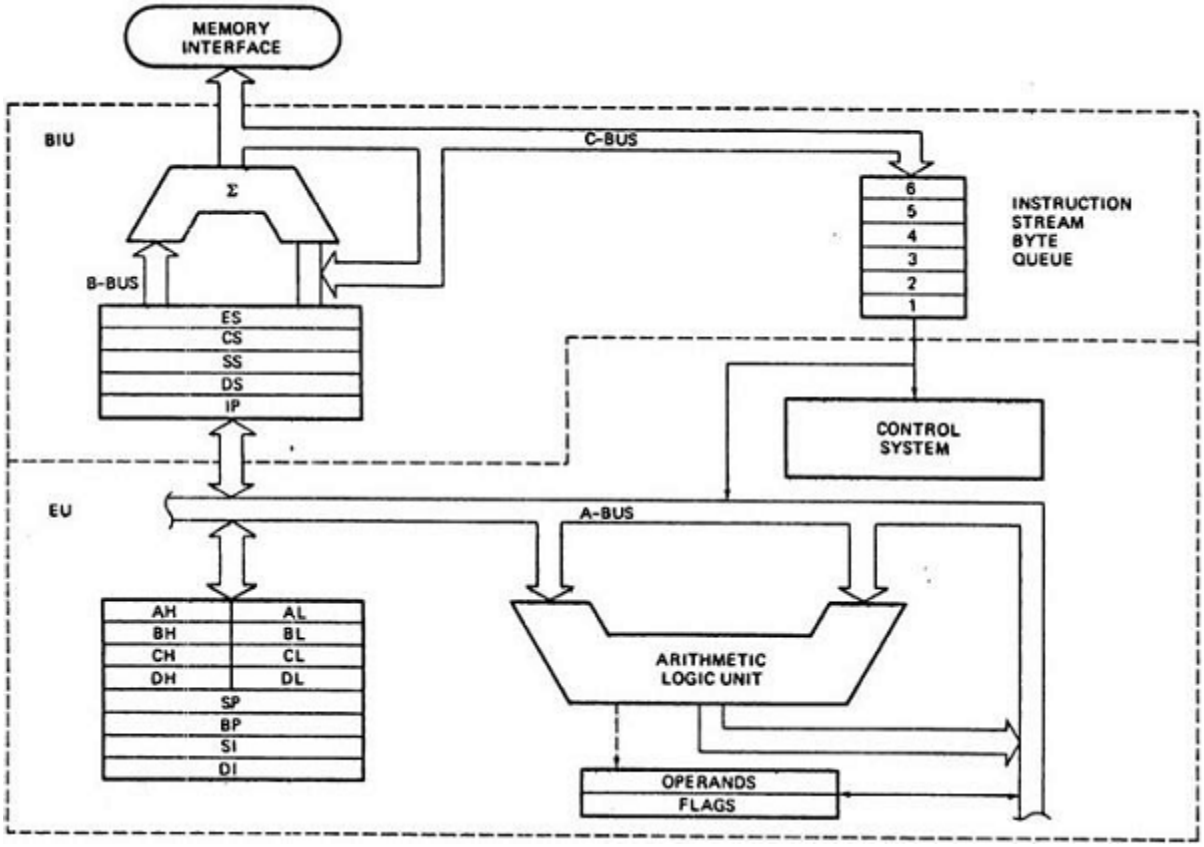
UNIT – 1

Introduction to 8086:

Microprocessor architecture: The following diagram depicts the architecture of an 8086 Microprocessor,



8086 Microprocessor is divided into two functional units, i.e., **EU** (Execution Unit) and **BIU** (Bus Interface Unit).

EU (Execution Unit):

Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions. Its function is to control operations on data using the instruction decoder & ALU. EU has no direct connection with system buses as shown in the above figure; it performs operations over data through BIU.

Let us now discuss the functional parts of 8086 microprocessors.

ALU:

It handles all arithmetic and logical operations, like +, −, ×, /, OR, AND, NOT operations.

Flag Register:

It is a 16-bit register that behaves like a flip-flop, i.e. it changes its status according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups − Conditional Flags and Control Flags.

Conditional Flags:

It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags −

- **Carry flag** − this flag indicates an overflow condition for arithmetic operations.

- **Auxiliary flag** − When an operation is performed at ALU, it results in a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.
- **Parity flag** − This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1's, then the Parity Flag is set. For odd number of 1's, the Parity Flag is reset.
- **Zero flag** − this flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.
- **Sign flag** − This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.
- **Overflow flag** − This flag represents the result when the system capacity is exceeded.

Control Flags:

Control flags controls the operations of the execution unit. Following is the list of control flags −

- **Trap flag** − It is used for single step control and allows the user to execute one instruction at a time for debugging. If it is set, then the program can be run in a single step mode.
- **Interrupt flag** − It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.
- **Direction flag** − It is used in string operation. As the name suggests when it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.

General purpose registers:

There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually to store 8-bit data and can be used in pairs to store 16bit data. The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.

- **AX register** − It is also known as accumulator register. It is used to store operands for arithmetic operations.
- **BX register** − It is used as a base register. It is used to store the starting base address of the memory area within the data segment.
- **CX register** − It is referred to as counter. It is used in loop instruction to store the loop counter.
- **DX register** − this register is used to hold I/O port address for I/O instruction.

Stack pointer register:

It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

BIU (Bus Interface Unit):

BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory

as well as writing data to the ports and the memory. EU has no direction connection with System Buses so this is possible with the BIU. EU and BIU are connected with the Internal Bus.

It has the following functional parts −

- **Instruction queue** − BIU contains the instruction queue. BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed.
- Fetching the next instruction while the current instruction executes is called **pipelining**.
- **Segment register** − BIU has 4 segment buses, i.e. CS, DS, SS& ES. It holds the addresses of instructions and data in memory, which are used by the processor to access memory locations. It also contains 1 pointer register IP, which holds the address of the next instruction to execute by the EU.
    - **CS** − It stands for Code Segment. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
    - **DS** − It stands for Data Segment. It consists of data used by the program andis accessed in the data segment by an offset address or the content of other register that holds the offset address.
    - **SS** − It stands for Stack Segment. It handles memory to store data and addresses during execution.
    - **ES** − It stands for Extra Segment. ES is additional data segment, which is used by the string to hold the extra destination data.
- **Instruction pointer** − It is a 16-bit register used to hold the address of the next instruction to be executed.

8086 was the first 16-bit microprocessor available in 40-pin DIP (Dual Inline Package) chip. Let us now discuss in detail the pin configuration of a 8086 Microprocessor.

INSTRUCTION SETS:

The 8086 microprocessor supports 8 types of instructions −

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

Let us now discuss these instruction sets in detail.

Data Transfer Instructions:

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

Instruction to transfer a word

- **MOV** − Used to copy the byte or word from the provided source to the provided destination.
- **PPUSH** − Used to put a word at the top of the stack.
- **POP** − Used to get a word from the top of the stack to the provided location.
- **PUSHA** − Used to put all the registers into the stack.
- **POPA** − Used to get words from the stack to all registers.
- **XCHG** − Used to exchange the data from two locations.
- **XLAT** − Used to translate a byte in AL using a table in the memory.

Instructions for input and output port transfer

- **IN** − Used to read a byte or word from the provided port to the accumulator.
- **OUT** − Used to send out a byte or word from the accumulator to the provided port.

Instructions to transfer the address

- **LEA** − Used to load the address of operand into the provided register.
- **LDS** − Used to load DS register and other provided register from the memory
- **LES** − Used to load ES register and other provided register from the memory.

Instructions to transfer flag registers

- **LAHF** − Used to load AH with the low byte of the flag register.
- **SAHF** − Used to store AH register to low byte of the flag register.
- **PUSHF** − Used to copy the flag register at the top of the stack.
- **POPF** − Used to copy a word at the top of the stack to the flag register.

Arithmetic Instructions:

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

Following is the list of instructions under this group −

Instructions to perform addition

- **ADD** − Used to add the provided byte to byte/word to word.
- **ADC** − Used to add with carry.
- **INC** − Used to increment the provided byte/word by 1.
- **AAA** − Used to adjust ASCII after addition.
- **DAA** − Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

- **SUB** − Used to subtract the byte from byte/word from word.
- **SBB** − Used to perform subtraction with borrow.
- **DEC** − Used to decrement the provided byte/word by 1.
- **NPG** − Used to negate each bit of the provided byte/word and add 1/2's complement.
- **CMP** − Used to compare 2 provided byte/word.

- **AAS** − Used to adjust ASCII codes after subtraction.
- **DAS** − Used to adjust decimal after subtraction.

Instruction to perform multiplication

- **MUL** − Used to multiply unsigned byte by byte/word by word.
- **IMUL** − Used to multiply signed byte by byte/word by word.
- **AAM** − Used to adjust ASCII codes after multiplication.

Instructions to perform division

- **DIV** − Used to divide the unsigned word by byte or unsigned double word by word.
- **IDIV** − Used to divide the signed word by byte or signed double word by word.
- **AAD** − Used to adjust ASCII codes after division.
- **CBW** − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
- **CWD** − Used to fill the upper word of the double word with the sign bit of the lower word.

Bit Manipulation Instructions:

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

Following is the list of instructions under this group −

Instructions to perform logical operation

- **NOT** − Used to invert each bit of a byte or word.
- **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
- **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
- **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
- **TEST** − Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

- **SHL/SAL** − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR** − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR** − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

Instructions to perform rotate operations

- **ROL** − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- **ROR** − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- **RCR** − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.

- **RCL** − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions:

String is a group of bytes/words and their memory is always allocated in a sequential order.

Following is the list of instructions under this group −

- **REP** − Used to repeat the given instruction till CX ≠ 0.
- **REPE/REPZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **REPNE/REPNZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **MOVS/MOVSB/MOVSW** − Used to move the byte/word from one string to another.
- **COMS/COMPSB/COMPSW** − Used to compare two string bytes/words.
- **INS/INSB/INSW** − Used as an input string/byte/word from the I/O port to the provided memory location.
- **OUTS/OUTSB/OUTSW** − Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCASB/SCASW** − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODSB/LODSW** − Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch and Loop Instructions):

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition −

- **CALL** − Used to call a procedure and save their return address to the stack.
- **RET** − Used to return from the procedure to the main program.
- **JMP** − Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions −

- **JA/JNBE** − Used to jump if above/not below/equal instruction satisfies.
- **JAE/JNB** − Used to jump if above/not below instruction satisfies.
- **JBE/JNA** − Used to jump if below/equal/ not above instruction satisfies.
- **JC** − Used to jump if carry flag CF = 1
- **JE/JZ** − Used to jump if equal/zero flag ZF = 1
- **JG/JNLE** − Used to jump if greater/not less than/equal instruction satisfies.
- **JGE/JNL** − Used to jump if greater than/equal/not less than instruction satisfies.
- **JL/JNGE** − Used to jump if less than/not greater than/equal instruction satisfies.
- **JLE/JNG** − Used to jump if less than/equal/if not greater than instruction satisfies.
- **JNC** − Used to jump if no carry flag (CF = 0)
- **JNE/JNZ** − Used to jump if not equal/zero flag ZF = 0
- **JNO** − Used to jump if no overflow flag OF = 0

- **JNP/JPO** − Used to jump if not parity/parity odd PF = 0
- **JNS** − Used to jump if not sign SF = 0
- **JO** − Used to jump if overflow flag OF = 1
- **JP/JPE** − Used to jump if parity/parity even PF = 1
- **JS** − Used to jump if sign flag SF = 1

Processor Control Instructions:

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group −

- **STC** − Used to set carry flag CF to 1
- **CLC** − Used to clear/reset carry flag CF to 0
- **CMC** − Used to put complement at the state of carry flag CF.
- **STD** − Used to set the direction flag DF to 1
- **CLD** − Used to clear/reset the direction flag DF to 0
- **STI** − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- **CLI** − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Iteration Control Instructions:

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group −

- **LOOP** − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
- **LOOPE/LOOPZ** − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0
- **LOOPNE/LOOPNZ** − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0
- JCXZ − Used to jump to the provided address if CX = 0

Interrupt Instructions;

These instructions are used to call the interrupt during program execution.

- **INT** − Used to interrupt the program during execution and calling service specified.
- **INTO** − Used to interrupt the program during execution if OF = 1
- **IRET** − Used to return from interrupt service to the main program

ADDRESSING MODES:

The different ways in which a source operand is denoted in an instruction is known as **addressing modes**. There are 8 different addressing modes in 8086 programming.

Immediate addressing mode:

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

Example

MOV CX, 4929 H, ADD AX, 2387 H, MOV AL, FFH

Register addressing mode:

It means that the register is the source of an operand for an instruction.

Example
```
MOV CX, AX   ; copies the contents of the 16-bit AX register into
          ; the 16-bit CX register),
ADD BX, AX
```

Direct addressing mode:

The addressing mode in which the effective address of the memory location is written directly in the instruction.

Example
MOV AX, [1592H], MOV AL, [0300H]

Register indirect addressing mode:

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI& SI.

Example
```
MOV AX, [BX]; suppose the register BX contains 4895H, then the contents
          ; 4895H are moved to AX
ADD CX, {BX}
```

Based addressing mode:

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

Example
MOV DX, [BX+04], ADD CL, [BX+08]

Indexed addressing mode:

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

Example
MOV BX, [SI+16], ADD AL, [DI+16]

Based-index addressing mode:

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

Example
ADD CX, [AX+SI], MOV AX, [AX+DI]

Based indexed with displacement mode:

In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

Example
MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]

ASSEMBLER DIRECTIVES:

Definition: Assembler directives are the **instructions** used by the assembler at the time of assembling a source program. More specifically, we can say, assembler directives are the commands or instructions that control the operation of the assembler.
Assembler directives are the instructions provided to the assembler, not the processor as the processor has nothing to do with these instructions. These instructions are also known as **pseudo-instructions** or **pseudo-opcode**.

Assembler directives:

- show the beginning and end of a program provided to the assembler,
- used to provide storage locations to data,
- used to give values to variables,
- Define the start and end of different segments, procedures or macros etc. of a program.
    - We know that assembly language is a less complex and programmer-friendly language used to program the processors.

    - In assembly language programming, the instructions are specified in the form of mnemonics rather in the form of machine code i.e., 0 and 1.
    - But the microprocessor or microcontrollers are specifically designed in a way that they can only understand machine language.

    - Thus assembler is used to convert assembly language into machine code so that it can be understood and executed by the processor.

    - Therefore, to control the generation of machine codes from the assembly language, assembler directives are used.
    - However, machine codes are only generated for the program that must be provided to the processor and not for assembler directives because they do not belong to the actual program.

Assembler Directives of 8086:
These assembler directives are specifically used by 8086:
ASSUME: *Shows the segment name to the assembler*
It provides information to the assembler regarding the name of the program or data segment for that particular segment.

```
ASSUME CS : _DONE
```

This directive specifies that the instruction of the source program is stored in logical segment _DONE.

DD: *Define Double word*
This directive allows the initialization of single or multiple data in the form of double words (i.e., 4 bytes). This is used to inform the assembler that the stored data in memory is a double word.

```
MEM DD 10D50F7B
```

Thus memory stores the given data in the form:

MEM	7BH
	0FH
	D5H
	10H
	00H
	00H
	00H
	00H

DQ: *Define Quad words*
It is used to initialize quad words (8-bytes) either one or more than one. Thereby informing the assembler that the data stored in memory is quad-word.

DT: *Define ten bytes*
It is used to allocate and initialize 10 bytes of a variable.

DUP: *Duplicate*
DUP allows initialization of multiple locations and assigning of values to them. This allows storing of repeated characters or variables in different locations.

```
Book DB 8 DUP (10H, 20H, 30H, 40H)
```

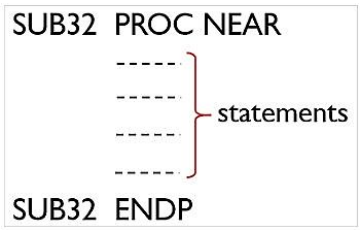
So this permits the storing of these data in memory and creating 8 identical sets in the memory identified as Book.

Book	10H
	20H
	30H
	40H
	10H
	20H
	30H
	40H
	10H
	20H
	30H
	40H

DWORD: *Double word*
This directive is used to indicate that the operand is of double word size.

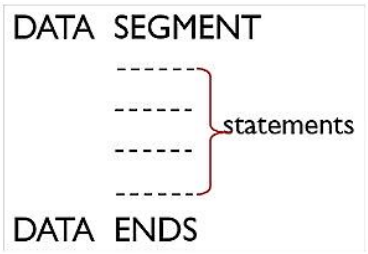
PROC: *Procedure*
It defines the starting of a procedure/subroutine.

FAR: This directive is a type specifier that is used by the assembler to declare intersegment call (i.e., call from different segment).
NEAR: This is used for intersegment call i.e., a call within the same segment.
ENDP: *End of procedure*
This directive shows the termination of a procedure.



SEGMENT: *Beginning of a memory segment.*
It is used to show the beginning of a memory segment with a specific name.

ENDS: *End of segment*
This directive defines the termination of a particular memory segment as it is specified by its name.



The statements within the segment are nothing but the program code.

EVEN: *It is used to inform the assembler to align the data beginning from an even address.*
As data specified with an odd starting address requires 2 byte accessing. Thus using this directive, data can be aligned with an even starting address.

PTR: *Pointer*
This directive shows information regarding the size of the operand.

```
JMP BYTE PTR [BX]
```

This shows legal near jump to BX.

PUBLIC: This directive is used to provide a declaration to variables that are common for different program modules.
STACK: This directive shows the presence of a stack segment.

```
STACK [size]
```

SHORT: This is used in reference to jump instruction in order to assign a displacement of one byte.
THIS: It is used along with EQU directive for setting the label to either, byte, word or double-word.
So, these assembler directives are used by the processors for controlling the generation of machine code and organization of the program.

ASSEMBLY LANGUAGE PROGRAMMING:

1. Need
2. Syntax

3. Development Tools

Need of Assembly language Programming:

We know that programs are a well-defined set of instructions used by programming devices like microprocessor, to perform a specific task.
The instructions in machine level language are written in binary form i.e., using 0 and 1. However, machine level language is quite complex and is not very much user-friendly, leading to cause difficulty in program development.

