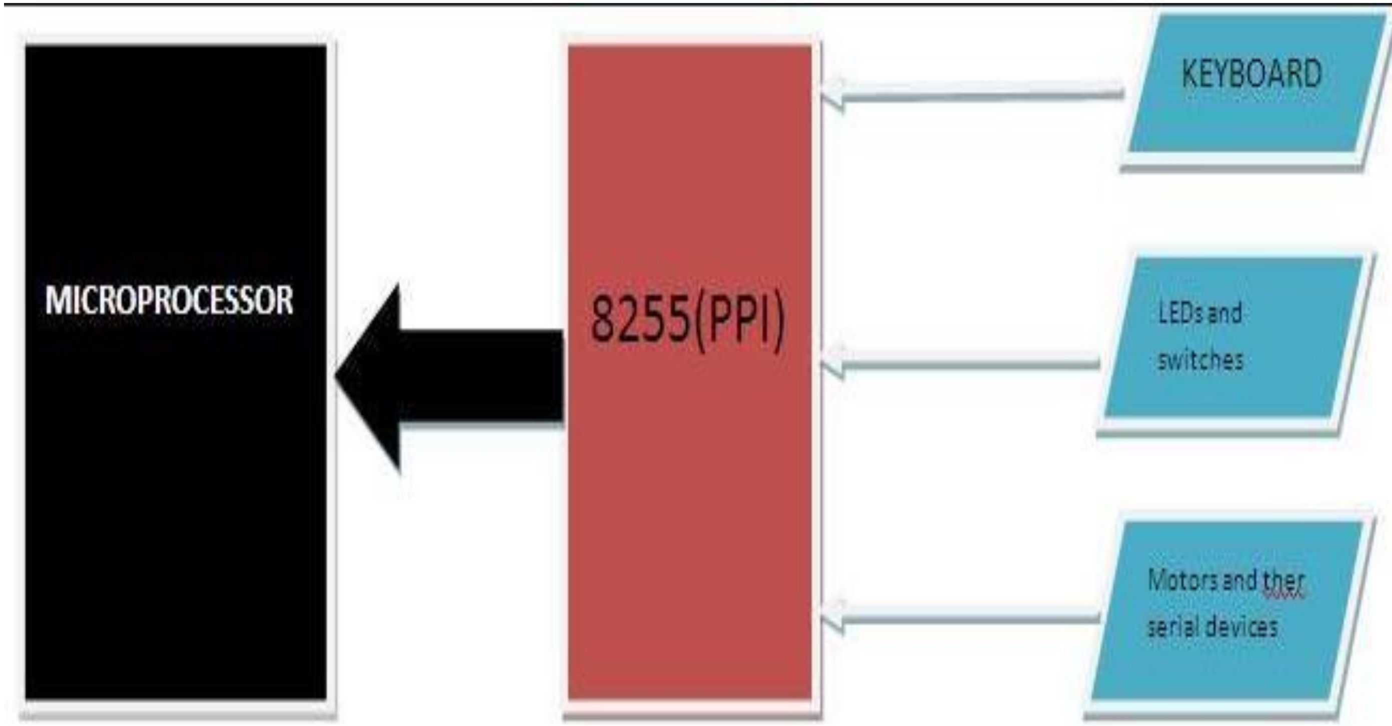
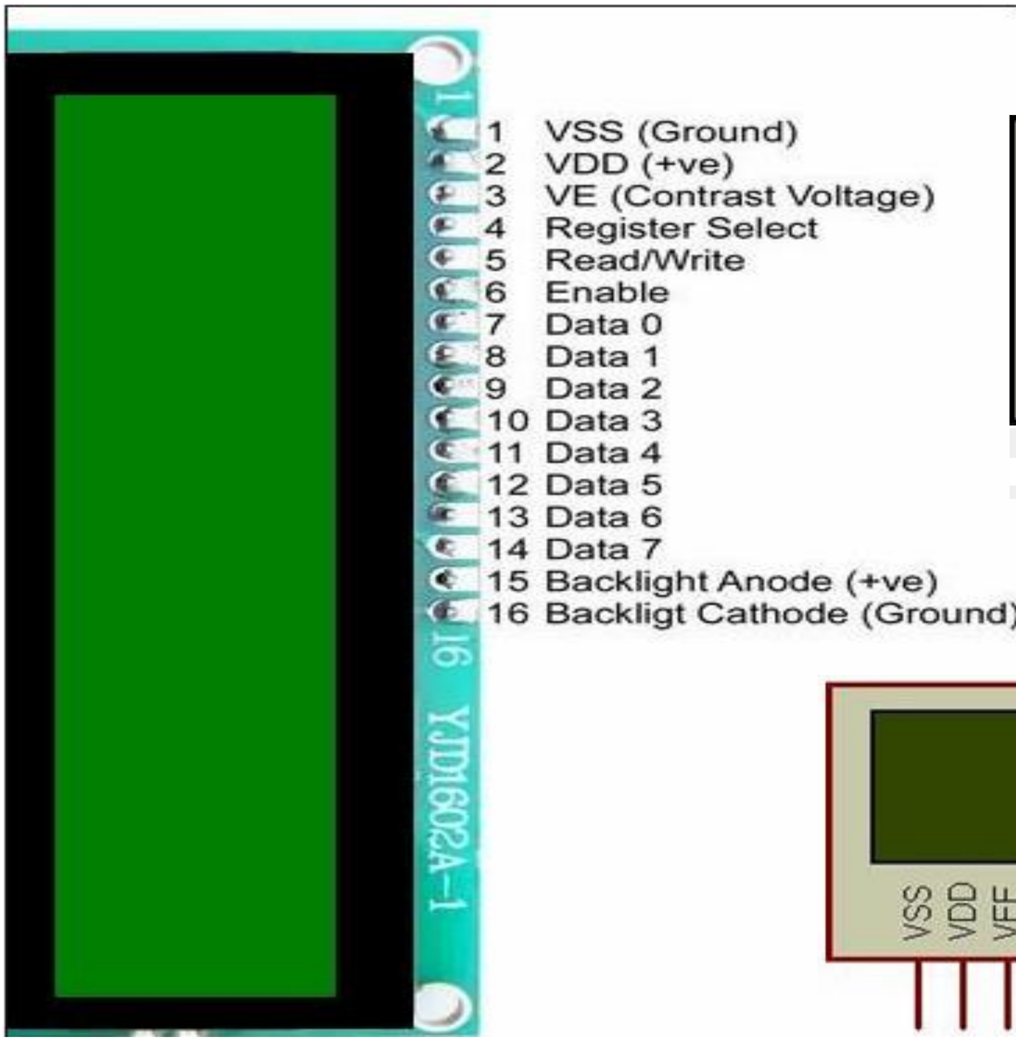
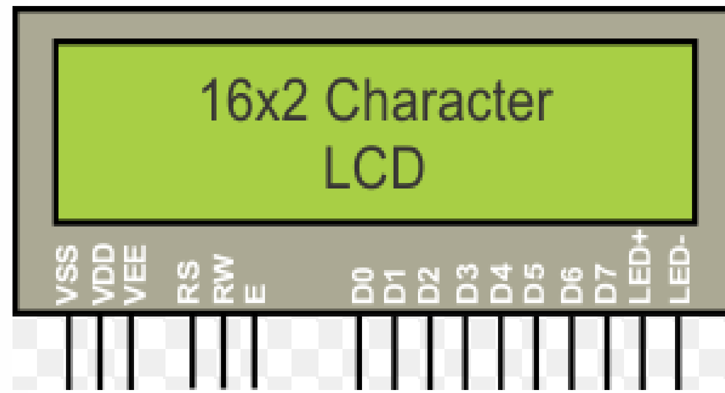
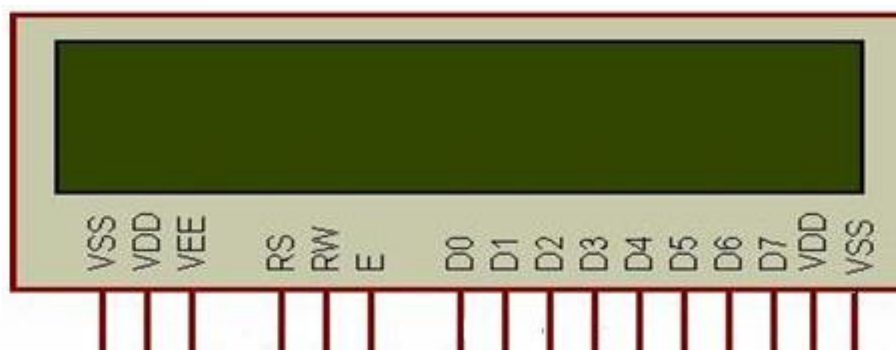


Fig 1.c (I/O interfacing using 8255)

16*2 LCD Module







16*2 LCD Pin Functions

Pin No	Function	Name
1	Ground (0V)	Ground
2	Supply voltage; 5V (4.7V – 5.3V)	Vcc
3	Contrast adjustment; through a variable resistor	V_{EE}
4	Selects command register when low; and data register when high	Register Select
5	Low to write to the register; High to read from the register	Read/write
6	Sends data to data pins when a high to low pulse is given	Enable
7	8-bit data pins	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6

Keyboard Interface

- In most keyboards, the key switches are connected in a matrix of Rows and Columns.

- Getting meaningful data from a keyboard requires three major tasks:
  - Detect a key press
  - Debounce the key press.
  - Encode the key press (produce a standard code for the pressed key).

- Logic ‘0’ is read by the microprocessor when the key is pressed.

Keyboard Interface(Contd…)

Key Debounce:

Whenever a mechanical push-bottom is pressed or released once, the mechanical components of the key do not change the position smoothly; rather it generates a transient response. These may be interpreted as the multiple pressures and responded accordingly.

Keyboard Interface(Contd…)

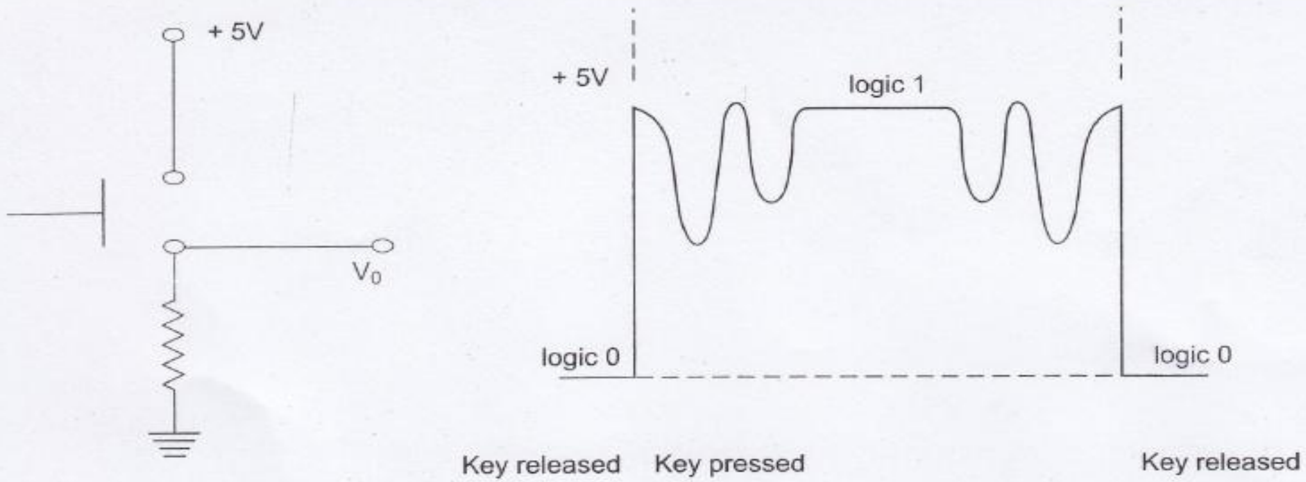


Fig. 5.23 *A Mechanical Key and Its Response*

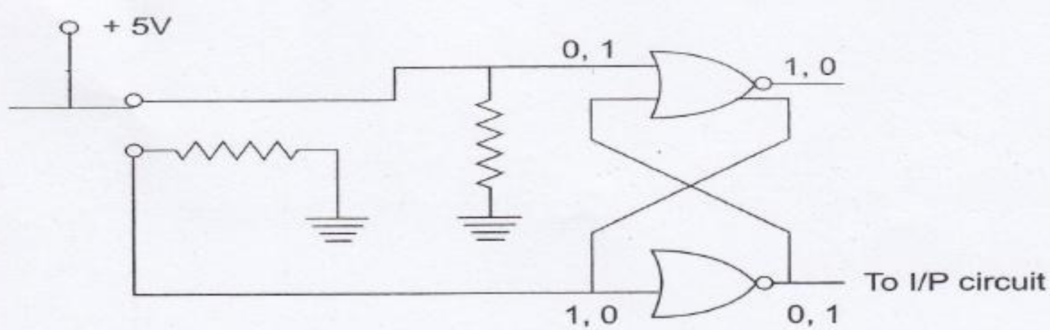
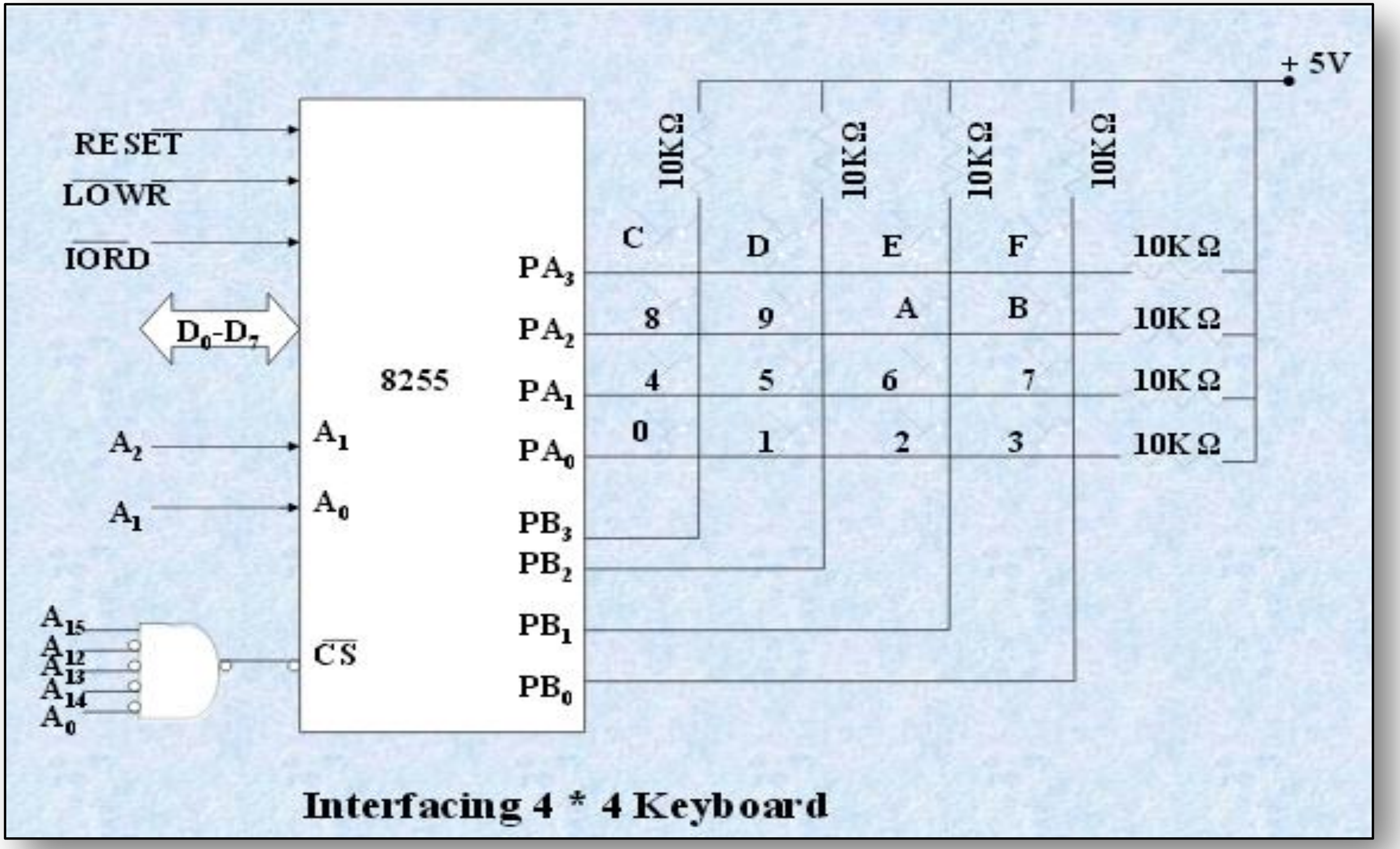
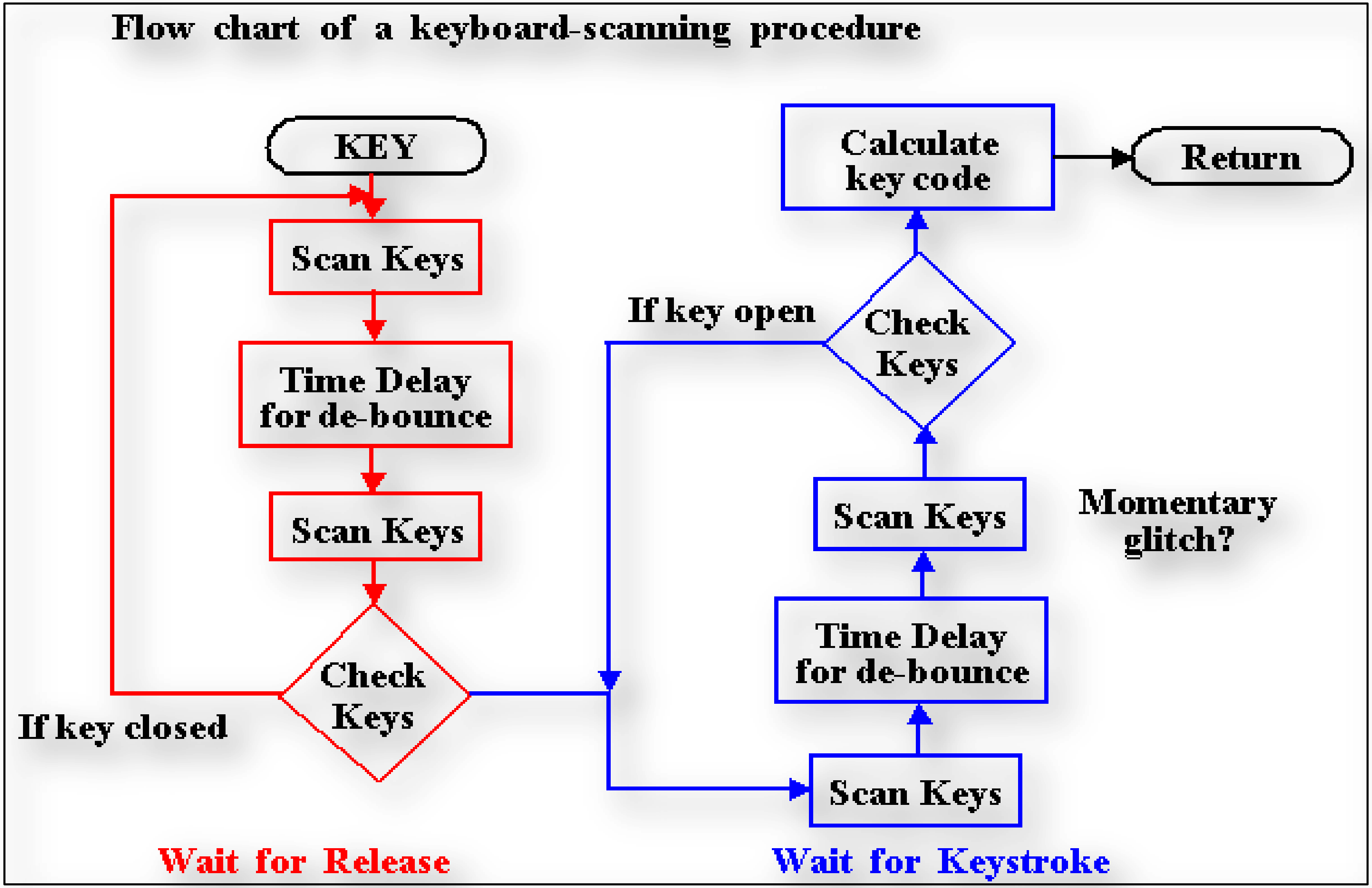


Fig. 5.24 *Hardware Debouncing Circuit*

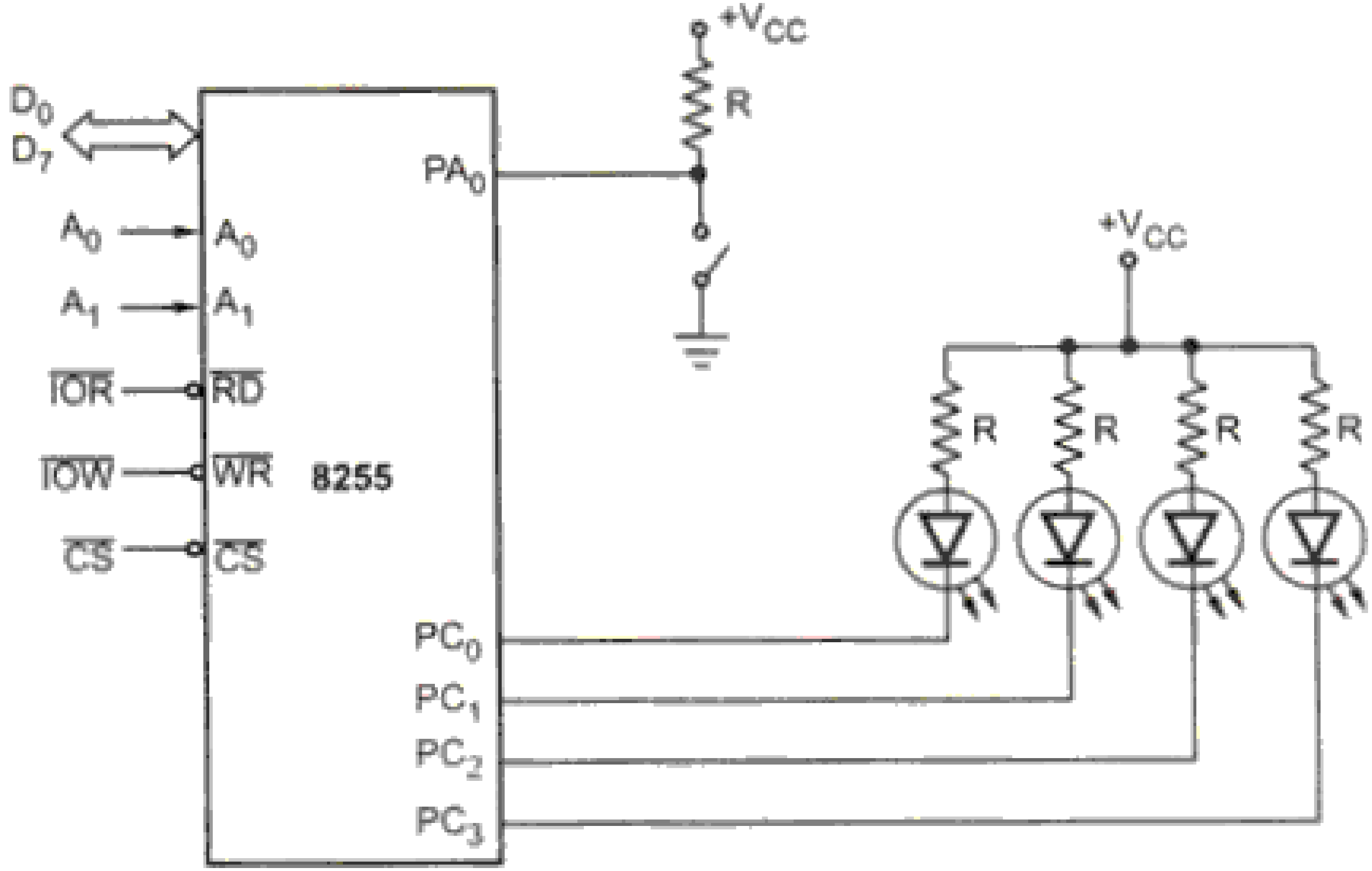
Keyboard Interface(Contd…)



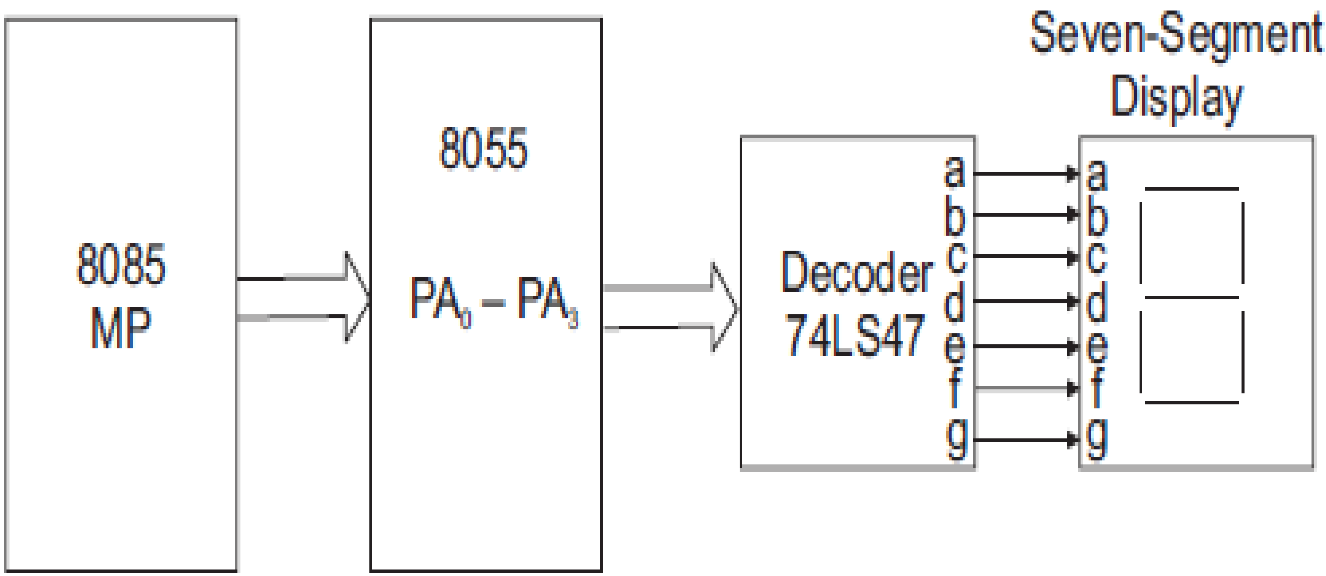
Keyboard Interface(Contd…)



Led interfacing with 8086 using 8255

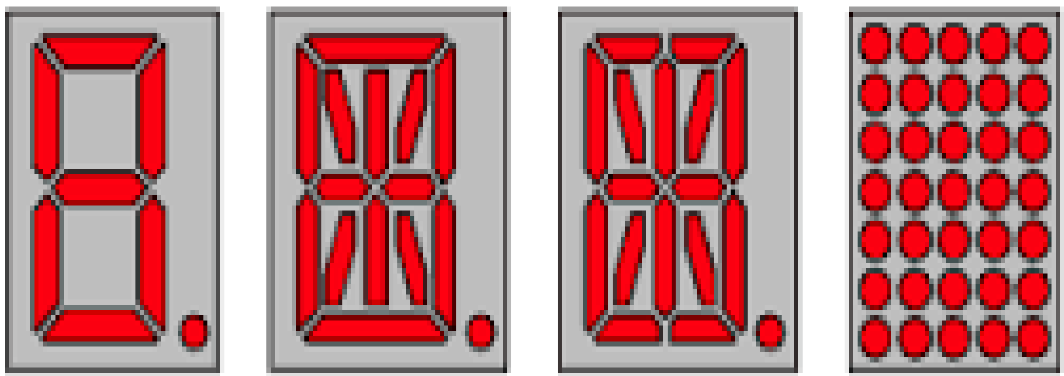


Seven segment LED Interfacing

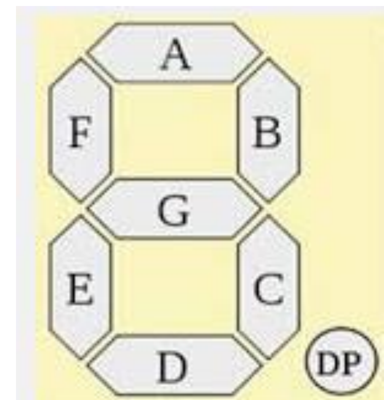


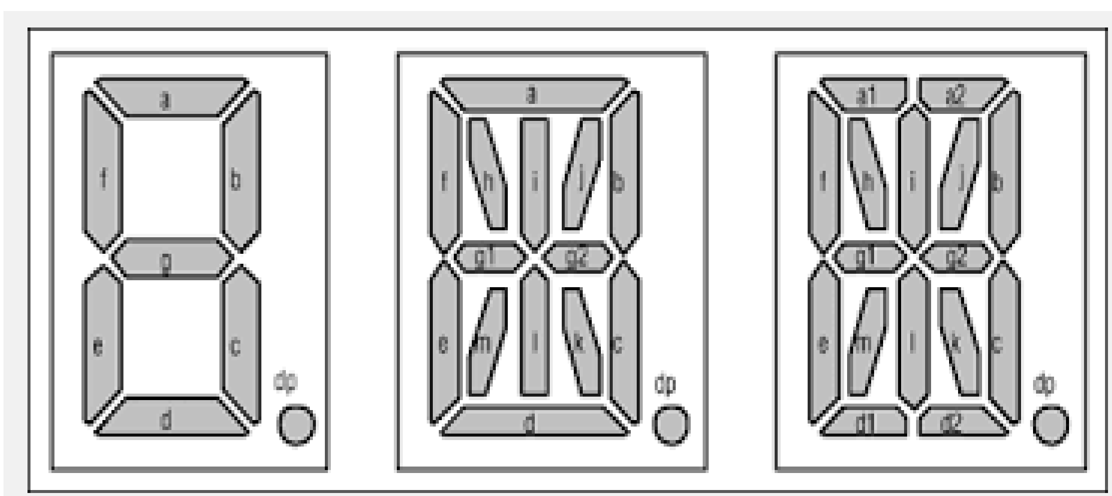
Block diagram of single-digit display

7, 14, 16 LED Segments



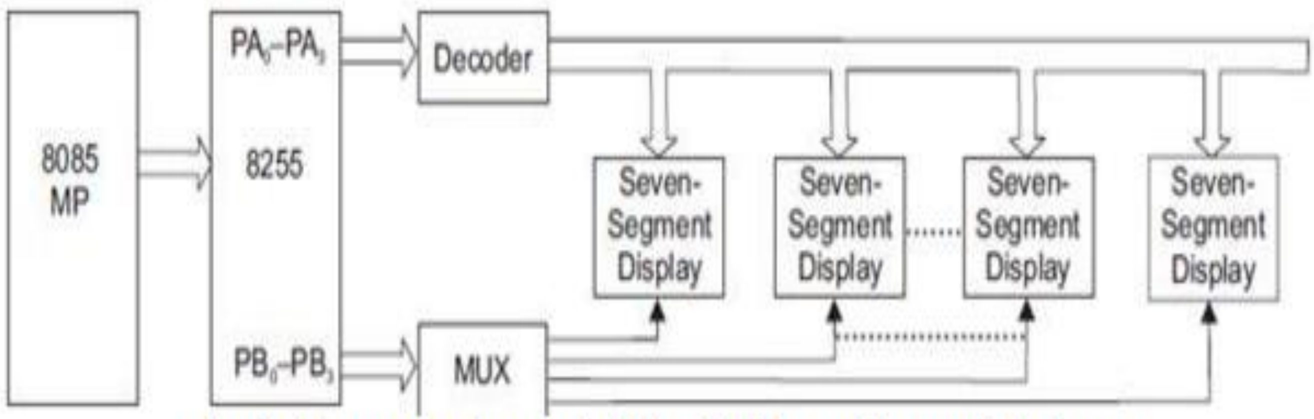
7-Segment plus DP | 14-Segment plus DP | 16-Segment plus DP | 5 x 7 Matrix





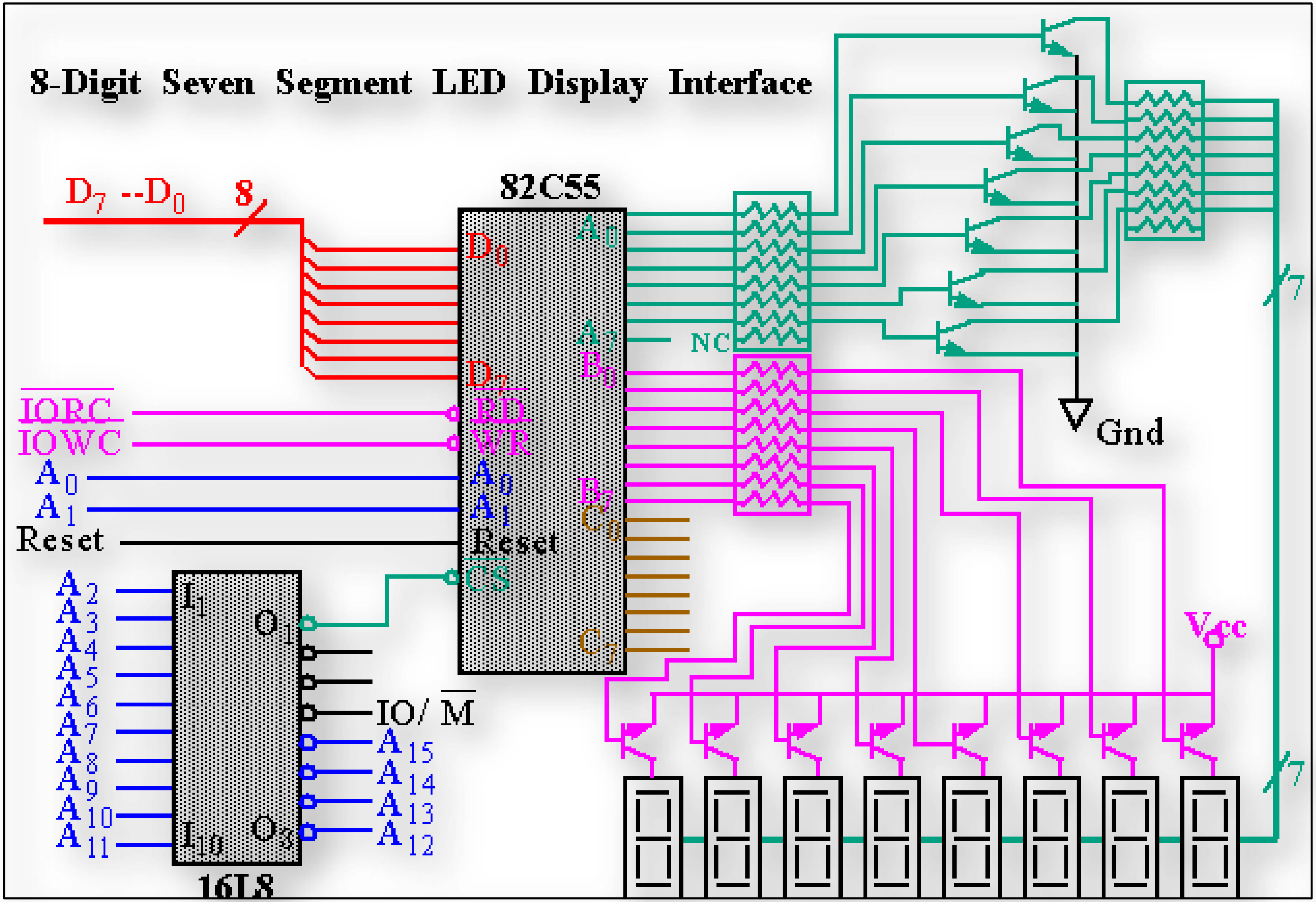
Four seven segment displays

Four-Digit Display



Block diagram of multi-digit display using multiplexer

8 Digit LED Display

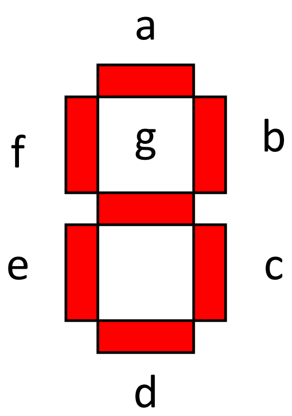


Seven-Segment Display

Table : Truth table for seven-segment display

Decimal Number	Inputs				Output						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

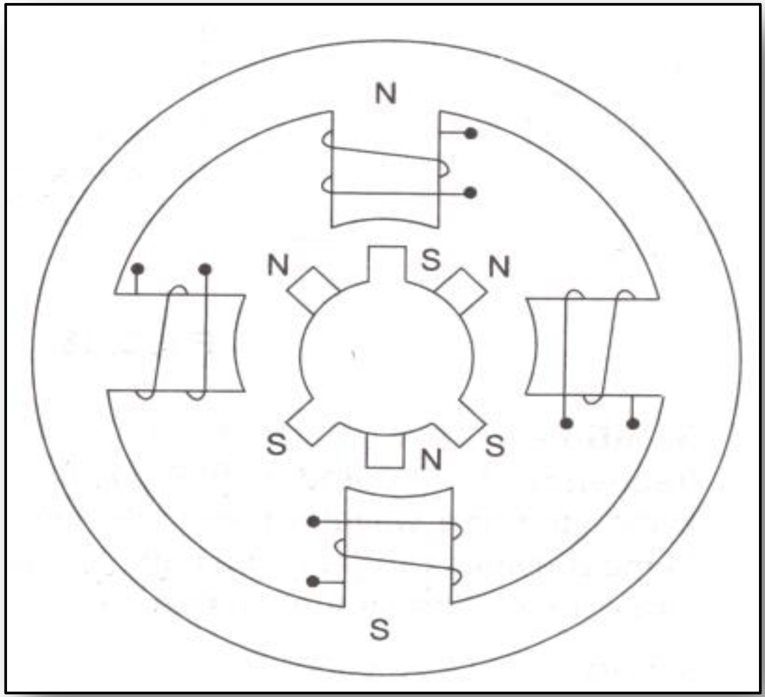
7 segment display



Digit-abcdefg-hex

0-1111110-7E	1-0110000-30
2-1101101-6D	3-1111001-79
4-0110011-33	5-1011011-5B
6-1011111-5F	7-1110000-70
8-1111111-7F	9-1111011-7B
A-1110111-77	B-0011111-1F
C-1001110-4E	D-0111101-3D
E-1001111-4F	F-1000111-47

Stepper Motor Interface



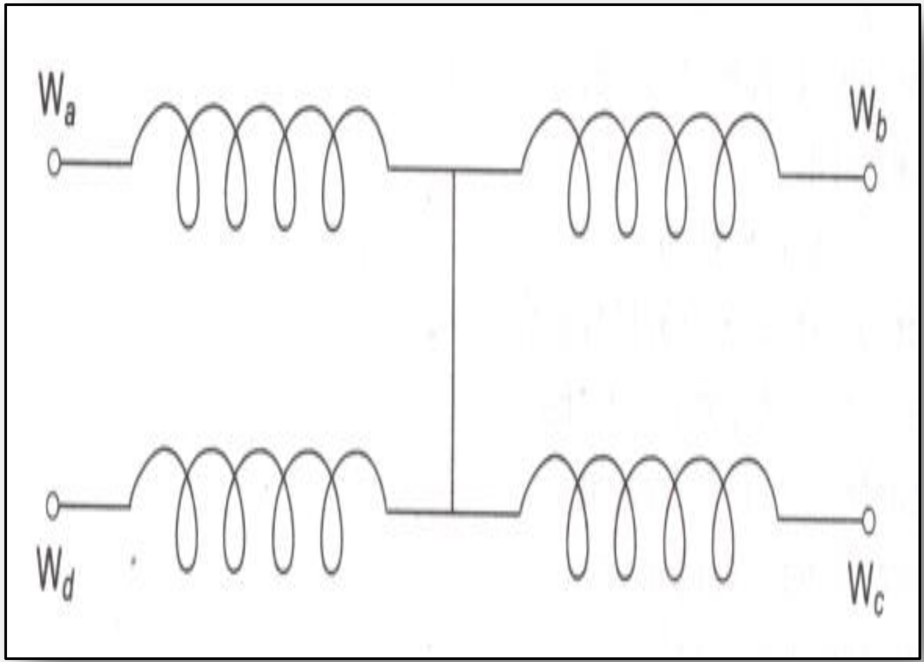
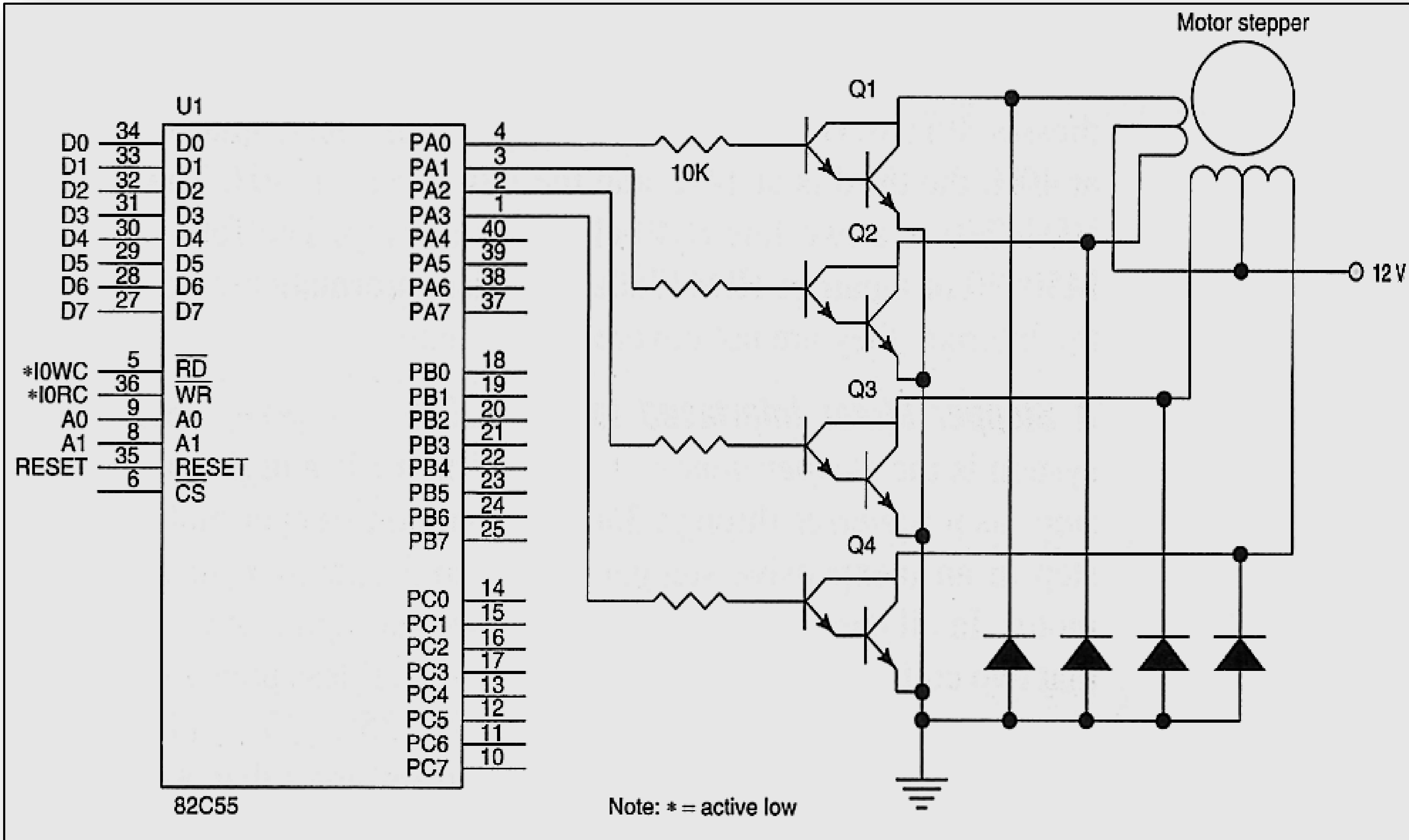


Fig.1 Internal schematic of a four winding stepper motor

Fig.2 Winding arrangement of a stepper motor.

Contd...

Stepper Motor Interface



Interfacing Analog to Digital Data Converters

General algorithm for ADC interfacing contains the following steps:

- Ensure the stability of analog input, applied to the ADC.
- Issue start of conversion pulse to ADC
- Read end of conversion signal to mark the end of conversion processes.
- Read digital data output of the ADC as equivalent digital output.
- Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
- If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

Interfacing Analog to Digital Data Converters

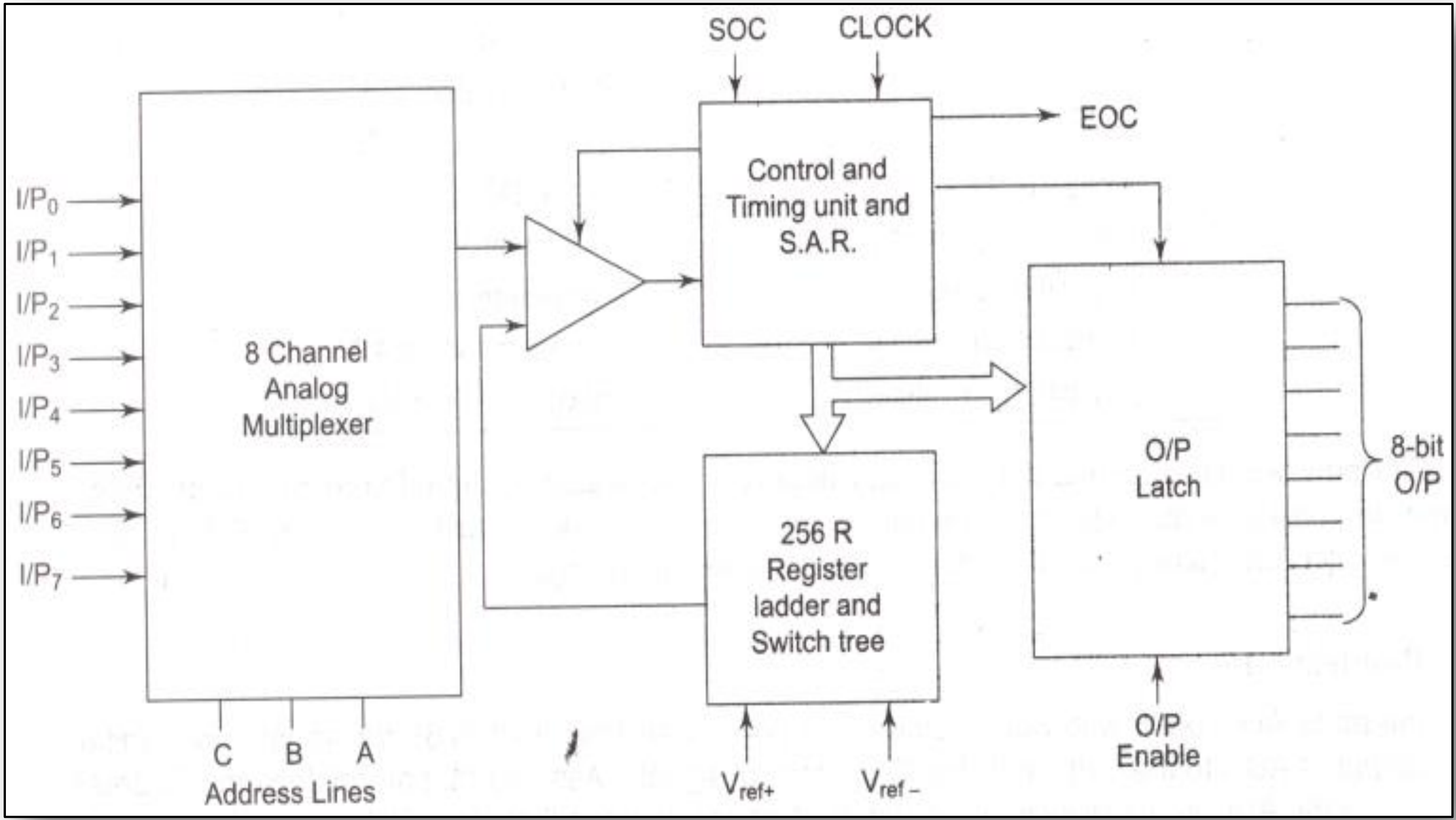


Fig.1 Block Diagram of ADC 0808/0809

Interfacing Analog to Digital Data Converters

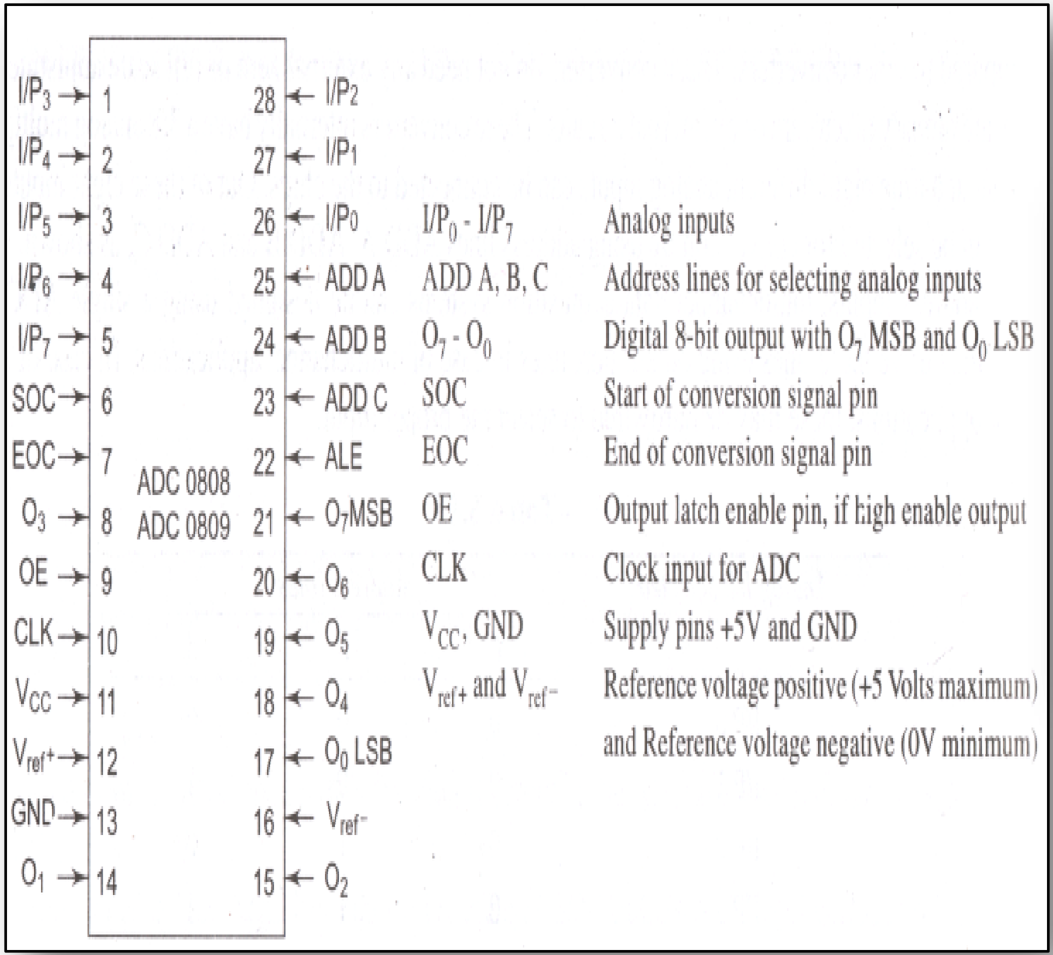


Fig.2 Pin Diagram of ADC 0808/0809

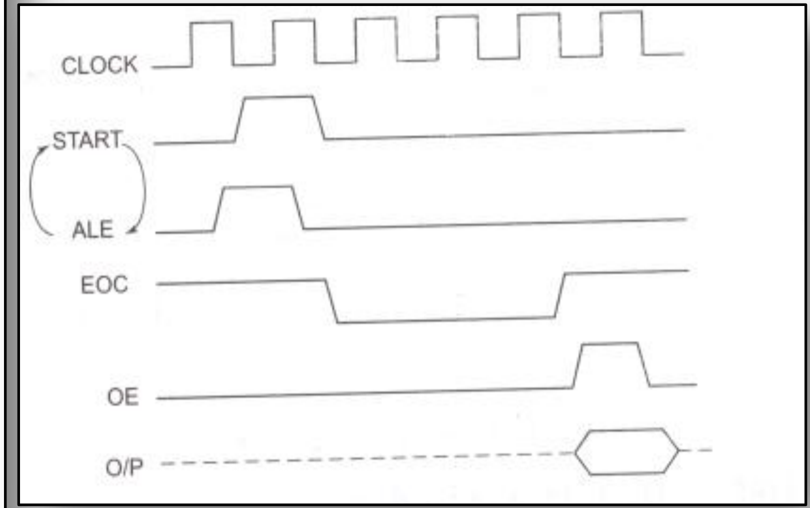


Fig.3 Timing Diagram Of ADC 0808.

Interfacing Analog to Digital Data Converters

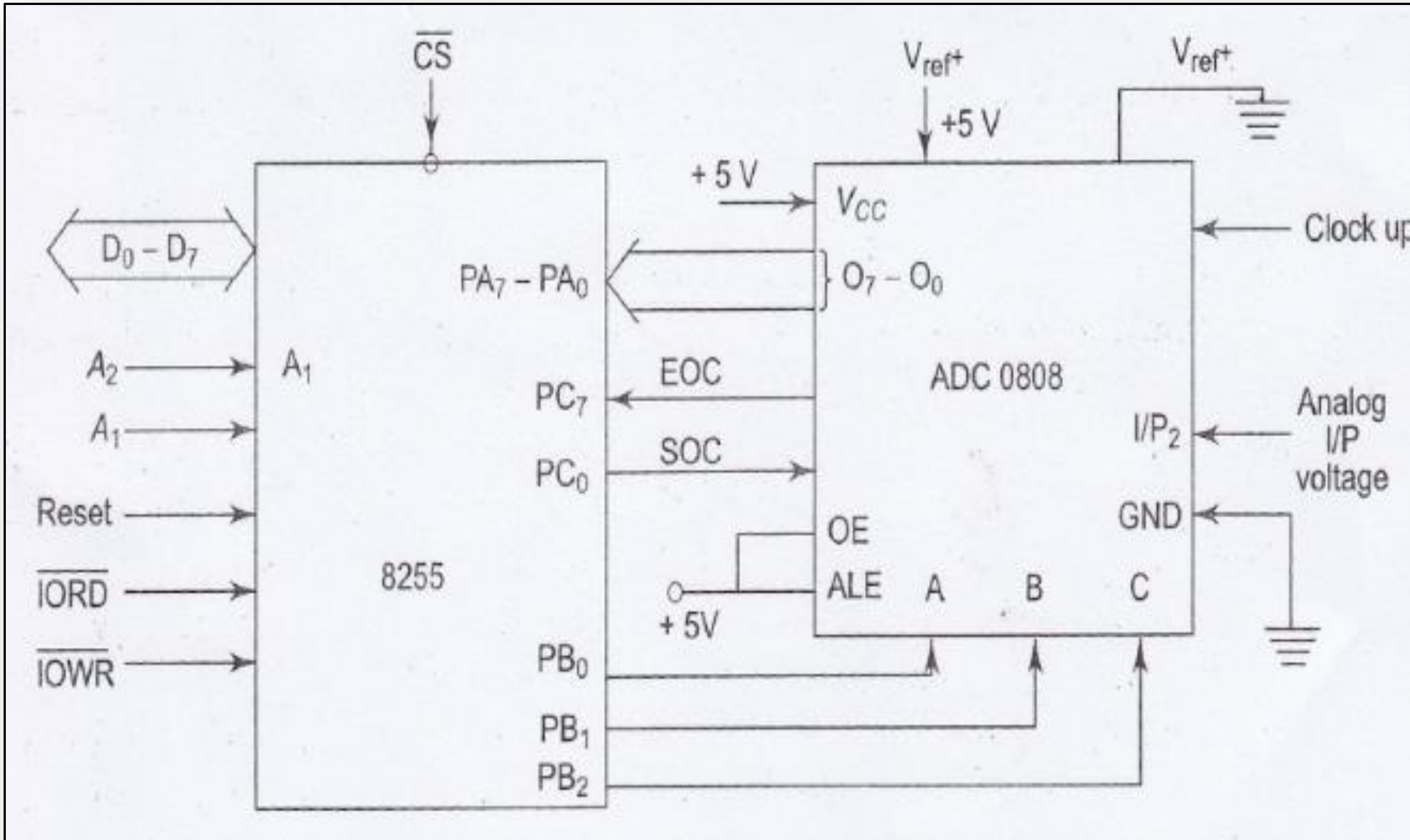
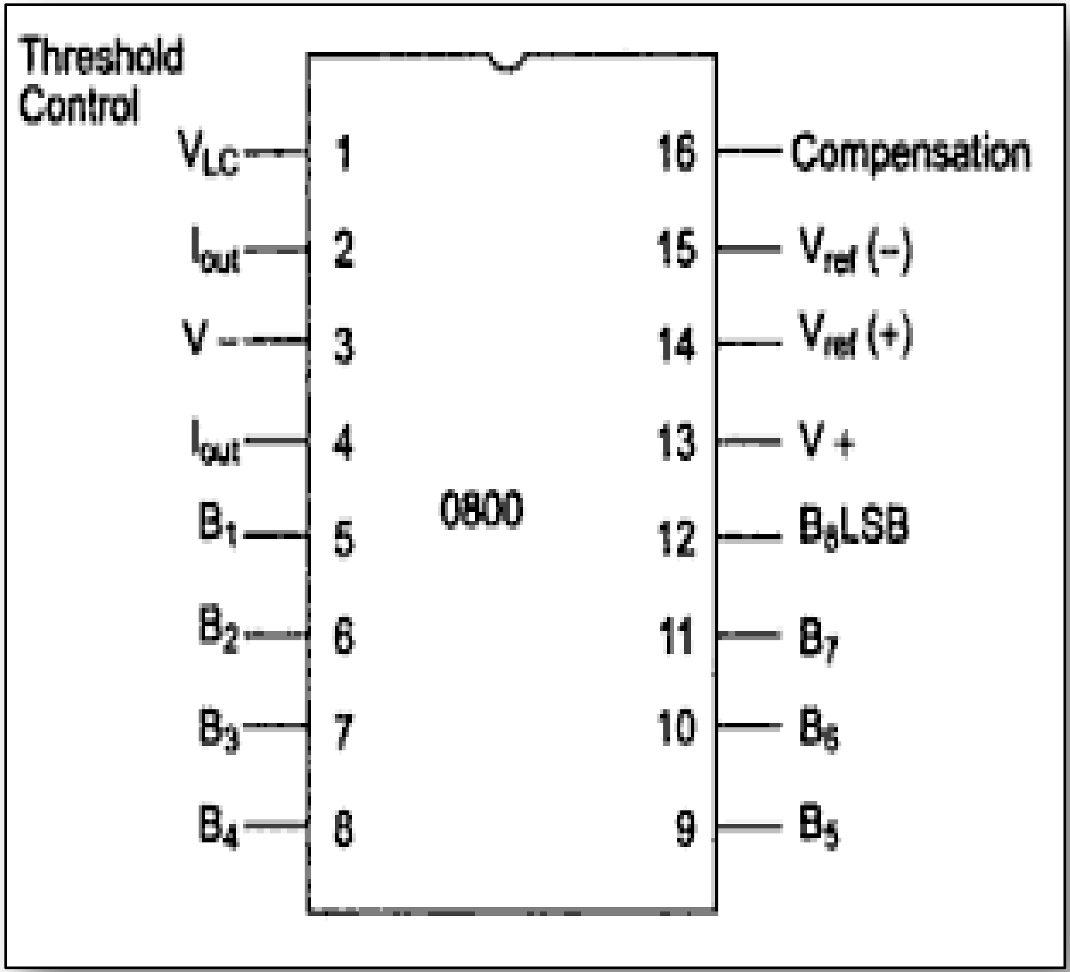


Fig: Interfacing ADC0808 with 8086

Interfacing Digital To Analog Converters



Pin Diagram of DAC 0800

Interfacing Digital To Analog Converters

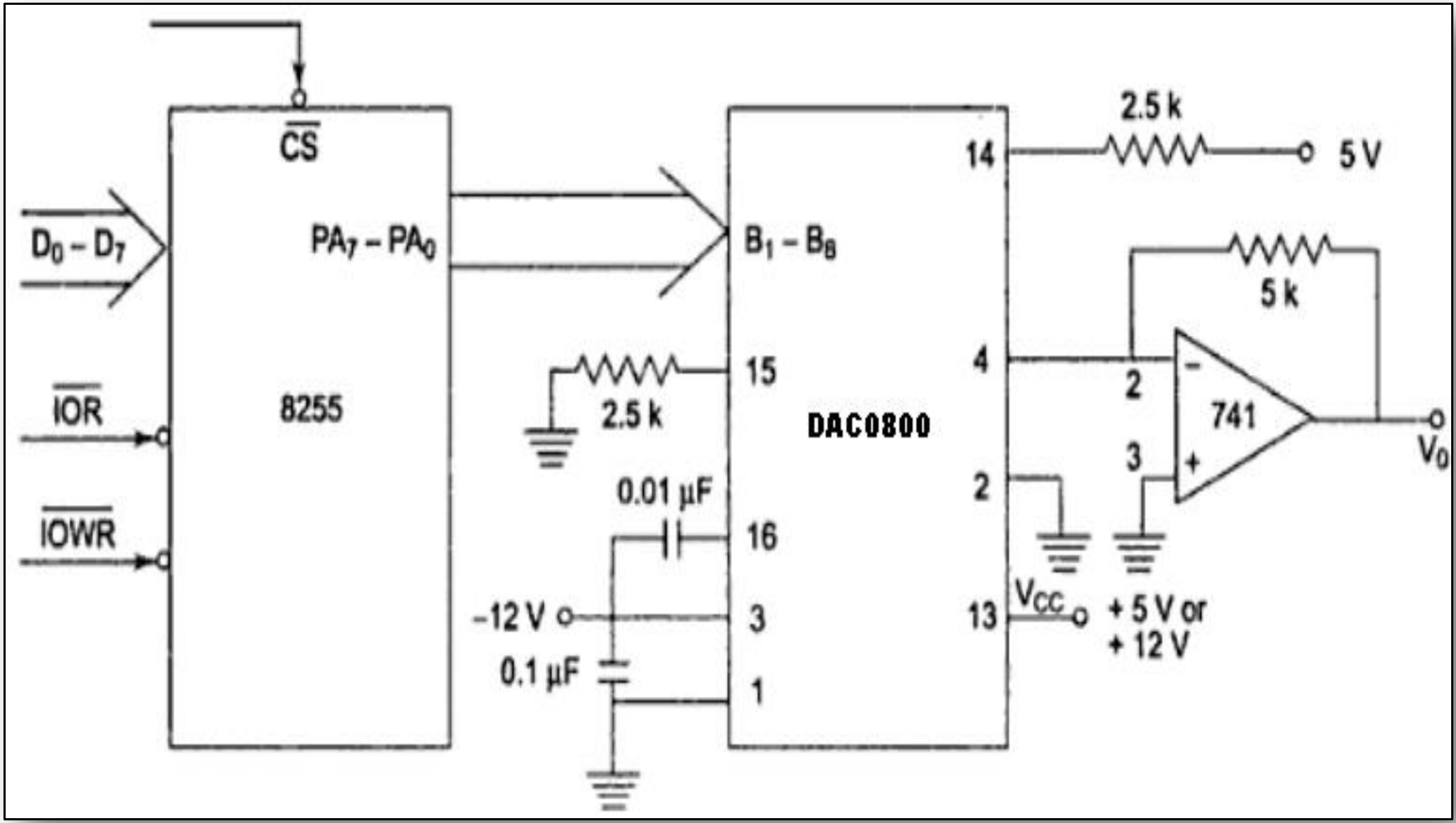


Fig:Interfacing DAC0800 with 8086

Interfacing with advanced devices

Bus Architecture

- Address:
  - If I/O, a value between 0000H and FFFFH is issued.
  - If memory, it depends on the architecture:
    - 20 -bits (8086/8088)
    - 24 -bits (80286/80386SX)
    - 25 -bits (80386SL/SLC/EX)
    - 32 -bits (80386DX/80486/Pentium)
    - 36 -bits (Pentium Pro/II/III)

Bus Architecture

- Data:
  - 8 -bits (8085)
  - 16 -bits (8086/80286/80386SX/SL/SLC/EX)
  - 32 -bits (80386DX/80486/Pentium)
  - 64 -bits (Pentium/Pro/II/III)
- Control:
  - Most systems have at least 4 control bus connections (active low).
  - MRDC (Memory Read Control), MWRC , IORC (I/O Read Control), IOWC

MEMORY

Memory Types

- Two basic types:
  - ROM: Read-only memory
  - RAM: Read-Write memory
- Four commonly used memories:
  - ROM
  - Flash (EEPROM)
  - Static RAM (SRAM)
  - Dynamic RAM (DRAM)

Memory Chips

- The data pins are typically bi-directional in read-write memories.
  - The number of data pins is related to the size of the memory location. For example, an 8-bit wide (byte-wide) memory device has 8 data pins.

- Each memory device has at least one chip select (CS) or chip enable (CE) or select (S) pin that enables the memory device.
  - This enables read and/or write operations.
  - If more than one are present, then all must be 0 in order to perform a read or write.

SRAM vs. DRAM

- SRAMs
  - SRAMs used for caches have access times as low as 10ns .
- DRAMs
  - SRAMs are limited in size (up to about 128Kb).
  - DRAMs are available in much larger sizes, e.g., 64M X 1.
  - DRAMs MUST be refreshed every 2 to 4 ms
  - Since they store their value on an integrated capacitor that loses charge over time.

Memory Address Decoding

Memory Address Decoding

- The processor can usually address a memory space that is much larger than the memory space covered by an individual memory chip.

- In order to splice a memory device into the address space of the processor, decoding is necessary.

- For example, the 8088 issues 20-bit addresses for a total of 1MB of memory address space.

Ex. Memory Address Decoding

- The BIOS on a 2716 EPROM has only 2KB of memory and 11 address pins.

- A decoder can be used to decode the additional 9 address pins and allow the EPROM to be placed in any 2KB section of the 1MB address space.

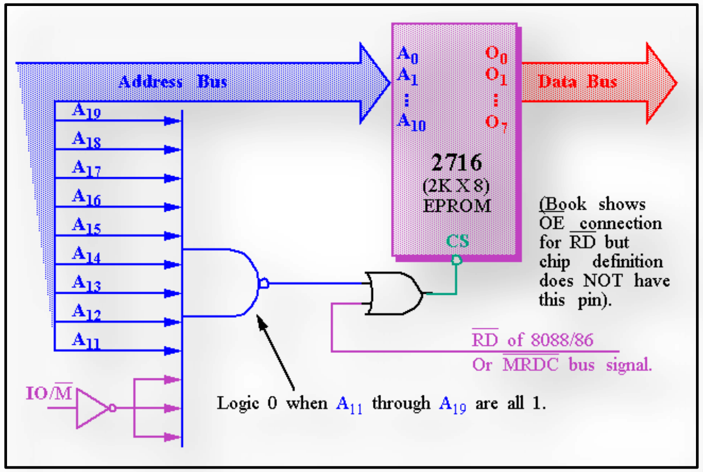
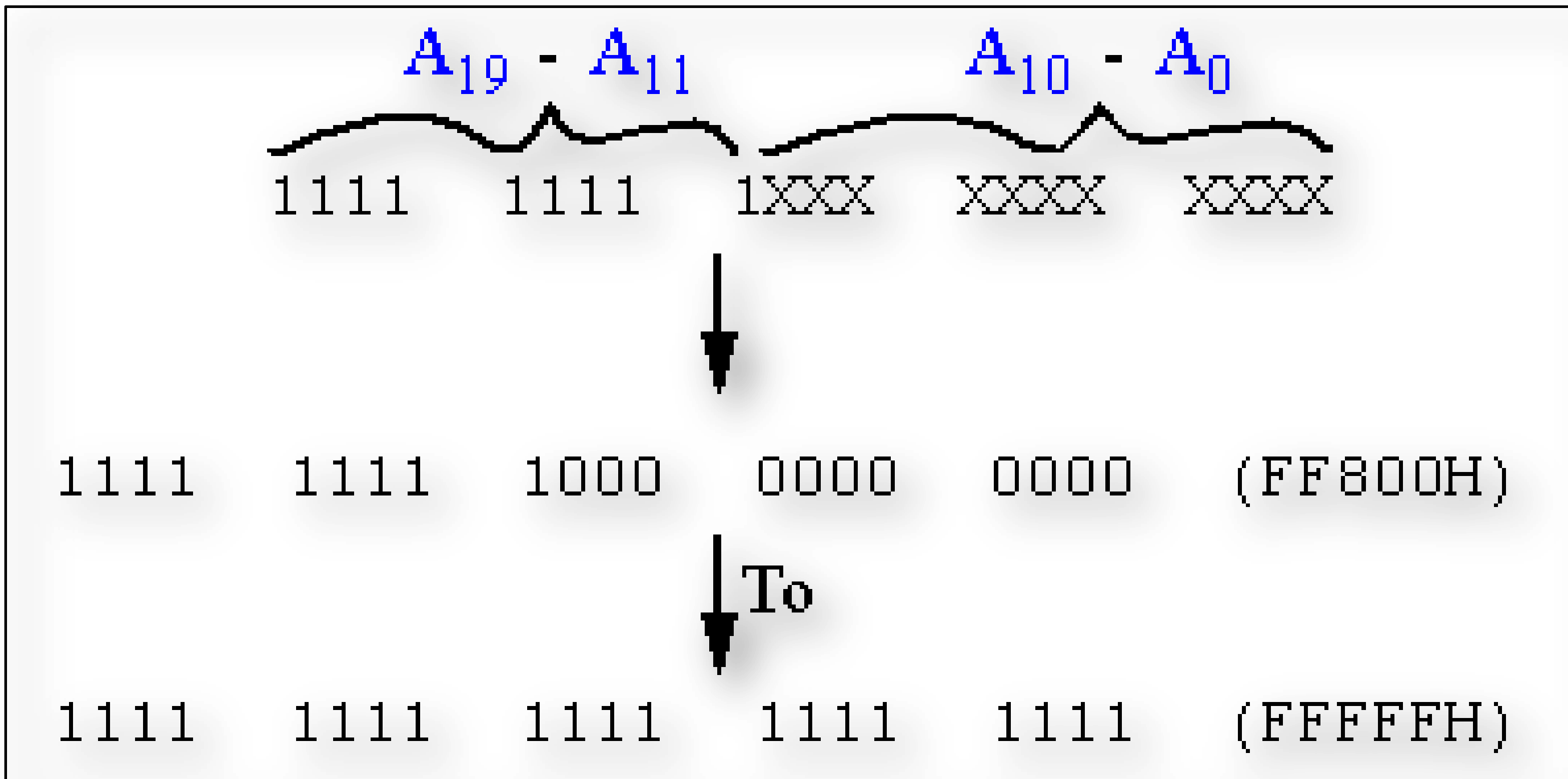


Figure: Memory Address Decoding

Ex. Memory Address Decoding

➢ To determine the address range that a device is mapped into:

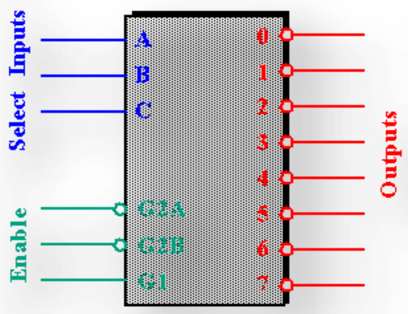


Ex. Memory Address Decoding

- This 2KB memory segment maps into the reset location of the 8086/8088 (FFFF0H).

- NAND gate decoders are not often used. Rather the 3-to-8 Line Decoder (74LS138) is more common.

3-to-8 Line Decoder



Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	0	1	2	3	4	5	6	7
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

- G2A, G2B, and G1 must be active.
- Each output of the decoder can be attached to an 2764 EPROM (8K X 8).

EPROM 2764 x 8

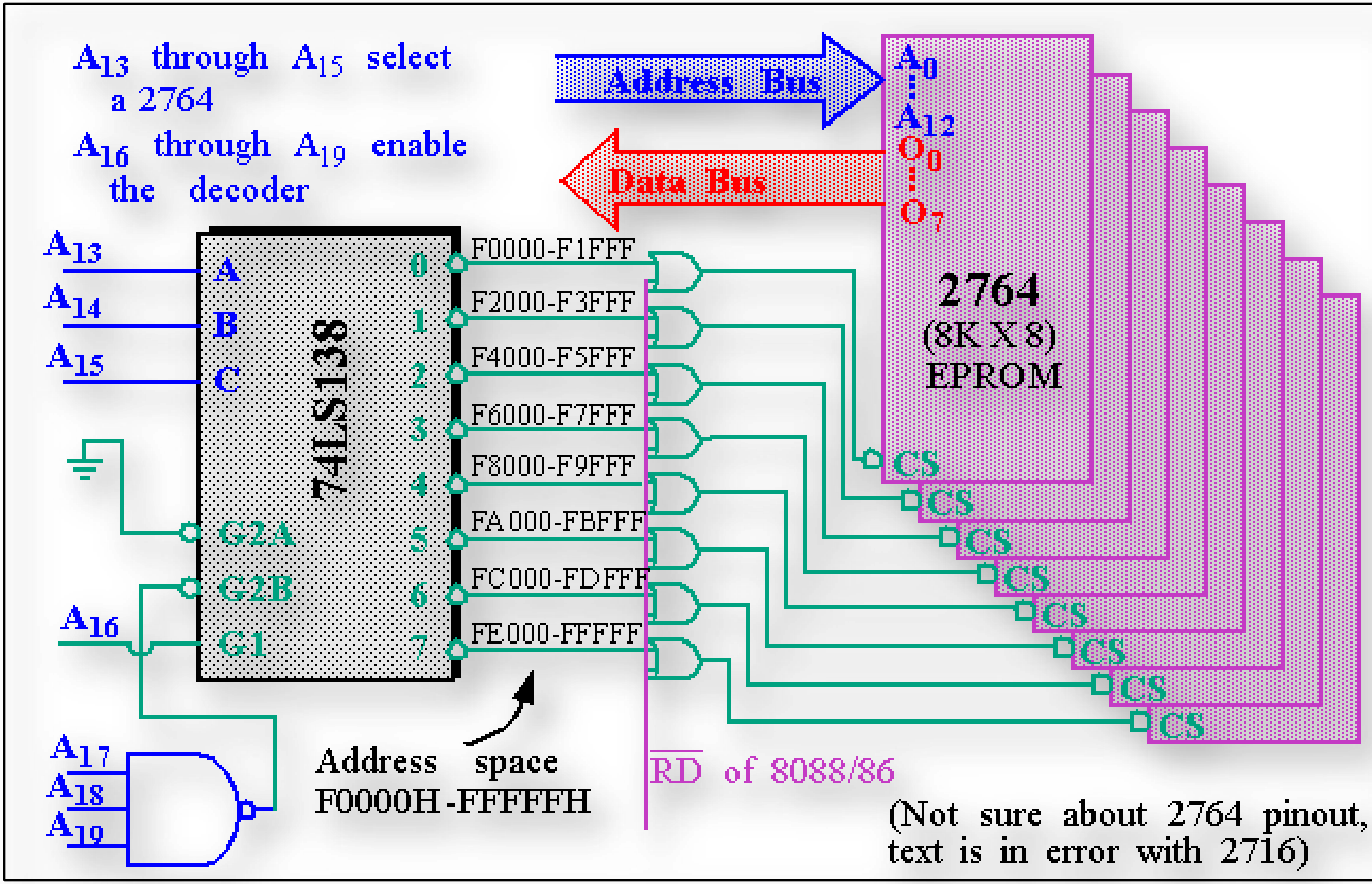


Figure: EPROM 2764 x 8

	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
f0000:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
fffff:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ROMx:								0	0	0	0	0	0	0	0	0	0	0	0	0
								1	1	1	1	1	1	1	1	1	1	1	1	1
ROM1:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
					0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM2:	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
					0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM3:	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
					0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM8:	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

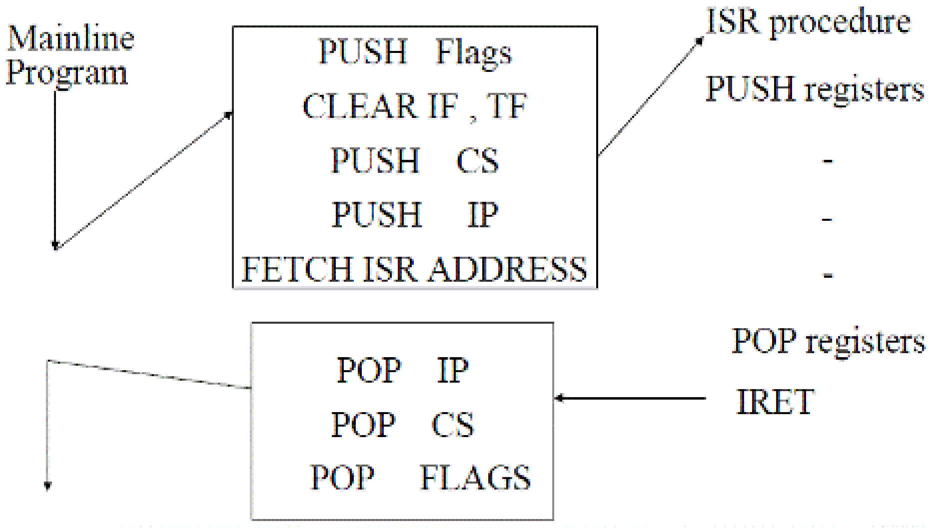
MEMORY INTERFACING WITH 8086

- The semi conductor memories are of two types:
    - Static RAM
    - Dynamic RAM
- The semiconductor memories are organized as two-dimensional arrays of memory locations.
- For Ex: 4K*8 or 4K byte memory contains 8-bit data and only one o the 4096 locations can be selected at a time.
- For addressing 4k bytes of memory, 12 address lines are required.
- For N memory locations, n address lines are required where n = $Log2^N$
- For 4096 Locations, n = $\log 2^{4096}$

N=12

INTERRUPT STRUCTURE OF 8086

- While the CPU is executing a program, an interrupt breaks the normal sequence of execution of instructions, diverts its execution to some other program called “Interrupt Service Routine (ISR).
- •After executing ISR, the control is transferred back again to the main program which was being executed at the time of interruption.



Interrupt Response

- While the CPU is executing a program, an interrupt breaks the normal sequence of execution of instructions, diverts its execution to some other program called “Interrupt Service Routine (ISR).
- After executing ISR, the control is transferred back again to the main program which was being executed at the time of interruption.
- Nested interrupts.
- In 8086, there are two interrupts pins: 1. NMI 2. INTR
- NMI : Non Maskable Interrupt input pin which means that any interrupt request at NMI input cannot to masked or disabled by any means.
- INTR: It can be masked using the Interrupt Flag (IF).

- If more than one type of INTR interrupt occurs at a time, then an external chip called programmable interrupt controller is required to handle them. (eg: 8259 interrupt controller).