Hence, a less complex and user-friendly language was developed in which the instructions are specified using a few letters of an English word, thus making it quite easier. This language is known as assembly level language.

But it is noteworthy here that a microprocessor does not hold the ability to execute the programs written in assembly language. Thus it becomes necessary to convert assembly language programs into machine language so that the microprocessor can execute it.

So, a software tool known as assembler is used for converting assembly language into a machine language.

Thus a program written in assembly language is first converted into machine language and then these programs are executed by the microprocessor.

Syntax for Assembly Language Instruction:

Assembly language programming follows a unique syntax, in which the instructions are written in the format:

```
Label : Mnemonic  Operand1, Operand2  ;Comment
```

Label is an optional field of instruction in assembly language and acts as an identifier that provides a symbolic name to the first byte of the instruction and is generally used at the time of branching.

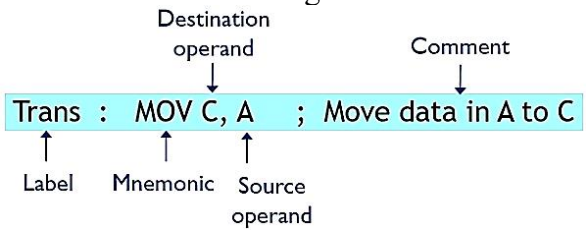
Mnemonics provide the information about the instruction to be performed and is a compulsory field of the instruction format.

Operand specifies the data over which operation is performed. It is to be noted here that the number of operations in an instruction depends on the type of instruction.

This is so because some hold no operand while some hold one or two operands. A comma is used to separate two operands of an instruction.

The comment field starts with a semicolon and it signifies the objective of a particular instruction.

For understanding consider the instruction given below:`



Assembly language program development tools:

Basically, before the fabrication of a microprocessor-based system, the designers first need to design and test the software and hardware of the system. So the development system requires a set of tools. In this section, we will discuss the various tools.

1. **Editor**: An editor is a software tool that allows user, the construction of an assembly language program by providing a set of commands. By using this tool, the user can type and modify the program in assembly code. The program created by an editor is termed as source program.
2. **Assembler**: We have already discussed that an assembler allows conversion of assembly level language into machine language. Assemblers are of different types like one pass; two pass, cross, meta macro and resident assembler.
   Source program generated by editor acts as input to the assembler.
3. **Library Builder**: Library builder is a tool used to generate library files. Basically library files contain the functions that are frequently used by a program.
   For any particular application, whenever software is developed, then the programmers can link the library files with the programs. Linking library files with the program permits the copying of procedure needed by the program from library files to the program.
4. **Linker**: A linker allows the combining of relocatable object files of program modules with the library functions to generate a single executable file.
   Basically **program modules** are nothing but separate procedures of subdivisions of a large program into smaller tasks. This is so because whenever a program is divided into smaller tasks then each task has its individual procedure. Library files are also used by certain tasks according to their availability.
   Each program module allows separate assembling, testing and debugging. Further to have a complete executable file the object files of the modules and library files are linked together. *The* linker creates a link map file the holds the information about the address of the linked files.
5. **Debugger**: A debugger executes a program according to the controlling of the user. A debugger allows the location and correction of errors if present in any program, and this is known as **debugging**.
6. **Simulator**: A program that runs on the PC for simulating the operations of the recently designed system is known as a simulator. It also displays the contents of registers and memory locations on the computer screen. So, as the program operates, the change in contents can be monitored.
7. **Emulator**: An emulator is a tool utilized for testing and debugging both hardware and software of a microprocessor-based system. The system designer loads and runs the program on the emulator, and allows examining and changing of information inside the registers and memory locations.

It is to be noted here that assembly language programs are machine-dependent. This is so because every microprocessor supports different mnemonics. Thus different programs are developed for different microprocessors.

6. MODULAR PROGRAMMING

Modules in **modular programming** enforce logical boundaries between components and improve maintainability. They are incorporated through interfaces. They are designed in such a way as to minimize dependencies between different modules. Teams can develop modules separately and do not require knowledge of all modules in the system. Each and every modular application has a version number associated with it. This provides developers flexibility in module maintenance. If any changes have to be applied to a module, only the affected subroutines have to be changed. This makes the program easier to read and understand. Modular programming has a main module and many auxiliary modules. The main module is compiled as an executable (EXE), which calls the auxiliary module functions. Auxiliary modules exist as separate executable files, which load when the main EXE runs. Each module has a unique name assigned in the PROGRAM statement. Function names across modules should be unique for easy access if functions used by the main module must be exported. Languages that support the module concept are IBM Assembler, COBOL, RPG, FORTRAN, Morph, Zonnon and Erlang, among others. The benefits of using modular programming include:

- Less code has to be written.
- A single procedure can be developed for reuse, eliminating the need to retype the code many times.
- Programs can be designed more easily because a small team deals with only a small part of the entire code.
- Modular programming allows many programmers to collaborate on the same application.
- The code is stored across multiple files.
- Code is short, simple and easy to understand.
- Errors can easily be identified, as they are localized to a subroutine or function.
- The same code can be used in many applications.
- The scoping of variables can easily be controlled.

Example of Modular Programming in C:

```
ALP for Multiplication of two 32-bit numbers
DATA SEGMENT
MULD DW 0FFFFH, 0FFFFH
MULR DW 0FFFFH, 0FFFFH
RES DW 6 DUP (0)
DATA ENDS
ASSUME CS: CODE, DS: DATA
CODE SEGMENT
START: MOV AX, DATA
MOV DS, AX
MOV AX, MULD
MUL MULR
MOV RES, AX
MOV RES+2, DX
MOV AX, MULD+2
MUL MULR
ADD RES+2, AX
```

```
ADC RES+4, DX
MOV AX, MULD
MUL MULR+2
ADD RES+2, AX
ADC RES+4, DX
JNC K
INC RES+6
K: MOV AX, MULD+2
MUL MULR+2
ADD RES+4, AX
ADC RES+6, DX
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

Linking and Relocation:

The DOS linking program links the different object modules of a source program and function library routines to generate an integrated executable code of the source program. The main input to the linker is the .OBJ file that contains the object modules of the source programs. Other supporting information may be obtained from the files generated by the MASM. The linker program is invoked using the following options.

```
C> LINK
```

Or

```
C>LINK MS.OBJ
```

The .OBJ extension is a must for a file to be accepted by the LINK as a valid object file. The first object may generate a display asking for the object file, list file and libraries as inputs and an expected name of the .EXE file to be generated. The output of the link program is an executable file with the entered filename and .EXE extension. This executable filename can further be entered at the DOS prompt to execute the file.

The linked file in binary for run on a computer is commonly known as executable file or simply '.exe.' file. After linking, there has to be re-allocation of the sequences of placing the codes before actually placement of the codes in the memory.

The loader program performs the task of reallocating the codes after finding the physical RAM addresses available at a given instant. The DOS linking program links the different object modules of a source program and functionlibrary routines to generate an integrated executable code of the source program. The main input to the linker is the .OBJ file that contains the object modules of the source programs. Other supporting information may be obtained from the files generated by the MASM. The linked file in binary for run on a computer is commonly known as executable file or simply '.exe.' file. After linking, there has to be re-allocation of the sequences of placing the codes before actually placement of the codes in the memory.

The loader program performs the task of reallocating the codes after finding the physical RAM addresses available at a given instant. The loader is a part of the operating system and places codes into the memory after reading the '.exe' file. This step is necessary because the available memory addresses may not start from 0x0000, and binary codes have to be loaded at the

different addresses during the run. The loader finds the appropriate startaddress. In a computer, the loader is used and it loads into a section of RAM the programthat is ready to run. A program called locator reallocates the linked file and creates a file forpermanent location of codes in a standard format.

Segment combination:

In addition to the linker commands, the assembler provides a means of regulating the way segments in different object modules are organized by the linker.

Segments with same name are joined together by using the modifiers attached to the SEGMENT directives. SEGMENT directive may have the form

Segment name SEGMENT Combination-type

Where the combine-type indicates how the segment is to be located within the load module.

Segments that have different names cannot be combined and segments with the same name but no combine-type will cause a linker error. The possible combine-types are:

PUBLIC – If the segments in different modules have the same name and combine- type PUBLIC, then they are concatenated into a single element in the load module. The ordering in the concatenation is specified by the linker command.

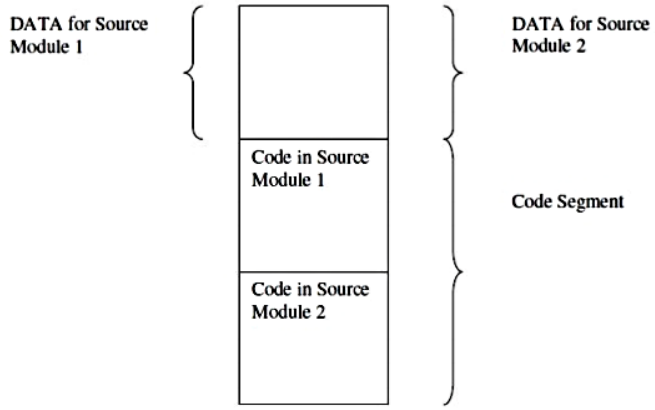
COMMON – If the segments in different object modules have the same name and the combine-type is COMMON, then they are overlaid so that they have the same starting address. The length of the common segment is that of the longest segment being overlaid.

STACK – If segments in different object modules have the same name and the combine type STACK, then they become one segment whose length is the sum of the lengths of the individually specified segments. In effect, they are combined to form one large stack

AT – The AT combine-type is followed by an expression that evaluates to a constant which is to be the segment address. It allows the user to specify the exact location of the segment in memory.

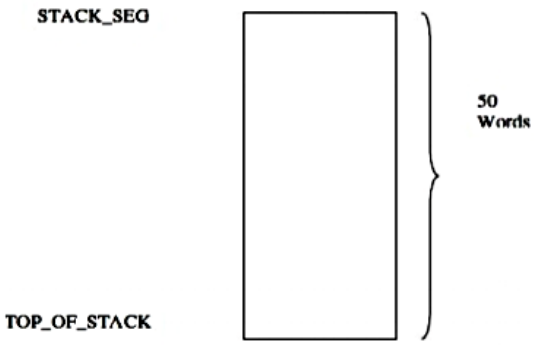
MEMORY – This combine-type causes the segment to be placed at the last of the load module. If more than one segment with the MEMORY combine-type is being linked, only the first one will be treated as having the MEMORY combine type; the others will be overlaid as if they had COMMON combine-type.

```
Source module 1
DATA    SEGMENT   COMMON
DATA        ENDS
CODE        SEGMENT   PUBLIC
CODE        ENDS

Source module 2
DATA    SEGMENT   COMMON
        .
        .
DATA    ENDS
CODE    SEGMENT   PUBLIC
        .
        .
CODE    ENDS
```



Above diagram indicates Segment combinations resulting from the PUBLIC and Common Combination types

```
Source module 1
STACK_SEG  SEGMENT STACK
       DW    20 DUP (?)
       TOP-OF_STACK LABEL WORD
STACK_SEG ENDS
       END

Source module 2
       STACK_SEG        SEGMENT    STACK
        DW   30 DUP (?)
STACK_SEG        ENDS
              .
              .
END
```



Formation of a stack from two segments

Access to External Identifiers:

If an identifier is defined in an object module, then it is said to be a local (or internal) identifier relative to the module. If it is not defined in the module but is defined in one of the other modules being linked, then it is referred to as an external (or global) identifier relative to the module. In order to permit other object modules to reference some of the identifiers in a given module, the given module must include a list of the identifiers towhich it will allow access. Therefore, each module in multi-module programs may contain two lists, one containing the external identifiers that can be referred to by other modules. Two lists are implemented by the EXTRN and PUBLIC directives, which have the forms:

```
EXTRN Identifier: Type..., Identifier: Type
  and
PUBLIC       Identifier,..., Identifier
```

Where the identifiers are the variables and labels being declared or as being available to other modules.

The assembler must know the type of all external identifiers before it can generate the proper machine code; a type specifier must be associated with each identifier in an EXTRN statement. For a variable the type may be BYTE, WORD, or DWORD and for a label it may be NEAR or FAR.

One of the primary tasks of the linker is to verify that every identifier appearing in an EXTRN statement is matched by one in a PUBLIC statement. If this is not the case, then there will be an undefined reference and a linker error will occur. The offsets for the local identifier will be inserted by the assembler, but the offsets for the external identifiers and all segment addresses must be inserted by the linking process. The offsets associated with all external references can be assigned once all of the object modules have been found and their external symbol tables have been examined. The assignment of the segment addresses is called relocation and is done after the linking process has determined exactly where each segment is to be put in memory.

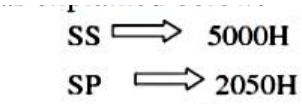
Stacks:

The stack is a block of memory that may be used for temporarily storing the contents of the registers inside the CPU. It is a top-down data structure whose elements are accessed using the stack pointer (SP) which gets decremented by two as we store a data word into the stack and gets incremented by two as we retrieve a data word from the stack back to the CPU register.

The process of storing the data in the stack is called 'pushing into' the stack and the reverse process of transferring the data back from the stack to the CPU register is known as 'popping off' the stack. The stack is essentially Last-In-First-Out (LIFO) data segment. This means that the data which is pushed into the stack last will be on top of stack and will be popped off the stack first.

The stack pointer is a 16-bit register that contains the offset address of the memorylocation in the stack segment. The stack segment, like any other segment, may have amemory block of a maximum of 64 Kbytes locations, and thus may overlap with any other segments. Stack Segment register (SS) contains the base address of the stack segment in the memory.

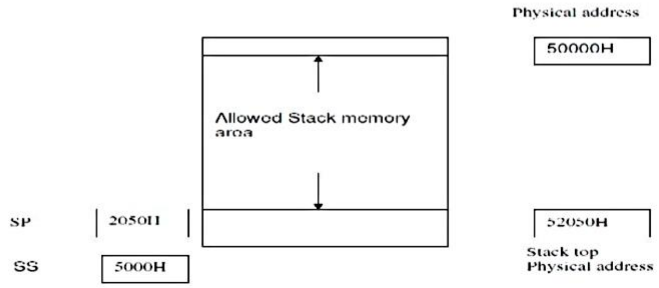
The Stack Segment register (SS) and Stack pointer register (SP) together address the stack- top as explained below:



If the stack top points to a memory location 52050H, it means that the location 52050H is already occupied with the previously pushed data. The next 16 bit push operation will decrement the stack pointer by two, so that it will point to the new stack-top 5204EH and the decremented contents of SP will be 204EH. This location will now be occupied by the recently pushed data.

Thus for a selected value of SS, the maximum value of SP=FFFFH and the segment can have maximum of 64K locations. If the SP starts with an initial value of FFFFH, it will be decremented by two whenever a 16-bit data is pushed onto the stack. After successive push operations, when the stack pointer contains 0000H, any attempt to further push the data to the stack will result in stack overflow.

After a procedure is called using the CALL instruction, the IP is incremented to the next instruction. Then the contents of IP, CS and flag register are pushed automatically to the stack. The control is then transferred to the specified address in the CALL instruction i.e. starting address of the procedure. Then the procedure is executed.



Stack –top address calculation

Procedures:

A procedure is a set of code that can be branched to and returned from in such a way that the code is as if it were inserted at the point from which it is branched to. The branch to procedure is referred to as the call, and the corresponding branch back is known as the return. The return is always made to the instruction immediately following the call regardless of where the call is located.

Calls, Returns, and Procedure Definitions:

The CALL instruction not only branches to the indicated address, but also pushes thereturn address onto the stack. The RET instruction simply pops the return address from the stack. The registers used by the procedure need to be stored before their contents are changed, and then restored just before their contents are changed, and then restored just before the procedure is excited.

A CALL may be direct or indirect and intrasegment or intersegment. If the CALL is intersegment, the return must be intersegment. Intersegment call must push both (IP) and (CS) onto the stack. The return must correspondingly pop two words from the stack. In the case of intrasegment call, only the contents of IP will be saved and retrieved when call and return instructions are used.

Procedures are used in the source code by placing a statement of the form at the beginning of the procedure

Procedure name PROC Attribute and by terminating the procedure with a statement

Procedure name ENDP

The attribute that can be used will be either NEAR or FAR. If the attribute is NEAR, the

RET instruction will only pop a word into the IP registers, but if it is FAR, it will also pop a word into the CS register.

A procedure may be in:

1. The same code segment as the statement that calls it.

2. A code segment that is different from the one containing the statement that calls it, but in the same source module as the calling statement.

3. A different source module and segment from the calling statement.

In the first case, the attribute could be NEAR provided that all calls are in the same code segment as the procedure. For the latter two cases the attribute must be FAR. If theprocedure is given a FAR attribute, then all calls to it must be intersegment calls even if the call is from the same code segment. For the third case, the procedure name must be declared in EXTRN and PUBLIC statements.

Saving and Restoring Registers:

When both the calling program and procedure share the same set of registers, it is necessary to save the registers when entering a procedure, and restore them before returning to the calling program.

```
MSK PROC NEAR
PUSH AX
PUSH BX
PUSH CX
POP CX
POP BX
POP AX
RET
MSK ENDP
```

Procedure Communication:

There are two general types of procedures; those operate on the same set of data and those that may process a different set of data each time they are called. If a procedure is in the same source module as the calling program, then the procedure can refer to the variables directly.

When the procedure is in a separate source module it can still refer to the source module directly provided that the calling program contains the directive

PUBLIC ARY, COUNT, SUM

EXTRN ARY: WORD, COUNT: WORD, SUM: WORD

Recursive Procedures:

When a procedure is called within another procedure it called recursive procedure. To make sure that the procedure does not modify itself, each call must store its set of parameters, registers, and all temporary results in a different place in memory Eg. Recursive procedure to compute the factorial

Macros:

Disadvantages of Procedure:

1. Linkage associated with them.
2. It sometimes requires more code to program the linkage than is needed to perform the task. If this is the case, a procedure may not save memory and execution time is considerably increased.
3. Macros are needed for providing the programming ease of a procedure while avoiding the linkage. Macro is a segment of code that needs to be written only once but whose basic structure can be caused to be repeated several times within a source module by placing a single statement at the point of each reference.

A macro is unlike a procedure in that the machine instructions are repeated each time the macro is referenced. Therefore, no memory is saved, but programming time is conserved (no linkage is required) and some degree of modularity is achieved. The code that is to be repeated is called the prototype code. The prototype code along with the statements for referencing and terminating is called the macro definition.

Once a macro is defined, it can be inserted at various points in the program by using macro calls. When a macro call is encountered by the assembler, the assembler replaces the call with the macro code.Insertion of the macro code by the assembler for a macro call is referred to as a

macro expansion. In order to allow the prototype code to be used in a variety of situations, macro definition and the prototype code can use dummy parameters which can be replaced by the actual parameterswhen the macro is expanded. During a macro expansion, the first actual parameter replaces the first dummy parameter in the prototype code, the second actual parameter replaces the second dummy parameter, and so on.
A macro call has the form%Macro name (Actual parameter list) with the actual parameters being separated by commas.
%MULTIPLY (CX, VAR, XYZ[BX]

Interrupts And Interrupt Routines:

Interrupt and its Need:

The microprocessors allow normal program execution to be interrupted in order to carry out a specific task/work. The processor can be interrupted in the following ways
i) by an external signal generated by a peripheral,
ii) by an internal signal generated by a special instruction in the program,
iii) by an internal signal generated due to an exceptional condition which occurs while executing an instruction. (For example, in 8086 processor, divide by zero is an exceptional condition which initiates type 0 interrupt and such an interrupt is also called execution).
The process of interrupting the normal program execution to carry out a specific task/work is referred to as interrupt. The interrupt is initiated by a signal generated by an external device or by a signal generated internal to the processor.
When a microprocessor receives an interrupt signal it stops executing current normal program, save the status (or content) of various registers (IP, CS and flag registers in case of 8086) in stack and then the processor executes a subroutine/procedure in order to perform the specific task/work requested by the interrupt. The subroutine/procedure that is executed in response to an interrupt is also called Interrupt Service Subroutine (ISR). At the end of ISR, the stored status of registers in stack is restored to respective registers, and the processor resumes the normal program execution from the point {instruction) where it was interrupted.
The external interrupts are used to implement interrupt driven data transfer scheme.
The interrupts generated by special instructions are called software interrupts and they are used to implement system services/calls (or monitor services/calls). The system/monitor services are procedures developed by system designer for various operations and stored inmemory. The user can call these services through software interrupts. The interrupts generated by exceptional conditions are used to implement error conditions in the system.

Interrupt Driven Data Transfer Scheme:

The interrupts are useful for efficient data transfer between processor and peripheral.
When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal. Upon receiving an interrupt signal, the processor suspends the current program execution, save the status in stack and executes an ISR to perform the data transfer between the peripheral and processor.
At the end of ISR the processor status is restored from stack and processor resume its normal program execution. This type of data transfer scheme is called interrupt driven data transfer scheme.
The data transfer between the processor and peripheral devices can be implemented either by polling technique or by interrupt method. In polling technique, the processor has toperiodically poll or check the status/readiness of the device and can perform data transferonly when the device 'is ready. In polling technique the processor time is wasted, because the processor has to suspend its work and check the status of the device in predefined intervals.
If the device interrupts the processor to initiate a data transfer whenever it is ready then the processor time is effectively utilized because the processor need not suspend its work and check the status of the device in predefined intervals.
For an example, consider the data transfer from a keyboard to the processor.

Normally a keyboard has to be checked by the processor once in every 10 milliseconds for a key press. Therefore once in every 10 milliseconds the processor has to suspend its work and then check the keyboard for a valid key code. Alternatively, the keyboard can interrupt the processor, whenever a key is pressed and a valid key code is generated. In this way the processor need not waste its time to check the keyboard once in every 10 milliseconds.

Classification of Interrupts:

In general the interrupts can be classified in the following three ways:

1. Hardware and software interrupts
2. Vectored and Non Vectored interrupt:
3. Maskable and Non Maskable interrupts.

The interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor is called hardware interrupt. The 8086 processor has two interrupt pins INTR and NMI. The interrupts initiated by applying appropriate signal to these pins are called hardware interrupts of 8086.

The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, if software interrupt instruction is encountered then the processor initiates an interrupt. The 8086 processor has 256 types of software interrupts. The software interrupt instruction is INT n, where n is the type number in the range 0 to 255.

When an interrupt signal is accepted by the processor, if the program control automatically branches to a specific address (called vector address) then the interrupt is called vectored interrupt. The automatic branching to vector address is predefined by the manufacturer of processors. (In these vector addresses the interrupt service subroutines
(ISR) are stored). In non-vectored interrupts the interrupting device should supply the address of the ISR to be executed in response to the interrupt. All the 8086 interrupts are vectored interrupts. The vector address for an 8086 interrupt is obtained from a vector table implemented in the first 1kb memory space (00000h to 03FFFh).

The processor has the facility for accepting or rejecting hardware interrupts.

Programming the processor to reject an interrupt is referred to as masking or disabling and programming the processor to accept an interrupt is referred to as unmasking or enabling. In8086 the interrupt flag (IF) can be set to one to unmask or enable all hardware interrupts and
IF is cleared to zero to mask or disable a hardware interrupts except NMI.

The interrupts whose request can be either accepted or rejected by the processor are called maskable interrupts. The interrupts whose request has to be definitely accepted (orcannot be rejected) by the processor are called non-maskable interrupts. Whenever a requestis made by non-maskable interrupt, the processor has to definitely accept that request and service that interrupt by suspending its current program and executing an ISR. In 8086 processor all the hardware interrupts initiated through INTR pin are maskable by clearing interrupt flag (IF). The interrupt initiated through NMI pin and all software interrupts are non-maskable.

Sources of Interrupts in 8086:

An interrupt in 8086 can come from one of the following three sources.

1. One source is from an external signal applied to NMI or INTR input pin of the processor.

The interrupts initiated by applying appropriate signals to these input pins are called hardware interrupts.

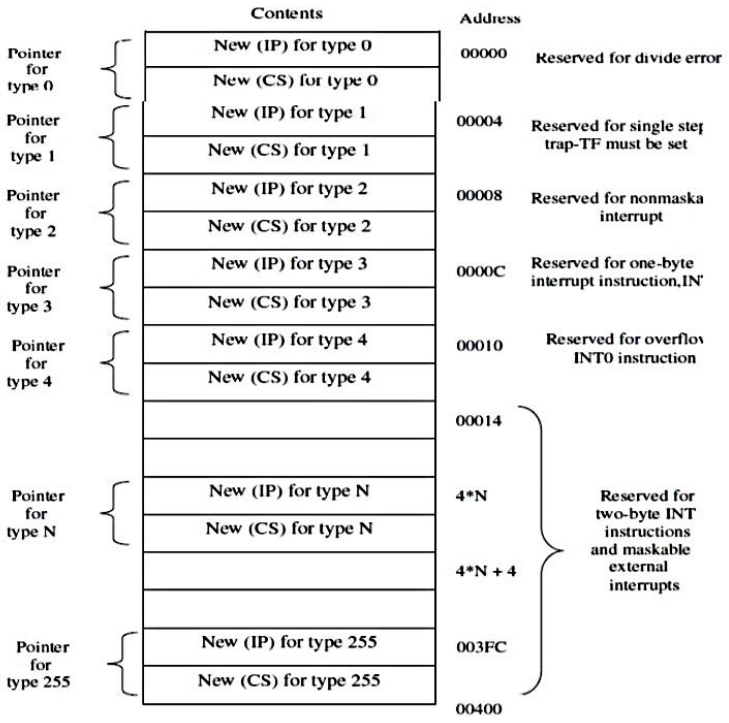
2. A second source of an interrupt is execution of the interrupt instruction "INT n", where n is the type number. The interrupts initiated by "INT n" instructions are called software interrupts.
3. The third source of an interrupt is from some condition produced in the 8086 by the execution of an instruction. An example of this type of interrupt is divide by zero interrupt.

Program execution will be automatically interrupted if you attempt to divide an operand by zero. Such conditional interrupts are also known as exceptions.

Interrupts of 8086:

The 8086 microprocessor has 256 types of interrupts. INTEL has assigned a type number to each interrupt. The type numbers are in the range of 0 to 255. The 8086 processor has dual facility of

initiating these 256 interrupts. The interrupts can be initiated either by executing "INT n" instruction where n is the type number or the interrupt can be initiated by sending an appropriate signal to INTR input pin of the processor.
For the interrupts initiated by software instruction" INT n ", the type number is specified by the instruction itself. When the interrupt is initiated through INTR pin, then the processor runs an interrupt acknowledge cycle to get the type number. (i.e., the interrupting device should supply the type number through D0- D7lines when the processor requests for the same through interrupt acknowledge cycle).



Organization of Interrupt vector table in 8086

Only the first five types have explicit definitions; the other types may be used by interrupt instructions or external interrupts. From the figure it is seen that the type associated with a division error interrupt is 0. Therefore, if a division by 0 is attempted, the processor will push the current contents of the PSW, CS and IP into the stack, fill the IP and CS registers from the addresses 00000 to 00003, and continue executing at the address indicated by the new contents of IP and CS.
A division error interrupt occurs any time a DIV or IDIV instruction is executed with the quotient exceeding the range, regardless of the IF (Interrupt flag) and TF (Trap flag) status.
The type 1 interrupt is the single-step interrupt (Trap interrupt) and is the only interrupt controlled by the TF flag. If the TF flag is enabled, then an interrupt will occur at the end of the next instruction that will cause a branch to the location indicated by the contents of 00004H to 00007H.The single step interrupt is used primarily for debugging which gives the programmer a snapshot of his program after each instruction is executed.

Name	Mnemonics	Description
Interrupt with Type	INT TYPE	(SP) ← (SP) – 2 ((SP)+1:(SP)) ← (PSW) (SP) ← (SP) – 2 ((SP)+1:(SP)) ← (CS) ((SP) ← (SP) – 2 ((SP)+1:(SP)) ← (IP) (IP) ← (TYPE * 4) (CS) ← (TYPE * 4) + 2
One byte interrupt	INT	(SP) ← (SP) – 2 ((SP)+1:(SP)) ← (PSW) (SP) ← (SP) – 2 ((SP)+1:(SP)) ← (CS) ((SP) ← (SP) 2 ((SP)+1:(SP)) ← (IP) (IP) ← (0CH) (CS) ← (0EH)
Interrupt on Overflow	INTO	If (OF) = 1, then (SP) ← (SP) – 2 ((SP)+1:(SP)) ← (PSW) (SP) ← (SP) – 2 ((SP)+1:(SP)) ← (CS) ((SP) ← (SP) – 2 ((SP)+1:(SP)) ← (IP) (IP) ← (10H) (CS) ← (12H)
Return from Interrupt	IRET	(IP) ←((SP)+1:(SP)) (SP) ← (SP) + 2 (CS) ←((SP)+1:(SP)) (SP) ← (SP) + 2 (PSW) ← ((SP)+1:(SP)) (SP) ← (SP) + 2

IRET is used to return from an interrupt service routine. It is similar to the RET instruction except that it pops the original contents of the PSW from the stack as well as the returnaddress. The INT instruction has one of the forms

INT or INT Type

The INT instruction is also often used as a debugging aid in cases where single stepping provides more detail than is wanted.

By inserting INT instructions at key points, called breakpoints. Within a program a programmer can use an interrupt routine to provide messages and other information at these points. Hence the 1 byte INT instruction (Type 3 interrupt) is also referred to as breakpoint interrupt.

The INTO instruction has type 4 and causes an interrupt if and only if the OF flag is set to 1. It is often placed just after an arithmetic instruction so that special processing will be done if the instruction causes an overflow. Unlike a divide-by-zero fault, an overflow condition does not cause an interrupt automatically; the interrupt must be explicitly specified by the INTO instruction. The remaining interrupt types correspond to interrupts instructions imbedded in the interrupt program or to external interrupts.

Byte and StringManipulation:

The 8086 microprocessor is equipped with special instructions to handle string operations.

By string we mean a series of data words or bytes that reside in consecutive memory locations. The string instructions of the 8086 permit a programmer to implement operations such as to move data from one block of memory to a block elsewhere in memory. A second type of operation that is easily performed is to scan a string and data elements stored in memory looking

for a specific value. Other examples are to compare the elements and two strings together in order to determine whether they are the same or different.

Move String: MOV SB, MOV SW:

An element of the string specified by the source index (SI) register with respect to the current data segment (DS) register is moved to the location specified by the destination index (DI) register with respect to the current extra segment (ES) register. The move can be performed on a byte (MOV SB) or a word (MOV SW) of data. After the move is complete, the contents of both SI & DI are automatically incremented or decremented by 1 for a byte move and by 2 for a word move. Address pointers SI and DI increment or decrement depends on how the direction flag DF is set.

Example: Block move program using the move string instruction

```
MOV AX, DATA SEG ADDR
MOV DS, AX
MOV ES, AX
MOV SI, BLK 1 ADDR
MOV DI, BLK 2 ADDR
MOV CK, N
CDF; DF=0
NEXT: MOV SB
LOOP NEXT
HLT
```

Load and store strings: (LOD SB/LOD SW and STO SB/STO SW)

LOD SB: Loads a byte from a string in memory into AL. The address in SI is used relative to DS to determine the address of the memory location of the string element.

(AL) ¬ [(DS) + (SI)]

(SI) ¬ (SI) + 1

LOD SW: The word string element at the physical address derived from DS and SI is to be loaded into AX. SI is automatically incremented by 2.

(AX) ¬ [(DS) + (SI)]

(SI) ¬ (SI) + 2

STO SB: Stores a byte from AL into a string location in memory. This time the contents of ES and DI are used to form the address of the storage location in memory.

[(ES) + (DI)] ¬ (AL)

(DI) ¬ (DI) + 1

STO SW: [(ES) + (DI)] ¬ (AX)

(DI) ¬ (DI) + 2

Repeat String: REP

The basic string operations must be repeated to process arrays of data. This is done by inserting a repeat prefix before the instruction that is to be repeated. Prefix REP causes the basic string operation to be repeated until the contents of register CX become equal to zero.

Each time the instruction is executed, it causes CX to be tested for zero, if CX is found to be nonzero it is decremented by 1 and the basic string operation is repeated.

Example: Clearing a block of memory by repeating STOSB