- There are two types of interrupts

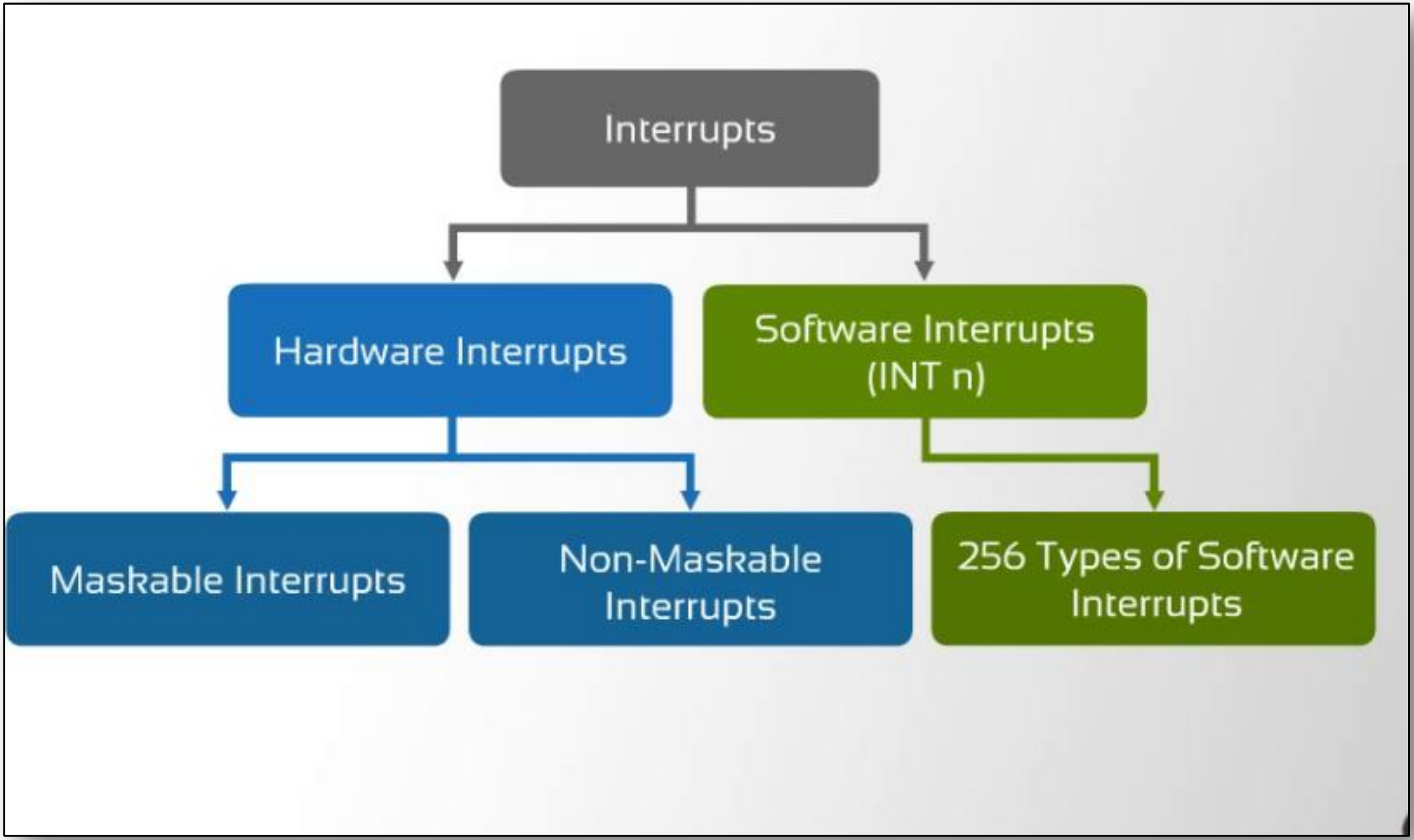
1. **External interrupts**

   These interrupts are generated by external devices i.e out side the processor (using NMI, INTR pins). Eg: Keyboard interrupt.

1. **Internal interrupts**

   It is generated internally by the process circuit or by the execution of an interrupt instruction. Eg: INT instruction, overflow interrupt, divide by zero. At the end of each instruction cycle, the 8086 checks to see if any interrupts have been requested.

Types of interrupts



8086 Interrupt Vector Table

- The first 1Kbyte of memory of 8086 (00000 to 003FF) is set aside as a table for storing the starting addresses of Interrupt Service Procedures (ISP).

- Since 4-bytes are required for storing starting addresses of ISPs, the table can hold 256 Interrupt procedures.

- The starting address of an ISP is often called the **Interrupt Vector** or **Interrupt Pointer**. Therefore the table is referred as **Interrupt Vector Table.**

- In this table, IP value is put in as low word of the vector & CS is put in high vector.

Structure of interrupt vector table

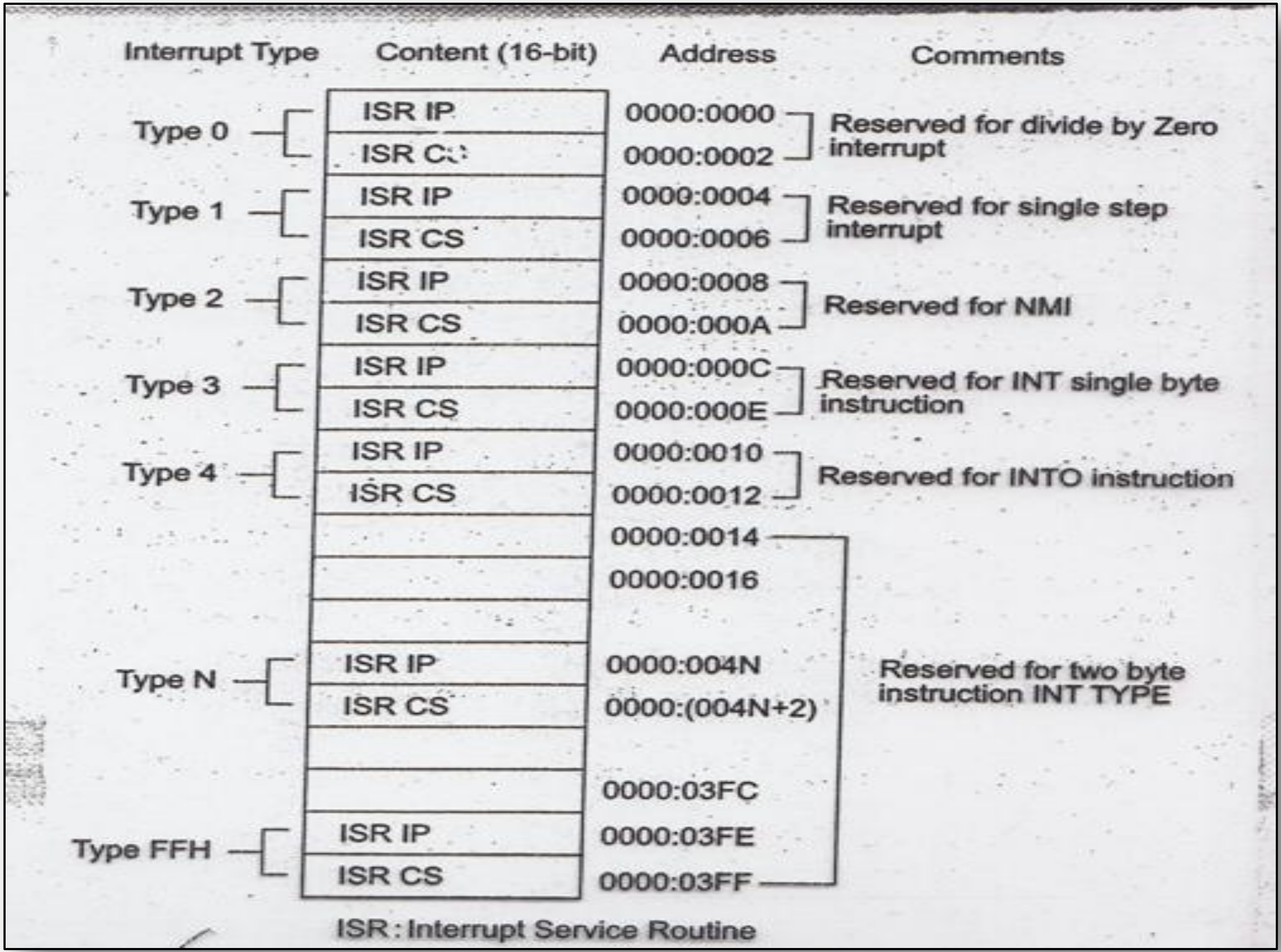


Figure: Structure of interrupt vector table

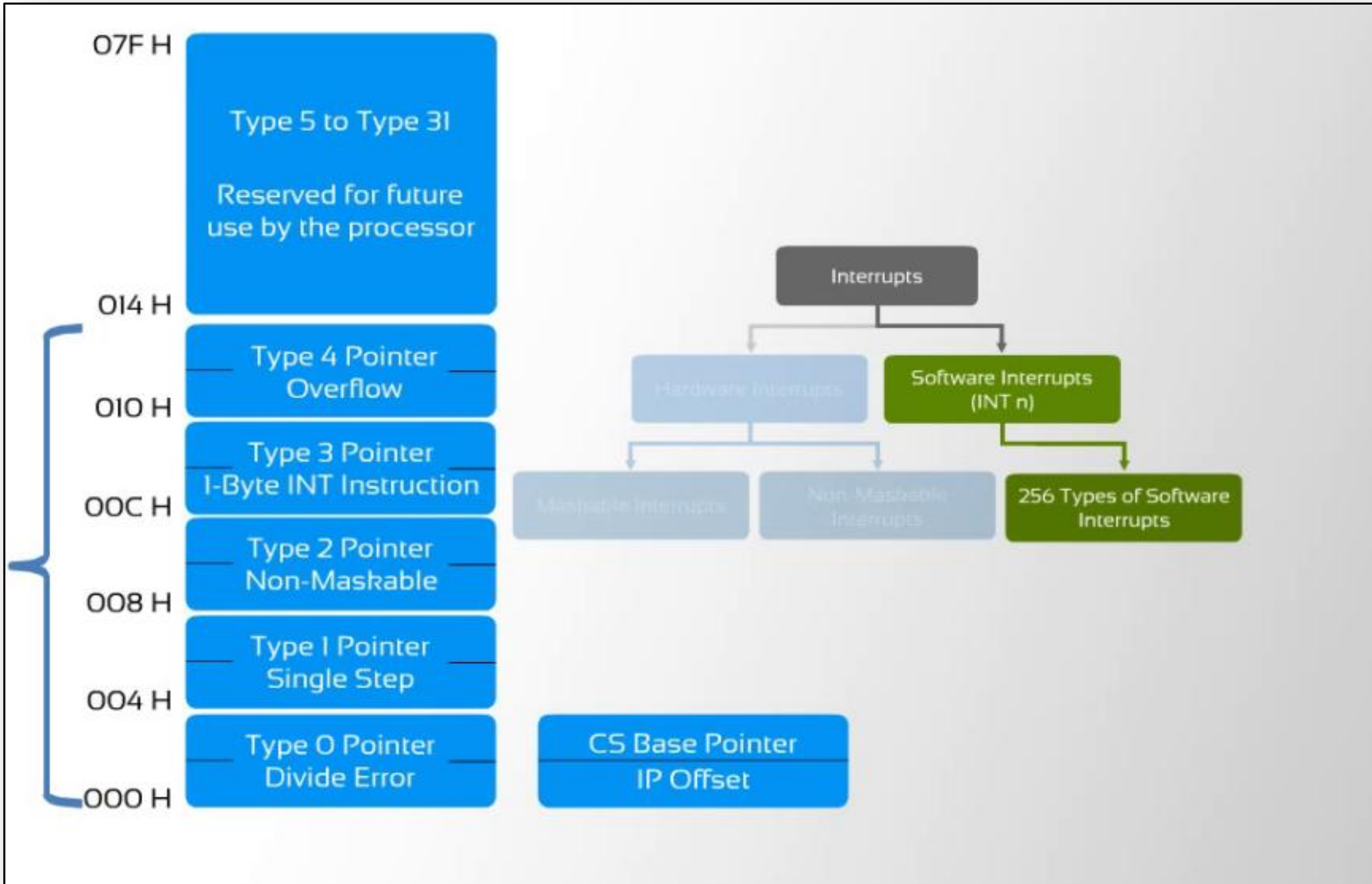


Figure: 8086 Interrupts

Special type interrupts

TYPE 0

The divide error : whenever the results from a division overflows or an attempt is made to divide by zero.

- Type 1: Single-Step Interrupts

  When bit 8 of the FLAGS register (trap flag) is set to 1, an interrupt number 1 is generated after the execution of every instruction. When the microprocessor enters the interrupt service routine, the T flag is automatically cleared (otherwise it would not be able to go beyond the first instruction of the interrupt service routine!).

Type 2

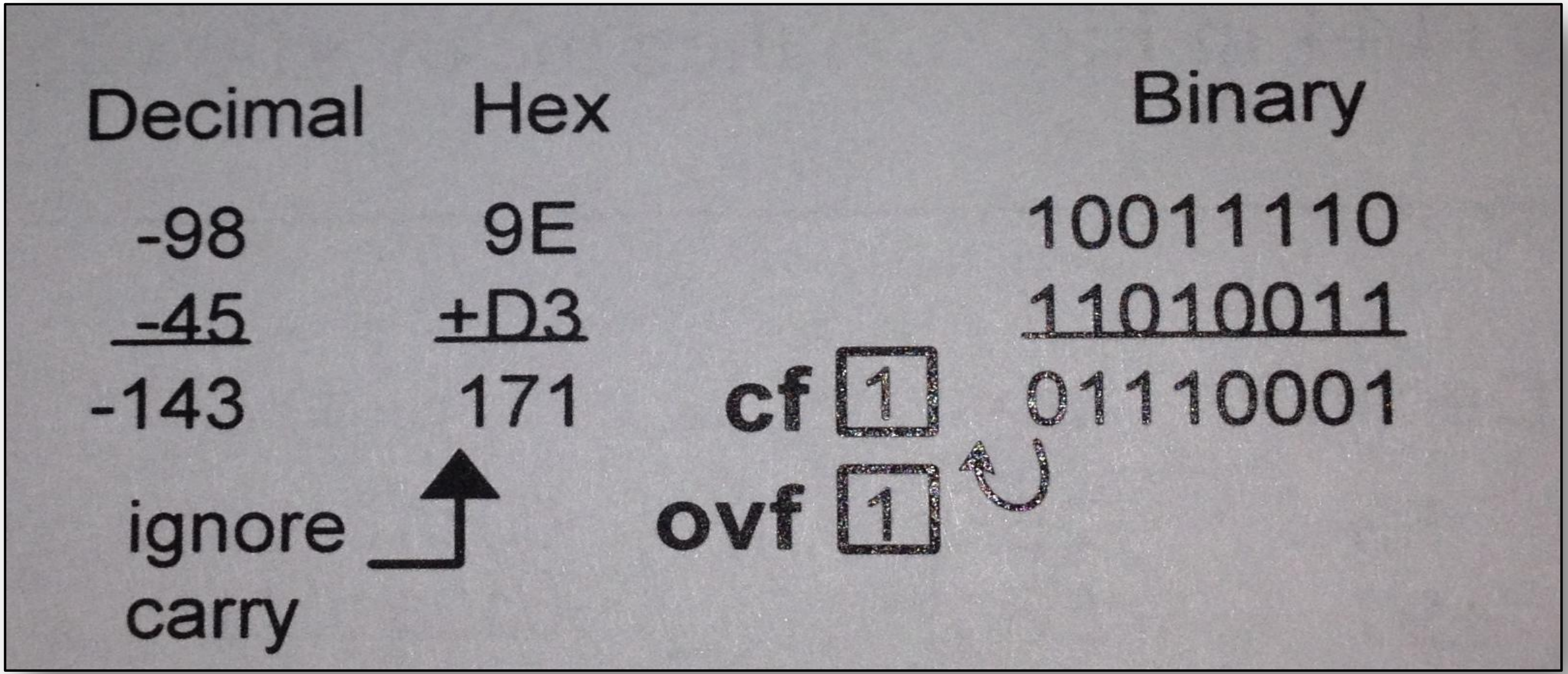
The non-maskable interrupt occurs when a logic 1 is placed on the NMI input pin to the microprocessor. non-maskable—it cannot be disabled

Type 3

A special one-byte instruction (INT 3) that uses this vector to access its interrupt-service procedure. often used to store a breakpoint in a program for debugging.

TYPE 4

Overflow is a special vector used with the INTO instruction. The INTO instruction interrupts the program if an overflow condition exists.



4. **Interrupt Mask Register (IMR):**

This register stores the bits required to mask the interrupts inputs. IMR operates on IRR at the direction of the Priority Resolver.

5. **Interrupt Control logic:**

- This block manages the interrupt and interrupt acknowledge signals to be sent to the CPU for serving one of the 8 interrupt requests.
- This also accepts the interrupt acknowledge (INTA) signal from CPU that causes the 8259A to release vector address on to the data bus.

6. **Data Bus Buffer:-**

- This Tri-state bidirectional buffer interfaces internal 8259A bus the microprocessor data bus.

- Control words, status & vector information pass through data buffer during read or write operations.

7. **Read/Write Control logic:-**

- This circuit accepts and decodes commands from the CPU. This block also allows the status of the 8259A to be transferred on to the data bus.

8. Cascade Buffer/Comparator:-

- This block stores & compares the IDs of all the 8259As used in the system.
- The 3 I/O pins CAS0 – CAS2 are outputs when the 8259A is used as a master.
- The same pins used as inputs when it is in the slave mode.
- 8259A in master mode, sends the ID of the interrupting slave device on these lines. In slave, will send its pre-programmed vector address on the data bus during the next INTA pulse.

Interrupt Sequence in an 8086 system

1. One or more IR lines are raised high that set corresponding IRR bits.
2. 8259A resolves priority and sends an INT signal to CPU.
3. The CPU acknowledges with INTA pulse.
4. Upon receiving an INTA signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data bus during this period.
5. The 8086 will initiate a second INTA pulse. During this period 8259A releases an 8-bit pointer on to data bus from where it is read by the CPU.
6. This completes the interrupt cycle. The ISR bit is reset at the end of the second INTA pulse if automatic end of interrupt (AEOI) mode is programmed. Otherwise ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

Once ICW_1 is loaded, the following initialization procedure is carried out internally.

a) The edge sense circuit is reset, i.e by default 8259A interrupts are edge sensitive

b) IMR is cleared

c) IR7 input is assigned the lowest priority

d) Slave mode address is set to 7

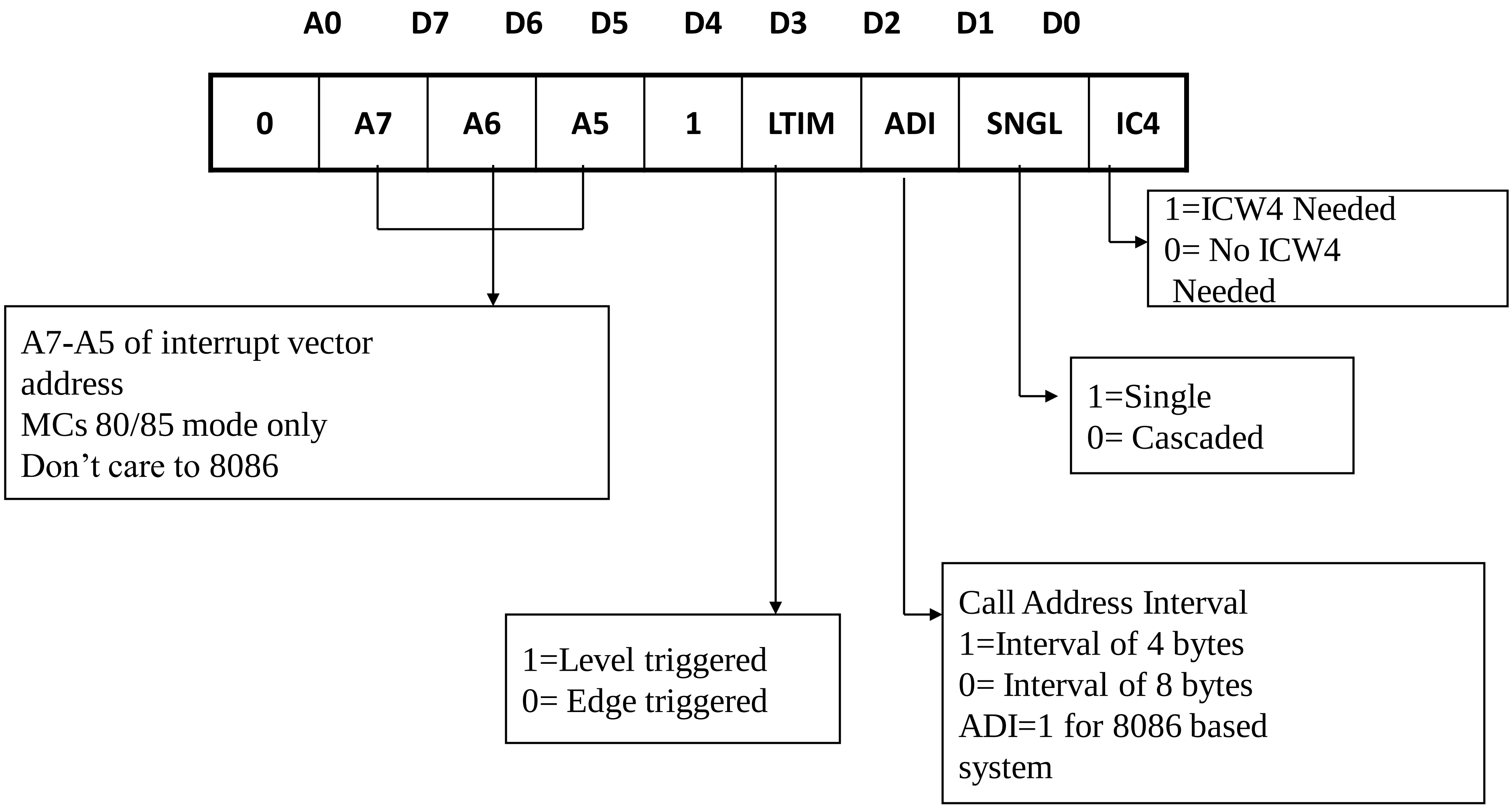
e) Special mask mode is cleared and the status read is set to IRR

f) If IC_4 =0, all the functions of ICW_4 are set to zero . Master/slave bit in ICW_4 is used in the buffered mode only.

ICW_1 , ICW_2 ---- are compulsory

ICW_3 , ICW_4 -- are optional.

ICW1


A0
D7
D6
D5
D4
D3
D2
D1
D0
0
A7
A6
A5
1
LTIM
ADI
SNGL
IC4
1=ICW4 Needed
0= No ICW4 Needed
A7-A5 of interrupt vector address
MCs 80/85 mode only
Don’t care to 8086
1=Single
0= Cascaded
Call Address Interval
1=Interval of 4 bytes
0= Interval of 8 bytes
ADI=1 for 8086 based system
1=Level triggered
0= Edge triggered

ICW2

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	T 7	T6	T5	T4	T3	A10	A9	A8

For 8085 system:

T7-T3 : they are filled by A15-A11 of the Interrupt Vector Address

A10-A8: these bits are same as the respective bits of vector address

For 8086 system:

T7-T3 : Interrupt type

A10-A8: 3 bits are 0, pointing to IR0.

ICW3

a) **Master Mode**: SP=1, in buffer mode M / S =1 in ICW4

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	**S7**	**S6**	**S5**	**S4**	**S3**	**S2**	**S1**	**S0**

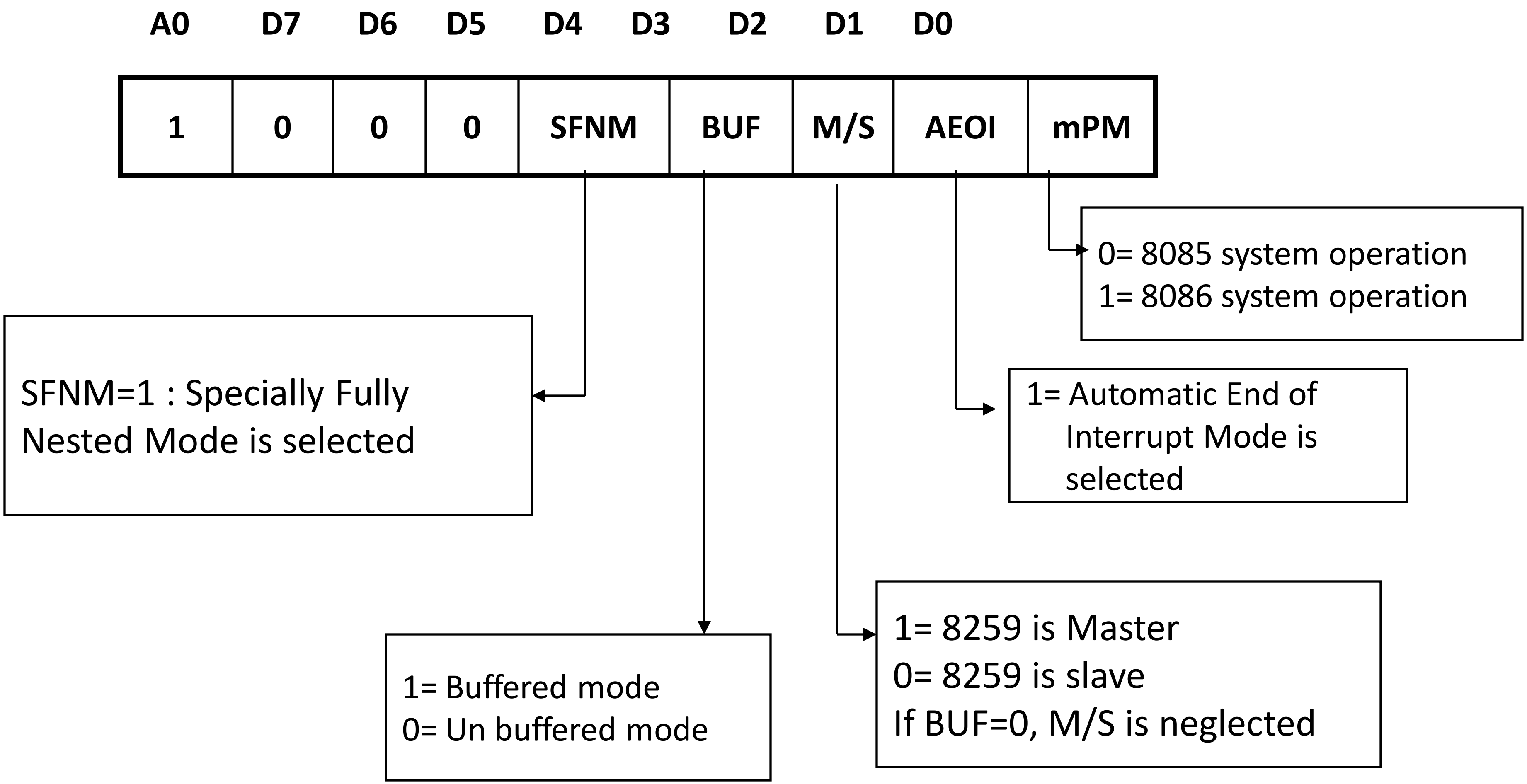
Sn = 1 → IRn input has a slave
Sn = 0 → IRn input does not have a slave

b) **Slave Mode**: SP=0, in buffer mode M / S = 0 in ICW4

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	**0**	**0**	**0**	**0**	**0**	**ID2**	**ID1**	**ID0**

ID2-ID0 → 000 to 111 for IR0-IR7 i.e slave1 to slave8

ICW4



A0
D7
D6
D5
D4
D3
D2
D1
D0
1
0
0
0
SFNM
BUF
M/S
AEOI
mPM
0= 8085 system operation
1= 8086 system operation
SFNM=1 : Specially Fully Nested Mode is selected
1= Automatic End of Interrupt Mode is selected
1= Buffered mode
0= Un buffered mode
1= 8259 is Master
0= 8259 is slave
If BUF=0, M/S is neglected

Operation command words (OCWs)

- Once ICW registers (accepting the interrupts) are initialized, 8259 is ready for its normal function.

- 8259 has its own ways of handling the received interrupts called as modes of operation. These can be selected by programming i.e writing 3 OCW registers.

- OCW1: It is for mask the unwanted interrupt requests.
- OCW2: It controls the end of interrupt, the rotate mode and their combination

- OCW3: It is for set or reset for special mask mode

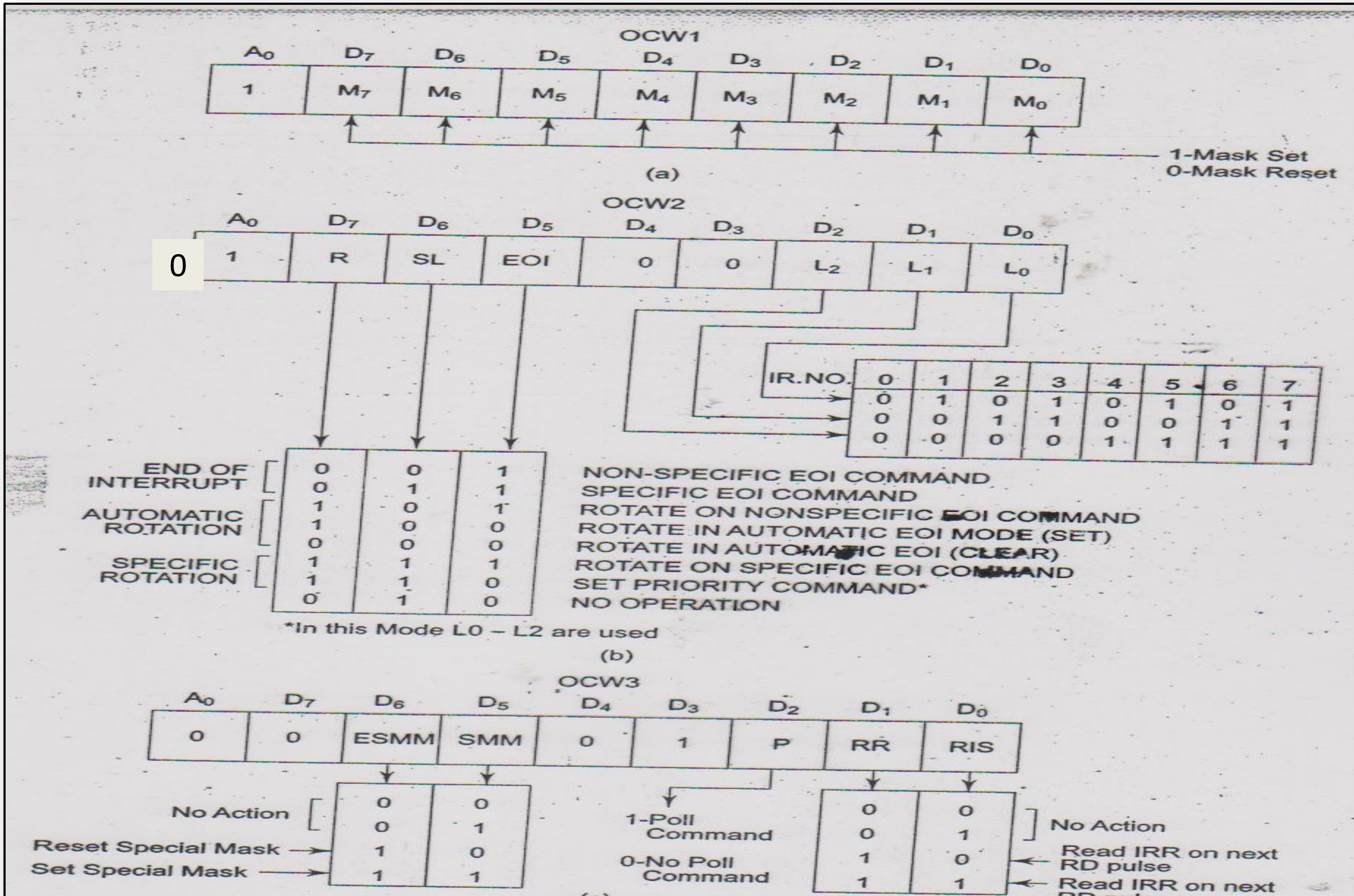


Fig. 6.18 *Operation Command Words*

Figure: Operation Command Words

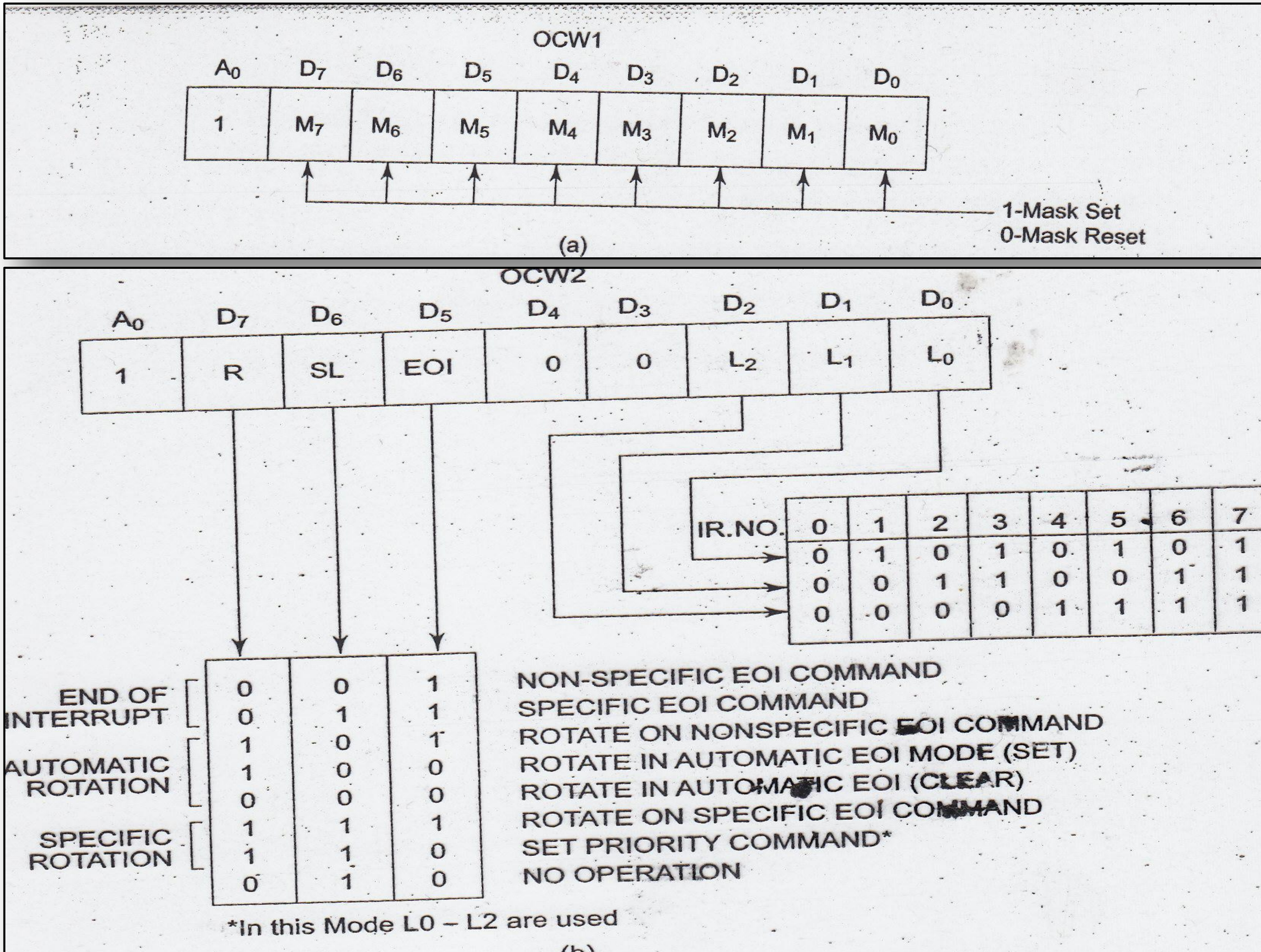
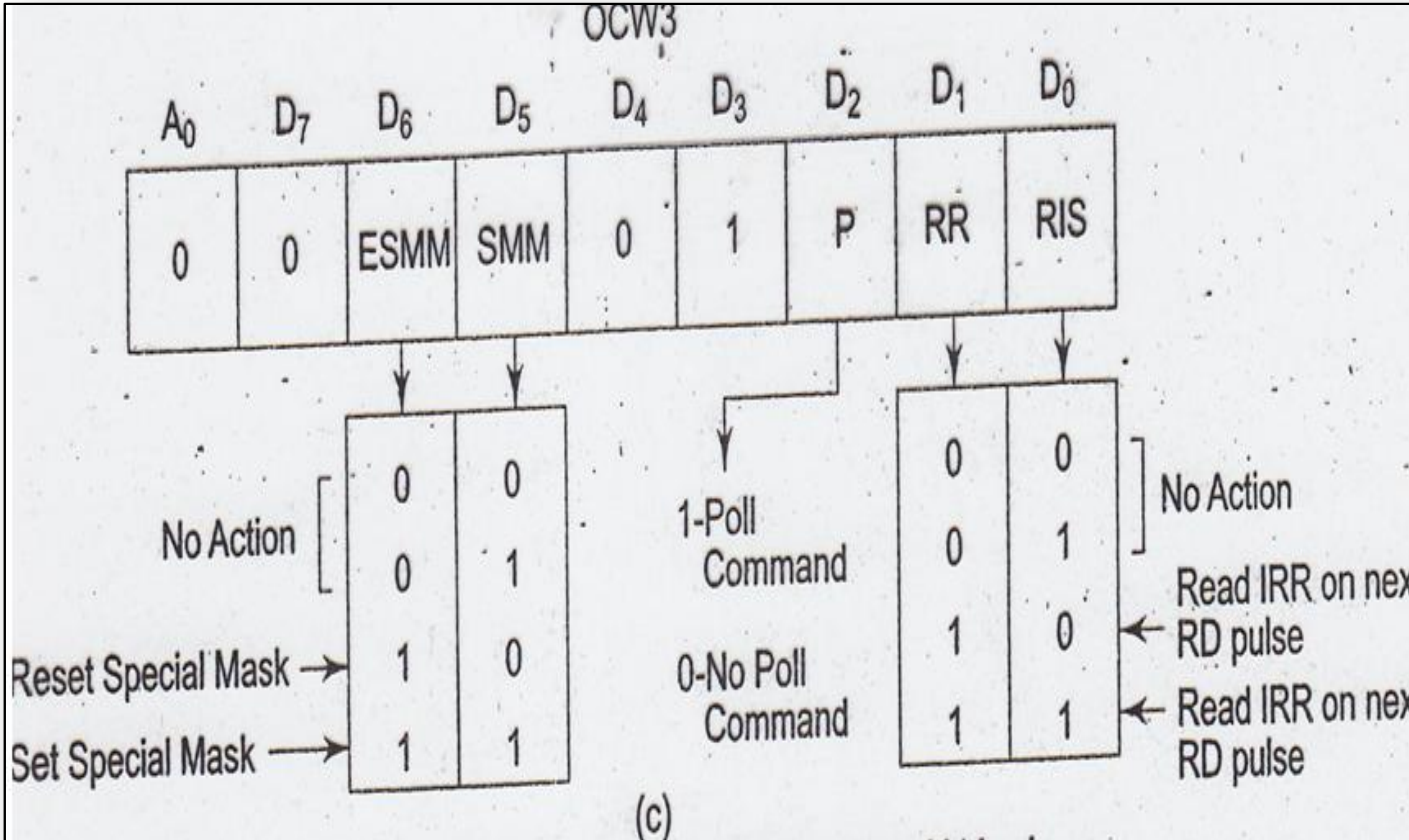