```
MOV AX, 0
MOV ES, AX
MOV DI, A000
MOV CX, OF
CDF
REP STOSB
NEXT:
```

The prefixes REPE and REPZ stand for same function. They are meant for use with the CMPS and SCAS instructions. With REPE/REPZ the basic compare or scan operation can be repeated as long as both the contents of CX are not equal to zero and zero flag is 1.

REPNE and REPNZ works similarly to REPE/REPZ except that now the operation is repeated as long as CX¹0 and ZF=0. Comparison or scanning is to be performed as long as the string elements are unequal (ZF=0) and the end of the string is not yet found (CX¹0).
Auto Indexing for String Instructions:
SI & DI addresses are either automatically incremented or decremented based on the setting of the direction flag DF.
When CLD (Clear Direction Flag) is executed DF=0 permits auto increment by 1.
When STD (Set Direction Flag) is executed DF=1 permits auto decrement by 1.

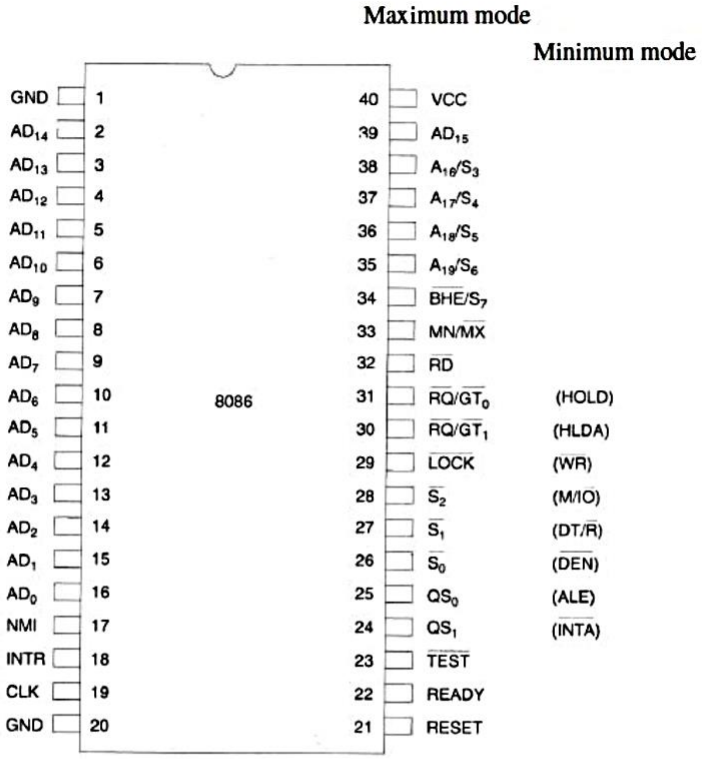
UNIT-II
8086 Processes

The 8086 Microprocessor is a 16-bit CPU available in 3 clock rates, i.e. 5, 8 and 10MHz,packaged in a 40 pin CERDIP or plastic package. The 8086 Microprocessor operates insingle processor or multiprocessor configurations to achieve high performance. The pinconfiguration is as shown in fig1. Some of the pins serve a particular function in minimummode (single processor mode) and others function in maximum mode (multiprocessor mode) configuration.

The 8086 signals can be categorized in three groups. The first are the signals havingcommon functions in minimum as well as maximum mode, the second are the signals whichhave special functions in minimum mode and third are the signals having special functionsfor maximum mode

8086 signals

The following signal description is common for both the minimum and maximum modes:



The 8086 signals can be categorized in three groups. The first are the signals having common functions in minimum as well as maximum mode, the second are the signals which have special functions in minimum mode and third are the signals having special functions for maximum mode.
The following signal description is common for both the minimum and maximum modes.

AD15-AD0: These are the time multiplexed memory I/O address and data lines. Addressremains on the lines during T1 state, while the data is available on the data bus during T2, T3, TW and T4. Here T1, T2, T3, T4 and TW are the clock states of a machine cycle. TW is a wait state. 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 or 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(displayed on S5) is updated at the beginning of each clock cycle. The S4 and S3 combinedly indicate which segment register is presently being used for memory accesses as shown in Table.

These lines float to tri-state off (tri-stated) during the local bus hold acknowledge. The status line S6 is always low (logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

Bus High Enable / status

BHE/S7-Bus High Enable/Status: The bus high enable signal is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in Table. It goes low for the data transfers over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, when- ever a byte is to be transferred on the higher byte of the data bus. The status information is available during T2, T3 and T4. The signal is active low and is tristated during 'hold'. It is low during T1 for the first pulses of the interrupt acknowledge cycle.

$\overline{BHE}$	A_0	Indication
0	0	Whole Word
0	1	Upper byte from or to odd address
1	0	Upper byte from or to even address
1	1	None

Bus high enables status

RD-Read: Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation. RD is active low and shows the state for T2, T3, and TW of any read cycle. The signal remains tristated during the 'hold acknowledge'.

READY: This is the acknowledgement from the slow devices or memory that they havecompleted the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. The signal is active high.

INTR- interrupt Request: This is a level triggered input. This is sampled during the last clock cycle of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. This signal is active high and internally synchronized.

TEST: This input is examined by a 'WAIT' instruction. If the TEST input goes low, execution will continue, else, the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

NMI-Non-maskable Interrupt: This is an edge-triggered input which causes a Type2 interrupt. The NMI is not maskable internally by software. A transition from low to high initiates the interrupt response at the end of the current instruction. This input is internally synchronized.

RESET: This input causes the processor to terminate the current activity and start execution from FFFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronized.

CLK-Clock Input: The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle. The range of frequency for different 8086 versions is from 5MHz to 10MHz.

VCC: +5V power supply for the operation of the internal circuit. GND ground for the internal circuit.

MN/MX: The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode.

The following pin functions are for the **minimum mode operation of 8086**.

M/IO -Memory/IO: This is a status line logically equivalent to S2 in maximum mode.

When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active in the previous T4 and remains active till final T4 of the current cycle. It is tristated during local bus "hold acknowledge".

INTA -Interrupt Acknowledge: This signal is used as a read strobe for interrupt acknowledge cycles. In other words, when it goes low, it means that the processor has accepted the interrupt. It is active low during T2, T3 and TW of each interrupt acknowledge cycle.

ALE-Address latch Enable: This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.

DT /R -Data Transmit/Receive: This output is used to decide the direction of data flowthrough the trans receivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low. Logically, this is equivalent to S1 in maximum mode. Its timing is the same as M/I/O. This is tristated during 'hold acknowledge'.

DEN-Data Enable This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle ofT2 until the middle of T4

DEN is tristated during 'hold acknowledge' cycle.

HOLD, HLDA-Hold/Hold Acknowledge: When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus (instruction) cycle. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input and it should be externally synchronized.

If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T 4 provided:

1. The request occurs on or before T 2 state of the current cycle.

2. The current cycle is not operating over the lower byte of a word (or operating on an odd address).
3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.
4. A Lock instruction is not being executed.
So far we have presented the pin descriptions of 8086 in minimum mode.
The following pin functions are applicable for maximum mode operation of 8086.
S2, S1, and S0 -Status Lines: These are the status lines which reflect the type of operation, being carried out by the processor. These become active during T4 of the previous cycle and remain active during T1 and T2 of the current bus cycle. The status lines return to passive state during T3 of the current bus cycle so that they may again become active for the next bus cycle during T4. Any change in these lines during T3 indicates the starting of a new cycle, and return to passive state indicates end of the bus cycle. These status lines are encoded in table.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

Status lines

LOCK: This output pin indicates that other system bus masters will be prevented fromgaining the system bus, while the LOCK signal is low. The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction.
This floats to tri-state off during "hold acknowledge". When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.
QS1, QS0-Queue Status: These lines give information about the status of the code prefetching queue. These are active during the CLK cycle after which the queue operation is performed. These are encoded as shown in Table.

QS_1,	QS_0	Indication
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

Queue Status

This modification in a simple fetch and execute architecture of a conventional microprocessor offers an added advantage of pipelined processing of the instructions. The 8086 architecture has a 6-byte instruction prefetching queue. Thus even the largest (6- bytes) instruction can be prefetched from the memory and stored in the prefetching queue. This results in a faster execution of the instructions. In 8085, an instruction (opcode and operand) is fetched, decoded and executed and only after the execution of this instruction, the next one is fetched. By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This scheme is known as instruction pipelining. At the starting the CS: IPis loaded with the

required address from which the execution is to be started. Initially, the queue will be empty and the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS:IP address is odd or two bytes at a time, if the CS:IP address is even. The first byte is a complete opcode in case of some instructions (one byte opcode instruction) and it is a part of opcode, in case of other instructions (two byte longopcode instructions), the remaining part of opcode may lie in the second byte. But invariably the first byte of an instruction is an opcode. These opcodes along with data are fetched and arranged in the queue. When the first byte from the queue goes for decoding and interpretation, one byte in the queue becomes empty and subsequently the queue is updated.

The microprocessor does not perform the next fetch operation till at least two bytes of the instruction queue are emptied. The instruction execution cycle is never broken for fetchoperation. After decoding the first byte, the decoding circuit decides whether the instruction is of single opcode byte or double opcode byte. If it is single opcode byte, the next bytes are treated as data bytes depending upon the decoded instruction length, otherwise, the next byte in the queue are treated as the second byte of the instruction opcode. The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data. The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least, two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions. The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program. The main point to be noted here is, that the fetch operation of the next instruction is overlapped with the execution of the current instruction. As shown in the architecture, there are two separate units, namely, execution unit and bus interface unit. While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes o(the next instruction from memory, depending upon the queue status. Figure 1.6 explains the queue operation.

RQ/GT0, RQ/GT1-ReQuest/Grant: These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with RQ/GT0 having higher priority than

RQ/ GT1, RQ/GT pins have internal pull-up resistors and may be left unconnected. The request! Grant sequence is as follows:

1. A pulse one clock wide from another bus master requests the bus access to 8086.
2. During T4 (current) or T1 (next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at next clock cycle. The CPU's bus interface unit is likely to be disconnected from the local bus of the system.
3. A one clock wide pulse from another master indicates to 8086 that the 'hold' request is about to end and the 8086 may regain control of the local bus at the next clock cycle. Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange. The request and grant pulses are active low. For the bus requests those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as discussed in case of HOLD and HLDA in minimum mode.

Basic configurations:

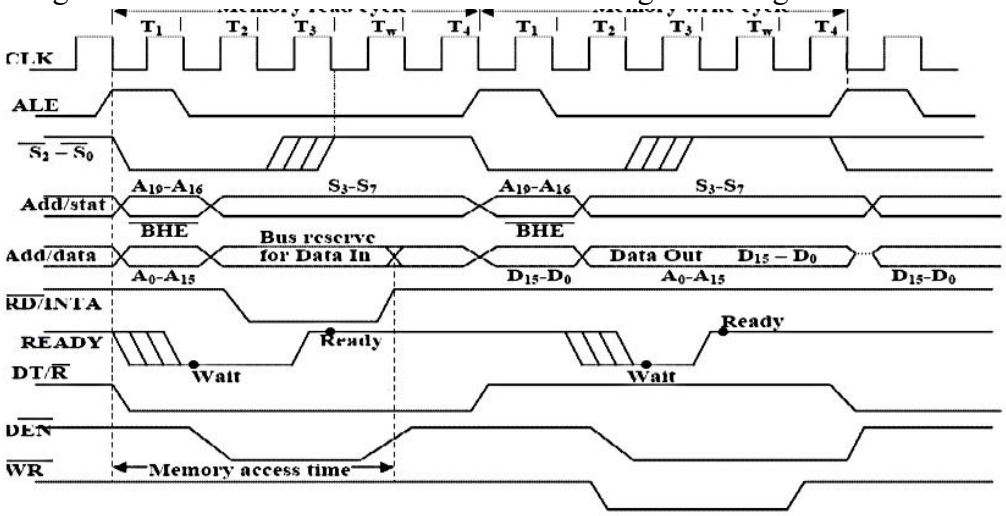
Read Write Timing Diagram

General Bus Operation:

The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus. The main reason behind multiplexing address and data over the same pins is the maximum utilization of processor pins and it facilitates the use of 40 pin standard DIP package. The bus can be demultiplexed using a few latches and trans receivers, whenever required.

Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, and T4. The address is transmitted by the processor during T1, It is present on the bus only for one cycle. The negative edge of this ALE pulse is used to separate the address and the data or status information.

In maximum mode, the status lines S0, S1 and S2 are used to indicate the type of operation. Status bits S3 to S7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.



General Bus operation cycle

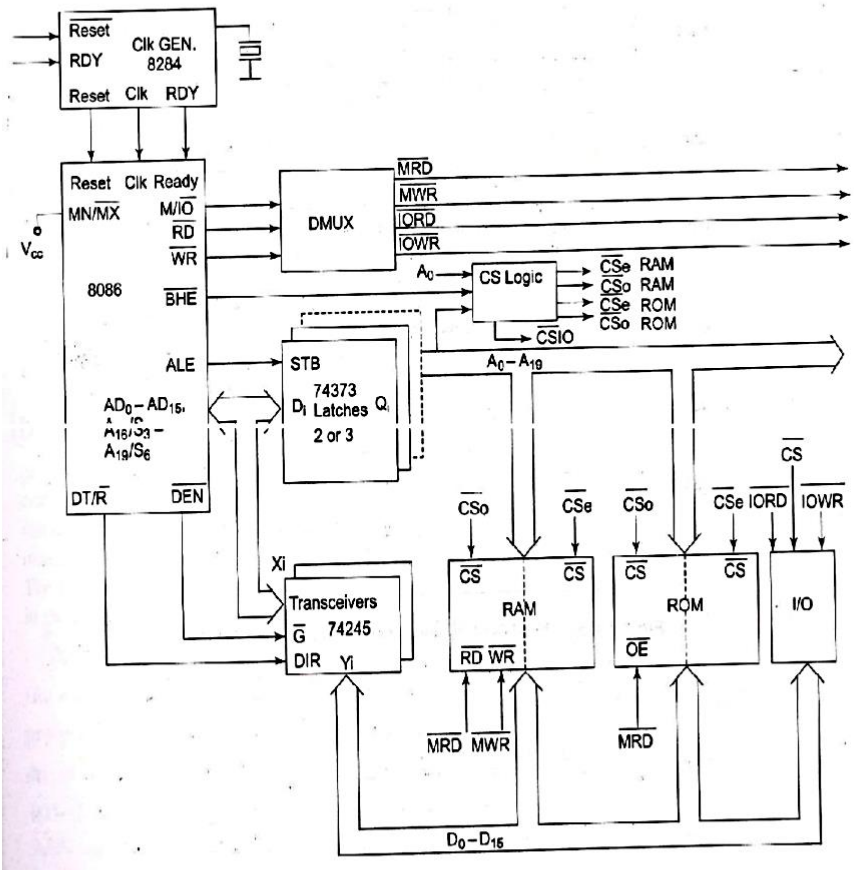
System Bus timings:

Minimum mode 8086 system and timings:

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX* pin to logic1. In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system. The remaining components in the system are latches, trans receivers, clock generator, memory and I/O devices.

The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.

Figure shows the read cycle timing diagram. The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal and also M/IO* signal. During the negative going edge of this signal, the valid address is latched on the local bus. The BHE* and A0 signals address low, high or both bytes. From Tl to T4, the M/IO* signal indicates amemory or I/O operation. At T2 the address is removed from the local bus and is sent to theoutput. The bus is then tristated. The read (RD*) control signal is also activated in T2 .The read (RD) signal causes the addressed device to enable its data bus drivers. After RD* goes low, the valid data is available on the data bus. The addressed device will drive the READY line high, when the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.

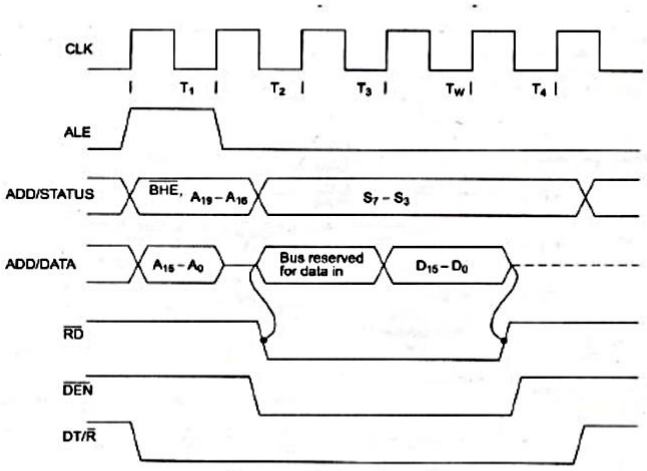


Minimum Mode 8086 System:

A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO* signal is again asserted to indicate a memory or I/O operation. In T2 after sending the address in Tl the processor sends the data to be written to the addressed location. The data remains on the bus until middle of T4 state. The WR* becomes active at the beginning ofT2 (unlike RD* is somewhat delayed in T2 to provide time for floating). The BHE* and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or written. The M/IO*, RD* and WR* signals indicate the types of data transfer as specified in Table.

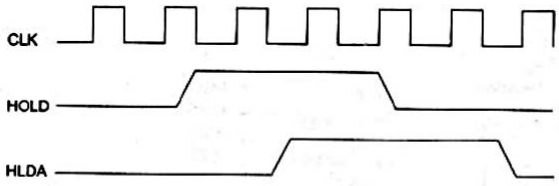
M/IO	RD	WR	Transfer Type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

Read write cycle

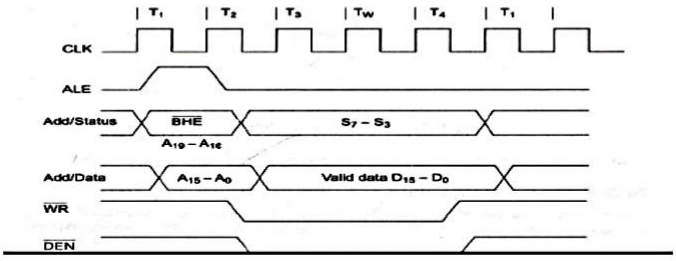


Readcycle timing diagram for minimum mode

HOLD Response Sequence

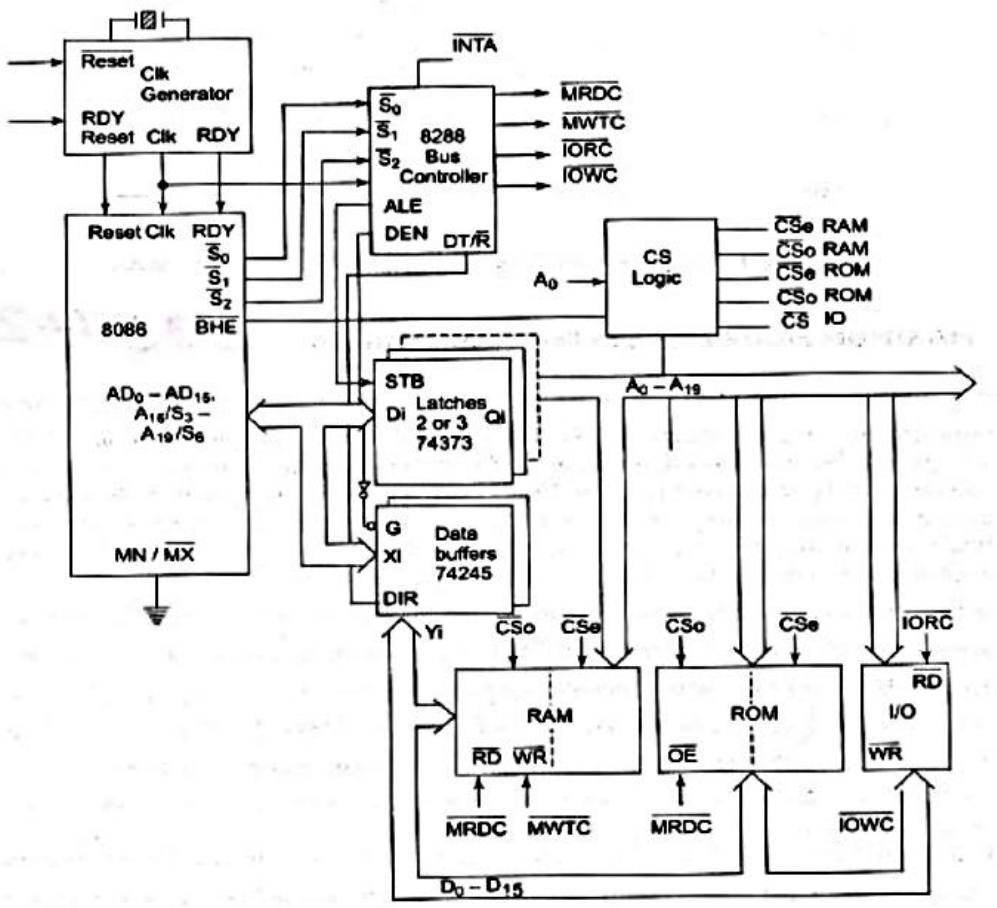


Bus request and bus grant timings in minimum mode system



System Design using 8086:
Maximum mode 8086 system and timings:

In the maximum mode, the 8086 is operated by strapping the MN/MX* pin to ground. In this mode, the processor derives the status signals S2*, S1* and S0*. Another chip called bus controller derives the control signals using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration.
The basic functions of the bus controller chip IC8288, is to derive control signals like
RD* and WR* (for memory and I/O devices), DEN*, DT/R*, ALE, etc. using the information made available by the processor on the status lines. The bus controller chip hasinput lines S2*, S1* and S0* and CLK. These inputs to 8288 are driven by the CPU. It derives the outputs ALE, DEN*, DT/R*, MWTC*, AMWC*, IORC*, IOWC* and AIOWC*. The AEN*, IOB and CEN pins are especially useful for multiprocessor systems. AEN* and IOB are generally grounded. CEN pin is usually tied to +5V.



Maximum mode configuration

The significance of the MCE/PDEN* output depends upon the status of the IOB pin. If IOB is grounded, it acts as master cascade enable to control cascaded 8259A; else it acts as peripheral data enable used in the multiple bus configurations.

INTA* pin is used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

IORC*, IOWC* are I/O read command and I/O write command signals respectively.

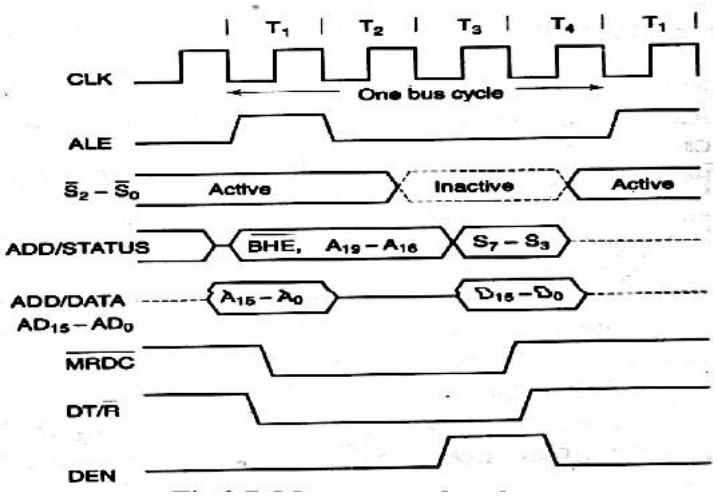
These signals enable an IO interface to read or write the data from or to the addressed port.

The MRDC*, MWTC* are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data from or to the bus.

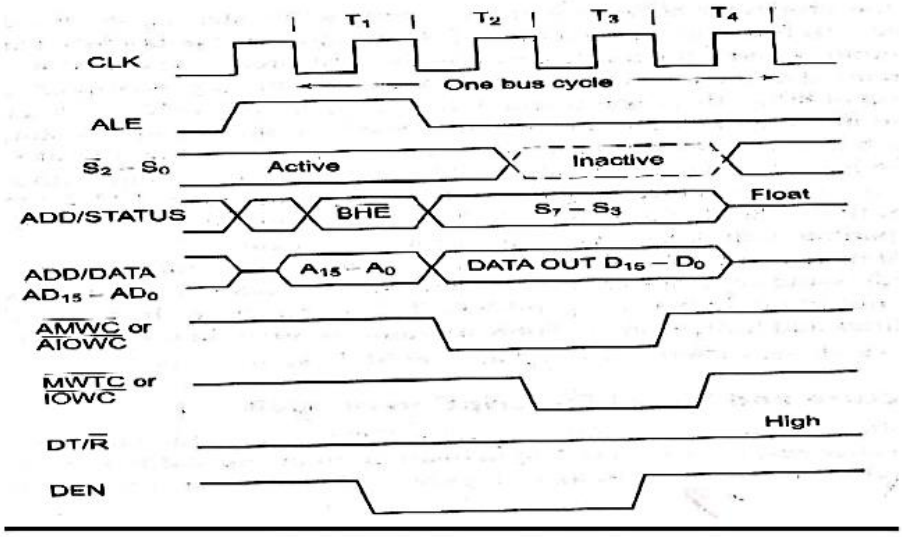
For both of these write command signals, the advanced signals namely AIOWC* and

AMWTC* are available. They also serve the same purpose, but are activated one clock cycle earlier than the IOWC* and MWTC* signals, respectively. The maximum mode system is shown in figure.

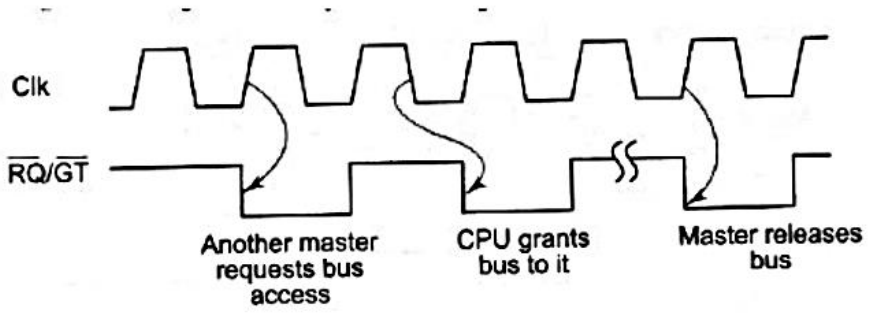
The maximum mode system timing diagrams are also divided in two portions as read (input) and write (output) timing diagrams. The address/data and address/status timings are similar to the minimum mode. ALE is asserted in T1, just like minimum mode. The only difference lies in the status signals used and the available control and advanced command signals. The figure shows the maximum mode timings for the read operation while the figure shows the same for the write operation.



Memory Read Cycle



Memory Write Cycle



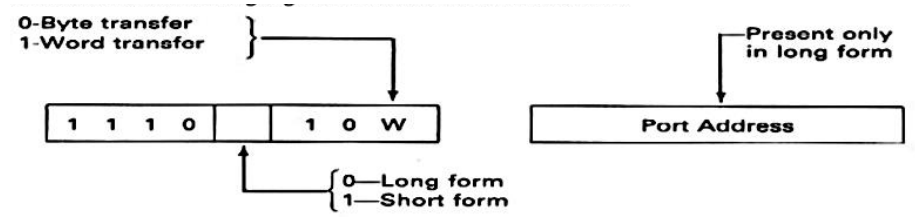
RG/GT* Timings in maximum mode*

IO programming:

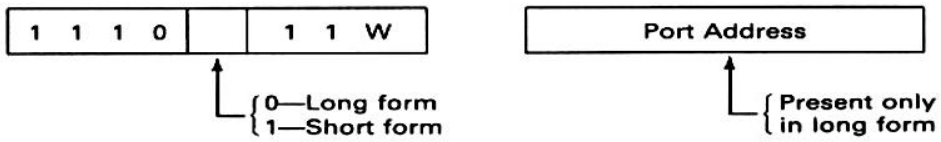
On the 8086, all programmed communications with the I/O ports is done by the IN and OUT instructions defined in Figure.
IN and OUT instructions

Name	Mnemonic and Format	Description
Input		
Long form, byte	IN AL, PORT	(AL) <- (PORT)
Long form, word	IN AX, PORT	(AX) <- (PORT+1: PORT)
Short form, byte	IN AL, DX	(AL) <- ((DX))
Short form, word	IN AX, DX	(AX) <- ((DX) + 1: (DX))
Output		
Long form, byte	OUT PORT, AL	(PORT) <- (AL)
Long form, word	OUT PORT, AX	(PORT+1: PORT) <- (AX)
Short form, byte	OUT DX, AL	((DX)) <- (AL)
Short form, word	OUT DX, AX	((DX)+1: (DX)) <- (AX)

Note: PORT is a constant ranging from 0 to 255
Flags: No flags are affected
Addressing modes: Operands are limited as indicated above.
If the second operand is DX, then there is only one byte in the instruction and the contents of DX are used as the port address. Unlike memory addressing, the contents of DX are not modified by any segment register. This allows variable access to I/O ports in the range 0 to 64K. The machine language code for the IN instruction is:



Although AL or AX is implied as the first operand in an IN instruction, either AL or AX must be specified so that the assembler can determine the W-bit.
Similar comments apply to the OUT instruction except that for it the port address is the destination and is therefore indicated by the first operand, and the second operand is eitherAL or AX. Its machine code is:



Note that if the long form of the IN or OUT instruction is used the port address must be in the range 0000 to 00FF, but for the short form it can be any address in the range 0000 to FFFF (i.e. any address in the I/O address space). Neither IN nor OUT affects the flags. The IN instruction may be used to input data from a data buffer register or the status from a status register. The instructions