Figure: Operation Command Words



OCW 3

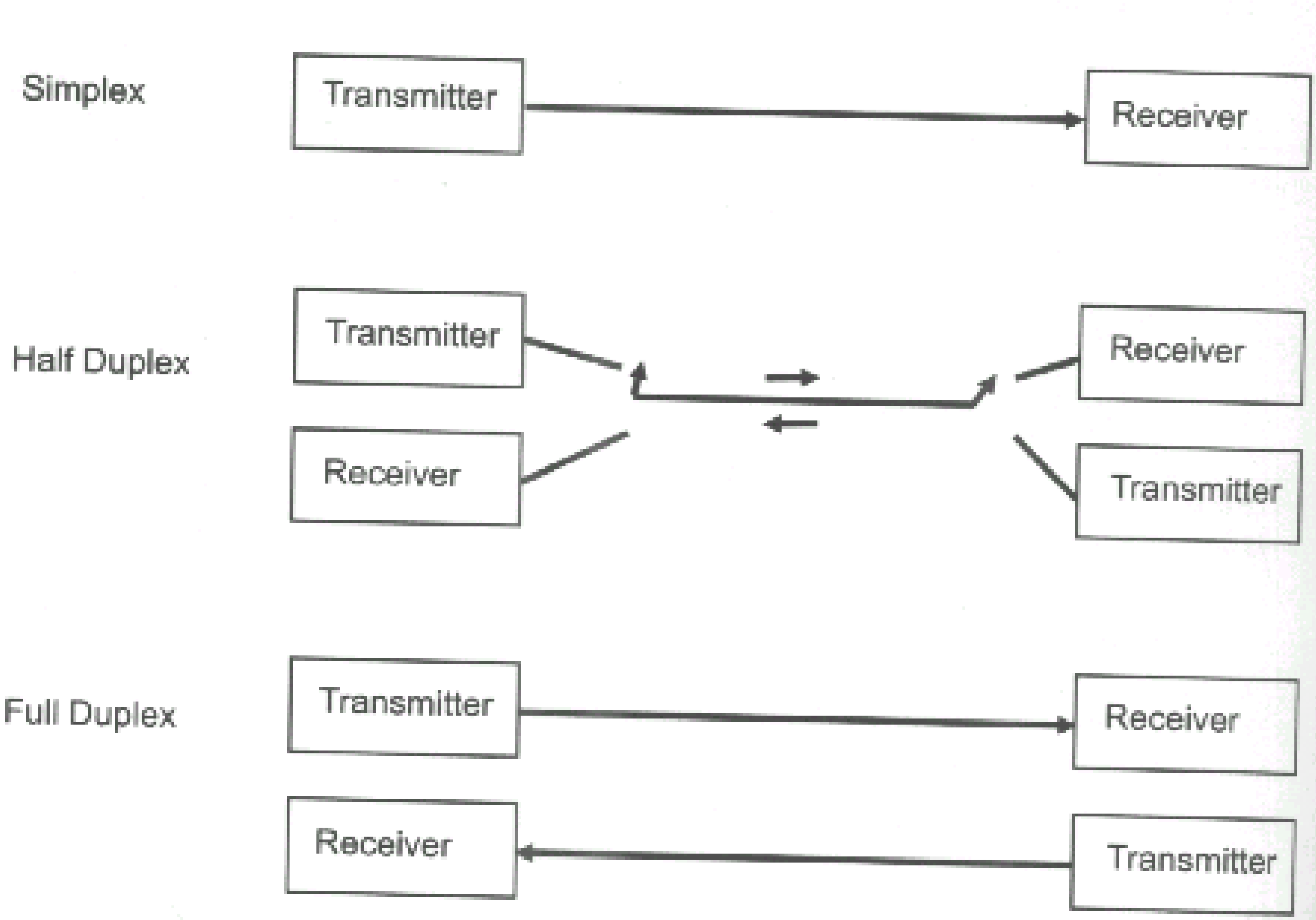
Figure: Operation Command Words

Communication Interface

Data Communications

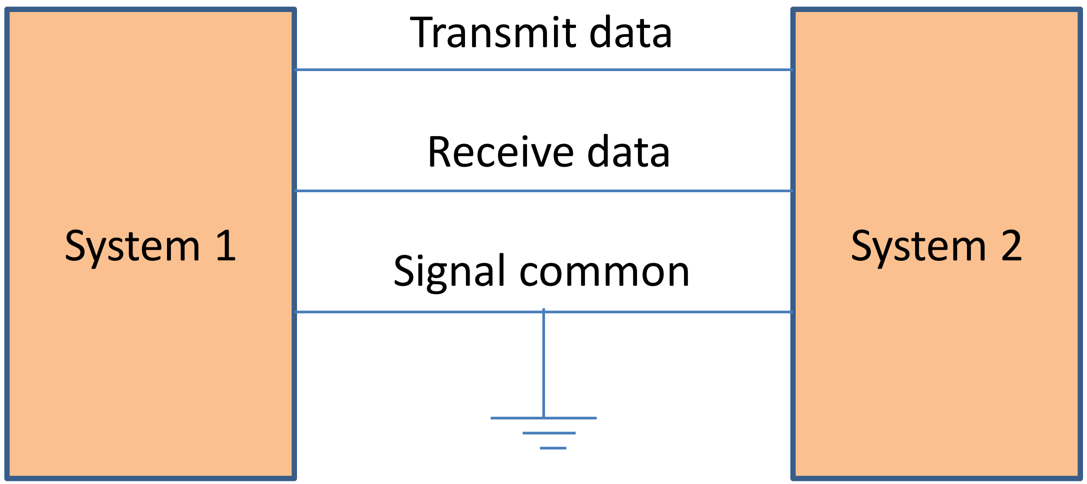
- Data communications refers to the ability of one computer to exchange data with another computer or a peripheral
- Physically, the data comm. path may be a short, 5 to 10 feet ribbon cable connecting a microcomputer and parallel printer; or it might be a high speed telecommunications port connecting two computers thousands of miles apart.
- Standard data communication interfaces and standards are needed
- Centronic's parallel printer interface
- RS-232 defines a serial communications standard
- 8251 USART (Universal Synchronous/Asynchronous Receiver/Transmitter) is the key component for converting parallel data to serial form and vice versa
- Two types of serial data communications are widely used
    - Asynchronous communications
    - Synchronous communications

Types of Transmission



Asynchronous Communications

- Eliminates the need for a clock signal between two microprocessor based systems

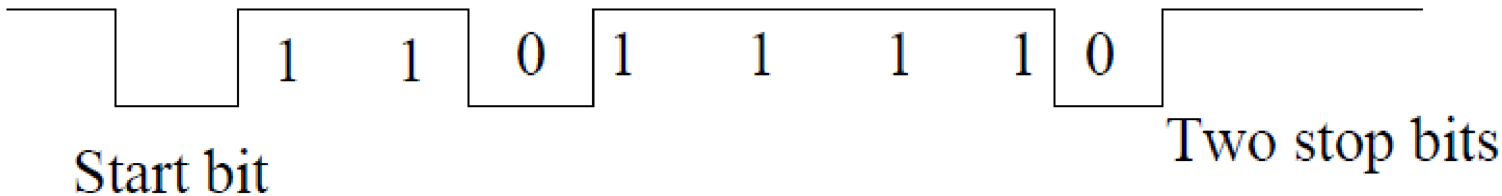


Asynchronous Transmission

- Asynchronous data transfer: sender provides a synchronization signal to the receiver before starting the transfer of each message
  - does not need clock signal between the sender and the receiver
  - slower data transfer rate

Asynchronous Communications

- Data to be transmitted is sent out one character at a time and the receiver end of the communication line synchronization is performed by examining synchronization bits that are included at the beginning and at the end of each character.



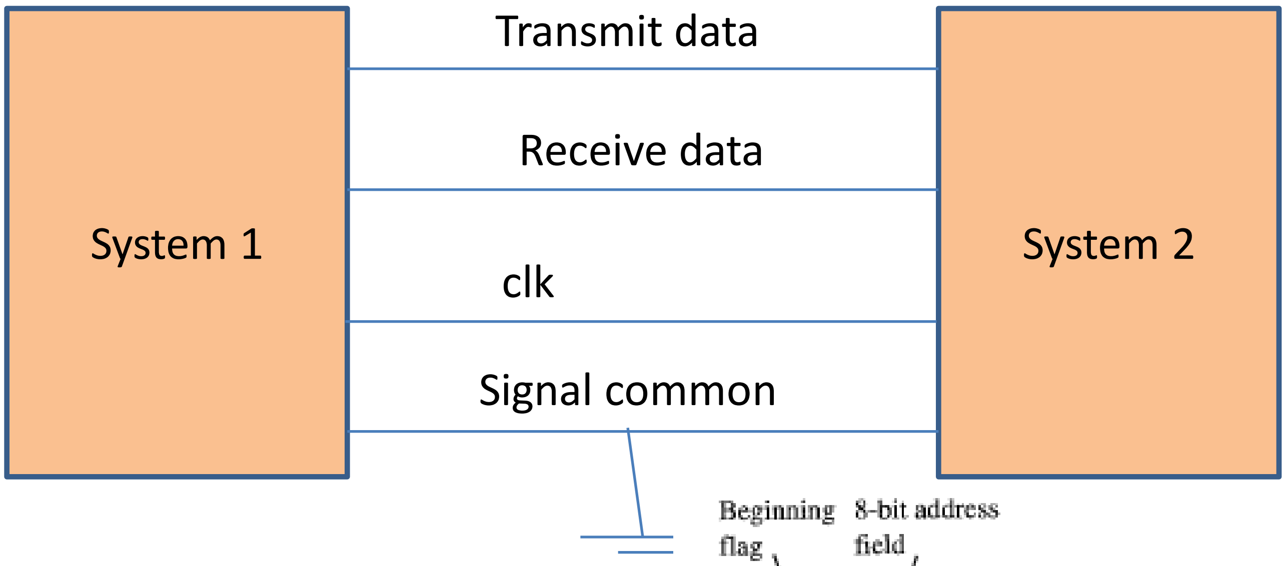
Examples

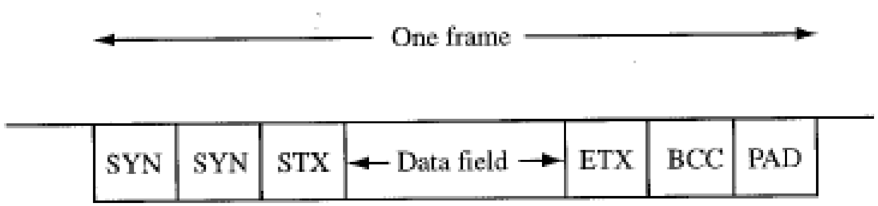
- What is the data rate in bits/sec and character rate if the bit time is 3.33 ms
  - Bit rate = 1 / 3.33 ms = 300 bits/sec
  - 11 x 3.33 ms = 36.63 ms required to transmit a character so character rate = 1/36.63 ms = 27.3 char/sec
- Modems typically transmit data over the telephone network at 9600, 14400, 28800 or 56K bps.
- Ex: If 1 MByte file is to be transmitted to another computer using a modem calculate the transmission time
  - 9600 bps: 1048576x10/ 9600 bits/sec = 1092 s = 18 minutes and 12 sec
  - 28800 bps: 364 s = 6 minutes and 4 sec

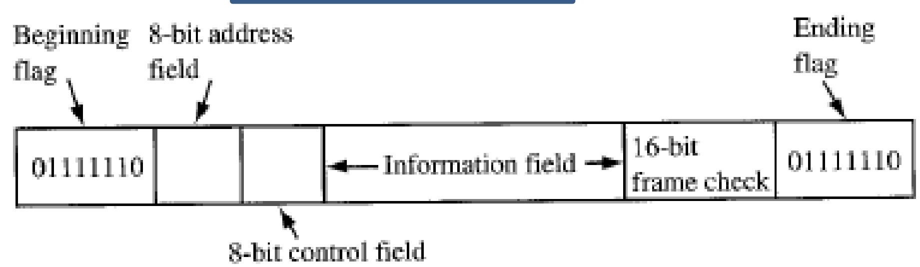
Synchronous Transmission

- Synchronous data transfer: sender and receiver use the same clock signal
  - supports high data transfer rate
  - needs clock signal between the sender and the receiver
  - requires master/slave configuration

Synchronous Communications







BISYNC: Each block of data has synch characters. The size of block data can be 100 or more bytes. BCC checks for errors.

Serial Data Link Control: Developed by IBM used for computer networking (Token Ring). After Flag byte the network address is sent. Control Byte stores information about sequence of data etc. Data is thousands of bits. 16 bit field is used for error checking.

USART Introduction

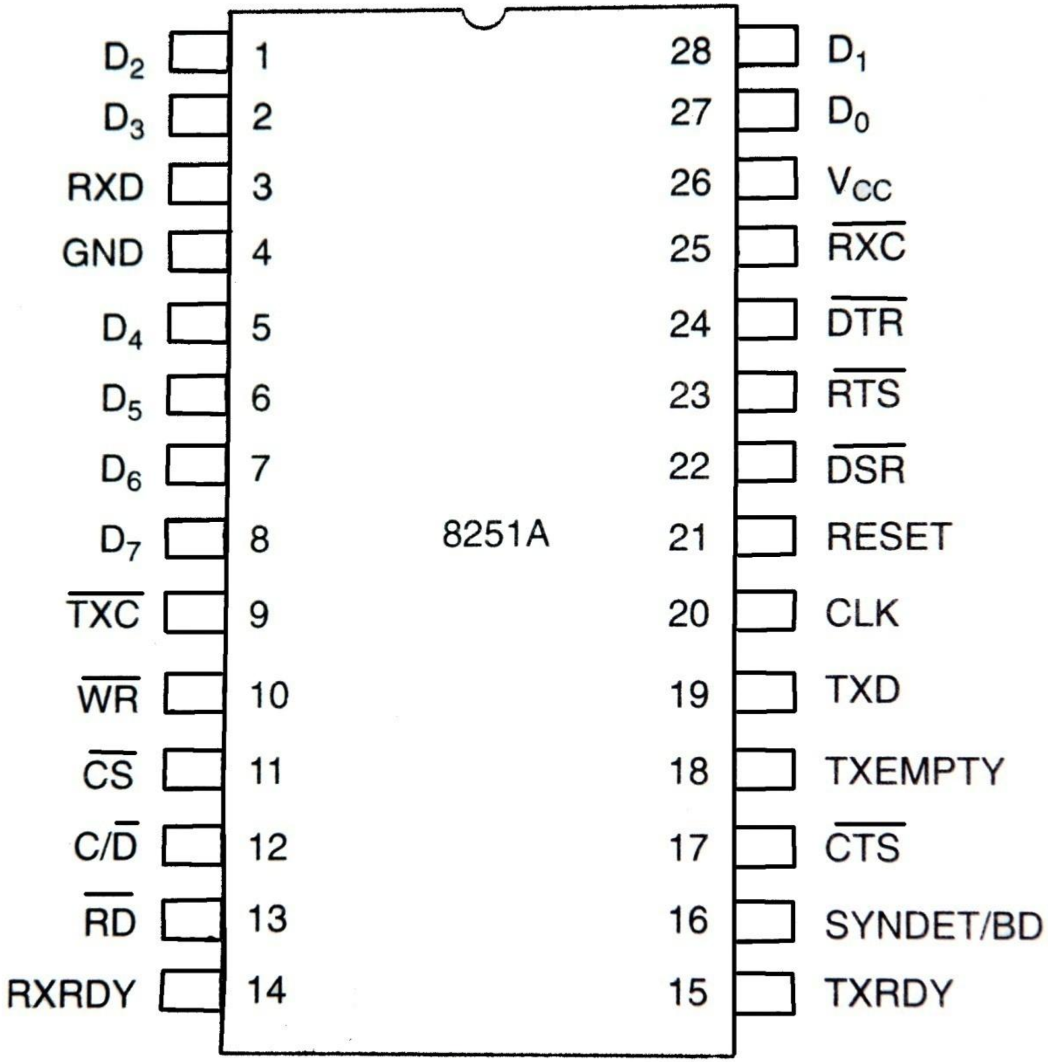
- 8251A is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication.

- Programmable peripheral designed for synchronous /asynchronous serial data communication, packaged in a 28-pin DIP.

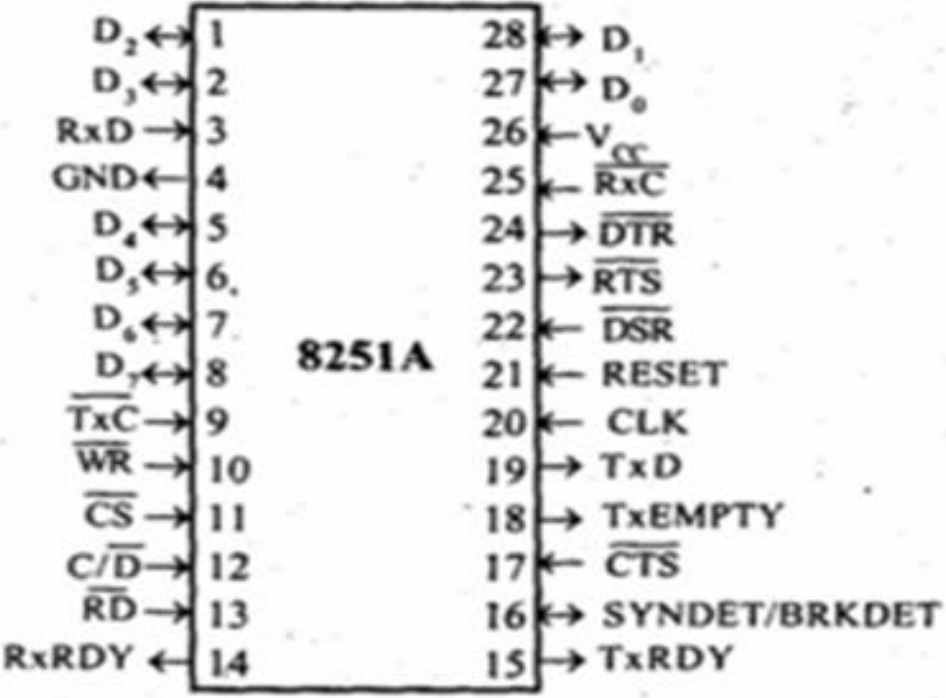
- Receives parallel data from the CPU & transmits serial data after conversion.

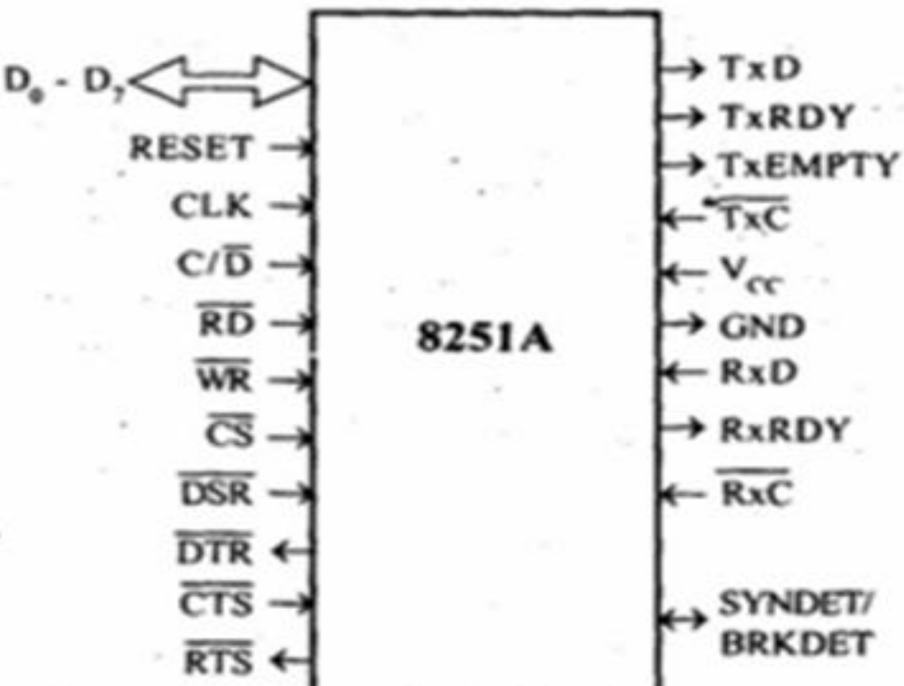
- Also receives serial data from the outside & transmits parallel data to the CPU after conversion.

Pin diagram



D2
D3
RXD
GND
D4
D5
D6
D7
TXC
WR
CS
C/D
RD
RXRDY
1
2
3
4
5
6
7
8
9
10
11
12
13
14
8251A
28
27
26
25
24
23
22
21
20
19
18
17
16
15
D1
D0
VCC
RXC
DTR
RTS
DSR
RESET
CLK
TXD
TXEMPTY
CTS
SYNDET/BD
TXRDY





Pin	Description
D_0-D_7	Parallel data
$C/\overline{D}$	Control register or Data buffer select
$\overline{RD}$	Read control
$\overline{WR}$	Write control
$\overline{CS}$	Chip Select
CLK	Clock pulse (TTL)
RESET	Reset
$\overline{TxC}$	Transmitter Clock
TxD	Transmitter Data
$\overline{RxC}$	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready
TxRDY	Transmitter Ready
$\overline{DSR}$	Data Set Ready
$\overline{DTR}$	Data Terminal Ready
SYNDET/ BRKDET	Synchronous Detect / Break Detect
$\overline{RTS}$	Reqnest To Send Data
$\overline{CTS}$	Clear To Send Data
TxEMPTY	Transmitter Empty
V_{cc}	Supply (+5V)
GND	Ground (0 V)

Block diagram of the 8251 USART

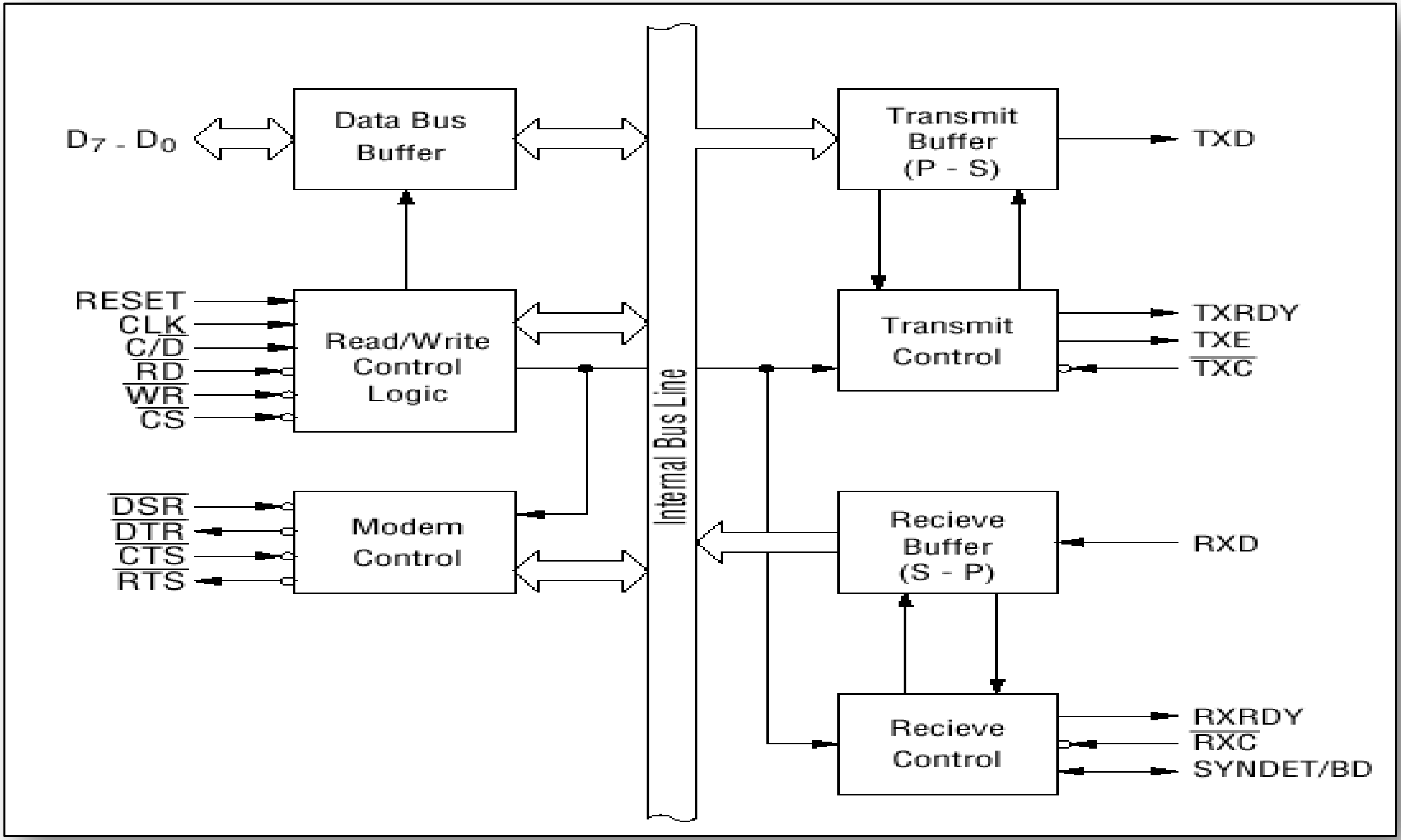


Figure: Block diagram of the 8251 USART

Sections of 8251A

- Data Bus buffer
- Read/Write Control Logic
- Modem Control
- Transmitter
- Receiver

1. Data Bus Buffer

- D0-D7 : 8-bit data bus used to read or write status, command word or data from or to the 8251A

2. Read/Write Control logic

- Includes a control logic, six input signals & three buffer registers: Data register, control register & status register.

- Control logic : Interfaces the chip with MPU, determines the functions of the chip according to the control word in the control register & monitors the data flow.

Input signals

- $\overline{CS}$ – Chip Select : When signal goes low, the 8251A is selected by the MPU for communication.
- C/$\overline{D}$ – Control/Data : When signal is high, the control or status register is addressed; when it is low, data buffer is addressed. (Control register & status register are differentiated by WR and RD signals)
- $\overline{WR}$: When signal is low, the MPU either writes in the control register or sends output to the data buffer.
- $\overline{RD}$: When signal goes low, the MPU either reads a status from the status register or accepts data from data buffer.
- RESET : A high on this signal reset 8252A & forces it into the idle mode.
- CLK : Clock input, usually connected to the system clock for communication with the microprocessor.

Control Register

- 16-bit register for a control word consist of two independent bytes namely mode word & command word.
- Mode word : Specifies the general characteristics of operation such as baud, parity, number of bits etc.
- Command word : Enables the data transmission and reception.
- Register can be accessed as an output port when the Control/Data pin is high.

Status register

- Checks the ready status of the peripheral.
- Status word in the status register provides the information concerning register status and transmission errors.

Data register

- Used as an input and output port when the C/D is low

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	
1	×	×	×	Data Bus 3-State
0	×	1	1	Data Bus 3-State
0	1	0	1	Status → CPU
0	1	1	0	Control Word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

3. Modem Control

- $\overline{\text{DSR}}$ - Data Set Ready : Checks if the Data Set is ready when communicating with a modem.

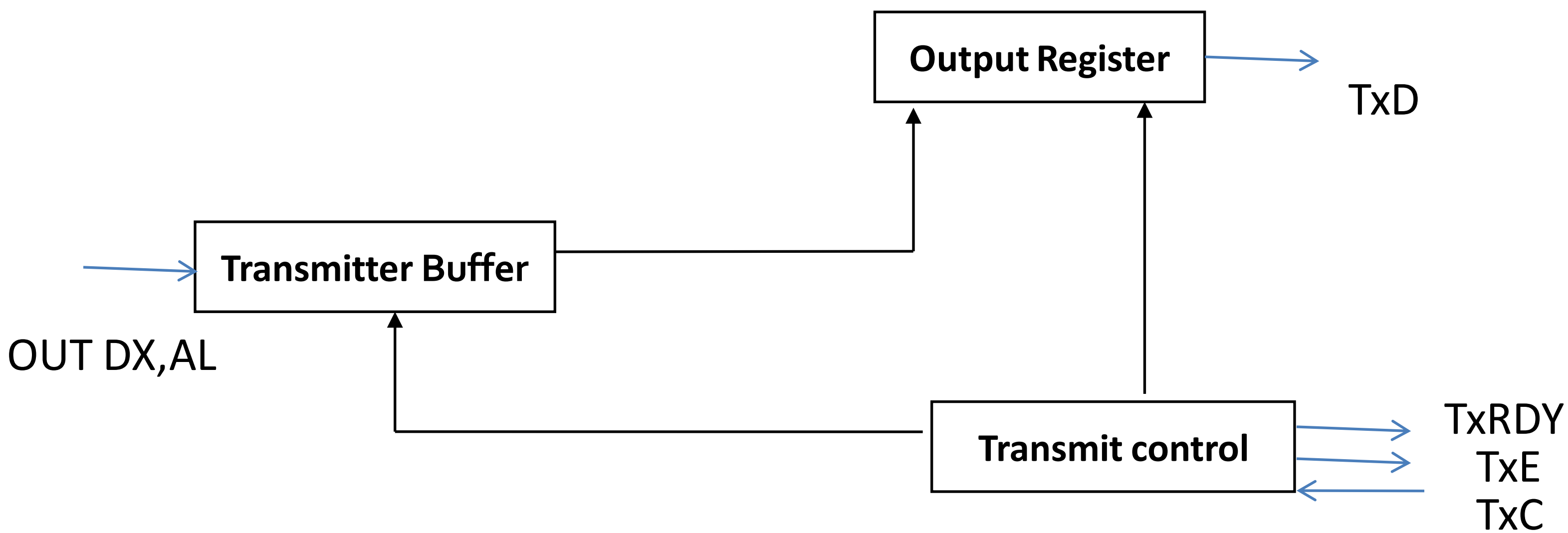
- $\overline{\text{DTR}}$ - Data Terminal Ready : Indicates that the device is ready to accept data when the 8251 is communicating with a modem.

- $\overline{\text{CTS}}$ - Clear to Send : If its low, the 8251A is enabled to transmit the serial data provided the enable bit in the command byte is set to '1'.

- $\overline{\text{RTS}}$ - Request to Send Data : Low signal indicates the modem that the receiver is ready to receive a data byte from the modem.

4. Transmitter section

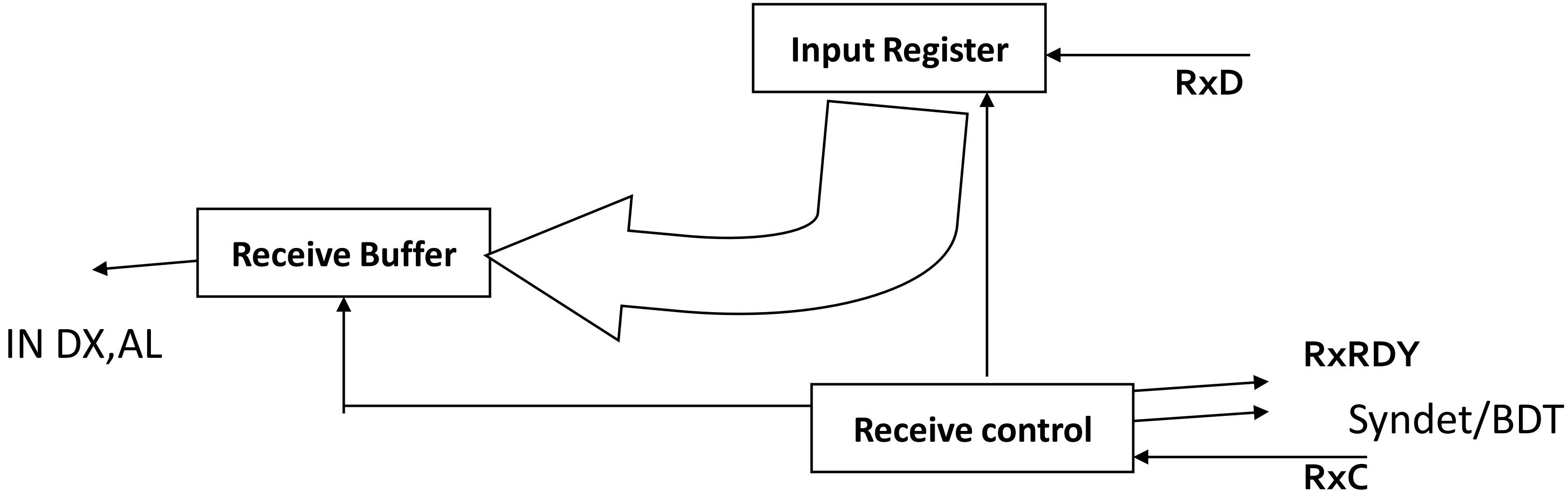
- Accepts parallel data from MPU & converts them into serial data.
- Has two registers:
  - Buffer register : To hold eight bits
  - Output register : To convert eight bits into a stream of serial bits.



- The MPU writes a byte in the buffer register.
- Whenever the output register is empty; the contents of buffer register are transferred to output register.
- Transmitter section consists of three output & one input signals
    - TxD - Transmitted Data Output : Output signal to transmit the data to peripherals.
    - $\overline{TxC}$ - Transmitter Clock Input : Input signal, controls the rate of transmission.
    - TxRDY - Transmitter Ready : Output signal, indicates the buffer register is empty and the USART is ready to accept the next data byte.
    - TxE - Transmitter Empty : Output signal to indicate the output register is empty and the USART is ready to accept the next data byte.

5. Receiver Section

- Accepts serial data on the RxD pin and converts them to parallel data.
- Has two registers :
  - Receiver input register
  - Buffer register



- When RxD goes low, the control logic assumes it is a start bit, waits for half bit time, and samples the line again. If the line is still low, the input register accepts the following data, and loads it into buffer register at the rate determined by the receiver clock.
- RxRDY - Receiver Ready Output: Output signal, goes high when the USART has a character in the buffer register & is ready to transfer it to the MPU.
- RxD - Receive Data Input : Bits are received serially on this line & converted into a parallel byte in the receiver input register.
- $\overline{RxC}$ - Receiver Clock Input : Clock signal that controls the rate at which bits are received by the USART.