```
IN  AX, 28H
MOV DATA_WORD, AX
```

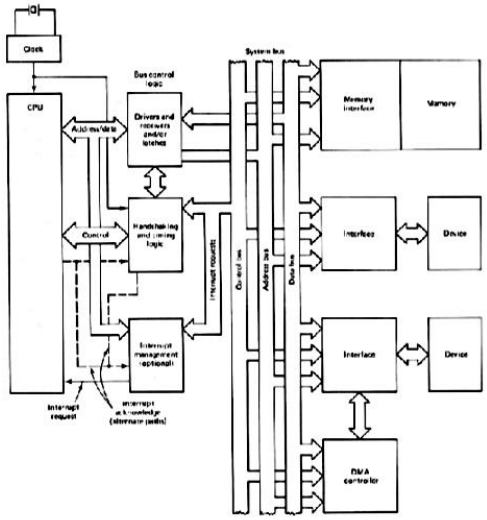
would move the word in the ports whose address are 0028 and 0029 to the memory location DATA_WORD.

Introduction to Multiprogramming:

In order to adapt to as many situations as possible both the 8086 and 8088 have been given two modes of operation, the minimum mode and the maximum mode. The minimum mode is used for a small system with a single processor, a system in which the 8086/8088 generates all the

necessary bus control signals directly (thereby minimizing the required bus control logic). The maximum mode is for medium-size to large systems, which often include two or more processors.

System Bus structure:



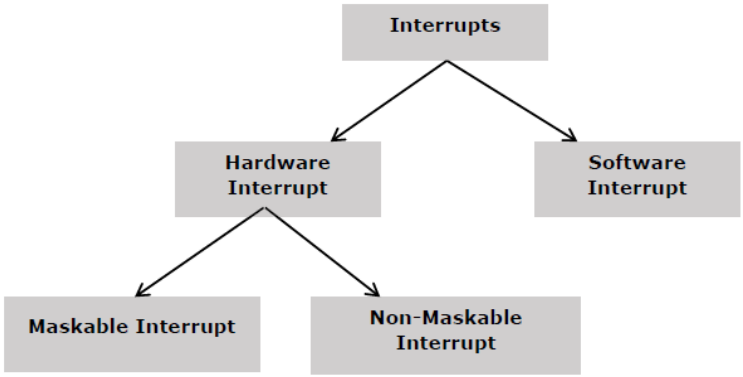
Operation	/BHE	A0	Data pins used
Write/read a word at an even address	0	0	AD15-AD0
Write/read a byte at an even address	1	0	AD7-AD0
Write/read a byte at an odd address	0	1	AD15-AD8
Write/read a word at an odd address	0	1	AD15-AD8 (first bus cycle: puts the least significant data byte on AD15-AD8)
	1	0	AD7-AD0 (Next bus cycle: puts the most significant data byte on AD7-AD0)

Pins for read/ write operation

8086 Interrupts:

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The following image shows the types of interrupts we have in a 8086 microprocessor –



Hardware Interrupts:

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI:

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

When this interrupt is activated, these actions take place −

- Completes the current instruction that is in progress.
- Pushes the Flag register values on to the stack.
- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
- IP is loaded from the contents of the word location 00008H.
- CS is loaded from the contents of the next word location 0000AH.
- Interrupt flag and trap flag are reset to 0.

INTR:

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends '0' on INTA pin twice. The first '0' means INTA informs the external device to get ready and during the second '0' the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor −

- First completes the current instruction.
- Activates INTA output and receives the interrupt type, say X.
- Flag register value; CS value of the return address and IP value of the return address are pushed on to the stack.
- IP value is loaded from the contents of word location X × 4
- CS is loaded from the contents of the next word location.
- Interrupt flag and trap flag is reset to 0

Software Interrupts:

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes −

INT- Interrupt instruction with type number

It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

Its execution includes the following steps −

- Flag register value is pushed on to the stack.

- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 'type number' × 4
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0

The starting address for type0 interrupt is 000000H, for type1 interrupt is 00004H similarly for type2 is 00008H and ……so on. The first five pointers are dedicated interrupt pointers. i.e. −

- **TYPE 0** interrupt represents division by zero situation.
- **TYPE 1** interrupt represents single-step execution during the debugging of a program.
- **TYPE 2** interrupt represents non-maskable NMI interrupt.
- **TYPE 3** interrupt represents break-point interrupt.
- **TYPE 4** interrupt represents overflow interrupt.

The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

INT 3-Break Point Interrupt Instruction

It is a 1-byte instruction having op-code is CCH. These instructions are inserted into the program so that when the processor reaches there, then it stops the normal execution of program and follows the break-point procedure.

Its execution includes the following steps −

- Flag register value is pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 3×4 = 0000CH
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0

INTO - Interrupt on overflow instruction

It is a 1-byte instruction and their mnemonic **INTO**. The op-code for this instruction is CEH. As the name suggests it is a conditional interrupt instruction, i.e. it is active only when the overflow flag is set to 1 and branches to the interrupt handler whose interrupt type number is 4. If the overflow flag is reset then, the execution continues to the next instruction.

Its execution includes the following steps −

- Flag register values are pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of word location 4×4 = 00010H
- CS is loaded from the contents of the next word location.
- Interrupt flag and Trap flag are reset to 0

Introduction to 80186 – 80286 – 80386 – 80486:

80186 ARCHITECTURE:

The 80186 and 80188 like the 8086 and 8088 are nearly identical. The only difference between the 80186 and 80188 is the width of their data buses. The 80186 (like the 8086) contains a 16-bit data bus, while the 80188 (like the 8088) contains an 8-bit data bus. The internal register structure of the 80186/80188 is virtually identical to that of the 8086/8088. About the only difference is that the 80186/80188 contains additional reserved interrupt vectors and some very powerful built-in I/O features. The 80186 and 80188 are often called embedded controllers because of their application as a controller, not as a microprocessor-based computer.

Versions of the 80186/80188:

As mentioned, the 80186 and 80188 are available in four different versions, which are all CMOS microprocessors. Table 16–1 lists each version and the major features provided. The 80C186XL and 80C188XL are the most basic versions of the 80186/80188; the 80C186EC and 80C188EC are the most advanced. This text details the 80C186XL/80C188XL, and then describes the additional features and enhancements provided in the other versions.

80186 Basic Block Diagram:

Figure 16–1 provides the block diagram of the 80188 microprocessor that generically represents all versions except for the enhancements and additional features outlined in Table 16–1. Notice that this microprocessor has a great deal more internal circuitry than the 8088. The block diagrams of the 80186 and 80188 are identical except for the prefect's queue, which is four bytes in the 80188 and six bytes in the 80186. Like the 8088, the 80188 contains a bus interface unit (BIU) and an execution unit (ED). In addition to the BIU and ED, the 80186/80188 family contains a clock generator, a programmable interrupt controller, programmable timers, a programmable DMA controller, and a programmable chip selection unit. These enhancements greatly increase the utility of the 80186/80188 and reduce the number of peripheral components required to implement a system. Many popular subsystems for the personal computer use the 80186/80188 microprocessors as caching disk controllers, local area network (LAN) controllers, and so forth. The 80186/80188 also finds application in the cellular telephone network as a switcher. Software for the 80186/80188 is identical to that for the 80286 microprocessor, without the memory management instructions. This means that the 80286-like instructions for immediate multiplication, immediate shift counts, string I/O, PUSHA, POPA, BOUND, ENTER, and LEAVE all function on the 80186/80188 microprocessors.

TABLE 16–1 The four versions of the 80186/80188 embedded controller.

Feature	*80C186XL 80C188XL*	*80C186EA 80C188EA*	*80C186EB 80C188EB*	*80C186EC 80C188EC*
80286-like instruction set	✓	✓	✓	✓
Power-save (green mode)	✓	✓		✓
Power down mode		✓	✓	✓
80C187 interface	✓	✓	✓	✓
ONCE mode	✓	✓	✓	✓
Interrupt controller	✓	✓	✓	✓ 8259-like
Timer unit	✓	✓	✓	✓
Chip selection unit	✓	✓	✓ enhanced	✓ enhanced
DMA controller	✓ 2-channel	✓ 2-channel		✓ 4-channel
Serial communications unit			✓	✓
Refresh controller	✓	✓	✓ enhanced	✓ enhanced
Watchdog timer				✓
I/O ports			✓ 16 bits	✓ 22 bits

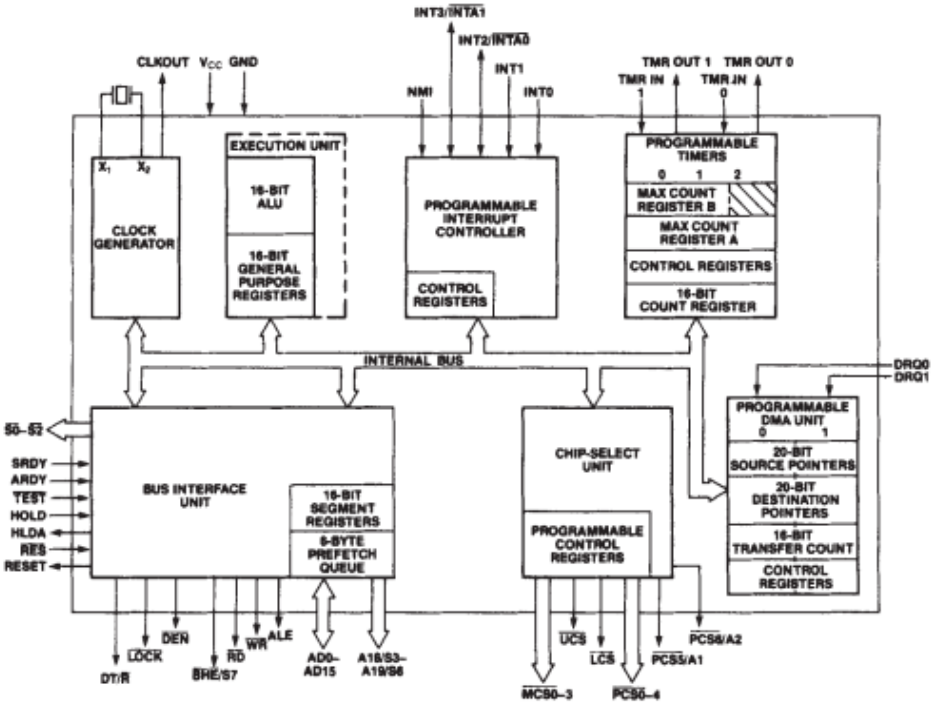


FIGURE 16–1 the block diagram of the 80186 microprocessor. Note that the block diagram of the 80188 is identical, except that is missing and AD15–AD8 are relabeled A15–A8. (Courtesy of Intel Corporation.)

80186/80188 Basic Features

In this segment of the text, we introduce the enhancements of the 80186/80188 microprocessors or embedded controllers that apply to all versions except where noted, but we do not provide

exclusive coverage. More details on the operation of each enhancement and details of each advanced version are provided later in the chapter.

Clock Generator. The internal clock generator replaces the external 8284A clock generator used with the 8086/8088 microprocessors. This reduces the component count in a system. The internal clock generator has three pin connections: X1, X2, and CLKOUT (or on some versions: CLKIN, OSCOUT, and CLKOUT). The X1 (CLKIN) and X2 (OSCOUT) pins are connected to a crystal that resonates at twice the operating frequency of the microprocessor. In the 8 MHz version of the 80186/80188, a 16 MHz crystal is attached to X1 (CLKIN) and X2 (OSCOUT). The 80186/80188 is available in 6 MHz, 8 MHz, 12 MHz, 16 MHz, or 25 MHz versions. The CLKOUT pin provides a system clock signal that is one half the crystal frequencies, with a 50% duty cycle. The CLKOUT pin drives other devices in a system and provides a timing source to additional microprocessors in the system.

In addition to these external pins, the clock generator provides the internal timing for synchronizing the READY input pin, whereas in the 8086/8088 system, READY synchronization is provided by the 8284A clock generator.

Programmable Interrupt Controller. The programmable interrupt controller (PIC) arbitrates the internal and external interrupts and controls up to two external 8259A PICs. When an external 8259 is attached, the 80186/80188 microprocessors function as the master and the 8259 functions as the slave. The 80C186EC and 80C188EC models contain an 8259A-compatible interrupt controller in place of the one described here for the other versions (XL, EA, and EB). If the PIC is operated without the external 8259, it has five interrupt inputs: INTO–INT3 and NMI. Note that the number of available interrupts depends on the version: The EB version has six interrupt inputs and the EC version has 16. This is an expansion from the two interrupt inputs available on the 8086/8088 microprocessors. In many systems, the five interrupt inputs are adequate.

Timers. The timer section contains three fully programmable l6-bit timers. Timers 0 and 1 generate waveforms for external use and are driven by either the master clock of the 80186/ 80188 or by an external clock. They are also used to count external events. The third timer, timer 2, is internal and clocked by the master clock. The output of timer 2 generates an interrupt after a specified number of clocks and can provide a clock to the other timers. Timer 2 can also be used as a watchdog timer because it can be programmed to interrupt the microprocessor after a certain length of time. The 80C186EC and 80C188EC models have an additional timer called a watchdog. The watchdog timer is a 32-bit counter that is clocked internally by the CLKOUT signal (one half the crystal frequencies). Each time the counter hits zero, it reloads and generates a pulse on the WDTOUT pin that is four CLKOUT periods wide. This output can be used for

any purpose: It can be wired to the reset input to cause a reset or to the NMI input to cause an interrupt. Note that if it is connected to the reset or NMI inputs, it is periodically reprogrammed so that it never counts down to zero. The purpose of a watchdog timer is to reset or interrupt the system if the software goes awry.

Programmable DMA Unit. The programmable DMA unit contains two DMA channels or four DMA channels in the 80C186EC/80C188EC models. Each channel can transfer data between memory locations, between memory and I/O, or between I/O devices. This DMA controller is similar to the 8237 DMA controller discussed in Chapter 13. The main difference is that the 8237 DMA controller has four DMA channels, as does the EC model.

Programmable Chip Selection Unit. The chip selection is a built-in programmable memory and I/O decoder. It has six output lines to select memory, seven lines to select I/O on the XL and EA models, and 10 lines that select either memory or I/O on the EB and EC models. On the XL and EA models, the memory selection lines are divided into three groups that select memory for the major sections of the 80186/80188 memory map. The lower memory select signal enables memory for the interrupt vectors, the upper memory select signal enables memory for reset, and the middle memory select signals enable up to four middle memory devices. The boundary of the lower memory begins at location 00000H and the boundary of the upper memory ends at location FFFFFH. The sizes of the memory areas are programmable, and wait states (0–3 waits) can be automatically inserted with the selection of an area of memory. On the XL and EA models, each programmable I/O selection signal addresses a 128-byte block of I/O space. The programmable I/O area starts at a base I/O address programmed by the user, and all seven 128-byte blocks are contiguous. On the EB and EC models, there is an upper and lower memory chip selection pin and eight general-purpose memory or I/O chip selection pins. Another difference is that from 0 to 15 wait states can be programmed in these two versions of the 80186/80188 embedded controllers.

Power Save/Power down Feature. The power save feature allows the system clock to be divided by 4, 8, or 16 to reduce power consumption. The power-saving feature is started by software and exited by a hardware event such as an interrupt. The power down feature stops the clock completely, but it is not available on the XL version. The power down mode is entered by execution of an HLT instruction and is exited by any interrupt.

Refresh Control Unit. The refresh control unit generates the refresh row address at the interval programmed. The refresh control unit does not multiplex the address for the DRAM—this is still the responsibility of the system designer. The refresh address is provided to the memory system at the end of the programmed refresh interval, along with the control signal. The memory system

must run a refresh cycle during the active time of the control signal. More on memory and refreshing is provided in the section that explains the chip selection unit.

Pin-Out Figure 16–2 illustrates the pin-out of the 80C186XL microprocessor. Note that the 80C186XL is packaged in either a 68-pin leadless chip carrier (LCC) or in a pin grid array (PGA). The LCC package and PGA packages are illustrated in Figure 16–3.

Pin Definitions. The following list defines each 80C186XL pin and notes any differences between the 80C186XL and 80C188XL microprocessors. The enhanced versions are described later in this chapter.

VCC This is the system power supply connection for ±10%, +5.0 V.

VSS This is the system ground connection.

X1 and X2 The clock pins are generally connected to a fundamental-mode parallel resonant crystal that operates an internal crystal oscillator. An external clock signal may be connected to the X1 pin. The internal master clock operates at one half the external crystal or clock input signal. Note that these pins are labeled CLKIN (Xl) and OSCOUT (X2) on some versions of the 80186/80188.

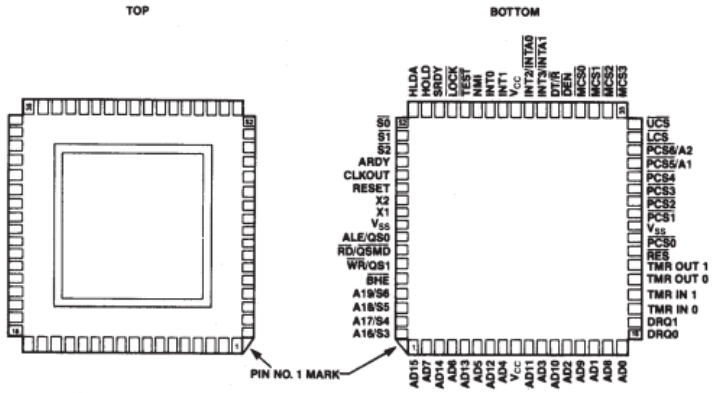


FIGURE 16–2 Pin-out of the 80186 microprocessor. (Courtesy of Intel Corporation.)

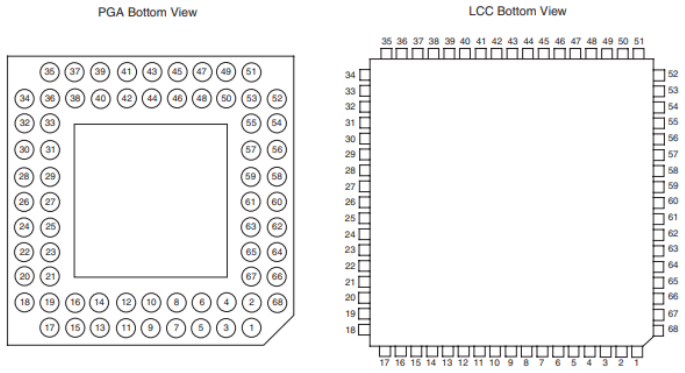


FIGURE 16–3 The bottom views of the PGA and LCC style versions of the 80C188XL microprocessor.

FIGURE 16–3 the bottom views of the PGA and LCC style versions of the 80C188XL microprocessor.

CLKOUT	Clock out provides a **timing signal** to system peripherals at one half the clock input frequency with a 50% duty cycle.
$\overline{\text{RES}}$	The **reset input** pin resets the 80186/80188. For a proper reset, the $\overline{\text{RES}}$ must be held low for at least 50 ms after power is applied. This pin is often connected to an RC circuit that generates a reset signal after power is applied. The reset location is identical to that of the 8086/8088 microprocessor—FFFF0H.
RESET	The companion **reset output** pin (goes high for a reset) connects to system peripherals to initialize them whenever the $\overline{\text{RES}}$ input goes low.
$\overline{\text{TEST}}$	This test pin connects to the BUSY output of the 80187 numeric coprocessor. The $\overline{\text{TEST}}$ pin is interrogated with the FWAIT or WAIT instruction.
$T_{in}0$ and $T_{in}1$	These pins are used as **external clocking sources** to timers 0 and 1.
$T_{out}0$ and $T_{out}1$	These pins provide the **output signals** from timers 0 and 1, which can be programmed to provide square waves or pulses.
DRQ0 and DRQ1	These pins are active-high-level triggered **DMA request** lines for DMA channels 0 and 1.
NMI	This is a **non-maskable interrupt** input. It is positive edge-triggered and always active. When NMI is activated, it uses interrupt vector 2.
INT_0, INT_1, $INT_2/\overline{INTA0}$, and $INT_3/\overline{INTA1}$	These are **maskable interrupt inputs.** They are active-high and are programmed as either level or edge-triggered. These pins are configureed as four interrupt inputs if no external 8259 is present, or as two interrupt inputs if the 8259A interrupt controller is present.

A_{19}/$\overline{ONCE}$, A_{18}, A_{17}, and A_{16}	These are **multiplexed address/status** connections that provide the address (A_{19}–A_{16}) and status (S_6–S_3). Status bits found on address pins A_{18}–A_{16} have no system function and are used during manufacturing for testing. The A_{19} pin is an input for the $\overline{ONCE}$ function on a reset. If $\overline{ONCE}$ is held low on a reset, the microprocessor enters a testing mode.
AD_{15}–AD_0	These are **multiplexed address/data** bus connections. During T_1, the 80186 places A_{15}–A_0 on these pins; during T_2, T_3, and T_4, the 80186 uses these pins as the data bus for signals D_{15}–D_0. Note that the 80188 has pins AD_7–AD_0 and A_{15}–A_8.
$\overline{BHE}$	The **bus high enable** pin indicates (when a logic 0) that valid data are transferred through data bus connections D_{15}–D_8.
ALE	**Address latch enable** is an output pin that contains ALE one-half clock cycle earlier than in the 8086. It is used to demultiplex the address/data and address/status buses. (Even though the status bits on A_{19}–A_{16} are not used in the system, they must still be demultiplexed.)
$\overline{WR}$	The **write** pin causes data to be written to memory or I/O.
$\overline{RD}$	The **read** pin causes data to be read from memory or I/O.
ARDY	The **asynchronous READY** input informs the 80186/80188 that the memory or I/O is ready for the 80186/80188 to read or write data. If this pin is tied to +5.0 V, the microprocessor functions normally; if it is grounded, the microprocessor enters wait states.
SRDY	The **synchronous READY** input is synchronized with the system clock to provide a relaxed timing for the ready input. Like ARDY, SRDY is tied to +5.0 V for no wait states.
$\overline{LOCK}$	The **lock** pin is an output controlled by the LOCK prefix. If an instruction is prefixed with LOCK, the $\overline{LOCK}$ pin becomes a logic 0 for the duration of the locked instruction.
S_2, S_1, and S_0	These are status bits that provide the system with the type of bus transfer in effect. See Table 16–2 for the states of the status bits. The **upper-memory chip select** pin selects memory on the upper portion of the memory map.
$\overline{UCS}$	The **upper-memory chip select** output is programmable to enable memory sizes of 1K to 256K bytes ending at location FFFFFH. Note that this pin is programmed differently on the EB and EC versions and enables memory between 1K and 1M long.

TABLE 16–2 The S_2, S_1, and S_0 status bits.

S_2	S_1	S_0	*Function*
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

$\overline{\text{LCS}}$	The **lower-memory chip select** pin enables memory beginning at location 00000H. This pin is programmed to select memory sizes from 1K to 256K bytes. Note that this pin functions differently for the EB and EC versions and enables memory between 1K and 1M bytes long.
$\overline{\text{MCS0}}$–$\overline{\text{MCS3}}$	The **middle-memory chip select** pins enable four middle memory devices. These pins are programmable to select an 8K to 512K byte block of memory, containing four devices. Note that these pins are not present on the EB and EC versions.
$\overline{\text{PCS0}}$–$\overline{\text{PCS4}}$	These are five different **peripheral selection** lines. Note that the lines are not present on the EB and EC versions.
$\overline{\text{PCS5}}/A_1$ and $\overline{\text{PCS6}}/A_2$	These are programmed as peripheral selection lines or as internally latched address bits A_2 and A_1. These lines are not present on the EB and EC versions.
DT/$\overline{\text{R}}$	The **data transmit/receive** pin controls the direction of data bus buffers if attached to the system.
$\overline{\text{DEN}}$	The **data bus enable** pin enables the external data bus buffers.

DC Operating Characteristics:

It is necessary to know the DC operating characteristics before attempting to interface or operate the microprocessor. The 80C186/801C88 microprocessors require between 42 mA and 63 mA of power-supply current. Each output pin provides 3.0 mA of logic 0 current and -2 mA of logic 1 current.

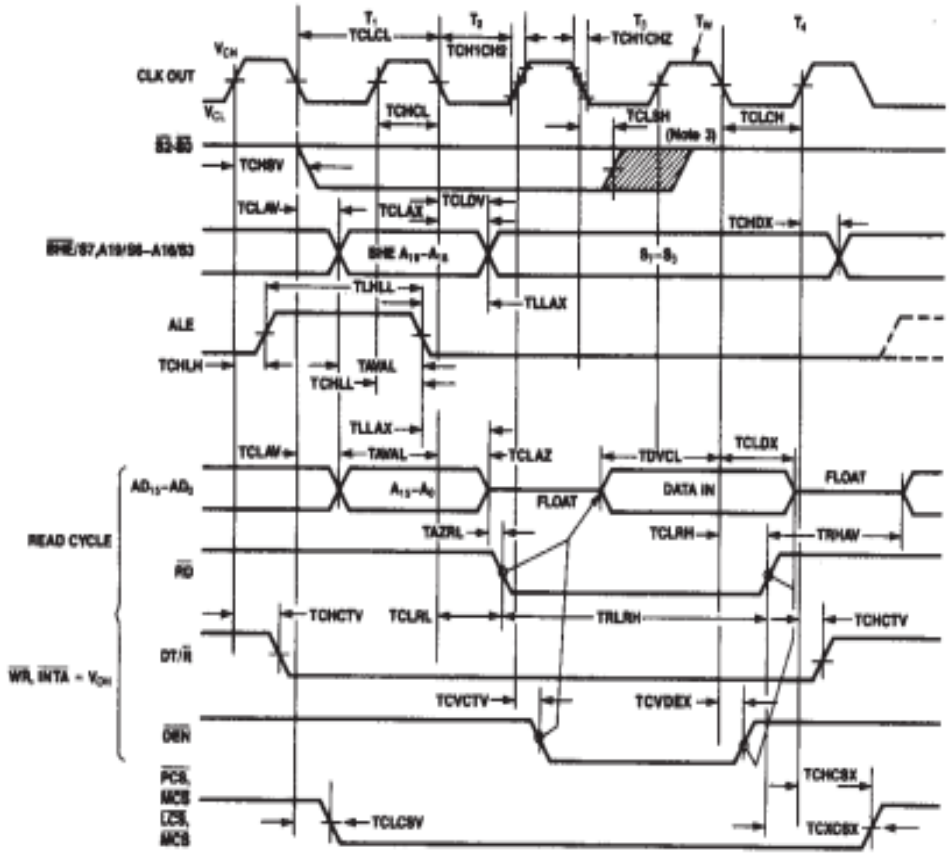
80186/80188 Timing:

The timing diagram for the 80186 is provided in Figure 16–4. Timing for the 80188 is identical except for the multiplexed address connection, which are AD7–AD0 instead of AD15–AD0, and the, which does not exist on the 80188.

The basic timing for the 80186/80188 is composed of four clocking periods just as in the 8086/8088. A bus cycle for the 8 MHz version requires 500 ns, while the 16 MHz version requires 250 ns.

There are very few differences between the timing for the 80186/80188 and the 8086/8088. The most noticeable difference is that ALE appears one-half clock cycle earlier in the 80186/ 80188.

Memory Access Time. One of the more important points in any microprocessor's timing diagram is the memory access time. Access time calculations for the 80186/80188 are identical to that of the 8086/8088. Recall that the access time is the time allotted to the memory and I/O to provide data to the microprocessor after the microprocessor sends the memory or I/O its address. A close examination of the timing diagram reveals that the address appears on the address bus TCLAV time after the start of T1. TCLAV is listed as 44 ns for the 8 MHz version. (See Figure 16–5.) Data are sampled from the data bus at the end of T3, but a setup time is required before the clock defined as TDVCL. The times listed for TDVCL are 20 ns for both versions of the microprocessor. Access time is therefore equal to three clocking periods minus both TCLAV and TDVCL. Access time for the 8 MHz microprocessor is 375 ns - 44 ns - 20 ns, or 311 ns. The access time for the 16 MHz version is calculated in the same manner, except that TCLAV is 25 ns and TDVCL is 15 ns.



(a)

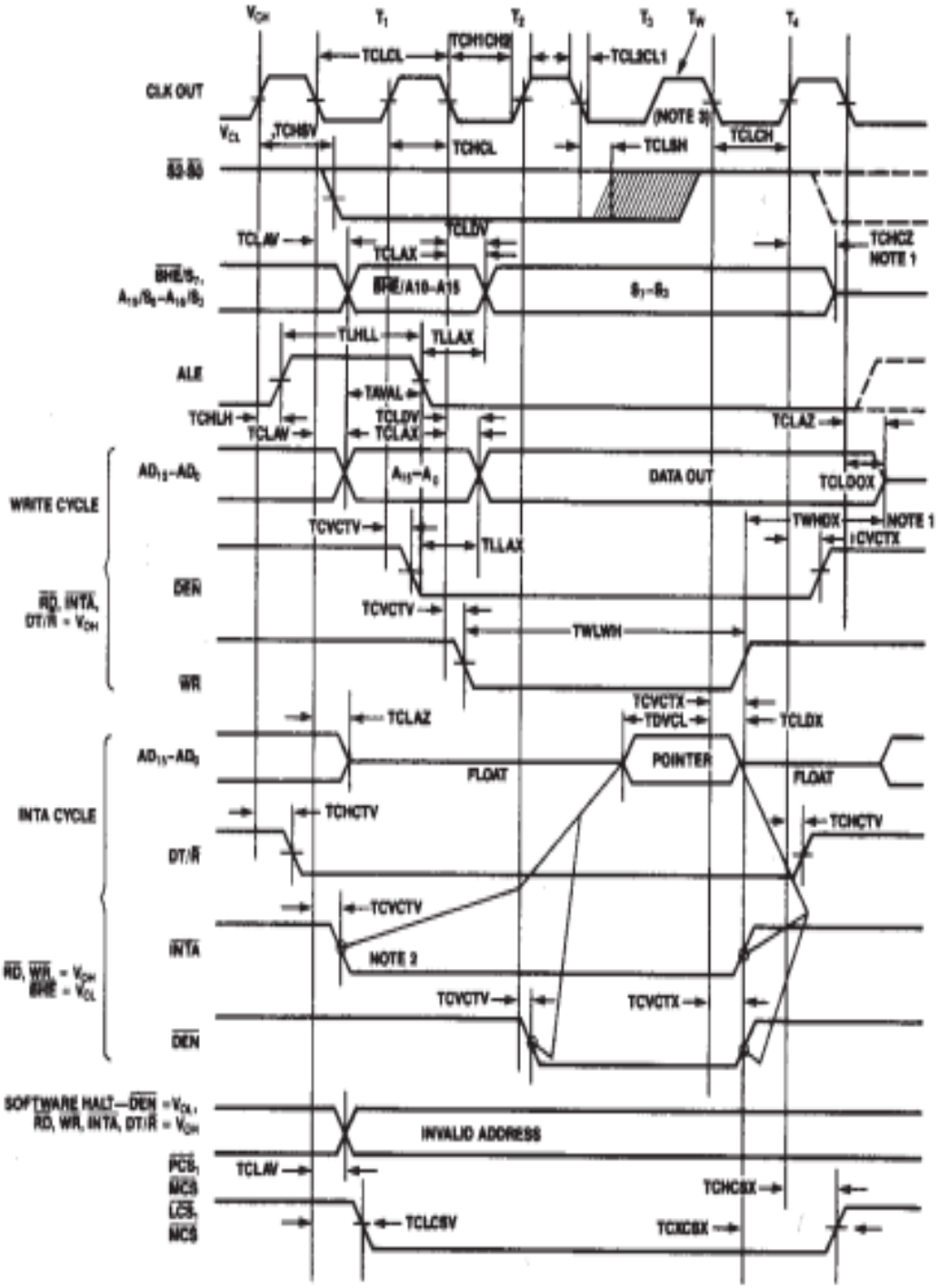


FIGURE 16–4 80186/80188 timing. (a) Read cycle timing and (b) write cycle timing. (Courtesy of Intel Corporation.)

80186 Master Interface Timing Responses

		80188 (8 MHz)		80188-6 (6 MHz)			
Symbol	**Parameters**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CLAV}	Address Valid Delay	5	44	5	63	ns	C_L = 20-200 pF all outputs
T_{CLAX}	Address Hold	10		10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	44	ns	
T_{CHCZ}	Command Lines Float Delay		45		56	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55		76	ns	
T_{LHLL}	ALE Width	$T_{CLCL-35}$		$T_{CLCL-35}$		ns	
T_{CHLH}	ALE Active Delay		35		44	ns	
T_{CHLL}	ALE Inactive Delay		35		44	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL-25}$		$T_{CHCL-30}$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	55	ns	
T_{CLDOX}	Data Hold Time	10		10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL-40}$		$T_{CLCL-50}$		ns	
T_{CVCTV}	Control Active Delay 1	5	70	5	87	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	76	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	76	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)		70		87	ns	
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	87	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	76	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL-40}$		$T_{CLCL-50}$		ns	
T_{CLHAV}	HLDA Valid Delay	10	50	10	67	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL-50}$		$2T_{CLCL-50}$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL-40}$		$2T_{CLCL-40}$		ns	
T_{AVAL}	Address Valid to ALE Low	$T_{CLCH-25}$		$T_{CLCH-45}$		ns	
T_{CHSV}	Status Active Delay	10	55	10	76	ns	
T_{CLSH}	Status Inactive Delay	10	55	10	76	ns	
T_{CLTMV}	Timer Output Delay		60		75	ns	100 pF max
T_{CLRO}	Reset Delay		60		75	ns	
T_{CHQSV}	Queue Status Delay		35		44	ns	

80186 Chip-Select Timing Responses

Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CLCSV}	Chip-Select Active Delay		66		80	ns	
T_{CKCSX}	Chip-Select Hold from Command Inactive	35		35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	47	ns	

Symbol	**Parameter**	**Min.**	**Max.**	**Units**	**Test Conditions**
TDVCL	Data in Setup (A/D)	20		ns	
TCLDX	Data in Hold (A/D)	10		ns	
TARYHCH	Asynchronous Ready (AREADY) active setup time*	20		ns	
TARYLCL	AREADY inactive setup time	35		ns	
TCHARYX	AREADY hold time	15		ns	
TSRYCL	Synchronous Ready (SREADY) transition setup time	35		ns	
TCLSRY	SREADY transition hold time	15		ns	
THVCL	HOLD Setup*	25		ns	
TINVCH	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		ns	
TINVCL	DRQ0, DRQ1, Setup*	25		ns	

*To guarantee recognition at next clock.

FIGURE 16–5 80186 AC characteristics. (Courtesy of Intel Corporation.)

INTRODUCTION TO THE 80286

The 80286 microprocessor is an advanced version of the 8086 microprocessor that was designed for multiuser and multitasking environments. The 80286 addresses 16M bytes of physical memory and 1G bytes of virtual memory by using its memory-management system. This section of the text introduces the 80286 microprocessor, which finds use in earlier AT-style personal computers that once pervaded the computer market and still find some applications. The 80286 is basically an 8086 that is optimized to execute instructions in fewer clocking periods than the

8086. The 80286 is also an enhanced version of the 8086 because it contains a memory manager. At this time, the 80286 no longer has a place in the personal computer system, but it does find applications in control systems as an embedded controller.

Hardware Features

Figure 16–29 shows the internal block diagram of the 80286 microprocessor. Note that like the 80186/80188, the 80286 does not incorporate internal peripherals; instead, it contains a memory management unit (MMU) that is called the address unit in the block diagram.

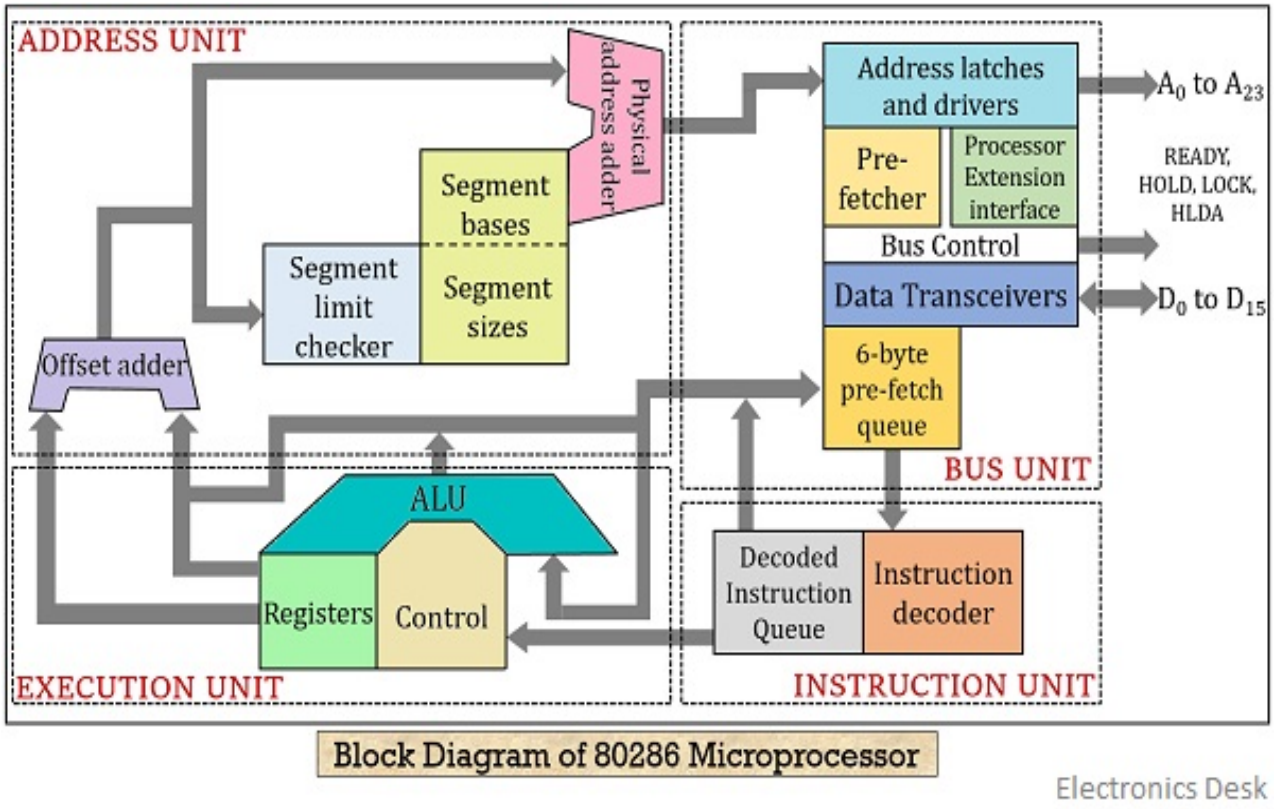


FIGURE 16–29 The block diagram of the 80286 microprocessor. (Courtesy of Intel Corporation.)

As a careful examination of the block diagram reveals, address pins A_{23}–A_0, $\overline{BUSY}$, CAP, $\overline{ERROR}$, $\overline{PEREQ}$, and $\overline{PEACK}$ are new or additional pins that do not appear on the 8086 microprocessor. The $\overline{BUSY}$, $\overline{ERROR}$, $\overline{PEREQ}$, and $\overline{PEACK}$ signals are used with the microprocessor extension or coprocessor, of which the 80287 is an example. (Note that the $\overline{TEST}$ pin is now referred to as the $\overline{BUSY}$ pin.) The address bus is now 24 bits wide to accommodate the 16M bytes of physical memory. The CAP pin is connected to a 0.047 μF, ±20% capacitor that acts as a 12 V filter and connects to ground. The pin-outs of the 8086 and 80286 are illustrated in Figure 16–30 for comparative purposes. Note that the 80286 does not contain a multiplexed address/data bus.

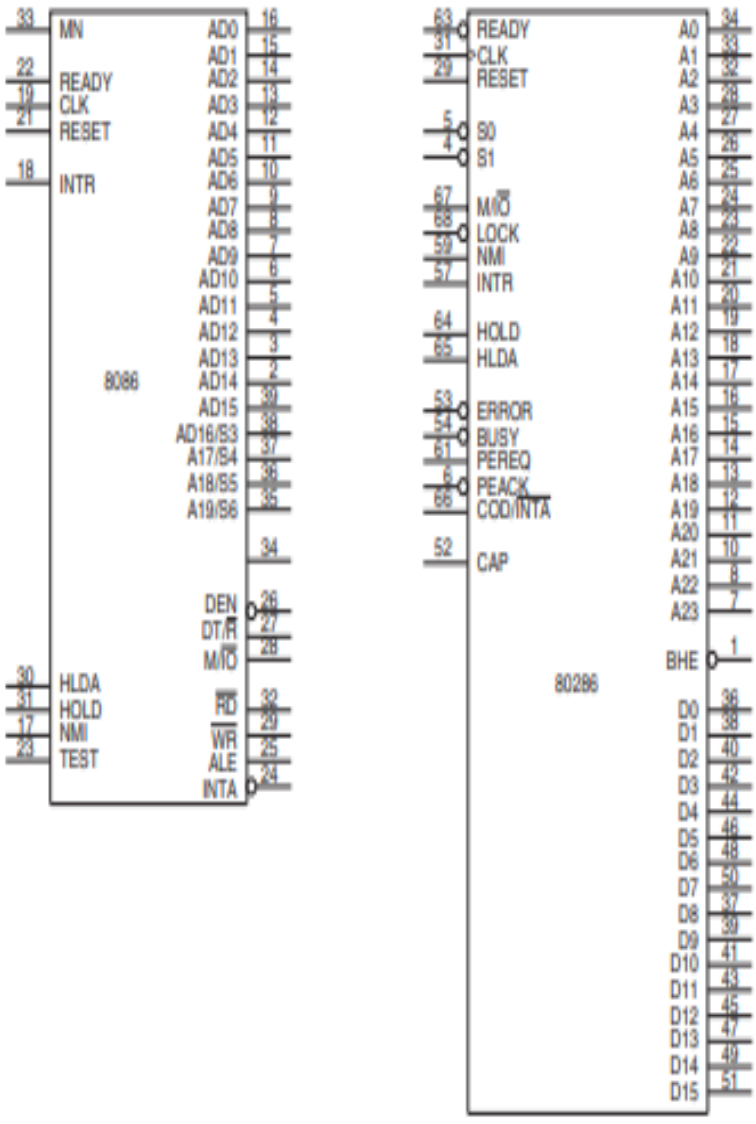


FIGURE 16–30 The 8086 and 80286 microprocessor pin-outs. Notice that the 80286 does not have a multiplexed address/data bus.

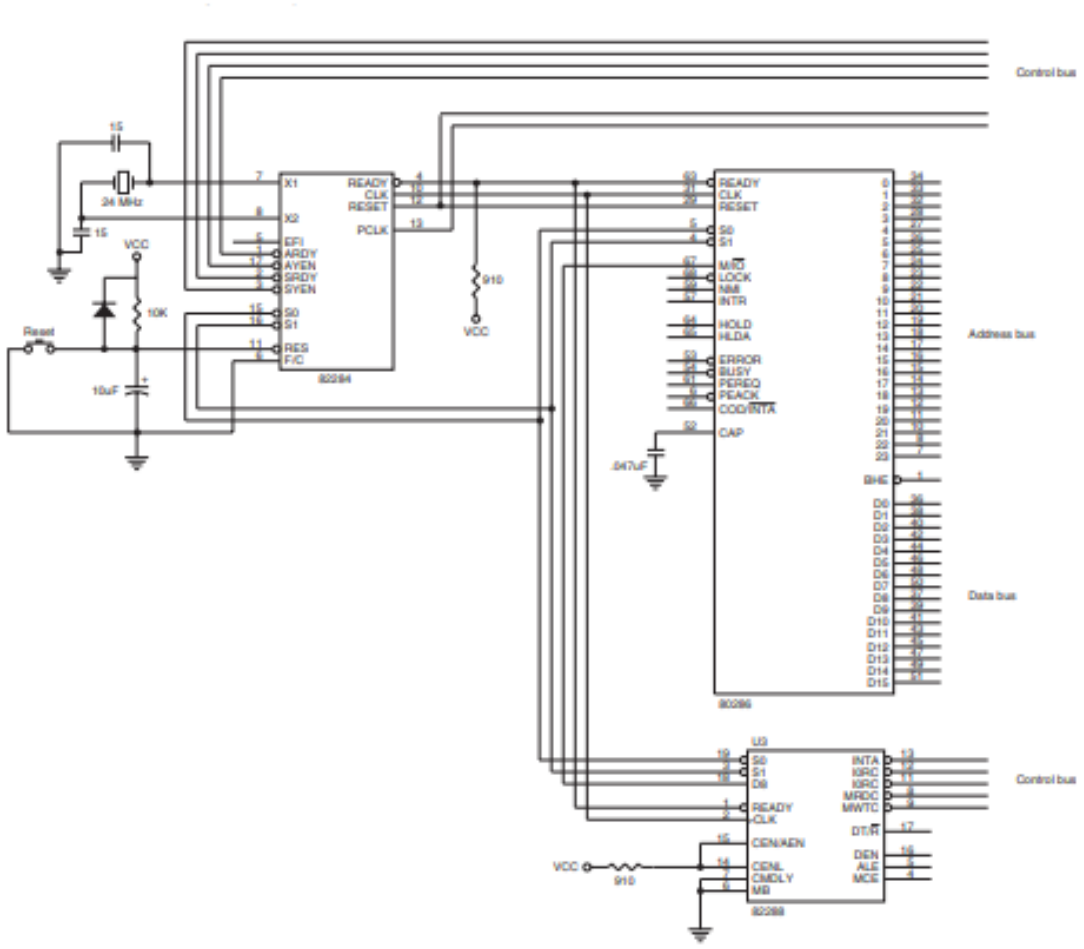


FIGURE 16–31 The interconnection of the 80286 microprocessor, 82284 clock generator, and 8288 system bus controller.

As mentioned in Chapter 1, the 80286 operates in both the real and protected modes. In the real mode, the 80286 addresses a 1M-byte memory address space and is virtually identical to the 8086. In the protected mode, the 80286 addresses a 16M-byte memory space. Figure 16–31 illustrates the basic 80286 microprocessor-based system. Notice that the clock is provided by the 82284 clock generator (similar to the 8284A) and the system control signals are provided by the 82288 system bus controller (similar to the 8288). Also, note the absence of the latch circuits used to de multiplex the 8086 address/data bus.

Additional Instructions the 80286 has even more instructions than its predecessors. These extra instructions control the virtual memory system through the memory manager of the 80286. Table 16–9 lists the additional 80286 instructions with a comment about the purpose of each instruction. These instructions are the only new instructions added to the 80286. Note that the 80286 contains the new instructions added to the 80186/80188 such as INS, OUTS, BOUND, ENTER, LEAVE, PUSHA, POPA, and the immediate multiplication and immediate shift and rotate counts.

TABLE 16–9 Additional 80286 instructions.

Instruction	*Purpose*
CLTS	Clear the task-switched bit
LDGT	Load global descriptor table register
SGDT	Store global descriptor table register
LIDT	Load interrupt descriptor table register
SIDT	Store interrupt descriptor table register
LLDT	Load local descriptor table register
SLDT	Store local descriptor table register
LMSW	Load machine status register
SMSW	Store machine status register
LAR	Load access rights
LSL	Load segment limit
SAR	Store access rights
ARPL	Adjust requested privilege level
VERR	Verify a read access
VERW	Verify a write access

Following are descriptions of instructions not explained in the memory-management section. The instructions described here are special and only used for the conditions indicated.

CLTS The **clear task-switched flag** (CLTS) instruction clears the TS (task-switched) flag bit to a logic 0. If the TS flag bit is a logic 1 and the 80287 numeric coprocessor is used by the task, an interrupt occurs (vector type 7). This allows the function of the coprocessor to be emulated with software. The CLTS instruction is used in a system and is considered a privileged instruction because it can be executed only in the protected mode at privilege level 0. There is no set TS flag instruction; this is accomplished by writing a logic 1 to bit position 3 (TS) of the machine status word (MSW) by using the LMSW instruction.