Mode word & command word for 8251 USART

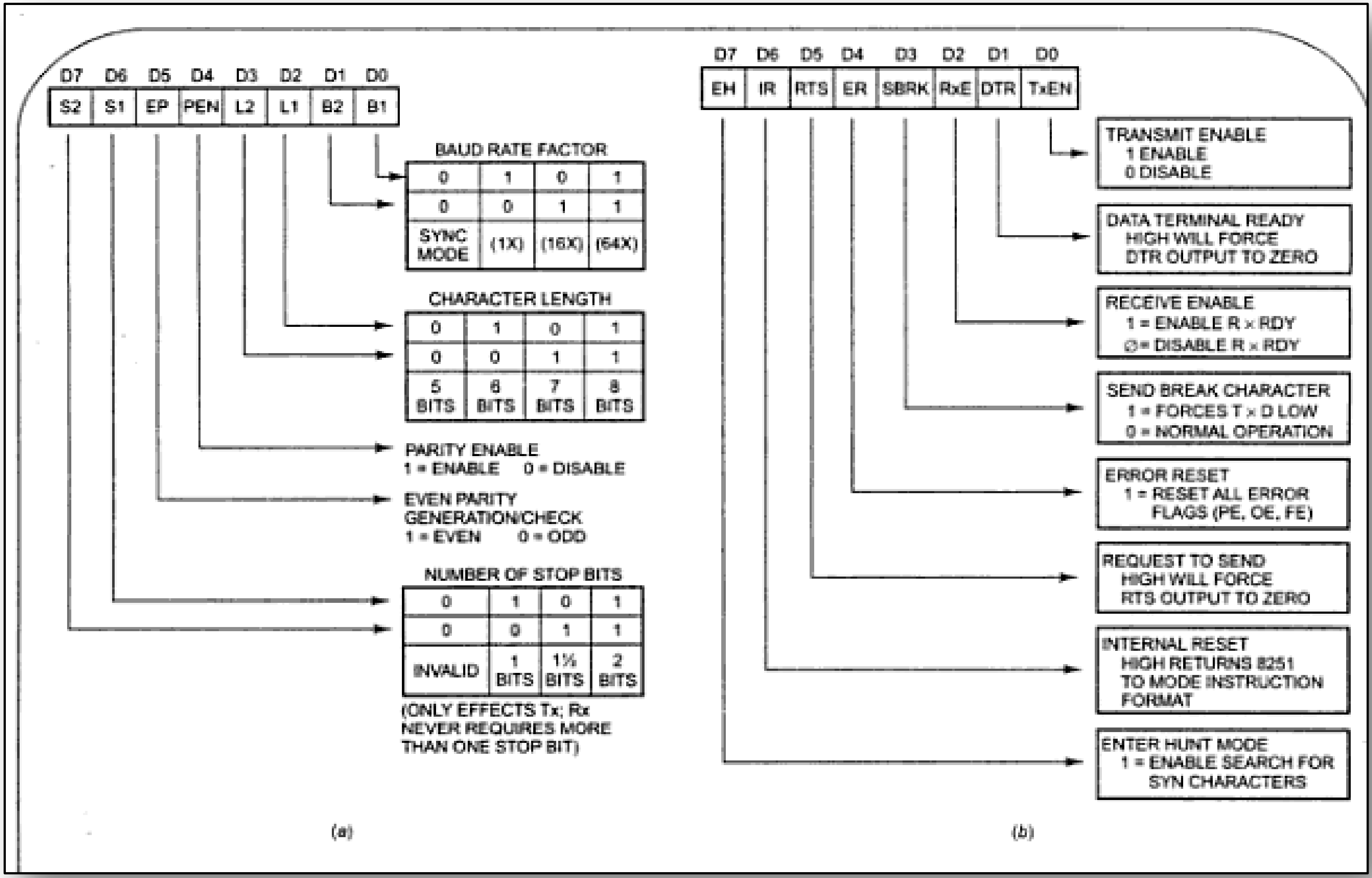


Figure: Mode word & command word for 8251 USART

Status word register of 8251

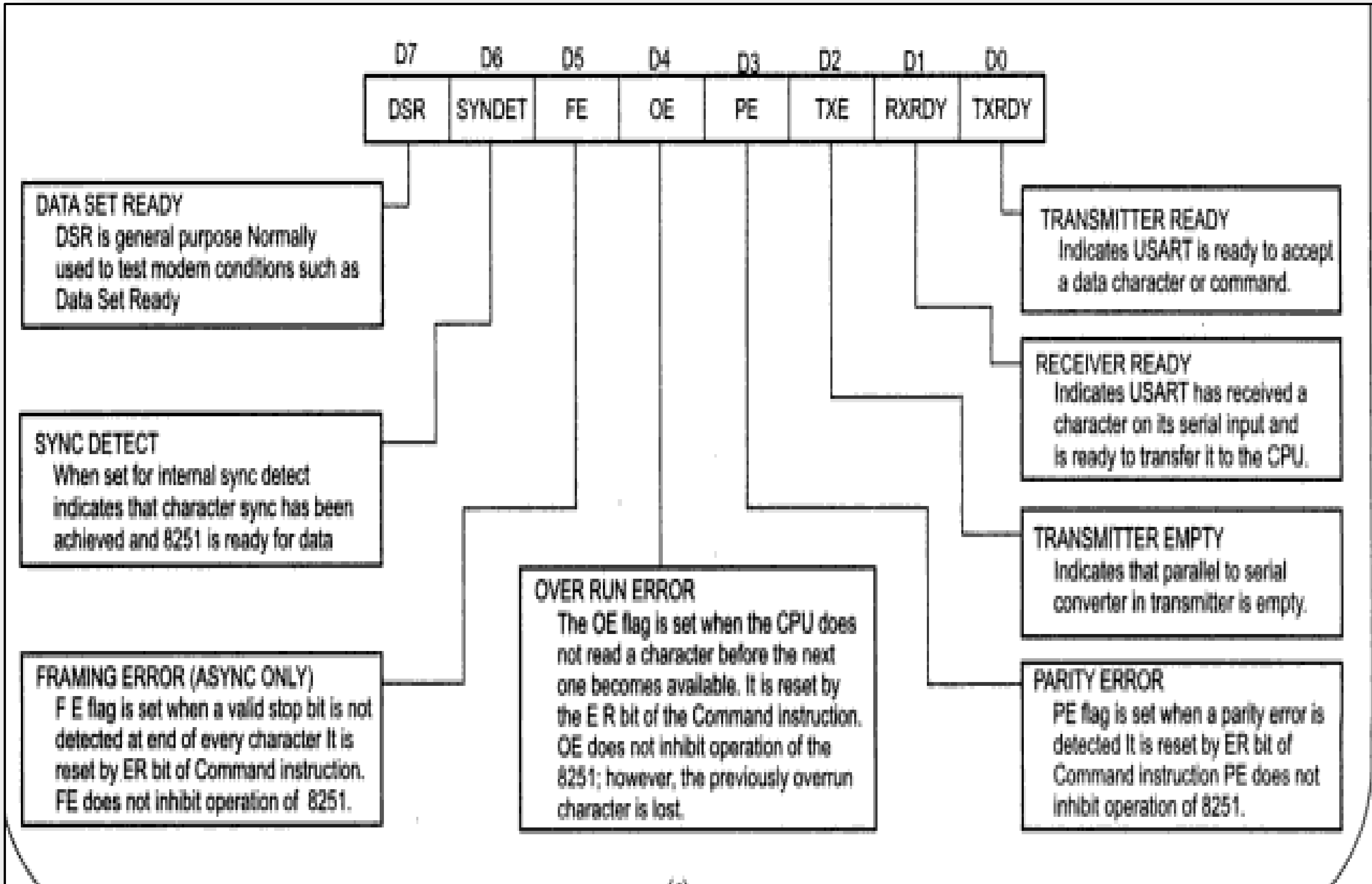
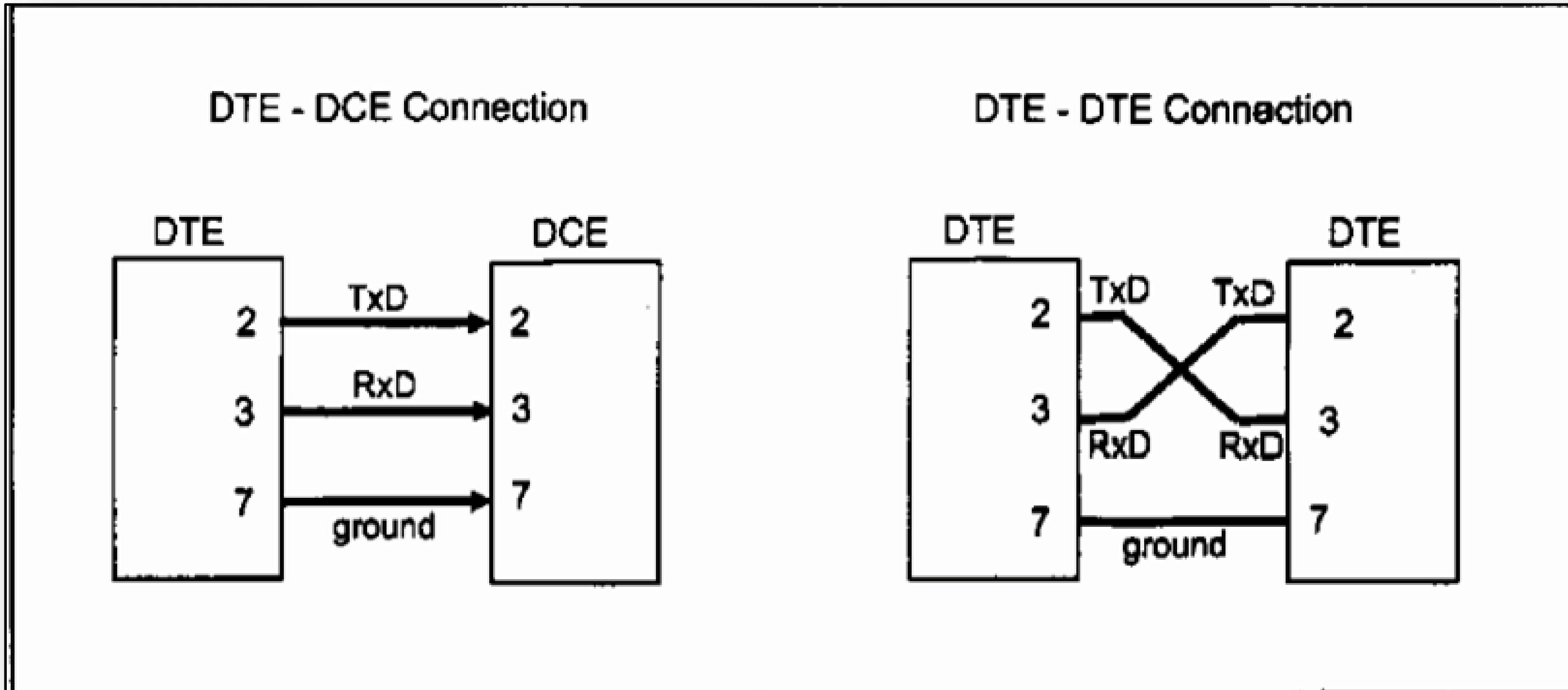


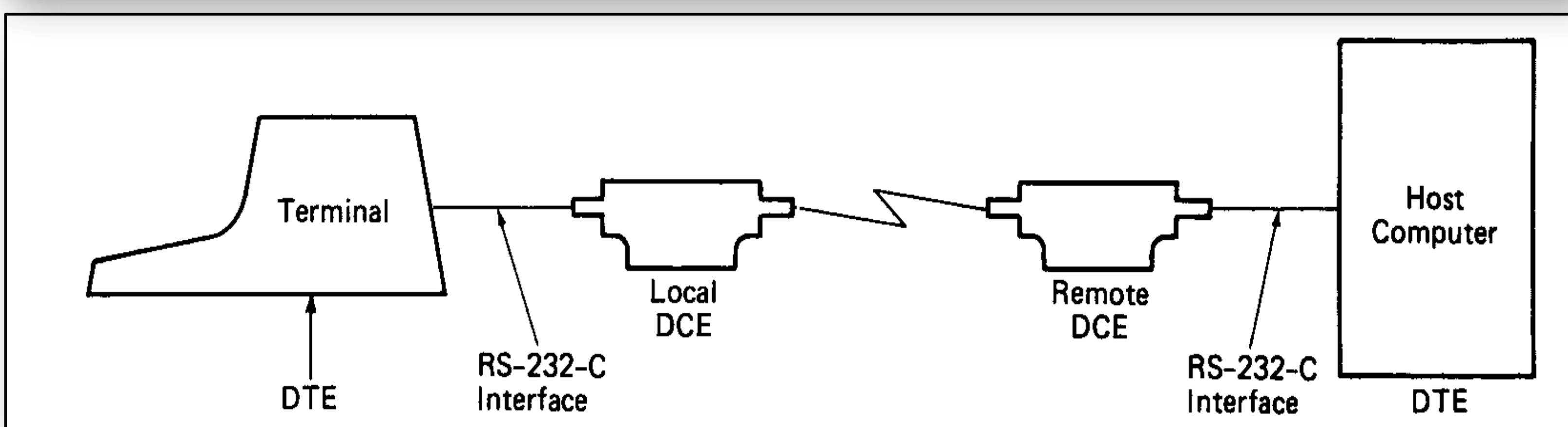
Figure: Status word register of 8251

RS-232

- Standard for transfer of characters across copper wire
- Produced by EIA
- Full name is *RS-232-C*
- RS-232 defines *serial*, *asynchronous* communication
  - Serial - bits are encoded and transmitted one at a time (as opposed to *parallel* transmission)
  - Asynchronous - characters can be sent at any time and bits are not individually synchronized

DTE Connections





Mechanical Characteristics

- 25-pin connector
  - 9-pin connector is more commonly found in IBM-PC but it covers signals for asynchronous serial communication only.
- Use male connector on DTE and female connector on DCE.
- Note: all signal names are viewed from DTE.

25-Pin RS232 Connector

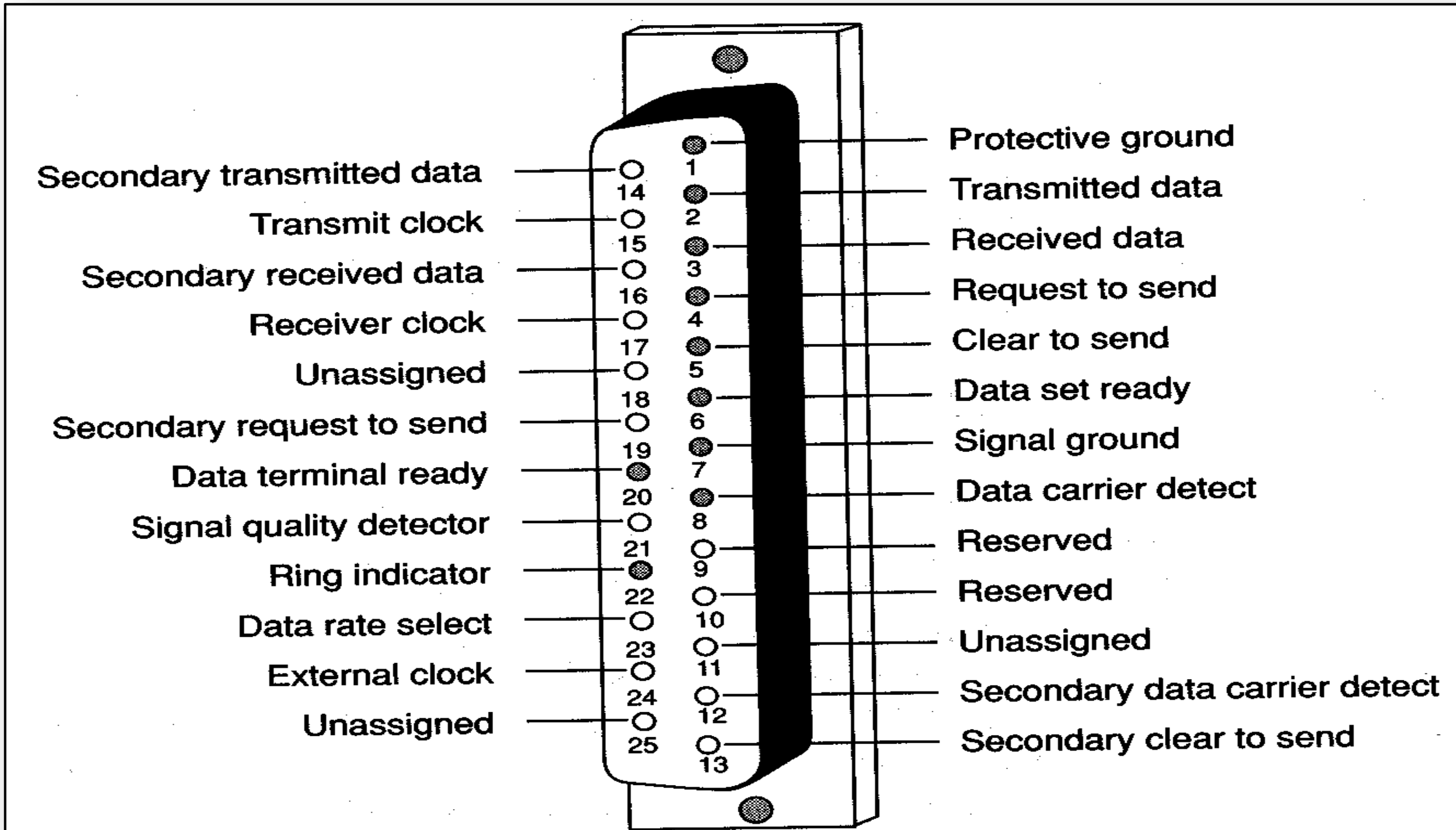
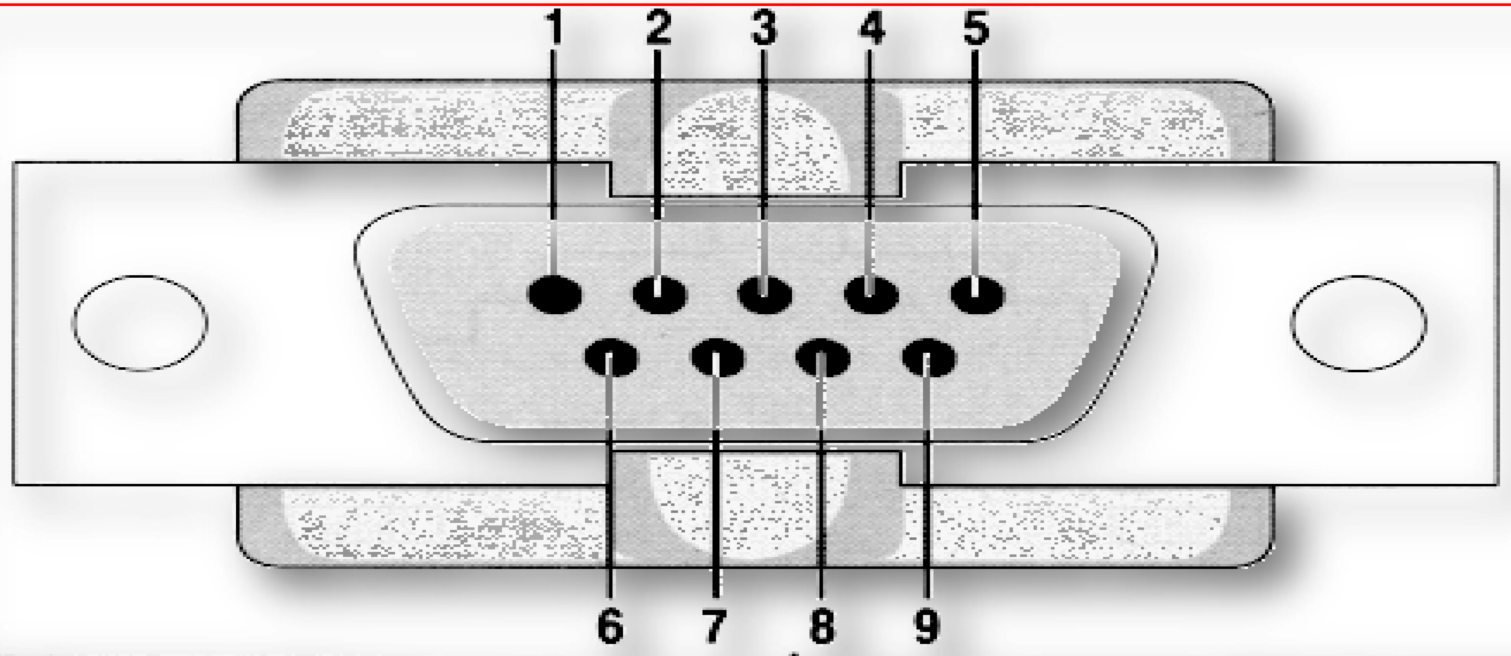


Figure: 25-Pin RS232 Connector

9-Pin RS232 Connector



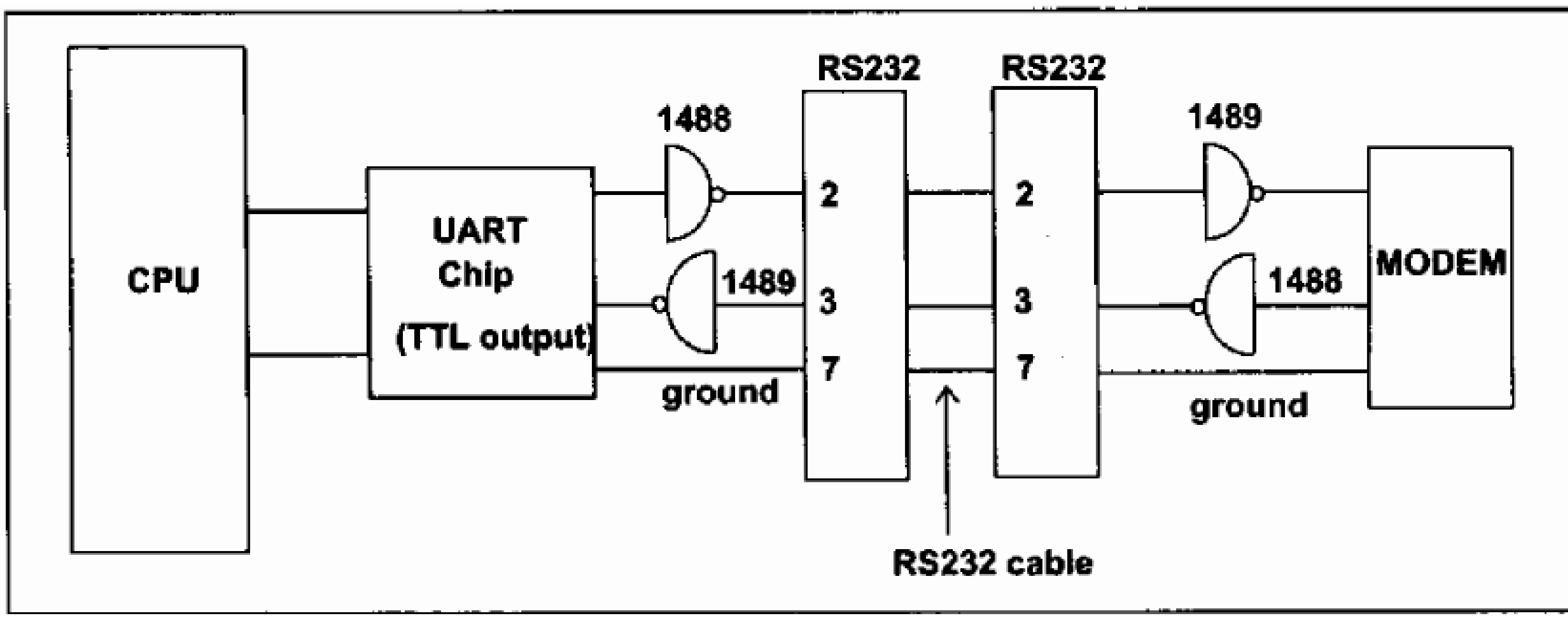
Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Figure: 9-Pin RS232 Connector

Electrical Characteristics

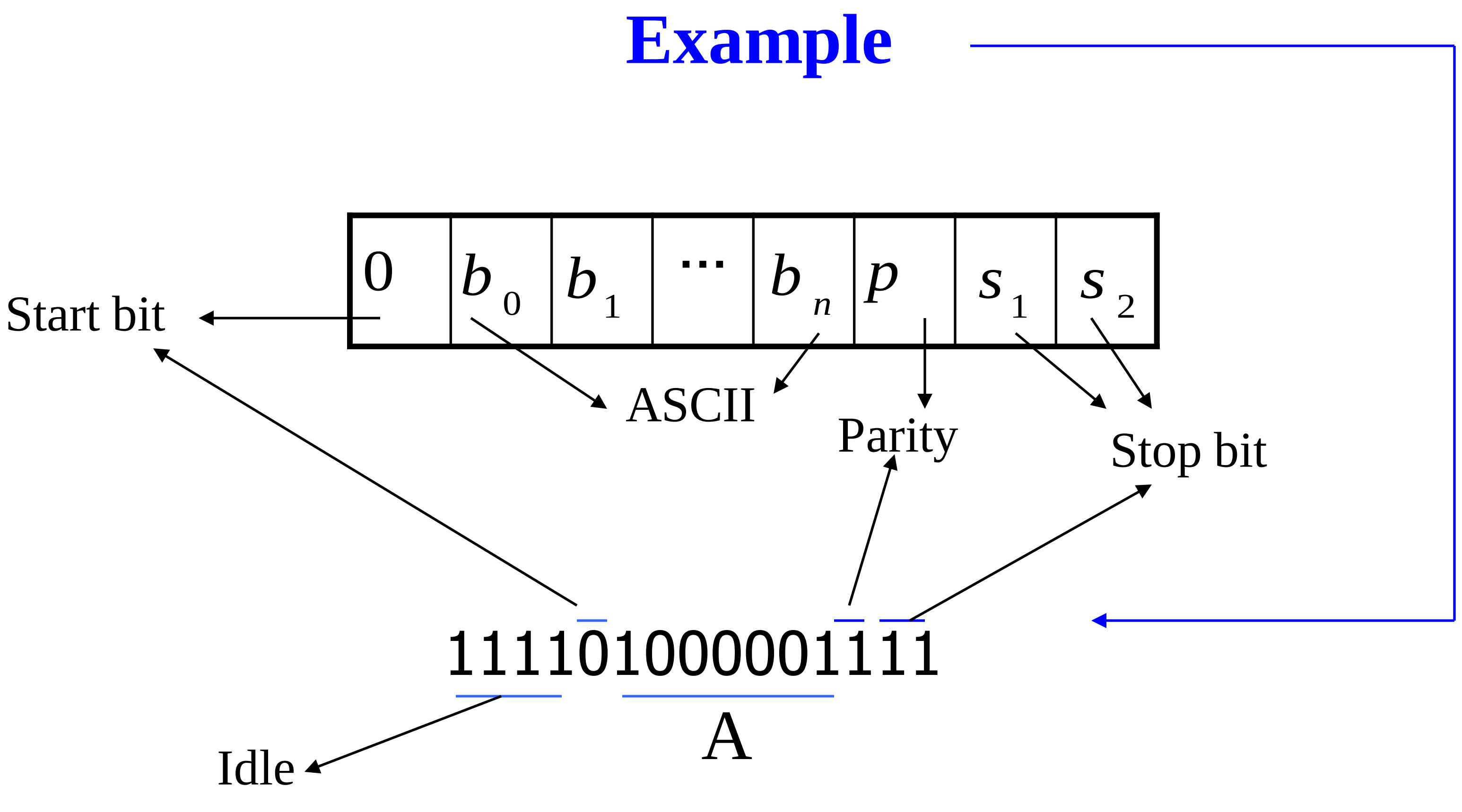
- Single-ended
  - one wire per signal, voltage levels are with respect to system common (i.e. signal ground)
- Mark: –3V to –15V
  - represent Logic 1, Idle State (OFF)
- Space: +3 to +15V
  - represent Logic 0, Active State (ON)
- Usually swing between –12V to +12V
- Recommended maximum cable length is 15m, at 20kbps

TTL to RS-232

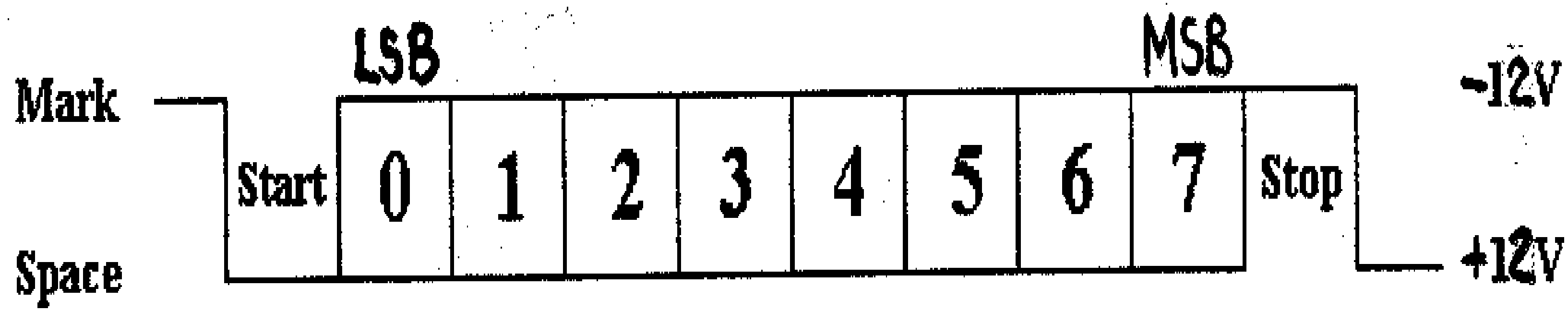


Line drivers and line receivers

RS-232 Frame Format



RS232 Logic Waveform



Function of Signals

- TD: transmitted data
- RD: received data
- DSR: data set ready
  - indicate whether DCE is powered on.
- DTR: data terminal ready
  - indicate whether DTR is powered on
  - turning off DTR causes modem to hang up the line
- RI: ring indicator
  - ON when modem detects phone call.

Function of Signals

- DCD: data carrier detect
  - ON when two modems have negotiated successfully and the carrier signal is established on the phone line
- RTS: request to send
  - ON when DTE wants to send data
  - Used to turn on and off modem's carrier signal in multi-point (i.e. multi-drop) lines
  - Normally constantly ON in point-to-point lines
- CTS: clear to send
  - ON when DCE is ready to receive data.
- SG: signal ground

Flow Control

- Means to ask the transmitter to stop/resume sending in data
- Required when:
  - DTE to DCE speed > DCE to DCE speed
  - (e.g. terminal speed = 115.2kbps and line speed = 33.6kbps, in order to benefit from modem's data compression protocol)
  - without flow control, the buffer within modem will overflow – sooner or later.
  - the receiving end takes time to process the data and thus cannot be always ready to receive

Hardware Flow Control

- RTS/CTS
  - the transmitting end activates RTS to inform the receiving end that it has data to send.
  - if the receiving end is ready to receive, it activates CTS.
  - normally used between computer and modem.
    - computer is always ready to receive data but modem is not, because terminal speed > link speed

Software Flow Control

Xon/Xoff

- when the buffer within the receiving end is nearly full, Xoff is sent to the transmitting end to ask it to stop.

- when data have been processed by the receiving end and the buffer has space again, Xon is sent to the transmitting end to notify it to resume

- advantage: only three wires are required (TD, RD and GND).

- disadvantage: confusion arises when the transmitted data (e.g. a graphics file) contains a byte equal to 13H (Xoff).

Other Standards

Table 9-3: RS232 Comparison with RS422 and RS423

	RS232	RS422	RS423
Max. cable length (ft)	50	4000	4000
Maximum speed (baud)	20K	10M/40 ft	100K/30 ft
		1M/400 ft	10K/300 ft
		100K/4000 ft	1K/4000 ft
Logic 1 voltage level	–3 to –25	A > B	–4 to –6
Logic 0 voltage level	+3 to +25	B > A	+4 to +6

8250/16450/16550 UART

Table 9-4. 8250A Register Addresses

DLAB	A2	A1	A0	Description
0	0	0	0	Receive buffer register for read, transmitter holding register for write
0	0	0	1	Interrupt enable register
x	0	1	0	Interrupt identification register (read only)
x	0	1	1	Line control register (data format register)
x	1	0	0	MODEM control register
x	1	0	1	Line status register
x	1	1	0	MODEM status register
x	1	1	1	Scratch register
1	0	0	0	Divisor latch register (LSB)
1	0	0	1	Divisor latch register (MSB)

8051 Microcontroller

Disadvantages of microprocessor

- The overall system cost is high.
- A large sized PCB is required for assembling all the components.
- Overall product design requires more time.
- Physical size of the product is big.
- A discrete components are used, the system is not reliable.

Advantages of Microcontroller based System

- As the peripherals are integrated into a single chip, the overall system cost is very less.
- The product is of small size compared to micro processor based system.
- The system design now requires very little efforts
- As the peripherals are integrated with a microprocessor the system is more reliable.
- Though microcontroller may have on chip ROM,RAM and I/O ports, addition ROM, RAM I/O ports may be interfaced externally if required.
- On chip ROM provide a software security.

Why we Choosing a Microcontroller

- meeting the computing needs of the task efficiently and cost effectively.
  - speed, the amount of ROM and RAM, the number of I/O ports and timers, size, packaging, power consumption.
  - easy to upgrade.
  - cost per unit.
  - Noise of environment.
- availability of software development tools
  - assemblers, debuggers, C compilers, emulator, simulator, technical support
- wide availability and reliable sources of the microcontrollers

Comparison of the 8051 Family Members

- ROM type
  - 8031 no ROM
  - 80xx mask ROM
  - 87xx EPROM
  - 89xx Flash EEPROM
- 89xx
  - 8951
  - 8952
  - 8953
  - 8955
  - 898252
  - 891051
  - 892051

Example (AT89C51,AT89LV51)

AT= ATMEL(Manufacture)

C = CMOS technology

LV= Low Power(3.0v)

Comparison some of the 8051 Family Members

	ROM	RAM	Timer
8051	4k	128	2
8031	-	128	2
8751	4k eprom	128	2
8052	8krom	256	3
8032	-	256	3
8752	8k eprom	256	3

8051 Basic Component

- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface
- 64k external memory for code
- 64k external memory for data
- 210 bit addressable

Microcontroller

The basic 8051 Core

- 8-bit CPU optimized for control applications
- Capability for single bit Boolean operations.
- Supports up to 64K of program memory.
- Supports up to 64K of data memory.
- 4 K bytes of on-chip program memory.
- Newer devices provide more.
- 128 or 256 bytes of on-chip data RAM.
- Four 8 bit ports.
- Two 16-bit timer/counters
- UART.
- Interrupts.
- On-chip clock oscillator.

Differences between 8086 and 8051

S. No	Microprocessor	Microcontroller
1	A microprocessor is a general purpose device which is called a CPU	A microcontroller is a dedicated chip which is also called single chip computer.
2	A microprocessor do not contain onchip I/OPorts, Timers, Memories etc..	A microcontroller includes RAM, ROM, serial and parallel interface, timers, interrupt circuitry (in addition to CPU) in a single chip.
3	Microprocessors are most commonly used as the CPU in microcomputer systems	Microcontrollers are used in small, minimum component designs performing control-oriented applications.
4	Microprocessor instructions are mainly nibble or byte addressable	Microcontroller instructions are both bit addressable as well as byte addressable.
5	Microprocessor instruction sets are mainly intended for catering to large volumes of data.	Microcontrollers have instruction sets catering to the control of inputs and outputs.

Differences between 8086 and 8051 cont…

6	Microprocessor based system design is complex and expensive	Microcontroller based system design is rather simple and cost effective
7	The Instruction set of microprocessor is complex with large number of instructions.	The instruction set of a Microcontroller is very simple with less number of instructions. For, ex: PIC microcontrollers have only 35 instructions**.**
8	A microprocessor has zero status flag	A microcontroller has no zero flag.

Block diagram of 8051

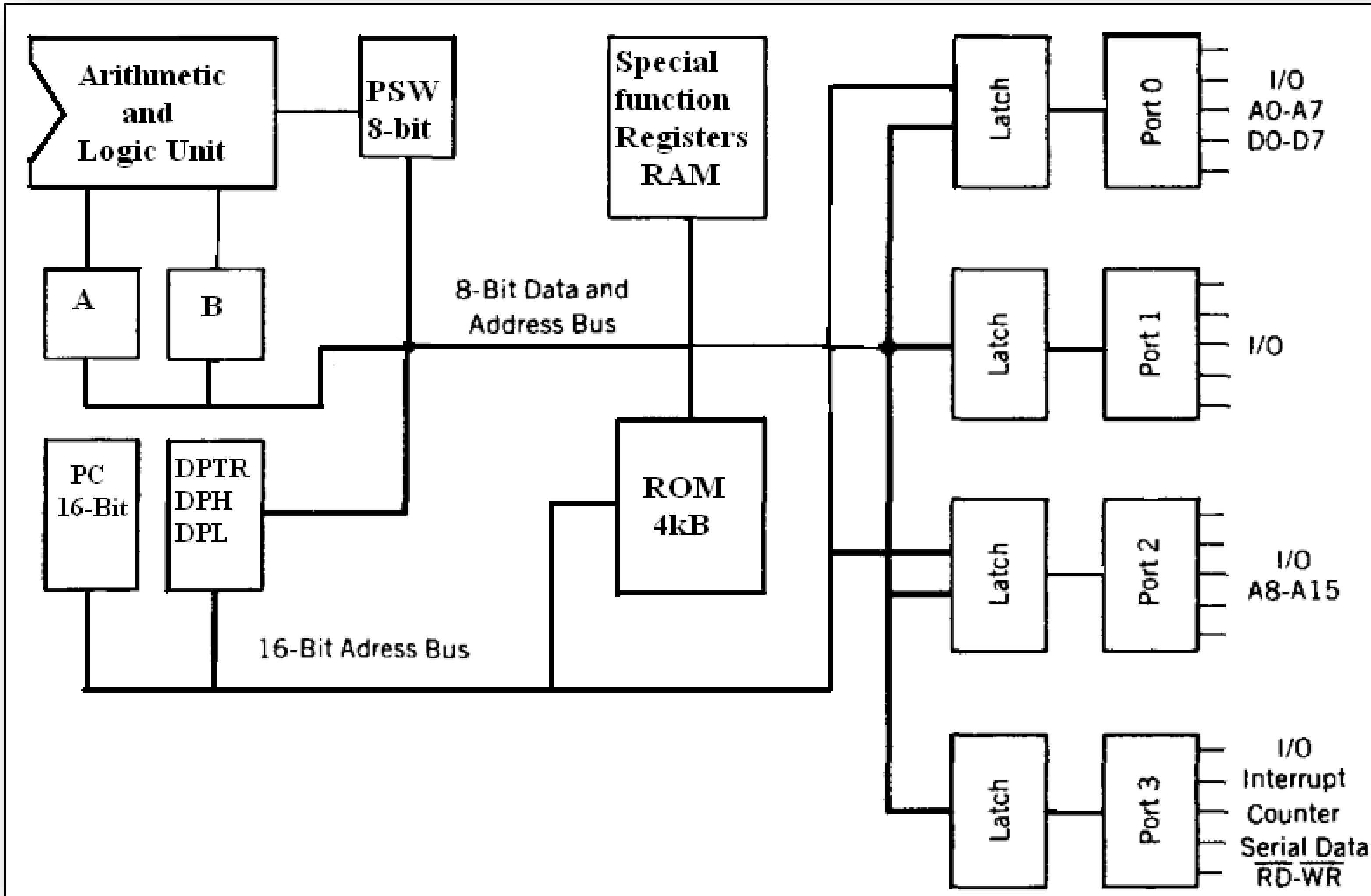
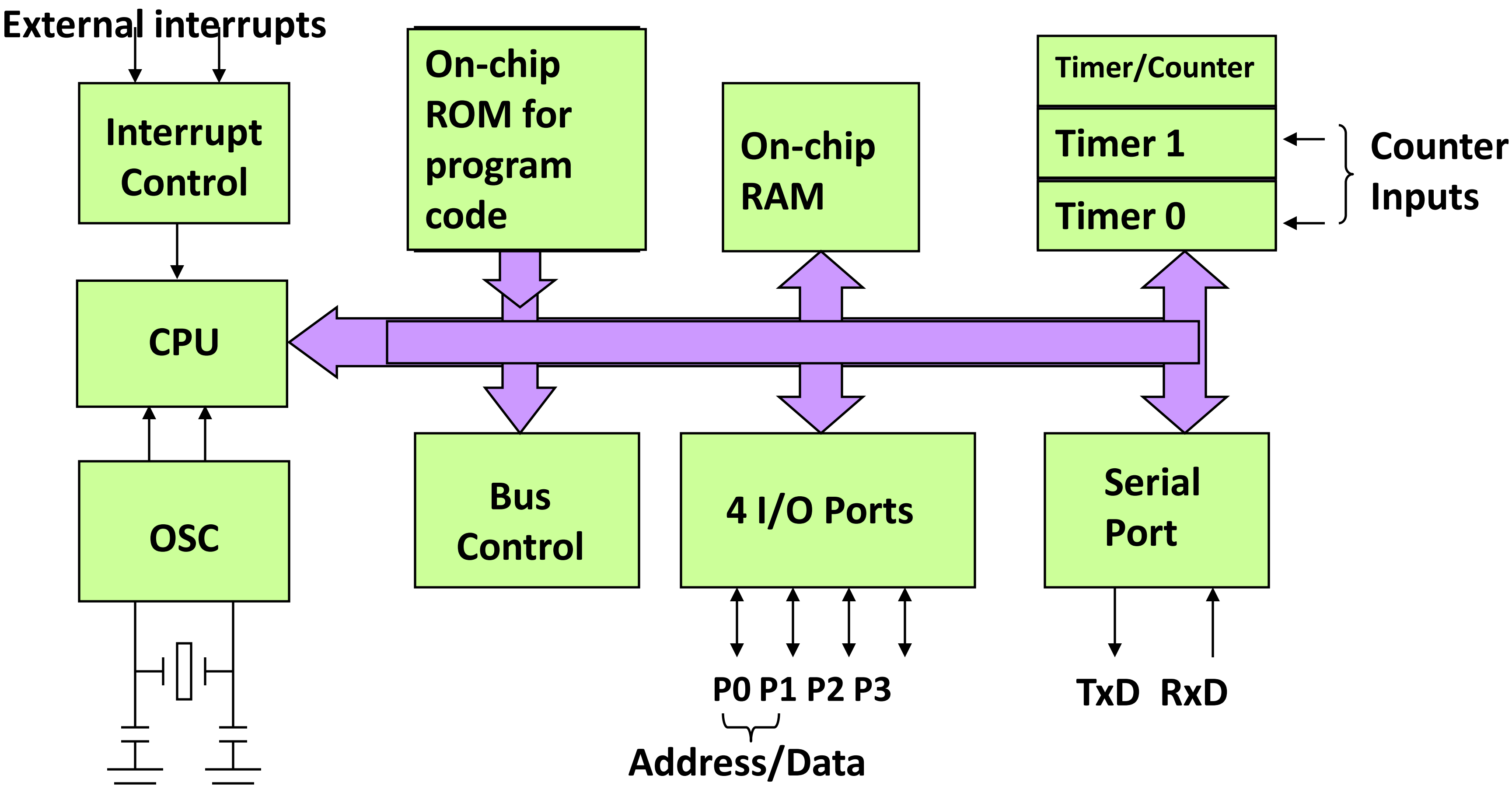


Figure: Block diagram of 8051

Block Diagram

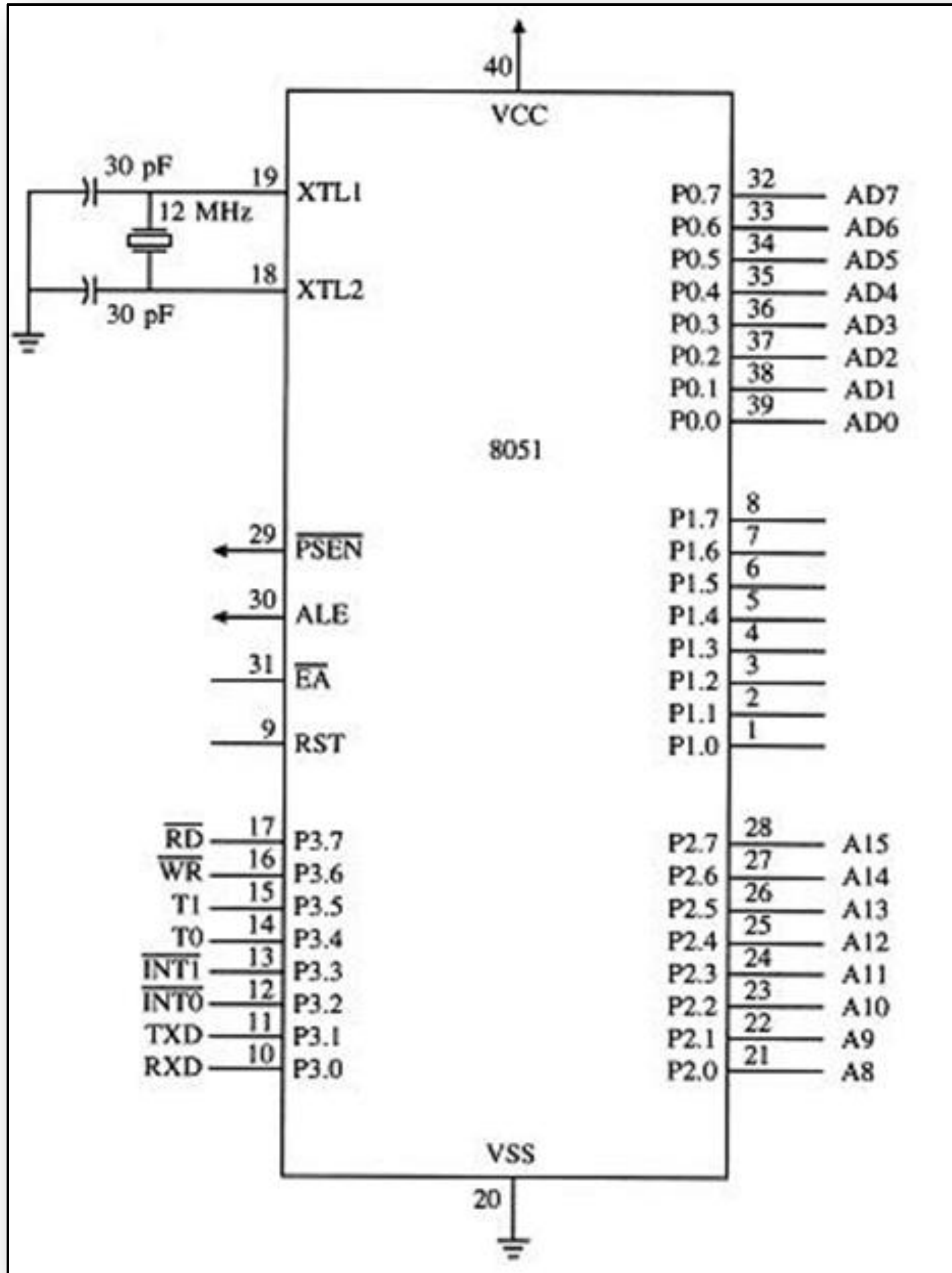


The figure also shows the usual CPU components: program counter, ALU, working registers, and clock circuits.[1]

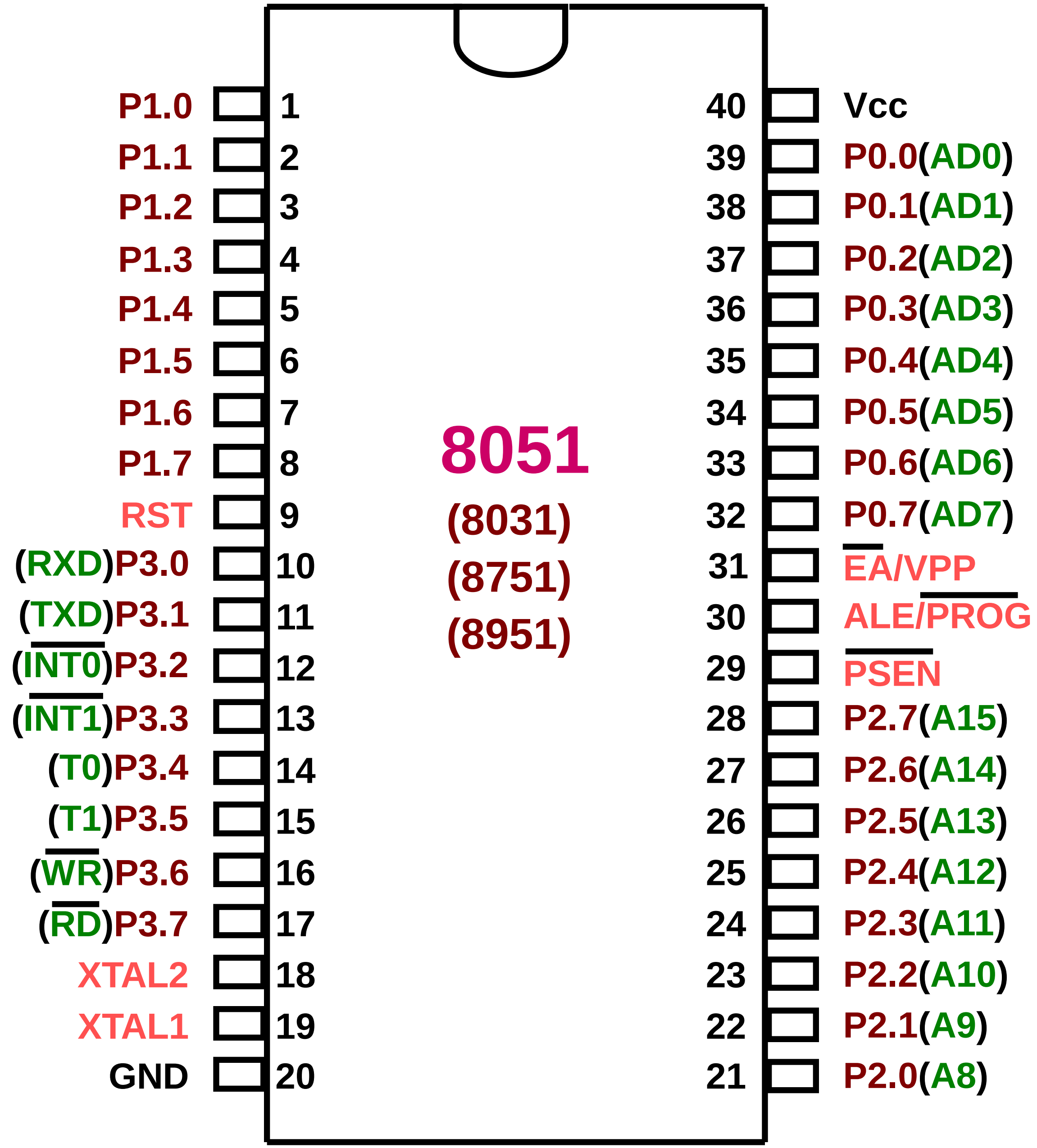
The 8051 architecture consists of these specific features:

- Eight-bit CPU with registers A (the accumulator) and B
- Sixteen-bit program counter (PC) and data pointer (DPTR)
- Eight-bit program status word (PSW)
- Eight-bit stack pointer (SP)
- Internal ROM or EPROM (8751) of 0 (8031) to 4K (8051)
- Internal RAM of 128 bytes:
    - Four register banks, each containing eight registers
    - Sixteen bytes, which may be addressed at the bit level
    - Eighty bytes of general-purpose data memory
- Thirty-two input/output pins arranged as four 8-bit ports: P0–P3
- Two 16-bit timer/counters: T0 and T1
- Full duplex serial data receiver/transmitter: SBUF
- Control registers: TCON, TMOD, SCON, PCON, IP, and IE
- Two external and three internal interrupt sources
- Oscillator and clock circuits

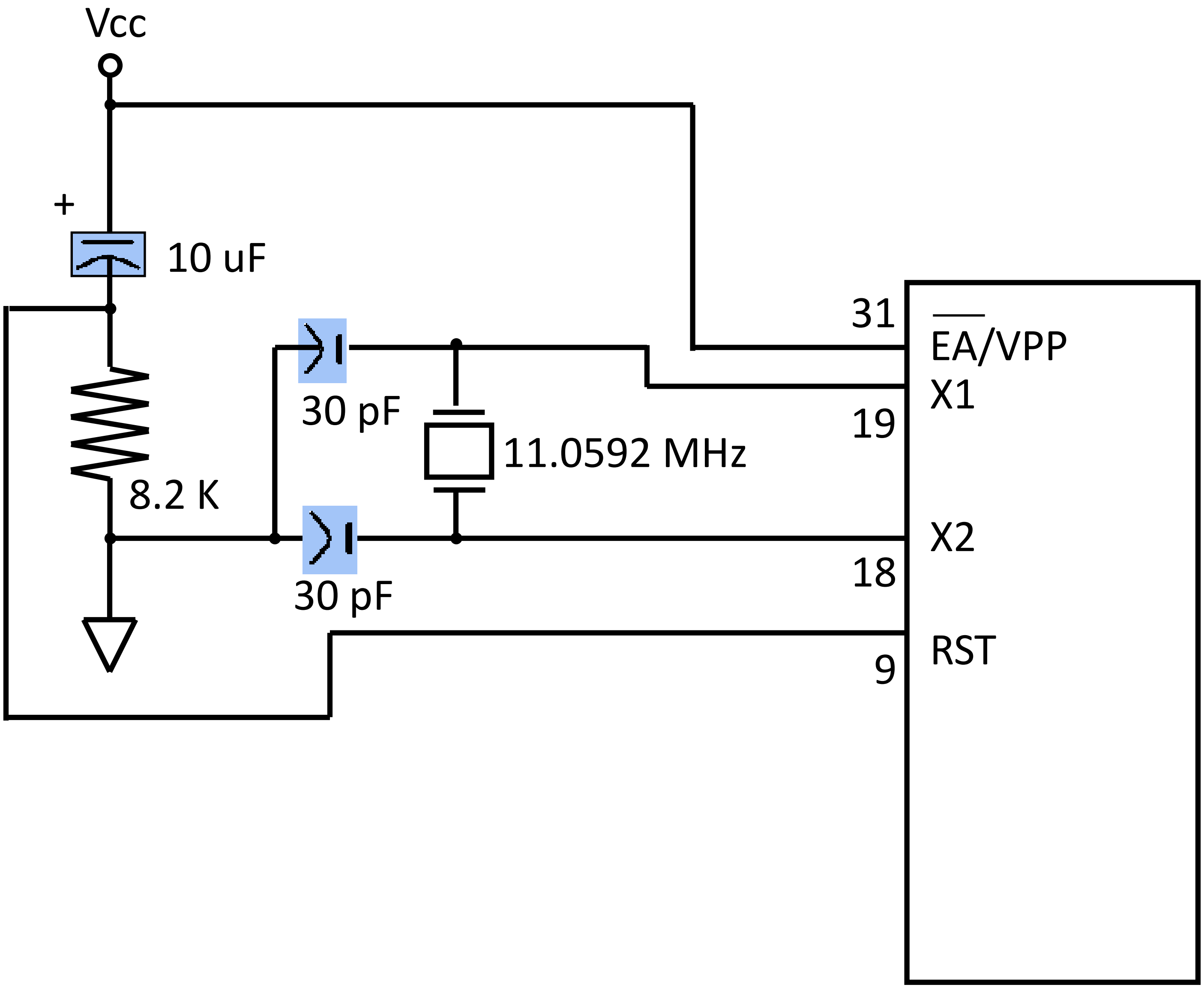
8051
Schematic Pin out



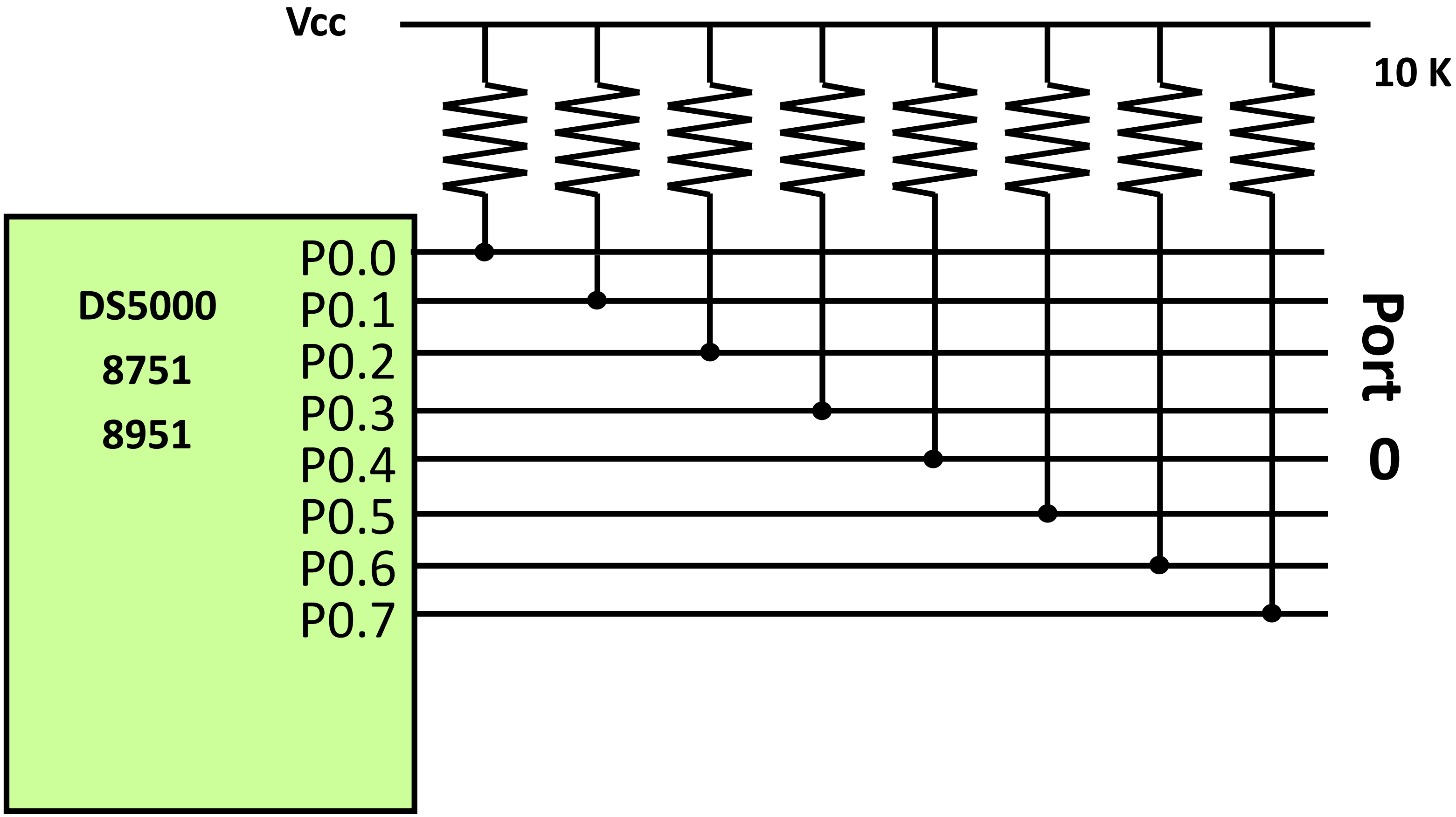
8051 Foot Print



Power-On RESET Circuit



Port 0 with Pull-Up Resistors



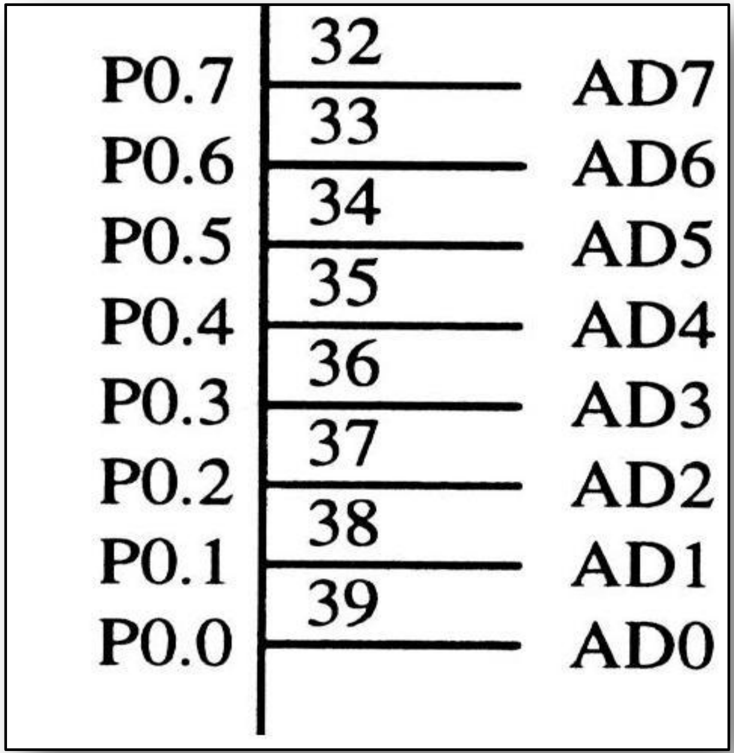
IMPORTANT PINS (IO Ports)

- One of the most useful features of the 8051 is that it contains four I/O ports (P0 - P3).
- Each port can be used as input or output (bi-direction).

- Port 0

pins 32-39 （P0.0～P0.7）

  - 8-bit R/W - General Purpose I/O.
  - Or acts as a multiplexed low byte address and data bus for external memory design.

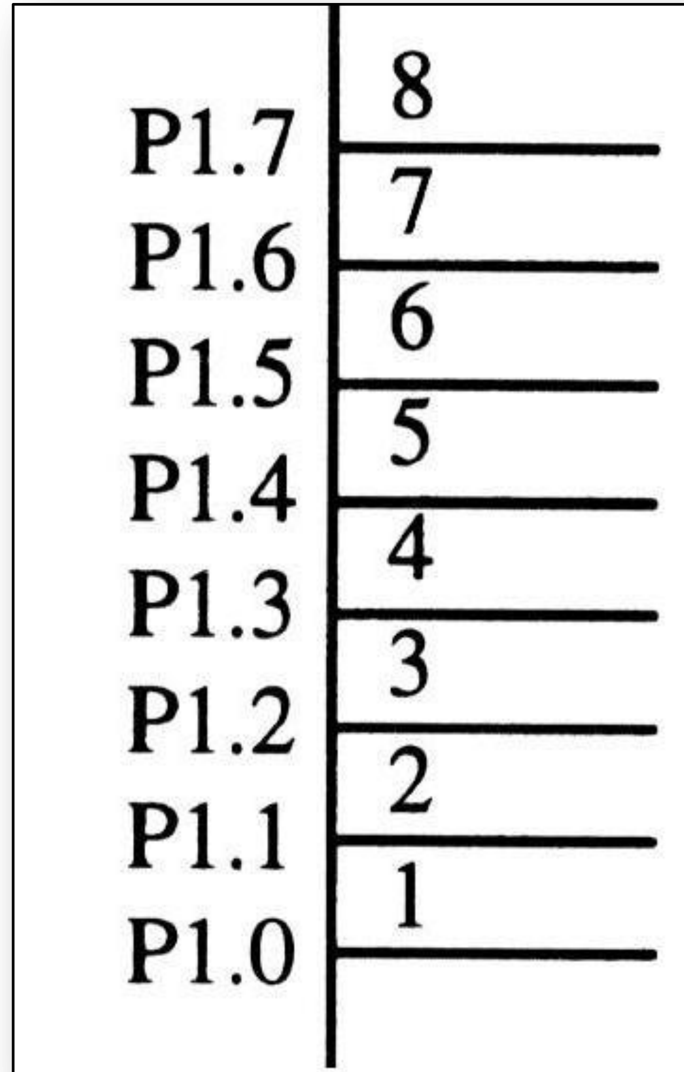


IMPORTANT PINS (IO Ports)

- Port 1

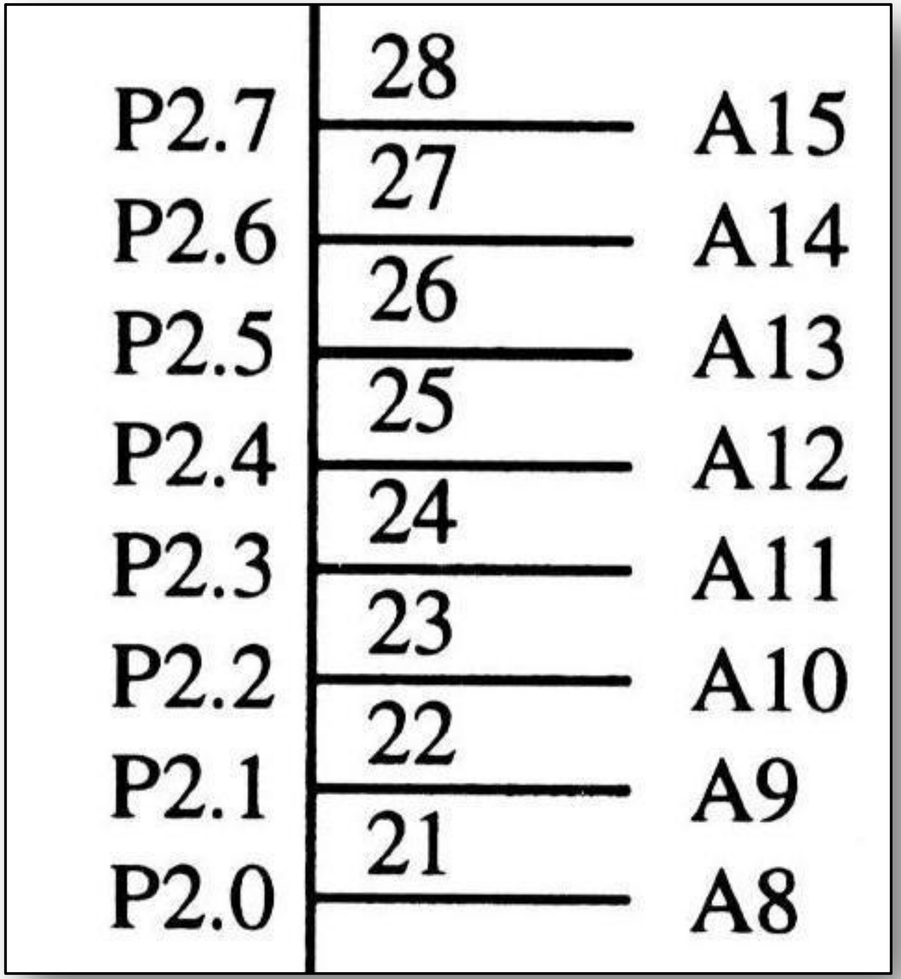
（pins 1-8）　（P1.0～P1.7）

- Only 8-bit R/W - General Purpose I/O



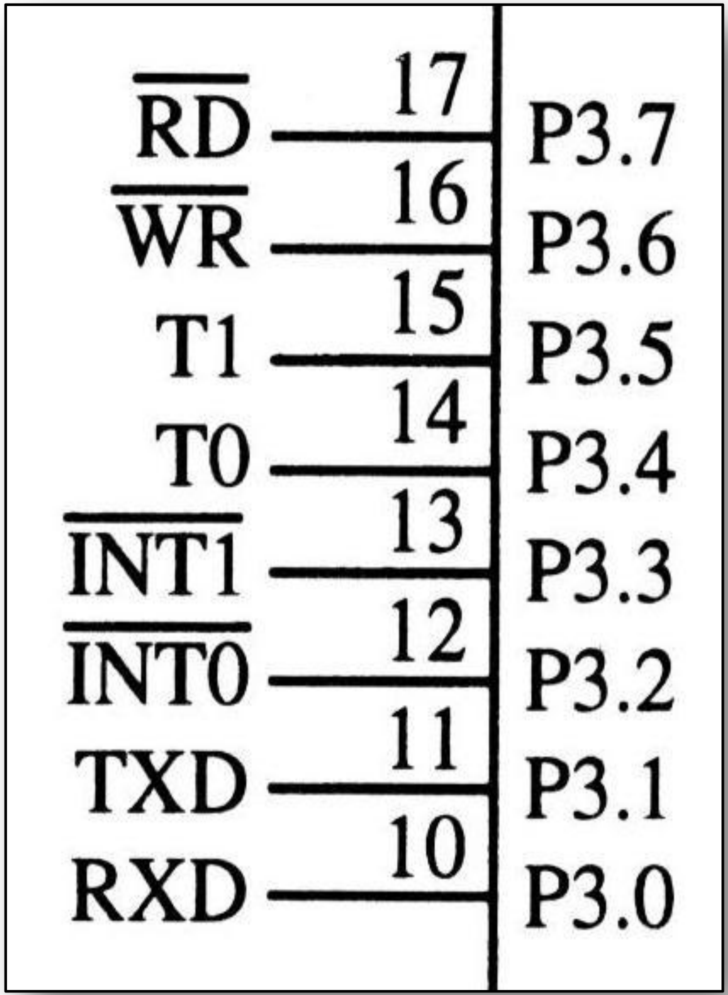
IMPORTANT PINS (IO Ports)

- Port 2
- （pins 21-28（P2.0～P2.7）
  - 8-bit R/W - General Purpose I/O
  - Or high byte of the address bus for external memory design



IMPORTANT PINS (IO Ports)

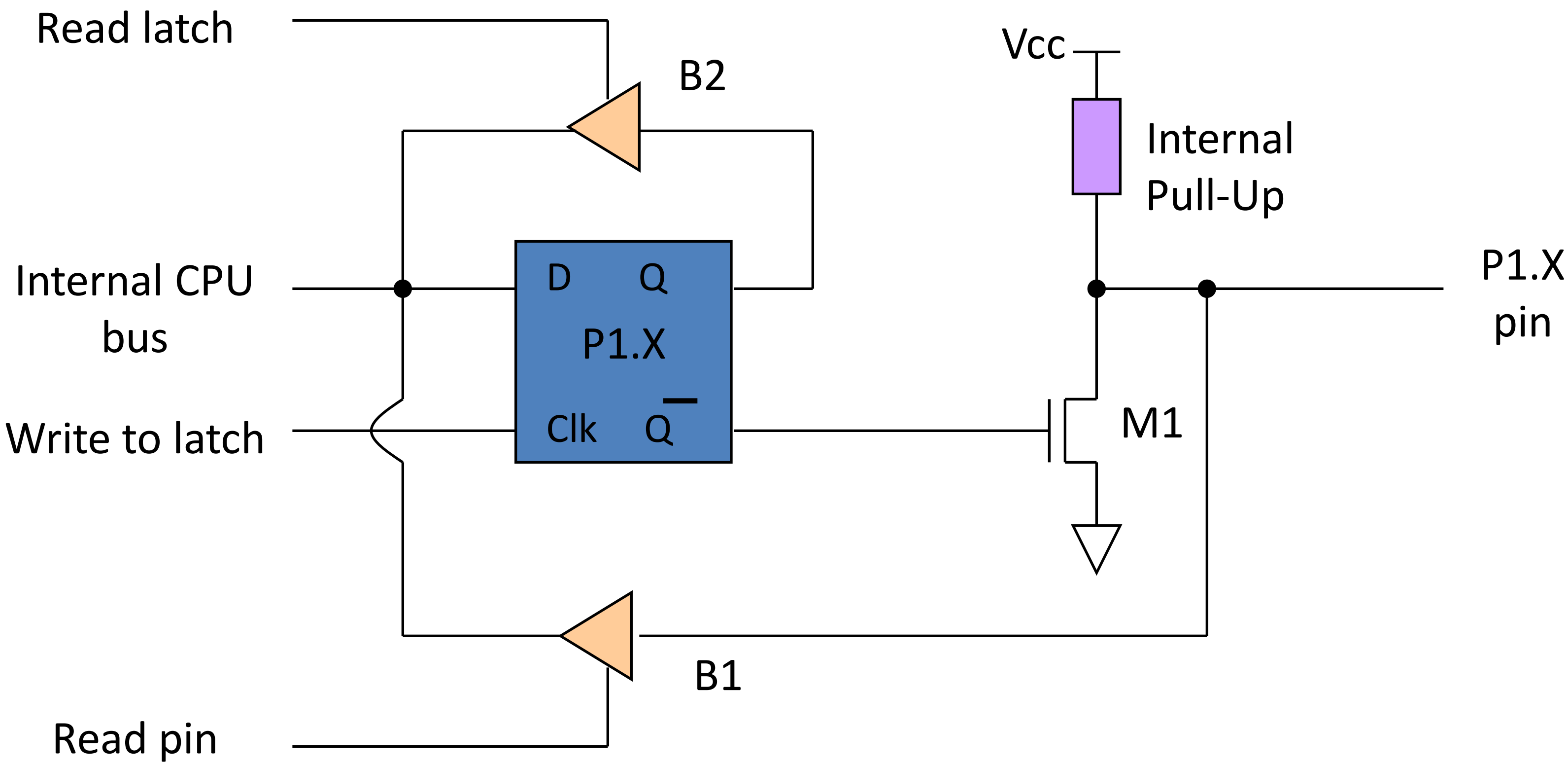
- Port 3
- （pins 10-17 （P3.0～P3.7）
  - General Purpose I/O
  - if not using any of the internal peripherals (timers) or external interrupts.



Port 3 Alternate Functions

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

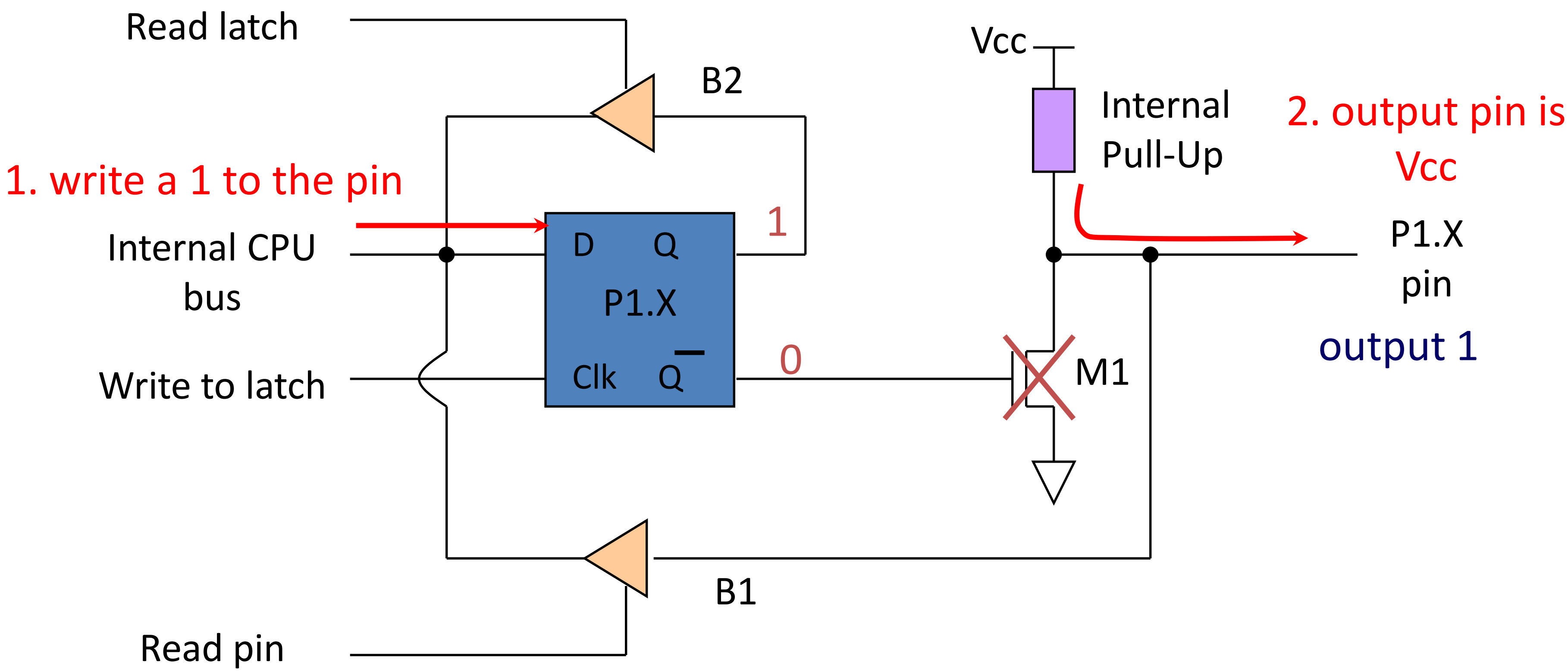
Hardware Structure of I/O Pin



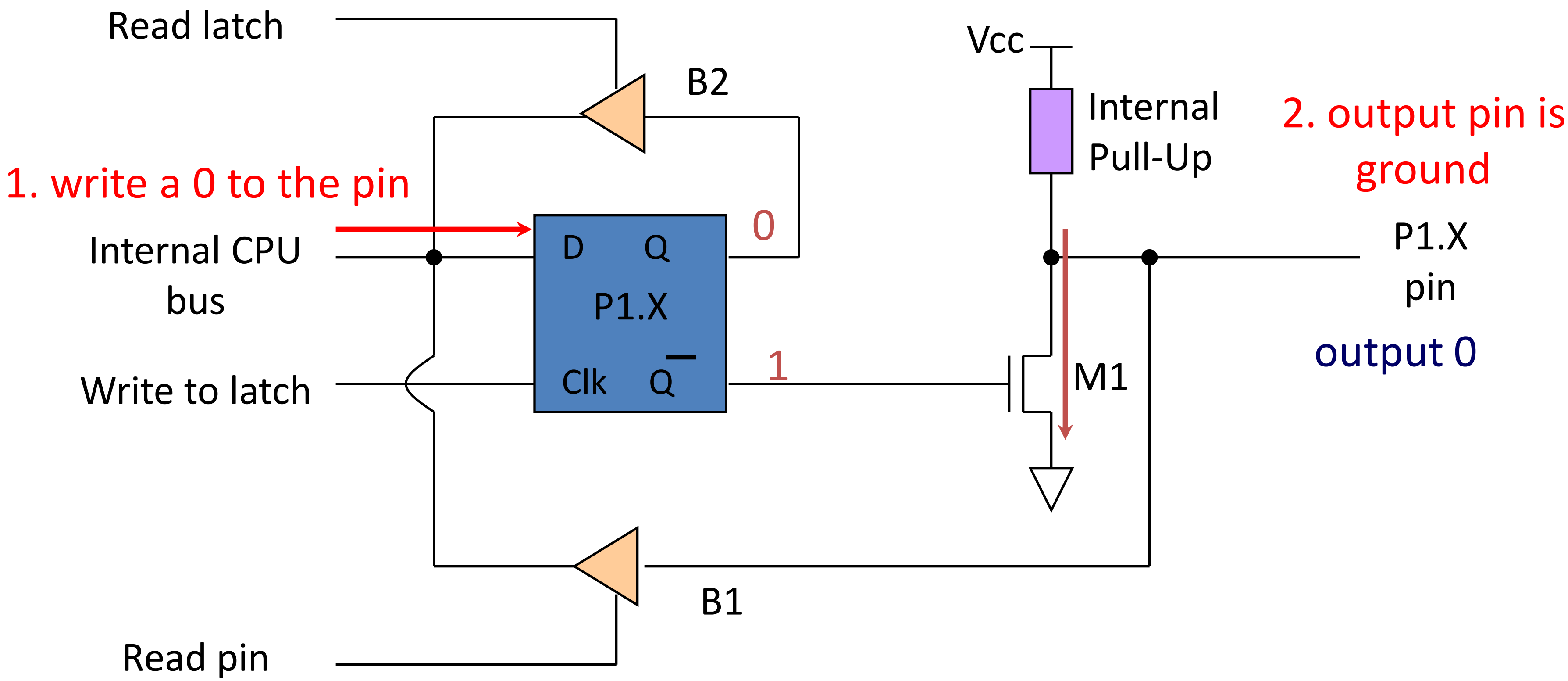
Hardware Structure of I/O Pin

- Each pin of I/O ports
  - Internally connected to CPU bus.
  - A D latch store the value of this pin.
  - Write to latch＝1：write data into the D latch.
- 2 Tri-state buffer：
  - B1: controlled by "Read pin".
  - Read pin＝1：really read the data present at the pin.
  - B2: controlled by "Read latch".
  - Read latch＝1：read value from internal latch.
- A transistor M1 gate
  - Gate=0: open
  - Gate=1: close

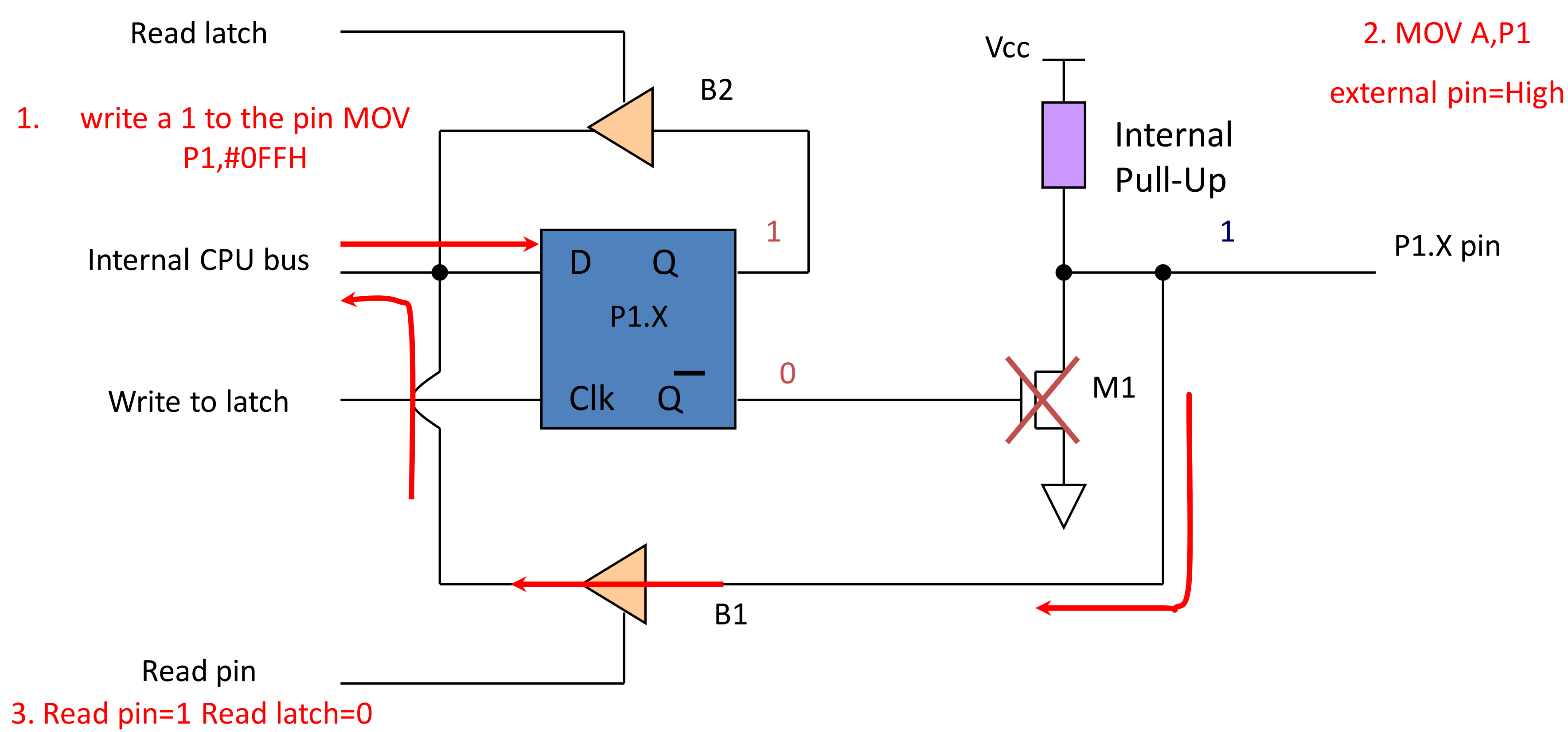
Writing "1" to Output Pin P1.X



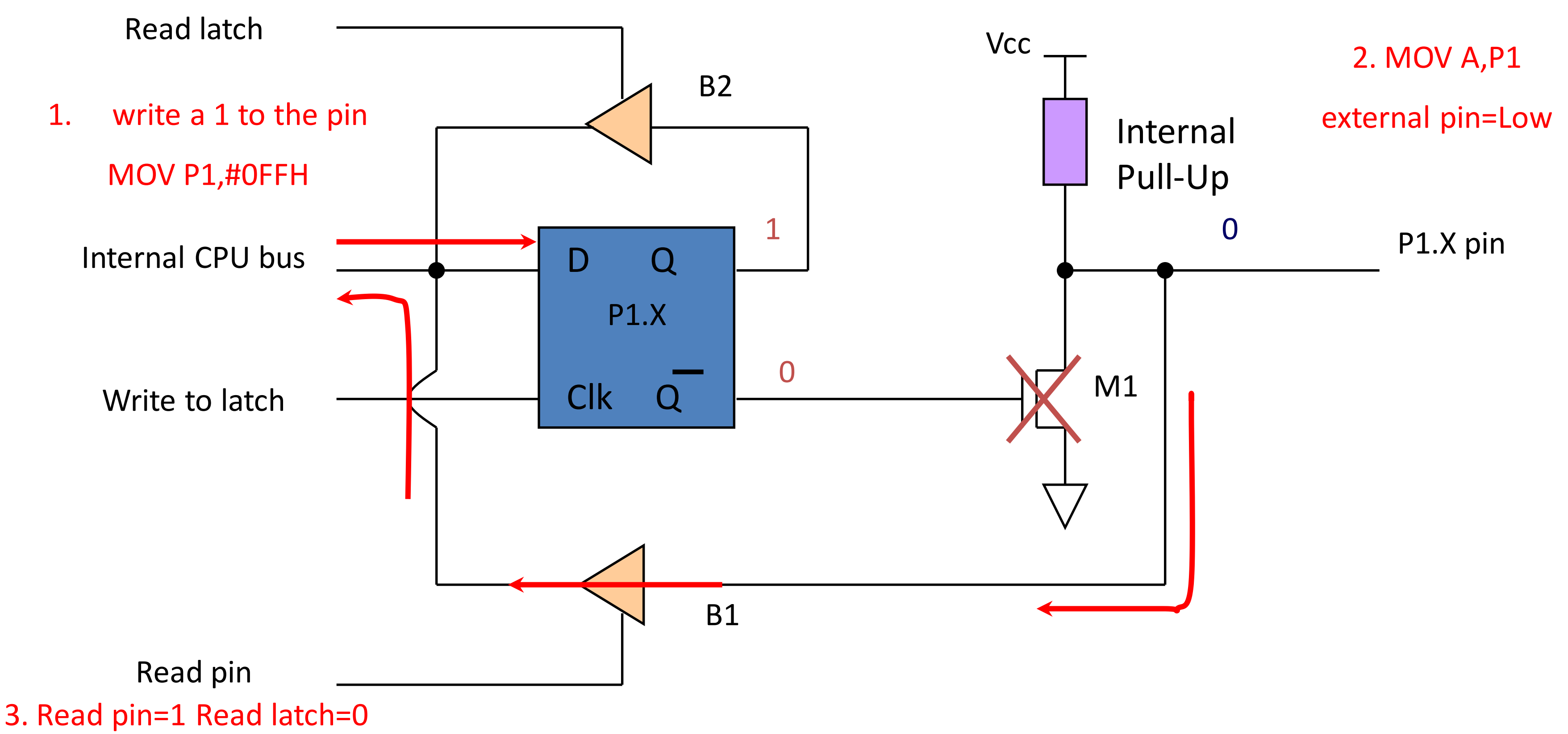
Writing "0" to Output Pin P1.X

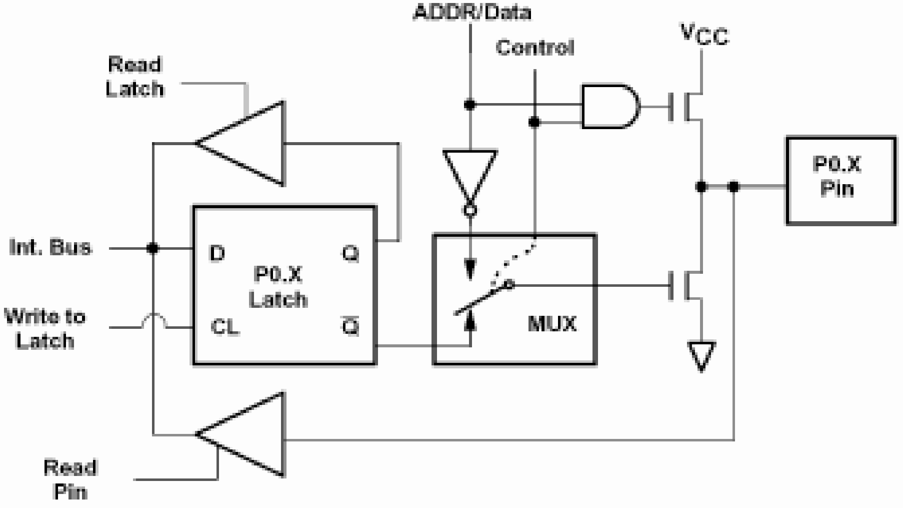


Reading "High" at Input Pin

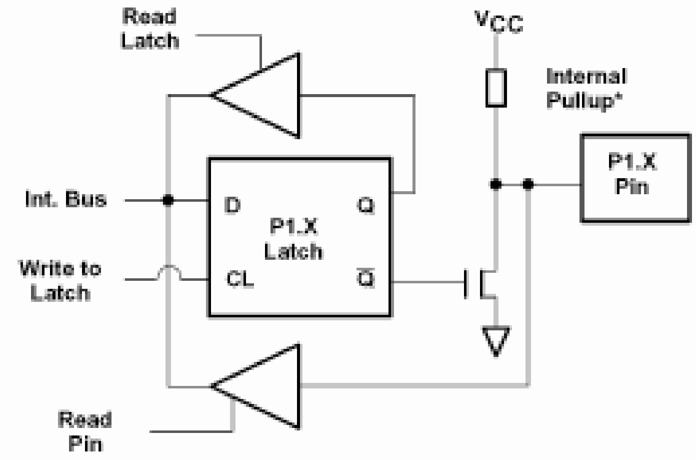


Reading "Low" at Input Pin

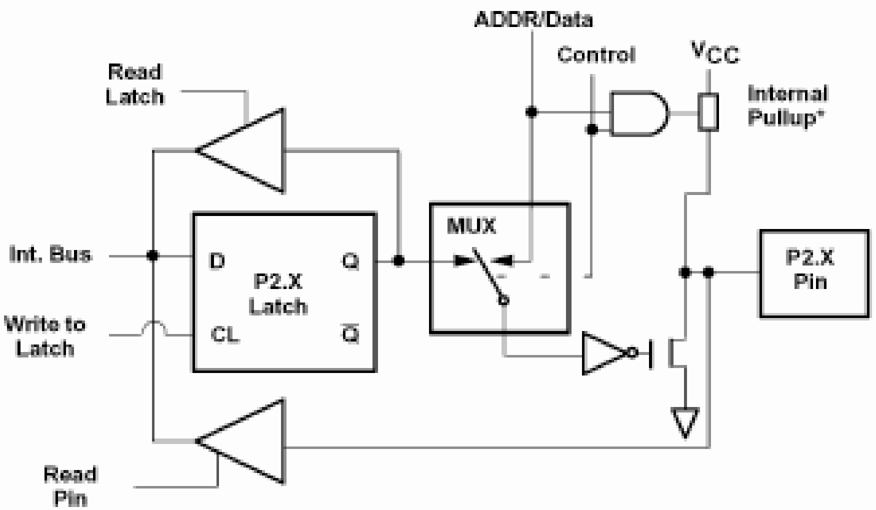




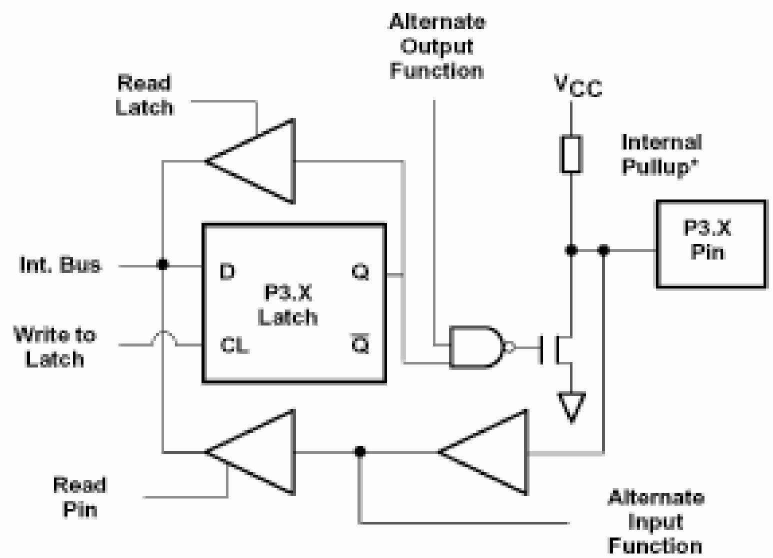
a. Port 0 Bit



b. Port 1 Bit

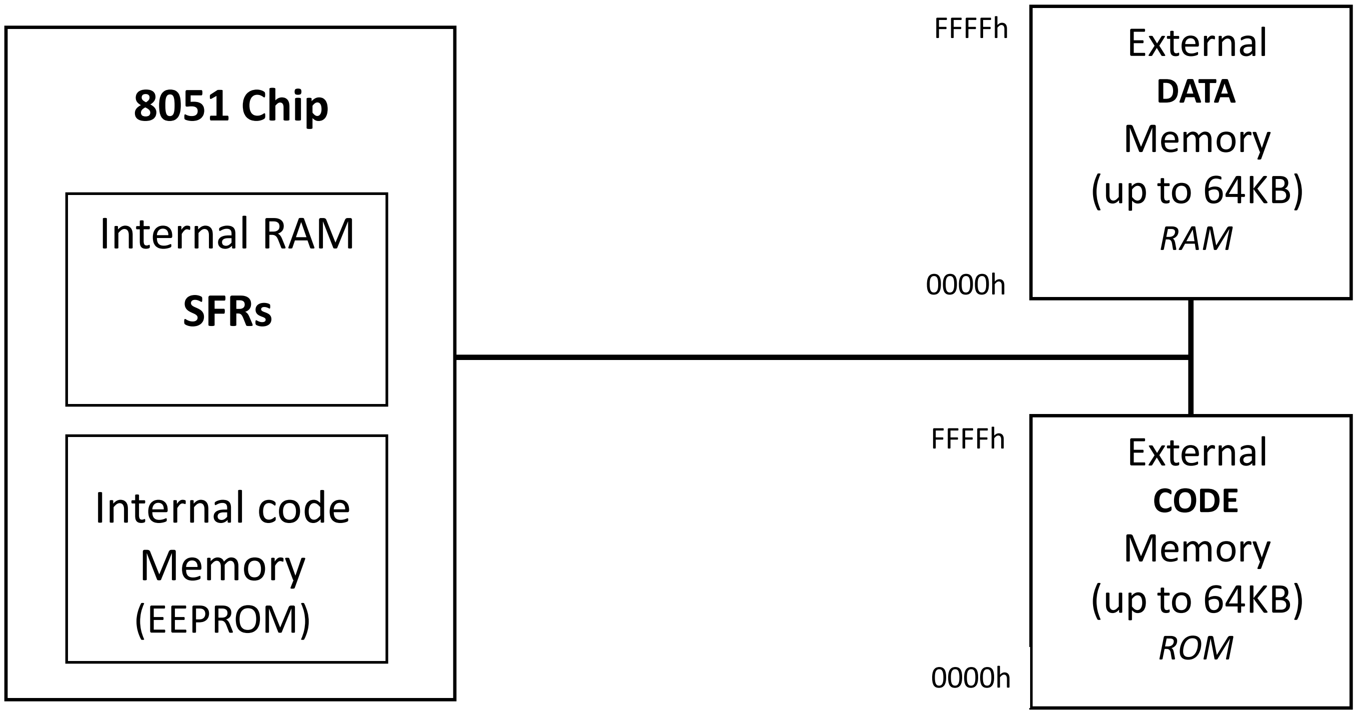


c. Port 2 Bit



d. Port 3 Bit

Memory organization



Types of Memory

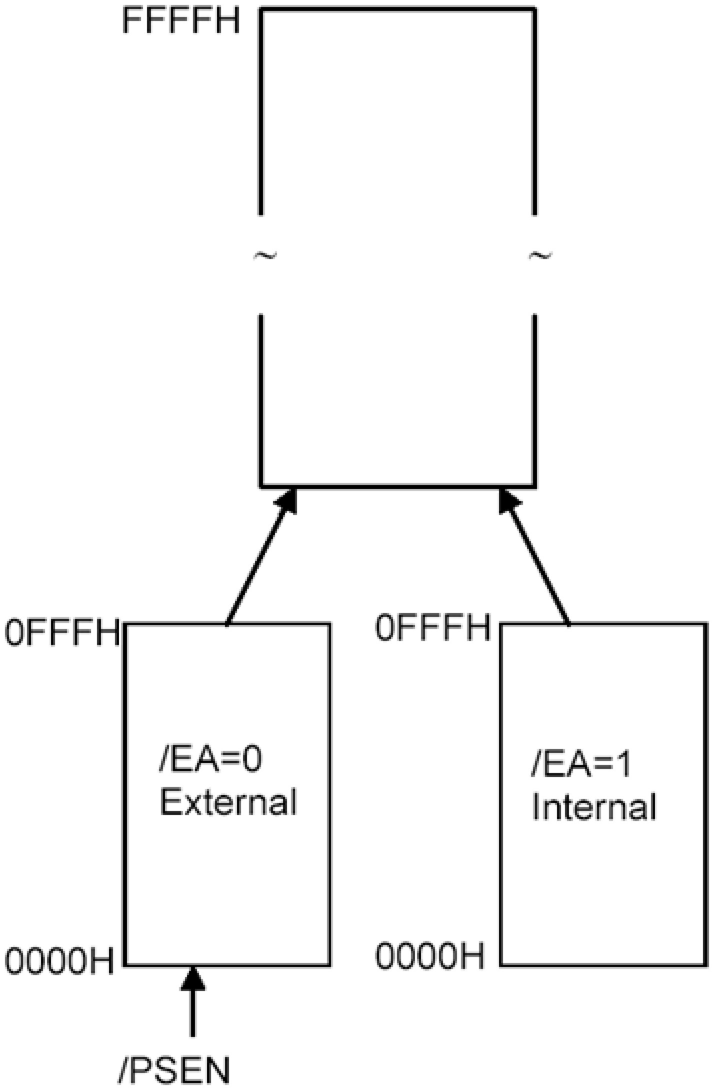
- External Code Memory (64k)
- External RAM Data Memory (64k)
- Internal Code Memory
  - 4k,8k,12k,20k
  - ROM, EPROM, EEPROM
- Internal RAM
  - First 128 bytes:
    - 00h to 1Fh Register Banks.
    - 20h to 2Fh Bit Addressable RAM.
    - 30 to 7Fh General Purpose RAM.
- Next 128 bytes:
  - 80h to FFh Special Function Registers.

External Memory

- **/EA**（pin 31）：External access
  - /EA='0' indicates that code is stored externally.
  - /PSEN & ALE are used for external ROM.
  - For 8051 internal code, /EA pin is connected to Vcc.
  - "/" means active low.
- **/PSEN**（pin 29）：program store enable.
  - Output- connected to OE of ROM.
  - Read signal – fetch from ROM
- ALE（pin 30): Address latch enable.
  - It is an output pin and is active high.
  - 8051 port 0 provides both address and data.
  - The ALE pin is used for de-multiplexing the address and data by connecting to the G pin of the 74LS373 latch.

Program or Code Memory

- May consist of internal or external program memory. The amount of internal program memory varies depending on the device.
- 4K bytes typical in older devices.
- The MOVC instruction can be use to read code memory.
- To reference code memory I will use the notation:
- CM = CM(0,…,FFFFH) = CM(0,…,FFFFH; 7,…,0)
- This notation can be used to specify particular bits and bytes of code memory.
- For example CM(1234H) refers to the byte of code memory at address 1234H. CM(1234H;7) refers to the most significant bit in that address.



MOVC A,@A + DPTR ;A ← CM(A+DPTR)
MOVC A,@A + PC ;A ← CM(A+PC)

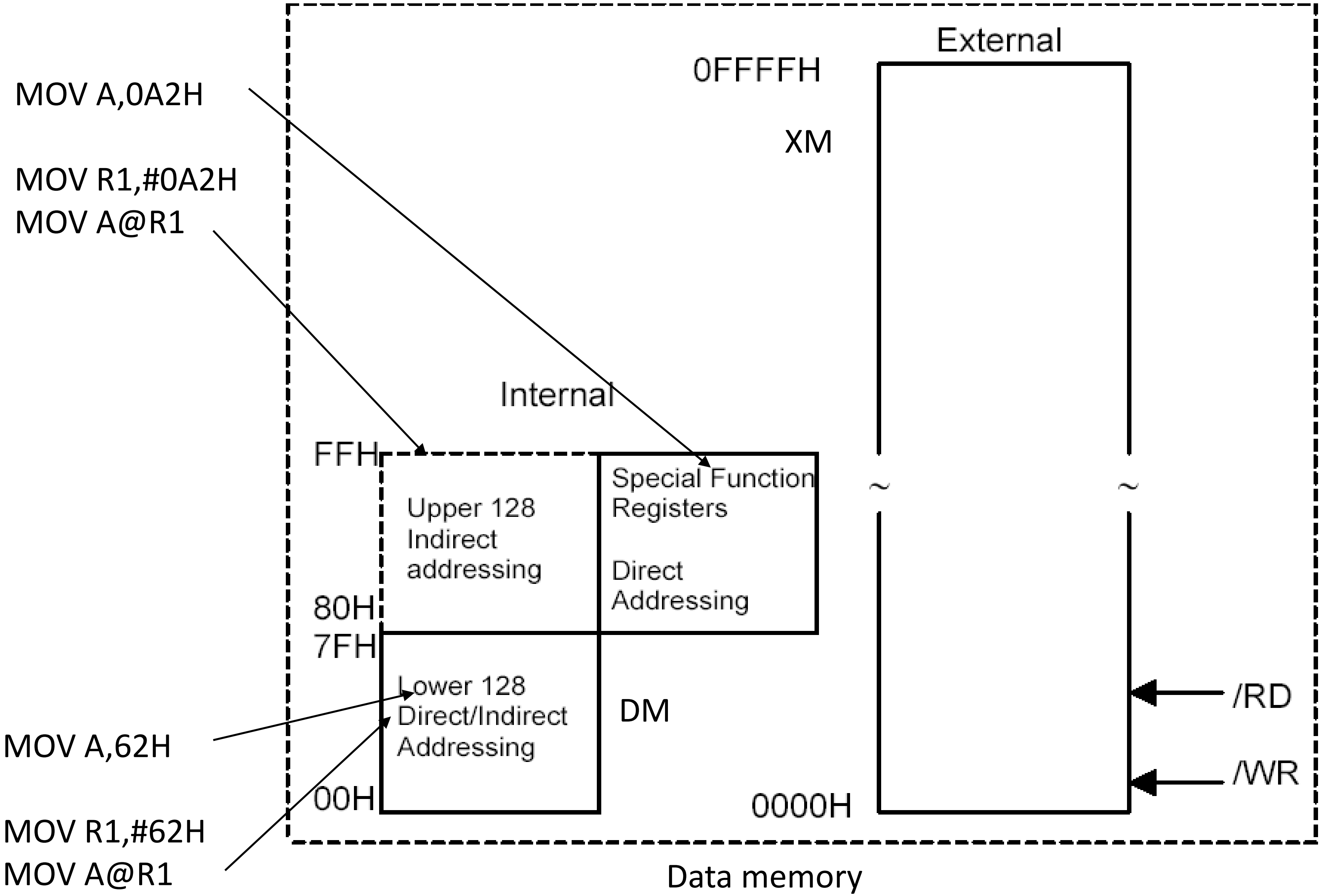
CM

PC = PC(15..0)

DPTR = DPTR(15..0)

Data Memory

- The original 8051 had 128 bytes of on-chip data RAM.
  - This memory includes 4 banks of general purpose registers at DM(00..1F)
  - Only one bank can be active at a time.
  - If all four banks are used, DM(20..7F) is available for program data.
  - DM(20..2F) is bit addressable as BADM(00..7F).
- DM(80,…,FF) contains the special function registers such as I/O ports, timers, UART, etc.
  - Some of these are bit addressable using BADM(80..FF)
- On newer versions of the 8051, DM(80,…,FF) is also use as data memory. Thus, the special functions registers and data memory occupy the same address space. Which is accessed is determined by the instruction being used.


MOV A,0A2H
MOV R1,#0A2H
MOV A@R1
MOV A,62H
MOV R1,#62H
MOV A@R1
External
0FFFFH
XM
Internal
FFH
Upper 128
Indirect
addressing
Special Function
Registers
Direct
Addressing
80H
7FH
Lower 128
Direct/Indirect
Addressing
DM
00H
0000H
/RD
/WR
Data memory

Bit Addressable

Byte Address	Bit Address							
7F	General Purpose RAM							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F	Bank 3							
18								
17	Bank 2							
10								
0F	Bank 1							
08								
07	Default Register Bank for R0 – R7							
00								

Byte Address	Bit Address								
FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	-	D0	PSW
B8	-	-	-	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	-	-	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	Not bit-addressable								SBUF
98	9F	96	95	94	93	92	91	90	SCON
90	97	96	95	94	93	92	91	90	P1
8D	Not bit-addressable								TH1
8C	Not bit-addressable								TH0
8B	Not bit-addressable								TL1
8A	Not bit-addressable								TL0
89	Not bit-addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	Not bit-addressable								PCON
83	Not bit-addressable								DPH
82	Not bit-addressable								DPL
81	Not bit-addressable								SP
80	87	86	85	84	83	82	81	80	P0

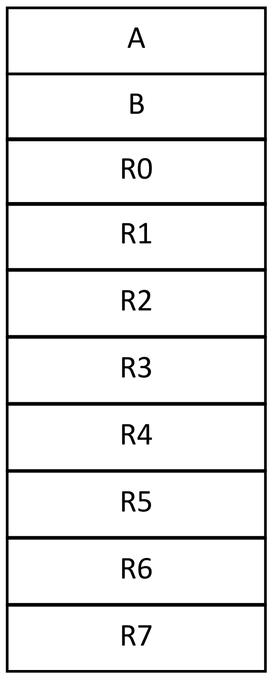
Data Memory (DM)

Table 1

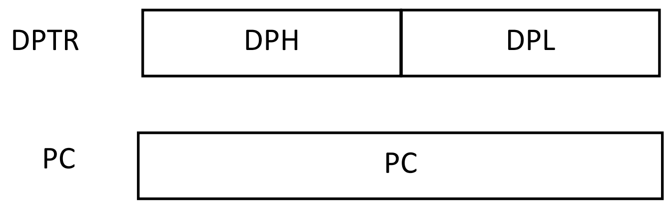
Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 2 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

* = Bit addressable
+ = 8052 only

Register set of 8051



Some 8-bitt Registers of
the 8051



Some 8051 16-bit Register

DPTR

- The data pointer consists of a high byte(DPH) and a low byte (DPL). Its function is to hold a 16 bit address. It may be manipulated as a 16 bit data register or two independent 8 bit register. It serves as a base register in indirect jumps, lookup table instructions and external data transfer.

PROGRAM STATUS WORD (PSW)

CY	AC	F0	RS1	RS0	OV		P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

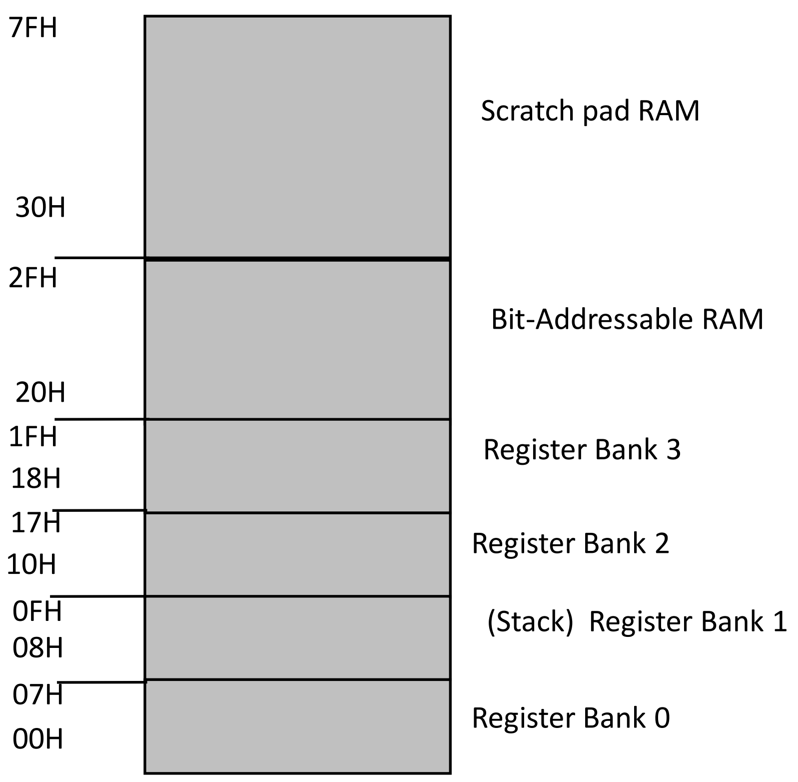
NOTE:
1. The value presented by RS0 and RS1 selects the corresponding register bank.

RS0	RS1	BANK SELECTION
0	0	00H – 07H BANK0
0	1	08H – 0FH BANK 1
1	0	10H – 17H BANK2
1	1	18H – 1FH BANK 3

Stack in the 8051

- The register used to access the stack is called SP (stack pointer) register.

- The stack pointer in the 8051 is only 8 bits wide, which means that it can take value 00 to FFH. When 8051 powered up, the SP register contains value 07.



SPECIAL FUNCTION REGISTERS (SFRs)

- In 8051 microcontroller there certain registers which uses the RAM addresses from 80h to FFh and they are meant for certain specific operations .These registers are called Special function registers (SFRs).Some of these registers are bit addressable also.
- The list of SFRs and their functional names are given below. In these SFRs some of them are related to I/O ports (P0,P1,P2 and P3) and some of them are meant for control operations (TCON,SCON, PCON..) and remaining are the auxiliary SFRs, in the sense that they don't directly configure the 8051.

<table>
<tr><th>S.No</th><th colspan="2">Symbol</th><th>Name of SFR</th><th>Address (Hex)</th></tr>
<tr><td>1</td><td colspan="2">ACC*</td><td>Accumulator</td><td>0E0</td></tr>
<tr><td>2</td><td colspan="2">B*</td><td>B-Register</td><td>0F0</td></tr>
<tr><td>3</td><td colspan="2">PSW*</td><td>Program Status word register</td><td>0DO</td></tr>
<tr><td>4</td><td colspan="2">SP</td><td>Stack Pointer Register</td><td>81</td></tr>
<tr><td rowspan="2">5</td><td rowspan="2">DPTR</td><td>DPL</td><td>Data pointer low byte</td><td>82</td></tr>
<tr><td>DPH</td><td>Data pointer high byte</td><td>83</td></tr>
<tr><td>6</td><td colspan="2">P0*</td><td>Port 0</td><td>80</td></tr>
<tr><td></td><td colspan="2">P1*</td><td>Port 1</td><td>90</td></tr>
<tr><td>8</td><td colspan="2">P2*</td><td>Port 2</td><td>0A</td></tr>
<tr><td>9</td><td colspan="2">P3*</td><td>Port 3</td><td>0B</td></tr>
<tr><td>10</td><td colspan="2">IP*</td><td>Interrupt Priority control</td><td>0B8</td></tr>
<tr><td>11</td><td colspan="2">IE*</td><td>Interrupt Enable control</td><td>0A8</td></tr>
<tr><td>12</td><td colspan="2">TMOD</td><td>Tmier mode register</td><td>89</td></tr>
<tr><td>13</td><td colspan="2">TCON*</td><td>Timer control register</td><td>88</td></tr>
<tr><td>14</td><td colspan="2">TH0</td><td>Timer 0 Higher byte</td><td>8C</td></tr>
<tr><td>15</td><td colspan="2">TL0</td><td>Timer 0 Lower byte</td><td>8A</td></tr>
<tr><td>16</td><td colspan="2">TH1</td><td>Timer 1Higher byte</td><td>8D</td></tr>
<tr><td>17</td><td colspan="2">TL1</td><td>Timer 1 lower byte</td><td>8B</td></tr>
<tr><td>18</td><td colspan="2">SCON*</td><td>Serial control register</td><td>98</td></tr>
<tr><td>19</td><td colspan="2">SBUF</td><td>Serial buffer register</td><td>99</td></tr>
<tr><td>20</td><td colspan="2">PCON</td><td>Power control register</td><td>87</td></tr>
</table>

The 8051 Assembly Language

Addressing Modes

- Register
- Direct
- Register Indirect
- Immediate
- Relative
- Absolute
- Long
- Indexed

Register Addressing Mode

8051 has access to eight working registers (R0 to R7)

- Instructions using register addressing are encoded using the
- three least significant bits of the instruction opcode to specify a register

  Example: ADD A,R7

- The opcode is 00101111. 00101 indicates the instruction and the three lower bits, 111, specify the register.
- Some instructions are specific to a certain register, such as the accumulator, data pointer etc.

  Example: INC DPTR

- A 1-byte instruction adding 1 to the data pointer

  Example: MUL AB

- A 1-byte instruction multiplying unsigned values in accumulator and register B

Direct Addressing Mode

- Direct addressing can access any on-chip memory location

  Example: ADD A,55H

  Example: MOV P1, A

- Transfers the content of accumulator to Port 1 (address90H)

Although the entire of 128 bytes of RAM can be accessed using direct addressing mode, it is most often used to access RAM loc. 30 – 7FH.

```
MOV   R0, 40H
MOV   56H, A
MOV   A, 4                 ; ≡ MOV A, R4
MOV   6, 2                 ; copy R2 to R6
                           ; MOV  R6,R2 is invalid !
```

Register Indirect Addressing Mode

- R0 or R1 may operate as pointer registers (their content indicates an address in internal RAM where data are written or read)
- In 8051 assembly language, indirect addressing is represented by an @ before R0 or R1.
- Example: MOV A, @R0
  - Moves a byte of data from internal RAM at location whose address is in R0 to the accumulator
- In this mode, register is used as a pointer to the data.

MOV A,@Ri ; move content of RAM loc. where address is held by Ri into A (i=0 or 1)

MOV @R1,B

Immediate Addressing Mode

- When the source operand is a constant rather than a variable,
- the constant can be incorporated into the instruction as a byte of immediate address
- In assembly language, immediate operands are preceded by #
- Operand my be a numeric constant, a symbolic variable or an
- arithmetic expression using constants, symbols and operators.
- Assembler computes the value and substitutes the immediate data into the instruction
- Example: MOV A,#12

```
MOV     DPTR,#2343H
MOV     P1,#65H
```

Relative Addressing

- Relative addressing is used with certain jump instructions Relative address (offset) is an 8-bit signed value (-128 to 127) which is added to the program counter to form the address of next instruction.
- Prior to addition, the program counter is incremented to the address following the jump (the new address is relative to the next instruction, not the address of the jump instruction).
- This detail is of no concern to the user since the jump destinations are usually specified as labels and the assembler determines the relative offset.
- Advantage of relative addressing: position independent codes.

Absolute Addressing

- Absolute addressing is only used with ACALL and AJMP.

- The 11 least significant bits of the destination address comes from the opcode and the upper five bits are the current upper five bits in the program counter (PC).

- The destination is in the same 2K (211) of the source.

Long addressing

- Long addressing is used only with the LCALL and LJMP instructions.
- These 3-bytes instructions include a full 16-bit destination address as bytes 2 and 3.
- The full 64K code space is available.
- The instruction is long and position dependent.
- Example: LJMP, 8AF2H.
- Jumps to memory location 8AF2H.

Indexed Addressing Mode

- Indexed addressing uses a base register (either the program counter or data pointer) and an offset (the accumulator) in forming the effective address for a JMP or MOVC instruction

- Example: MOVC A, @A+DPTR

  - This instruction moves a byte of data from code memory to the accumulator. The address in code memory is found by adding the accumulator to the data pointer

Instruction Groups

- The 8051 has 255 instructions
  - Every 8-bit opcode from 00 to FF is used except for A5.

- The instructions are grouped into 5 groups
  - Arithmetic
  - Logic
  - Data Transfer
  - Boolean
  - Branching

Arithmetic Instructions

- ADD
  - 8-bit addition between the accumulator (A) and a second operand.
  - The result is always in the accumulator.
  - The CY flag is set/reset appropriately.

- ADDC
  - 8-bit addition between the accumulator, a second operand and the previous value of the CY flag.
  - Useful for 16-bit addition in two steps.
  - The CY flag is set/reset appropriately.

Example – 16-bit Addition

Add 1E44H to 56CAH

```
CLR     C           ; Clear the CY flag
MOV     A, 44H      ; The lower 8-bits of the 1st number
ADD     A, CAH      ; The lower 8-bits of the 2nd number
MOV     R1, A       ; The result 0EH will be in R1. CY = 1.
MOV     A, 1EH      ; The upper 8-bits of the 1st number
ADDC    A, 56H      ; The upper 8-bits of the 2nd number
MOV     R2, A       ; The result of the addition is 75H
```

The overall result: 750EH will be in R2:R1. CY = 0.

Arithmetic Instructions

- DA
  - Decimal adjust the accumulator.
  - Format the accumulator into a proper 2 digit packed BCD number.
  - Operates only on the accumulator.
  - Works only after the ADD instruction.

- SUBB
  - Subtract with Borrow.
  - Subtract an operand and the previous value of the borrow (carry) flag from the accumulator.
  - $A \leftarrow A$ - <operand> - CY.
  - The result is always saved in the accumulator.
  - The CY flag is set/reset appropriately.

Example – BCD addition

Add 34 to 49 BCD

```
CLR   C          ; Clear the CY flag
MOV   A, #34H    ; Place 1st number in A
ADD   A, #49H    ; Add the 2nd number.
                 ; A = 7DH
DA    A          ; A = 83H
```

Arithmetic Instructions

- INC
  - Increment the operand by one.
  - The operand can be a register, a direct address, an indirect address, the data pointer.

- DEC
  - Decrement the operand by one.
  - The operand can be a register, a direct address, an indirect address.

- MUL AB / DIV AB
  - Multiply A by B and place result in A:B.
  - Divide A by B and place result in A:B.

Logical Operations

- ANL / ORL
- Work on byte sized operands or the CY flag.
    - ANL A, Rn
    - ANL A, direct
    - ANL A, @Ri
    - ANL A, #data
    - ANL direct, A
    - ANL direct, #data
    - ANL C, bit
    - ANL C, /bit

Logical Operations

- XRL
  - Works on bytes only.

- CPL / CLR
  - Complement / Clear.
  - Work on the accumulator or a bit.
    - CLR P1.2

Logical Operations

- RL / RLC / RR / RRC
  - Rotate the accumulator.
  - RL and RR without the carryRLC and RRC rotate through the carry.

- SWAP A
  - Swap the upper and lower nibbles of the accumulator.

- No compare instruction.
  - Built into conditional branching instructions.

Data Transfer Instructions

- MOV <destination>, <source>: allows data to be transferred between any two internal RAM or SFR locations
- Stack operations (pushing and popping data) are also internal data transfer instructions
- Pushing increments SP before writing the data
- Popping from the stack reads the data and decrements the SP
- 8051 stack is kept in the internal RAM8-bit data transfer for internal RAM and the SFR.

• MOV Rn, #data	MOV A, Rn	MOV A, direct
• MOV A, @Ri	MOV A, #data	MOV Rn, A
• MOV Rn, direct	MOV direct, A	MOV direct, Rn
• MOV direct, direct	MOV direct, @Ri	
• MOV direct, #data	MOV @Ri, A	
• MOV @Ri, direct	MOV @Ri, #data	

Data Transfer Operations

- MOV
  - 1-bit data transfer involving the CY flag
  - MOV C, bit
  - MOV bit, C

- MOV
  - 16-bit data transfer involving the DPTR.
  - MOV DPTR, #data

Data Transfer Instructions

- MOVC
  - Move Code Byte
    - Load the accumulator with a byte from program memory.
    - Must use indexed addressing

    - MOVC A, @A+DPTR
    - MOVC A, @A+PC

Data Transfer Instructions

- **MOVX**
  - Data transfer between the accumulator and a byte from external data memory.
    - MOVX A, @Ri
    - MOVX A, @DPTR
    - MOVX @Ri, A
    - MOVX @DPTR, A
- **PUSH / POP**
  - Push and Pop a data byte onto the stack.
  - The data byte is identified by a direct address from the internal RAM locations.
    - PUSH DPL
    - POP 40H

Data Transfer Instructions

- **XCH**
  - Exchange accumulator and a byte variable
    - XCH A, Rn
    - XCH A, direct
    - XCH A, @Ri

- **XCHD**
  - Exchange lower digit of accumulator with the lower digit of the memory location specified.
    - XCHD A, @Ri
    - The lower 4-bits of the accumulator are exchanged with the lower 4-bits of the internal memory location identified indirectly by the index register.
    - The upper 4-bits of each are not modified.

Boolean Operations

- 8051 contains a complete Boolean processor for single-bit operations.
- All bit accesses use direct addressing
- Bits may be set or cleared in a single instruction
- Example: SETB P1.7 CLR P1.7
- This group of instructions is associated with the single-bit operations of the 8051.
    - The P, OV, and AC flags cannot be directly altered.
    - This group includes:
        - Set, clear, and, or complement, move.
        - Conditional jumps.

Boolean Operations

- CLR
  - Clear a bit or the CY flag.
    - CLR P1.1
    - CLR C

- SETB
  - Set a bit or the CY flag.
    - SETB A.2
    - SETB C
- CPL
  - Complement a bit or the CY flag.
    - CPL 40H ; Complement bit 40 of the bit addressable memory

Boolean Operations

- **ORL / ANL**
  - OR / AND a bit with the CY flag.
    - ORL C, 20H ; OR bit 20 of bit addressable memory with the CY flag
    - ANL C, /34H ; AND complement of bit 34 of bit addressable memory with the CY flag.

- **MOV**
  - Data transfer between a bit and the CY flag.
    - MOV C, 3FH ; Copy the CY flag to bit 3F of the bit addressable memory.
    - MOV P1.2, C ; Copy the CY flag to bit 2 of P1.

Boolean Operations

- JC / JNC
  - Jump to a relative address if CY is set / cleared.
- JB / JNB
  - Jump to a relative address if a bit is set / cleared.
    - JB ACC.2, <label>
- JBC
  - Jump to a relative address if a bit is set and clear the bit.

Branching Instructions

- The 8051 provides four different types of unconditional jump instructions:
  - Short Jump – SJMP
    - Uses an 8-bit signed offset relative to the 1^{st} byte of the next instruction.
  - Long Jump – LJMP
    - Uses a 16-bit address.
    - 3 byte instruction capable of referencing any location in the entire 64K of program memory.

Branching Instructions

- Absolute Jump – AJMP
  - Uses an 11-bit address.
  - 2 byte instruction
    - The upper 3-bits of the address combine with the 5-bit opcode to form the 1st byte and the lower 8-bits of the address form the 2nd byte.
  - The 11-bit address is substituted for the lower 11-bits of the PC to calculate the 16-bit address of the target.
    - The location referenced must be within the 2K Byte memory page containing the AJMP instruction.

- Indirect Jump – JMP
  - JMP @A + DPTR

Branching Instructions

- The 8051 provides 2 forms for the CALL instruction:
  - Absolute Call – ACALL
    - Uses an 11-bit address similar to AJMP
    - The subroutine must be within the same 2K page.
  - Long Call – LCALL
    - Uses a 16-bit address similar to LJMP
    - The subroutine can be anywhere.

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

Branching Instructions

- The 8051 provides 2 forms for the return instruction:
  - Return from subroutine – RET
    - Pop the return address from the stack and continue execution there.
  - Return from ISV – RETI
    - Pop the return address from the stack.
    - Restore the interrupt logic to accept additional interrupts at the same priority level as the one just processed.
    - Continue execution at the address retrieved from the stack.
    - The PSW is not automatically restored.

Branching Instructions

- The 8051 supports 5 different conditional jump instructions.
  - ALL conditional jump instructions use an 8-bit signed offset.

  - Jump on Zero – JZ / JNZ
    - Jump if the A == 0 / A != 0
      - The check is done at the time of the instruction execution.

  - Jump on Carry – JC / JNC
    - Jump if the C flag is set / cleared.

Branching Instructions

- Jump on Bit – JB / JNB
  - Jump if the specified bit is set / cleared.
  - Any addressable bit can be specified.

- Jump if the Bit is set then Clear the bit – JBC
  - Jump if the specified bit is set.
  - Then clear the bit.

Branching Instructions

- Compare and Jump if Not Equal – CJNE
  - Compare the magnitude of the two operands and jump if they are not equal.
    - The values are considered to be unsigned.
    - The Carry flag is set / cleared appropriately.

    - CJNE A, direct, rel
    - CJNE A, #data, rel
    - CJNE Rn, #data, rel
    - CJNE @Ri, #data, rel

Branching Instructions

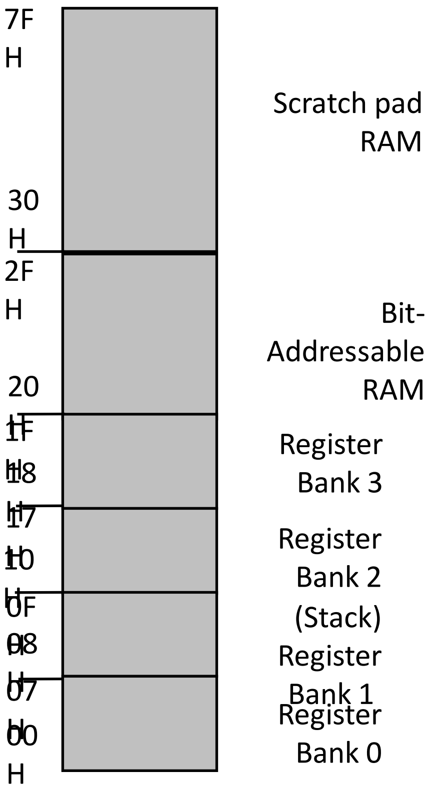
- Decrement and Jump if Not Zero – DJNZ
  - Decrement the first operand by 1 and jump to the location identified by the second operand if the resulting value is not zero.
    - DJNZ Rn, rel
    - DJNZ direct, rel

- No Operation
  - NOP

Stack in the 8051

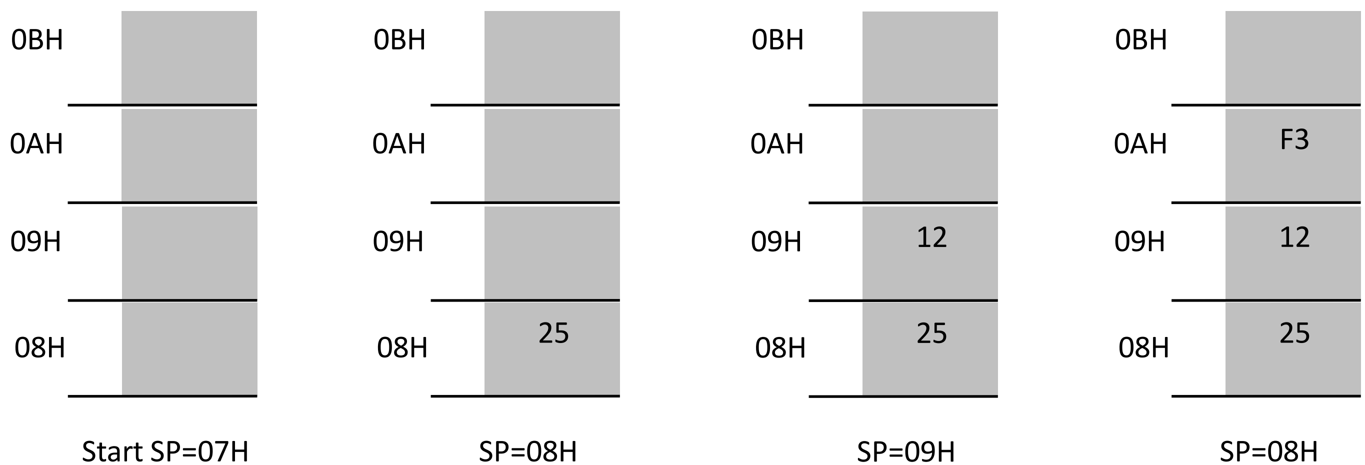
- The register used to access the stack is called SP (stack pointer) register.

- The stack pointer in the 8051 is only 8 bits wide, which means that it can take value 00 to FFH. When 8051 powered up, the SP register contains value 07.



Example:

```
MOV     R6,#25H
MOV     R1,#12H
MOV     R4,#0F3H
PUSH    6
PUSH    1
PUSH    4
```



Example:

Write a program to copy a block of 10 bytes from RAM location starting at 37h to RAM location starting at 59h.

Solution:

```
    MOV   R0,#37h        ; source pointer
    MOV   R1,#59h        ; dest pointer
    MOV   R2,#10         ; counter
L1: MOV   A,@R0
    MOV   @R1,A
    INC   R0
    INC   R1
    DJNZ  R2,L1
```

8051 REAL TIME CONTROL

Interrupts

- An *interrupt* is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.

Interrupts vs. Polling

- A single microcontroller can serve several devices.
- There are two ways to do that:
  - interrupts
  - polling.

- In Polling , the microcontroller ‘s program simply checks each of the I/O devices to see if any device needs servicing. If so, it performs the service.

- In the interrupt method, whenever any device needs microcontroller ‘s service, it tells to microcontroller by sending an interrupt signal.

- The program which is associated with the interrupt is called the ***interrupt service routine*** (ISR) or ***interrupt handler***.

Steps in executing an interrupt

- Finish current instruction and saves the PC on stack.
- Jumps to a fixed location in memory depend on type of interrupt.
- Starts to execute the interrupt service routine until RETI (return from interrupt).
- Upon executing the RETI the microcontroller returns to the place where it was interrupted. Get pop PC from stack.

Interrupt Sources

- Original 8051 has 6 sources of interrupts

1. Reset
2. Timer 0 overflow
3. Timer 1 overflow
4. External Interrupt 0
5. External Interrupt 1
6. Serial Port events buffer full, buffer empty, etc)

Interrupt Vectors

- Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.

External Interrupt 0	:	0003h
Timer 0 overflow	:	000Bh
External Interrupt 1	:	0013h
Timer 1 overflow	:	001Bh
Serial	:	0023h
Timer 2 overflow(8052+)	:	002bh

Note: that there are only 8 memory locations between vectors.

Interrupt Enable (IE) register

- All interrupt are disabled after reset
- We can enable and disable them by IE

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Enables / disables all interrupts
--	IE.6	No implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow interrupt
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 2 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt

Enabling an interrupt

- by bit operation
- Recommended in the middle of program

SETB EA	SETB IE.7	;Enable All
SETB ET0	SETB IE.1	;Enable Timer0 over flow
SETB ET1	SETB IE.3	;Enable Timer1 over flow
SETB EX0	SETB IE.0	;Enable INT0
SETB EX1	SETB IE.2	;Enable INT1
SETB ES	SETB IE.4	;Enable Serial port

- by mov instruction
- Recommended in the first of program
  - **MOV IE, #10010110B**

Disabling an interrupt

```
CLRB  EA       ;Disable All
CLRB  ET0      ; Disable Timer0 over flow
CLRB  ET1      ; Disable Timer1 over flow
CLRB  EX0      ; Disable INT0
CLRB  EX1      ; Disable INT1
CLRB  ES       ; Disable Serial port
```

Interrupt Priorities

- What if two interrupt sources interrupt at the same time?
- The interrupt with the highest PRIORITY gets serviced first.
- All interrupts have a power on default priority order.
  1. External interrupt 0 (INT0)
  2. Timer interrupt0 (TF0)
  3. External interrupt 1 (INT1)
  4. Timer interrupt1 (TF1)
  5. Serial communication (RI+TI)
- Priority can also be set to “high” or “low” by IP reg.

Interrupt Priorities (IP) Register

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

IP.7: reserved

IP.6: reserved

IP.5: timer 2 interrupt priority bit(8052 only)

IP.4: serial port interrupt priority bit

IP.3: timer 1 interrupt priority bit

IP.2: external interrupt 1 priority bit

IP.1: timer 0 interrupt priority bit

IP.0: external interrupt 0 priority bit

Interrupt Priorities Example

---	---	PT2	PS	PT1	PX1	PT0	PX0

- **MOV IP , #00000100B**

  or SETB IP.2 gives priority order

  1. Int1
  2. Int0
  3. Timer0
  4. Timer1
  5. Serial

- **MOV IP , #00001100B**

  gives priority order

  1. Int1
  2. Timer1
  3. Int0
  4. Timer0
  5. Serial

TIMER/COUNTER

- **Counter/timer** hardware is a crucial component of most embedded systems. ... In some cases, a **timer** measures elapsed time (counting processor clock ticks). In others, we want to **count** or time external events. The names **counter** and **timer** can be used interchangeably when talking about the hardware.

- 8051 has two 16-bit programmable timers/counters. They can be configured to operate either as timers or as event counters. The names of the two counters are T0 and T1 respectively.

- The timer content is available in four 8-bit special function registers, viz, TL0,TH0,TL1 and TH1 respectively.

- In the "timer" function mode, the counter is incremented in every machine cycle. Thus, one can think of it as counting machine cycles. Hence the clock rate is 1/12 th of the oscillator frequency.

- In the "counter" function mode, the register is incremented in response to a 1 to 0 transition at its corresponding external input pin (T0 or T1). It requires 2 machine cycles to detect a high to low transition. Hence maximum count rate is 1/24 th of oscillator frequency.

Operation of Timer/Counter

- The operation of the timers/counters is controlled by two special function registers, TMOD and TCON respectively.

Timer Mode control (TMOD) Special Function Register:

- TMOD register is not bit addressable.
- TMOD Address: 89 H

7	6	5	4	3	2	1	0
Gate	$C/\overline{T}$	M1	M0	Gate	$C/\overline{T}$	M1	M0
[Timer 1]				[Timer 0]			

Various bits of TMOD are described as follows -

Symbol	Function
Gate	OR gate enable bit which controls RUN/STOP of timer 1/0. Set to 1 by program to enable timer to run if bit TR1/0 in TCON is set and signal on external interrupt $\overline{INT1}$/0 pin is high. Cleared to 0 by program to enable timer to run if bit TR1/0 in TCON is set.
C/$\overline{T}$	Set to 1 by program to make timer 1/0 act as a counter by counting pulses from external input pins 3.5 (T1) or 3.4 (T0). Cleared to 0 by program to make timer act as a timer by counting internal frequency.
M1	Timer/counter operating mode select bit 1. Set/cleared by program to select mode.
M0	Timer/counter operating mode select bit 0. Set/cleared by program to select mode.

M1	M0	Operating Mode
0	0	3 13-bit Timer (MCS-48 compatible)
0	1	3 16-bit Timer/Counter
1	0	3 8-bit Auto-Reload Timer/Counter
1	1	3 (Timer 0). TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8-bit Timer and is controlled by Timer 1 control bits
1	1	3 (Timer 1) Timer/Counter 1 stopped

Timer/ Counter control logic:

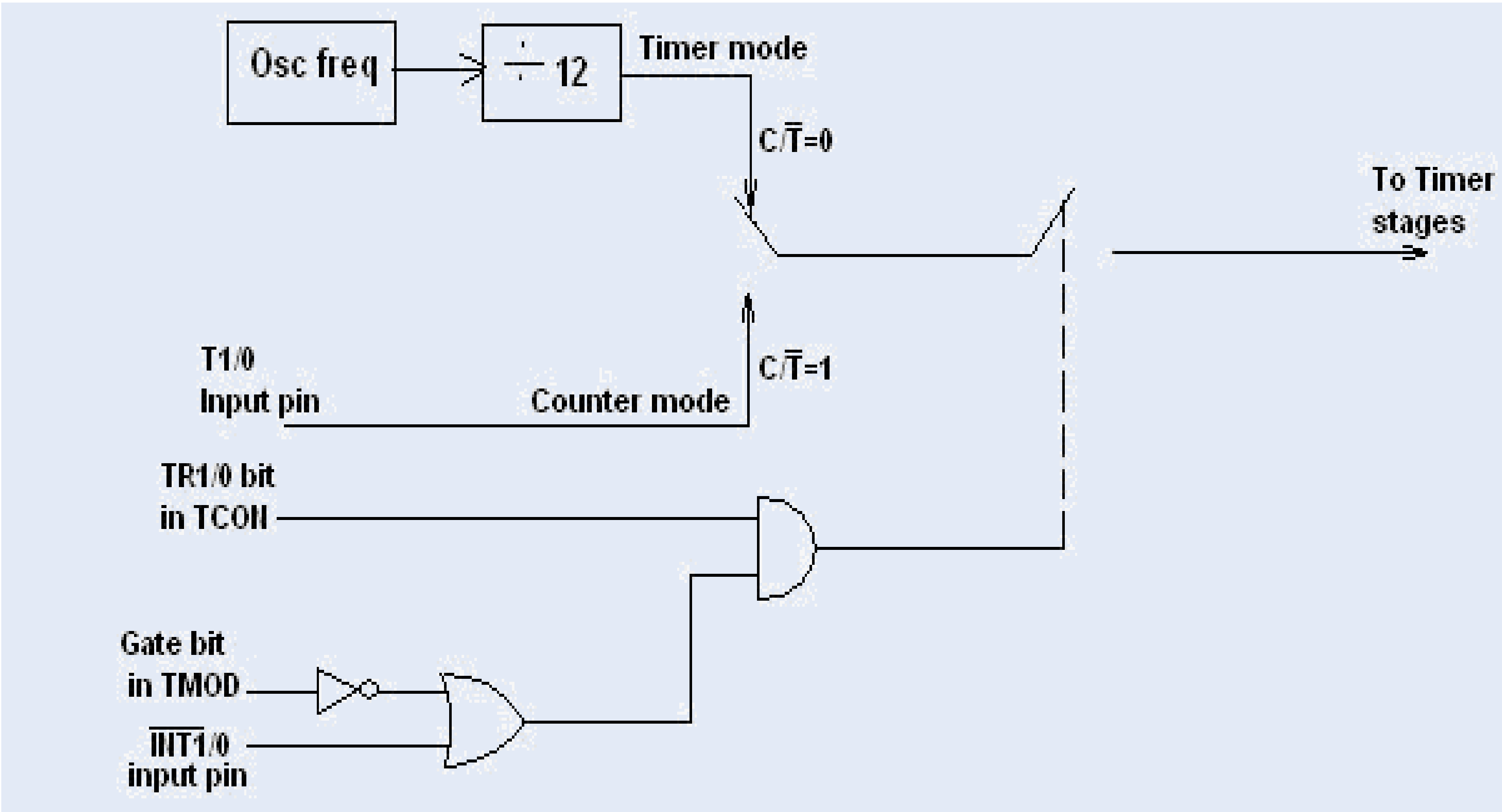


Figure: Timer/ Counter control logic Diagram

Timer modes of operation

Timer Mode-0:

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

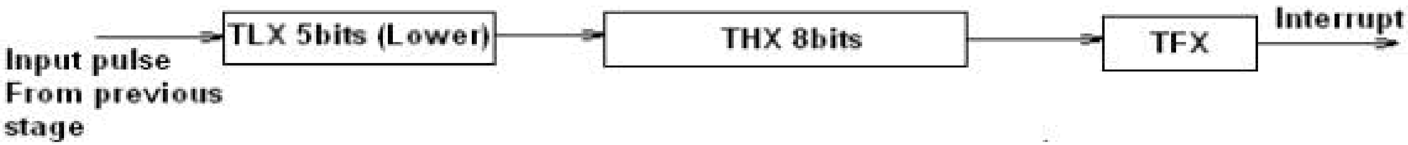


Fig: Operation of Timer in Mode 2

➢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.

➢The input pulse is obtained from the previous stage. If TR1/0 bit is 1 and Gate bit is 0, the counter continues counting up. If TR1/0 bit is 1 and Gate bit is 1, then the operation of the counter is controlled by input. This mode is useful to measure the width of a given pulse fed to input.

Timer Mode-1:

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

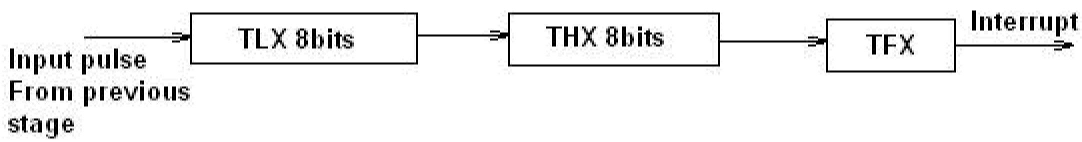


Fig: Operation of Timer in Mode 1

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. For example if we load THX with 50H then the timer in mode 2 will count from 50H to FFH. After that 50H is again reloaded. This mode is useful in applications like fixed time sampling

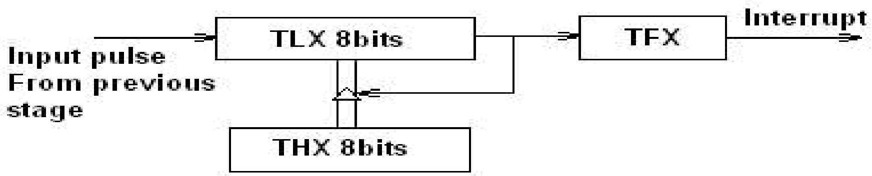


Fig: Operation of Timer in Mode 2

Timer Mode-3:

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.

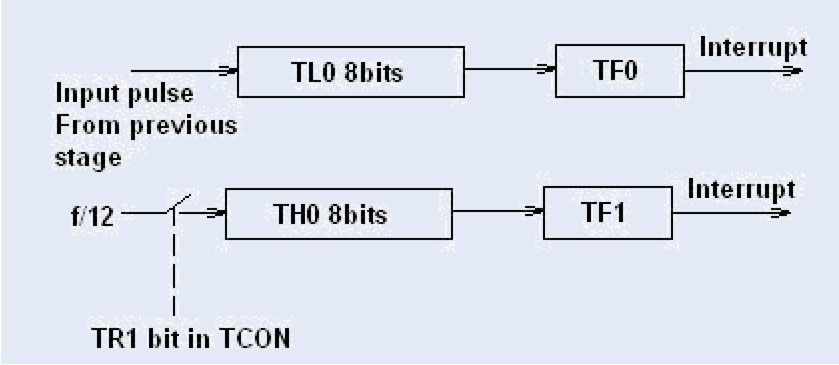
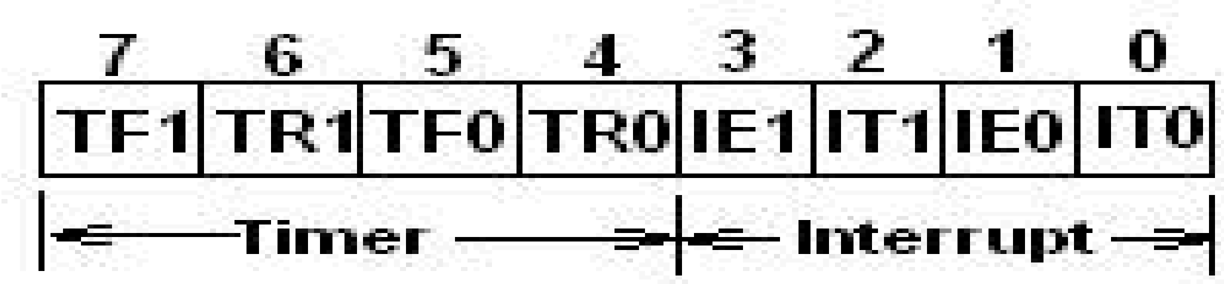


Fig: Operation of Timer in Mode 3

Control bits TR1 and TF1 are used by Timer-0 (higher 8 bits) (TH0) in Mode-3 while TR0 and TF0 are available to Timer-0 lower 8 bits(TL0).

Timer control (TCON) Special function register:

➢TCON is bit addressable. The address of TCON is 88H. It is partly related to Timer and partly to interrupt.



The various bits of TCON are as follows.

TF1 : Timer1 overflow flag. It is set when timer rolls from all 1s to 0s. It is cleared when processor vectors to execute ISR located at address 001BH.

TR1 : Timer1 run control bit. Set to 1 to start the timer / counter.

TF0 : Timer0 overflow flag. (Similar to TF1)

TR0 : Timer0 run control bit.

- IE1 : Interrupt1 edge flag. Set by hardware when an external interrupt edge is detected. It is cleared when interrupt is processed.

- IE0 : Interrupt0 edge flag. (Similar to IE1)

- IT1 : Interrupt1 type control bit. Set/ cleared by software to specify falling edge / low level triggered external interrupt.

- IT0 : Interrupt0 type control bit. (Similar to IT1)
  As mentioned earlier, Timers can operate in four different modes. They are as follows

Timer Delay and Timer Reload Value

- *Timer Delay = Delay Value × Timer Clock Cycle Duration*
- *Delay Value = how many counts before register(s) roll over*
- *Timer Clock Cycle Duration = 6/oscillator frequency*
- *Delay Value = Maximum Register Count – Timer Reload Value*
- *Maximum Register Count = 65535*

Serial communication

- The serial port of 8051 is full duplex, i.e., it can transmit and receive simultaneously.

- The register SBUF is used to hold the data. The special function register SBUF is physically two registers. One is, write-only and is used to hold data to be transmitted out of the 8051 via TXD. The other is, read-only and holds the received data from external sources via RXD. Both mutually exclusive registers have the same address 099H.

Serial Port Control Register (SCON)

Register SCON controls serial data communication.
Address: 098H (Bit addressable)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Mode select bits

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

SM2: multi processor communication bit
REN: Receive enable bit
TB8: Transmitted bit 8 (Normally we have 0-7 bits transmitted/received)
RB8: Received bit 8
TI: Transmit interrupt flag
RI: Receive interrupt flag

Power Mode control Register (PCON)

Register PCON controls processor powerdown, sleep modes and serial data baud rate, only one bit of PCON is used with respect to serial communication. The seventh bit (b7) (SMOD) is used to generate the baud rate of serial communication.

b7							b0
SMOD	—	—	—	GF1	GF0	PD	IDL

Figure: PCON Register

SMOD: Serial baud rate modify bit
GF1: General purpose user flag bit 1
GF0: General purpose user flag bit 0
PD: Power down bit
IDL: Idle mode bit

Generating the baud rates

- Serial port in mode-0
  - Baud rate = oscillating frequency / 12

- Serial port in mode-1

Timer 1 is used to generate the baud rate for mode 1 by using the overflow flag of the timer to determine the baud frequency. Typically, timer 1 is used in timer mode 2 as an autoload 8-bit timer that generates the baud frequency:

$$f_{baud} = \frac{2^{SMOD}}{32} X \frac{fosc}{12 X [256-(TH1)]}$$

- Serial port in mode-2

If smod = 1 then baud rate = 1/32 *oscillator frequency

If smod = 0 then baud rate = 1/64 *oscillator frequency

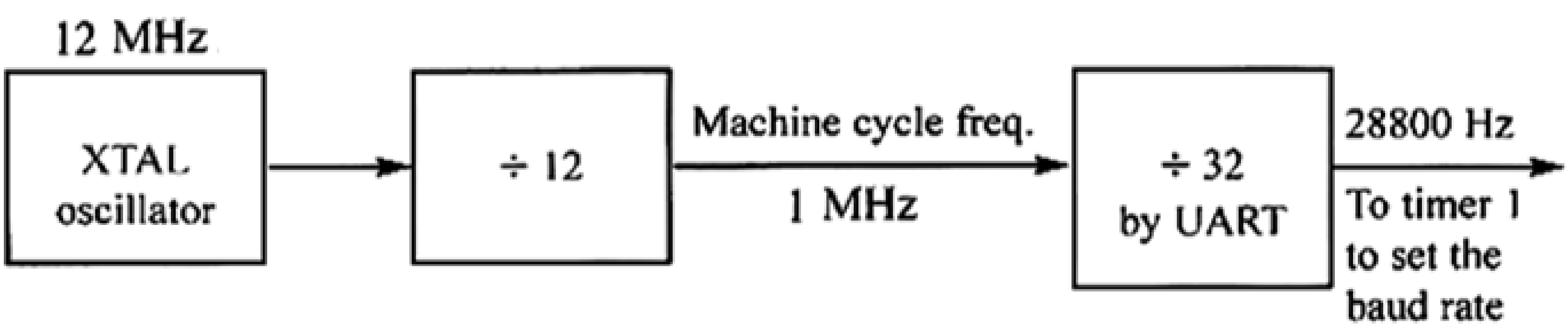
With XTAL = 12 MHz, find the TH1 value needed to have the following baud rates. (a) 9600 (b) 2400 (c) 1200

Solution:

With XTAL = 12 MHz, we have:

The machine cycle frequency of the 8051 = 12 MHz / 12 = 1 MHz, and
921.6 kHz/ 32 = 28,800 Hz is the frequency provided by UART to timer 1 to set baud rate.

(a) 28,800 / 3 = 9600 where –3 = FD (hex) is loaded into TH1
(b) 28,800 / 12 = 2400 where –12 = F4 (hex) is loaded into TH1
(c) 28,800 / 24 = 1200 where –24 = E8 (hex) is loaded into TH1



Timer 1 TH1 Register Values for Various Baud Rates

Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	–3	FD
4800	–6	FA
2400	–12	F4
1200	–24	E8

XTAL = 12 MHz.

- Baud rates for SMOD=0

- Machine cycle freq. = 12 MHz / 12 = 1 MHz

and

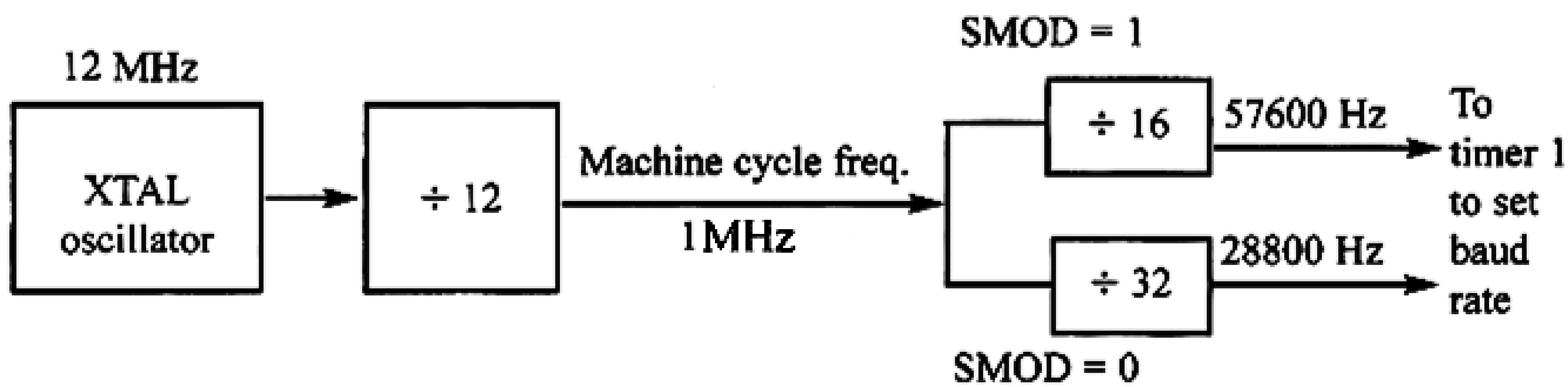
- 1MHz / 32 = 28,800 Hz since SMOD = 0

Examples

Find the TH1 value (in both decimal and hex) to set the baud rate to each of the following.
(a) 9600 (b) 4800 if SMOD = 1 Assume that XTAL = 12 MHz

Solution:

With XTAL = 12 MHz and SMOD = 1, we have timer 1 frequency = 57,600 Hz.
(a) 57,600 / 9600 = 6; therefore, TH1 = –6 or TH1 = FAH.
(b) 57,600 / 4800 = 12; therefore, TH1 = –12 or TH1 = F4H.



Programming Timer interrupts

- A 10khz square wave with 50% duty cycle
  XTAL = 12MHz

```
        ORG     0           ;Reset entry point
        LJMP    MAIN        ;Jump above interrupt
        ORG     000BH       ;Timer 0 interrupt vector
T0ISR:CPL       P1.0        ;Toggle port bit
        RETI                ;Return from ISR to Main program
        ORG 0030H           ;Main Program entry point
MAIN:MOV        TMOD,#02H ;Timer 0, mode 2
        MOV     TH0,#50     ;50 us delay
        SETB    TR0         ;Start timer
        MOV     IE,#82H     ;Enable timer 0 interrupt
        SJMP    main        ;Do nothing just wait
        END
```

Example

Show the instructions to (a) enable the serial interrupt, Timer 0 interrupt, and external hardware interrupt 1 (EX1), and (b) disable (mask) the Timer 0 interrupt, then (c) show how to disable all the interrupts with a single instruction.

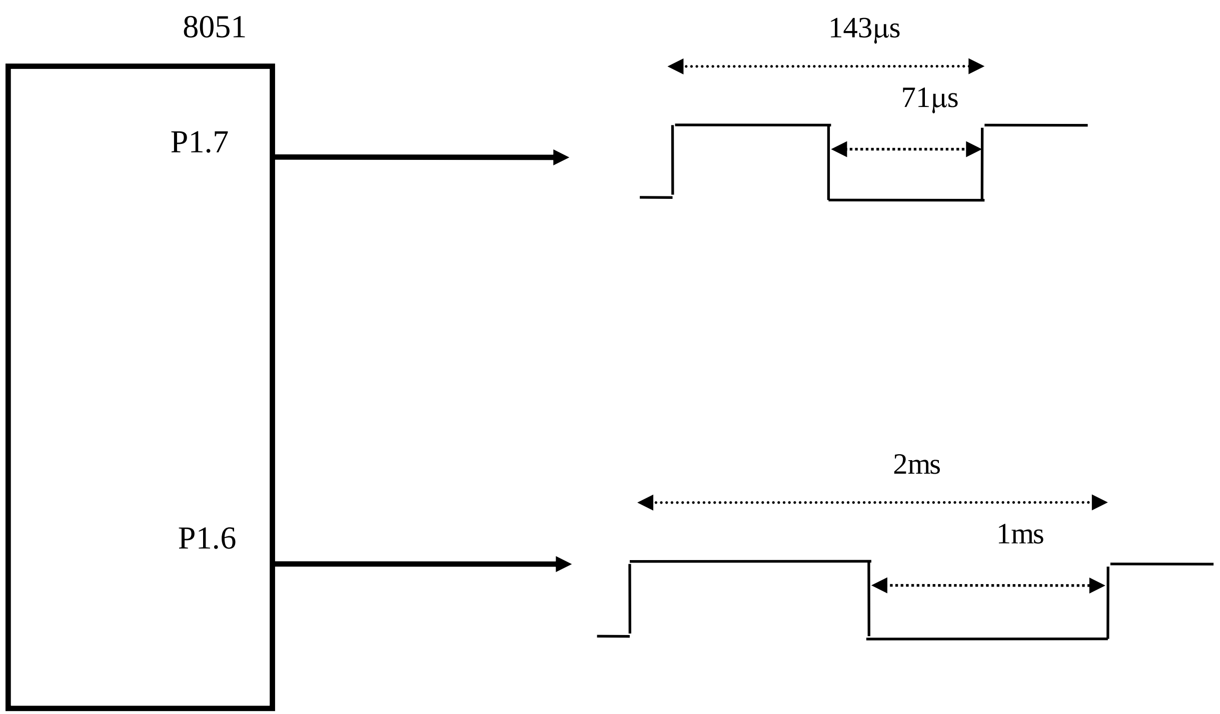
a) MOV IE, #10010110B

b) CLR IE.1

c) CLE IE.7

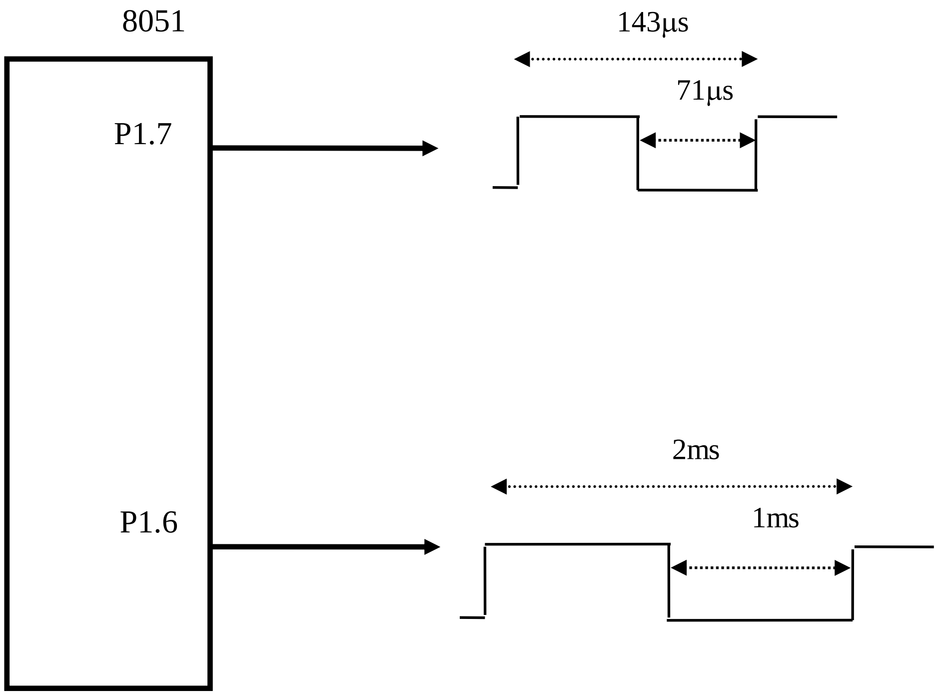
Timer0 & Timer1 Interrupt Example

- Write a program using interrupts to simultaneously create 7 kHz and 500 Hz square waves on P1.7 and P1.6. XTAL = 12MHz



Solution

```
        ORG     0
        LJMP    MAIN
        ORG     000BH
        LJMP    T0ISR
        ORG     001BH
        LJMP    T1ISR
        ORG     0030H
MAIN:   MOV     TMOD,#12H
        MOV     IE,#8AH
        MOV     TH0,#-71
        MOV     TH1,#0fcH
        MOV     TL1,#18H
        SETB    TR1
        SETB    TR0
        SJMP    main
T0ISR:  CPL     P1.7
        RETI
T1ISR:  CLR     TR1
        MOV     TH1,#0fcH
        MOV     TL1,#18H
        SETB    TR1
        CPL     P1.6
        RETI
        END
```



Example

Write a program that continuously gets 8-bit data from P0 and sends it to P1 while simultaneously creating a square wave of 200 ms period on pin P2.1. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

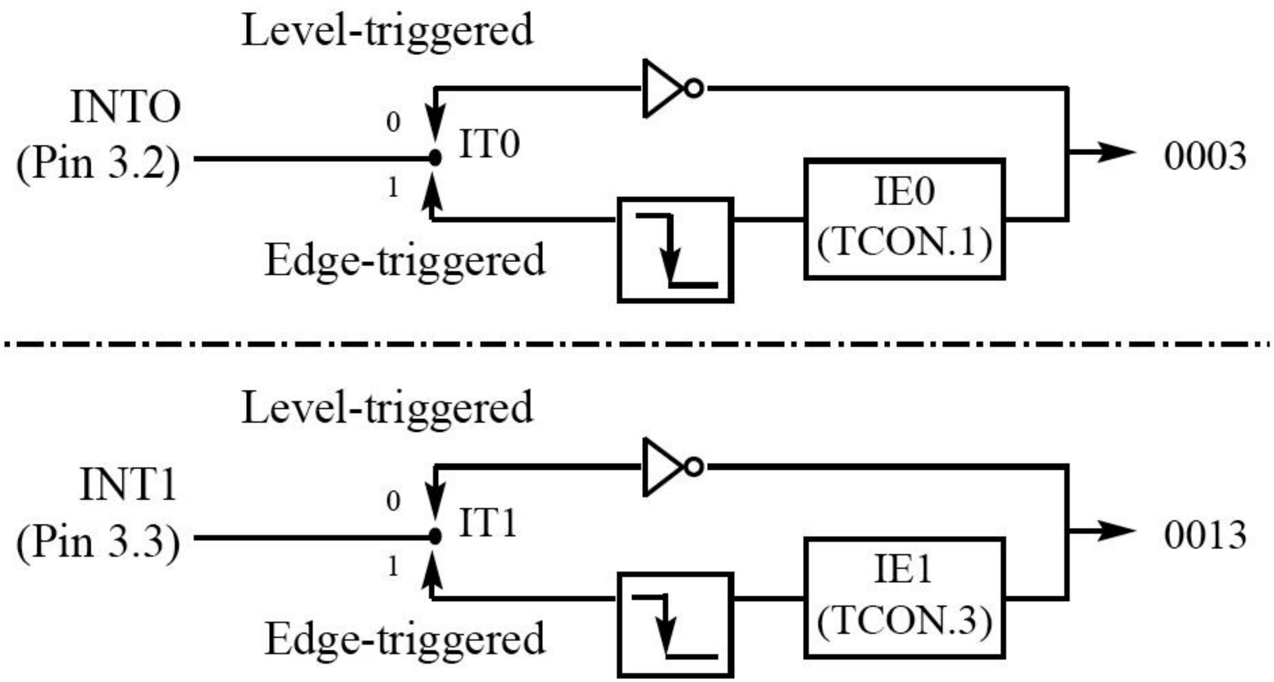
```
ORG 0000H
LJMP MAIN
//ISR FOR TIMER 0 TO GENERATE SQUARE WAVE
ORG 000BH
CPL P2.1
RETI
//
```

CONTINUE….

```
//MAIN PROGRAM FOR INITIALIZATION
ORG 0030H
MAIN: MOV TMOD, #02H
MOV P0, #0FFH
MOV TH0, #-92
MOV IE, #82H
SETB TR0
BACK: MOV A, P0
MOV P1, A
SJMP BACK
END
```

PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

External interrupts INT0 and INT1



Example

Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on.

```
ORG 0000H
 LJMP MAIN
//ISR FOR HARDWARE INTERRUPT
ORG 0013H
 SETB P1.3
MOV R3, #255
BACK: DJNZ R3, BACK
 CLR P1.3
RETI
// MAIN PROGRAM FOR INITIALIZATION
ORG 30H
MAIN: MOV IE, # 10000100B
HERE: SJMP HERE
END
```

Programming the serial communication interrupt

- RI and TI flags and interrupts
  - 1 interrupt is set for serial communication
  - used to both send and receive data
  - when RI or TI is raised the 8051 gets interrupted and jumps to memory address location 0023H to execute the ISR
  - the ISR must examine the TI and RI flags to see which one caused the interrupt and respond accordingly

Example

Write a program in which the 8051 reads data from P1 and writes it to P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

```
ORG 0
LJMP MAIN
ORG 23H
LJMP SERIAL
ORG 30H
MAIN: MOV P1, #0FFH
MOV TMOD, #20H
MOV TH1, #0FDH
MOV SCON, #50H
MOV IE, #10010000B
```

CONTINUE…

```
SETB TR1
BACK: MOV A, P1
MOV SBUF, A
MOV P2, A
SJMP BACK
//SERIAL PORT ISR
ORG 100H
SERIAL: JB TI,TRANS
MOV A, SBUF
CLR RI
RETI
TRANS: CLR TI
RETI
END
```

Programming timers and counters

TIMER-0 IN COUNTER MODE

```
MOV A, TMOD
ORL A, #05H
MOV TMOD,A
SETB TR0
LCALL 68EAH
LOOP: MOV DPTR, #0194H
MOV A, TL0
MOVX @DPTR,A
INC DPTR
MOV A, TH0
MOVX @DPTR, A
LCALL 6748H
SJMP LOOP
```

TIMER-1 IN COUNTER MODE

```
MOV A, TMOD
ORL A, #50H
MOV TMOD,A
SETB TR1
LCALL 68EAH
LOOP: MOV DPTR, #0194H
MOV A, TL1
MOVX @DPTR,A
INC DPTR
MOV A, TH1
MOVX @DPTR, A
LCALL 6748H
SJMP LOOP
```