LAR The **load access rights** (LAR) instruction reads the segment descriptor and places a copy of the access rights byte into a 16-bit register. An example is the LAR AX,BX instruction that loads AX with the access rights byte from the descriptor selected by the selector value found in BX. This instruction is used to get the access rights so that it can be checked before a program uses the segment of memory described by the descriptor.

LSL The **load segment limit** (LSL) instruction loads a user-specified register with the segment limit. For example, the LSL AX,BX instruction loads AX with the limit of the segment described by the descriptor selected by the selector in BX. This instruction is used to test the limit of a segment.

ARPL The **adjust requested privilege level** (ARPL) instruction is used to test a selector so that the privilege level of the requested selector is not violated. An example is ARPL AX,CX: AX contains the requested privilege level and CX contains the selector value to be used to access a descriptor. If the requested privilege level is of a lower priority than the descriptor under test, the zero flag is set. This may require that a program adjust the requested privilege level or indicate a privilege violation.

VERR The **verify for read access** (VERR) instruction verifies that a segment can be read. Recall from Chapter 1 that a code segment can be read-protected. If the code segment can be read, the zero flag bit is set. The VERR AX instruction tests the descriptor selected by the AX register.

VERW The **verify for write access** (VERW) instruction is used to verify that a segment can be written. Recall from Chapter 1 that a data segment can be write-protected. If the data segment can be written, the zero flag bit is set.

The Virtual Memory Machine

A virtual memory machine is a machine that maps a larger memory space (1G bytes for the 80286) into a much smaller physical memory space (l6M bytes for the 80286), which allows a very large system to execute in smaller physical memory systems. This is accomplished by spooling the data and programs between the fixed disk memory system and the physical memory. Addressing a 1G-byte memory system is accomplished by the descriptors in the 80286 microprocessor. Each 80286 descriptor describes a 64K-byte memory segment and the 80286 allows 16K descriptors. This (64K × 16K) allows a maximum of 1G bytes of memory to be described for the system.

As mentioned in Chapter 1, descriptors describe the memory segment in the protected mode. The 80286 has descriptors that define codes, data, stack segments, interrupts, procedures, and tasks. Descriptor accesses are performed by loading a segment register with a selector in the protected mode. The selector accesses a descriptor that describes an area of the memory. Additional details on descriptors and their applications are defined in Chapter 1, and also Chapters 17, 18, and 19. Please refer to these chapters for a detailed view of the protected mode memory-management system.

INTRODUCTION TO THE 80386 MICROPROCESSOR

Before the 80386 or any other microprocessor can be used in a system, the function of each pin must be understood. This section of the chapter details the operation of each pin, along with the external memory system and I/O structures of the 80386. Figure 17–1 illustrates the pin-out of the 80386DX microprocessor. The 80386DX is packaged in a 132-pin PGA (pin grid array). Two versions of the 80386 are commonly available: the 80386DX, which is illustrated and described in this chapter and the 80386SX, which is a reduced bus version of the 80386. A new version of the 80386—the 80386EX—incorporates the AT bus system, dynamic RAM controller, programmable chip selection logic, 26 address pins, 16 data pins, and 24 I/O pins. Figure 17–2 illustrates the 80386EX embedded PC. The 80386DX addresses 4G bytes of memory through its 32-bit data bus and 32-bit address. The 80386SX, more like the 80286, addresses 16M bytes of memory with its 24-bit address bus via its 16-bit data bus. The 80386SX was developed after the 80386DX for applications that didn't require the full 32-bit bus version. The 80386SX was found in many early personal computers that used the same basic motherboard design as the 80286. At the time that the 80386SX was popular, most applications, including Windows 3.11, required fewer than 16M bytes of memory, so the 80386SX is a popular and a less costly version of the 80386 microprocessor. Even though the 80486 has become a less expensive upgrade path for newer systems, the 80386 still can be used for many applications. For example, the 80386EX does not appear in computer systems, but it is becoming very popular in embedded applications.

FIGURE 17–1 The pin-outs of the 80386DX and 80386SX microprocessors.

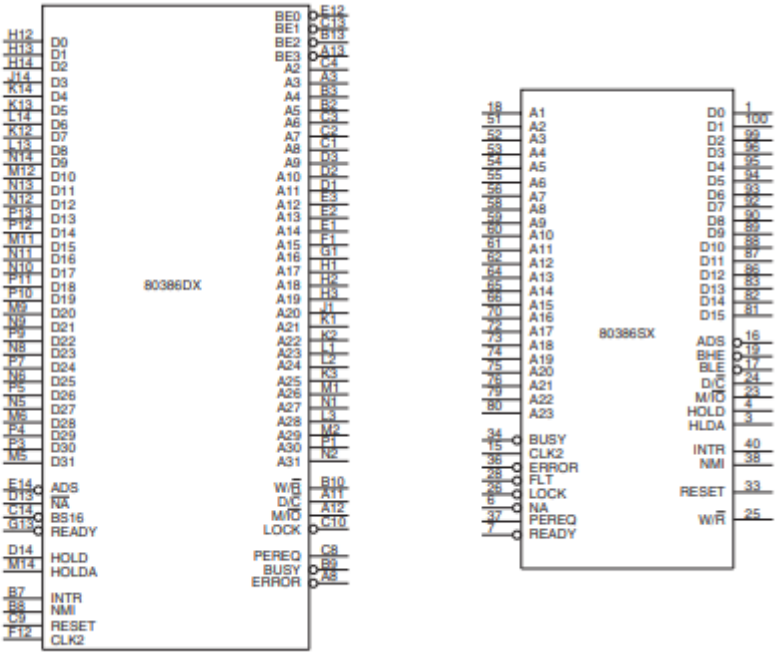
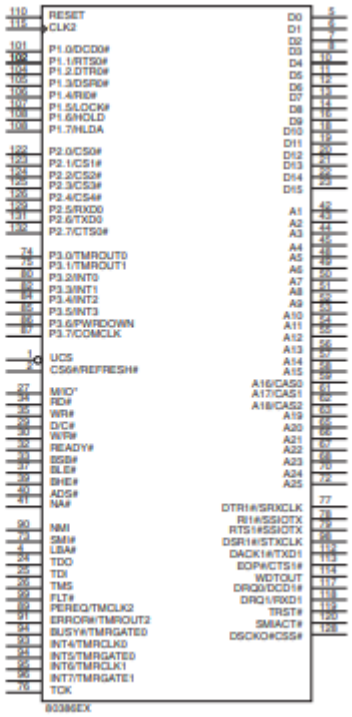


FIGURE 17–2 The 80386EX embedded PC.



As with earlier versions of the Intel family of microprocessors, the 80386 requires a single +5.0 V power supply for operation. The power supply current averages 550 mA for the 25 MHz version of the 80386, 500 mA for the 20 MHz version, and 450 mA for the 16 MHz version. Also available is a 33 MHz version that requires 600 mA of power supply current. The power supply current for the 80386EX is 320 mA when operated at 33 MHz. Note that during some modes of normal operation, power supply current can surge to over 1.0 A. This means that the power supply and power distribution network must be capable of supplying these current surges. This device contains multiple VCC and VSS connections that must all be connected to +5.0 V and grounded for proper operation. Some of the pins are labeled N/C (no connection) and must

not be connected. Additional versions of the 80386SX and 80386EX are available with a +3.3 V power supply. They are often found in portable notebook or laptop computers and are usually packaged in a surface mount device. Each 80386 output pin is capable of providing 4.0 mA (address and data connections) or 5.0 mA (other connections). This represents an increase in drive current compared to the 2.0 mA available on earlier 8086, 8088, and 80286 output pins. The output current available on most 80386EX output pins is 8.0 mA. Each input pin represents a small load, requiring only ±10 μA of current. In some systems, except the smallest, these current levels require bus buffers. The function of each 80386DX group of pins follows:

A_{31}–A_2	**Address bus** connections address any of the 1G × 32 (4G bytes) memory locations found in the 80386 memory system. Note that A_0 and A_1 are encoded in the bus enable ($\overline{BE3}$–$\overline{BE0}$) to select any or all of the four bytes in a

32-bit-wide memory location. Also note that because the 80386SX contains a 16-bit data bus in place of the 32-bit data bus found on the 80386DX, A_1 is present on the 80386SX, and the bank selection signals are replaced with $\overline{BHE}$ and $\overline{BLE}$. The $\overline{BHE}$ signal enables the upper data bus half; the $\overline{BLE}$ signal enables the lower data bus half.

D_{31}–D_0 **Data bus** connections transfer data between the microprocessor and its memory and I/O system. Note that the 80386SX contains D_{15}–D_0.

$\overline{BE3}$–$\overline{BE0}$ **Bank enable** signals select the access of a byte, word, or doubleword of data. These signals are generated internally by the microprocessor from address bits A_1 and A_0. On the 80386SX, these pins are replaced by $\overline{BHE}$, $\overline{BLE}$, and A_1.

M/$\overline{IO}$ **Memory/IO** selects a memory device when a logic 1 or an I/O device when a logic 0. During the I/O operation, the address bus contains a 16-bit I/O address on address connections A_{15}–A_2.

W/$\overline{R}$ **Write**/**$\overline{Read}$** indicates that the current bus cycle is a write when a logic 1 or a read when a logic 0.

$\overline{ADS}$ The **address data strobe** becomes active whenever the 80386 has issued a valid memory or I/O address. This signal is combined with the W/$\overline{R}$ signal to generate the separate read and write signals present in the earlier 8086–80286 microprocessor-based systems.

RESET **Reset** initializes the 80386, causing it to begin executing software at memory location FFFFFFF0H. The 80386 is reset to the real mode, and the leftmost 12 address connections remain logic 1s (FFFH) until a far jump or far call is executed. This allows compatibility with earlier microprocessors.

CLK_2 **Clock times 2** is driven by a clock signal that is twice the operating frequency of the 80386. For example, to operate the 80386 at 16 MHz, apply a 32 MHz clock to this pin.

$\overline{READY}$ **Ready** controls the number of wait states inserted into the timing to lengthen memory accesses.

$\overline{LOCK}$ **Lock** becomes a logic 0 whenever an instruction is prefixed with the LOCK: prefix. This is used most often during DMA accesses.

D/$\overline{C}$	**Data/control** indicates that the data bus contains data for or from memory or I/O when a logic 1. If D/$\overline{C}$ is a logic 0, the microprocessor is halted or executes an interrupt acknowledge.
$\overline{BS16}$	**Bus size 16** selects either a 32-bit data bus ($\overline{BS16}$ = 1) or a 16-bit data bus ($\overline{BS16}$ = 0). In most cases, if an 80386DX is operated on a 16-bit data bus, we use the 80386SX that has a 16-bit data bus. On the 80386EX, the $\overline{BS8}$ pin selects an 8-bit data bus.
$\overline{NA}$	**Next address** causes the 80386 to output the address of the next instruction or data in the current bus cycle. This pin is often used for pipelining the address.
HOLD	**Hold** requests a DMA action.
HLDA	**Hold acknowledge** indicates that the 80386 is currently in a hold condition.
$\overline{PEREQ}$	The **coprocessor request** asks the 80386 to relinquish control and is a direct connection to the 80387 arithmetic coprocessor.
$\overline{BUSY}$	**Busy** is an input used by the WAIT or FWAIT instruction that waits for the coprocessor to become not busy. This is also a direct connection to the 80387 from the 80386.
$\overline{ERROR}$	**Error** indicates to the microprocessor that an error is detected by the coprocessor.
INTR	An **interrupt request** is used by external circuitry to request an interrupt.
NMI	A **non-maskable interrupt** requests a non-maskable interrupt as it did on the earlier versions of the microprocessor.

The Memory System

The physical memory system of the 80386DX is 4G bytes in size and is addressed as such. If virtual addressing is used, 64T bytes are mapped into the 4G bytes of physical space by the memory management unit and descriptors. (Note that virtual addressing allows a program to be larger than 4G bytes if a method of swapping with a large hard disk drive exists.) Figure 17–3 shows the organization of the 80386DX physical memory system. The memory is divided into four 8-bit wide memory banks, each containing up to 1G bytes of memory. This 32-bit-wide memory organization allows bytes, words, or double words of memory data to be accessed directly. The 80386DX transfers up to a 32-bit-wide number in a single memory cycle, whereas the early 8088 requires four cycles to accomplish the same transfer, and the 80286 and 80386SX require two cycles. Today, the data width is important, especially with single-precision floating-point numbers that are 32 bits wide. High-level software normally uses floating-point numbers for data storage, so 32-bit memory locations speed the execution of high-level software when it is written to take advantage of this wider memory. Each memory byte is numbered in hexadecimal as they were in prior versions of the family. The difference is that the 80386DX uses a 32-bit-wide memory address, with memory bytes numbered from location 00000000H to FFFFFFFFH. The two memory banks in the 8086, 80286, and 80386SX system are accessed via (A0 on the 8086 and 80286) and . In the 80386DX, the memory banks are accessed via four bank enable signals, . This arrangement allows a single byte to be accessed when one bank enable signal is activated by the microprocessor. It also allows a word to be addressed when two

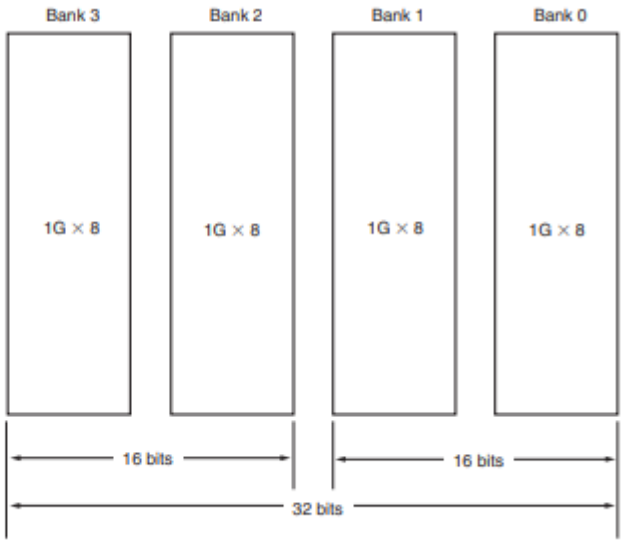


FIGURE 17–3 The memory system for the 80386 microprocessor. Notice that the memory is organized as four banks, each containing 1G byte. Memory is accessed as 8-, 16-, or 32-bit data.

bank enable signals are activated. In most cases, a word is addressed in banks 0 and 1, or in banks 2 and 3. Memory location 00000000H is in bank 0, location 00000001H is in bank 1, location 00000002H is in bank 2, and location 00000003H is in bank 3. The 80386DX does not contain address connections A0 and A1 because these have been encoded as the bank enable signals. Likewise, the 80386SX does not contain the A0 address pin because it is encoded in the and signals. The 80386EX addresses data either in two banks for a 16-bit-wide memory system if = 1 or as an 8-bit system if = 0.

Buffered System. Figure 17–4 shows the 80386DX connected to buffers that increase fan-out from its address, data, and control connections. This microprocessor is operated at 25 MHz using a 50 MHz clock input signal that is generated by an integrated oscillator module. Oscillator modules are usually used to provide a clock in modern microprocessor-based equipment. The HLDA signal is used to enable all buffers in a system that uses direct memory access. Otherwise, the buffer enable pins are connected to ground in a non-DMA system.

Pipelines and Caches. The cache memory is a buffer that allows the 80386 to function more efficiently with lower DRAM speeds. A pipeline is a special way of handling memory accesses so the memory has additional time to access data. A 16 MHz 80386 allows memory devices with access times of 50 ns or less to operate at full speed. Obviously, there are few DRAMs currently available with these access times. In fact, the fastest DRAMs currently in use have an access time of 40 ns or longer. This means that some technique must be found to interface these memory devices, which are slower than required by the microprocessor. Three techniques are available: interleaved memory, caching, and a pipeline.

The pipeline is the preferred means of interfacing memory because the 80386 microprocessor supports pipelined memory accesses. Pipelining allows memory an extra clocking period to access data. The extra clock extends the access time from 50 ns to 81 ns on an 80386 operating with a 16 MHz clock. The pipe, as it is often called, is set up by the microprocessor. When an

instruction is fetched from memory, the microprocessor often has extra time before the next instruction is fetched. During this extra time, the address of the next instruction is sent out from the address bus ahead of time. This extra time (one clock period) is used to allow additional access time to slower memory components.

Not all memory references can take advantage of the pipe, which means that some memory cycles are not pipelined. These non pipelined memory cycles request one wait state if the normal pipeline cycle requires no wait states. Overall, a pipe is a cost-saving feature that reduces the access time required by the memory system in low-speed systems. Not all systems can take advantage of the pipe. Those systems typically operate at 20, 25, or 33 MHz. In these higher-speed systems, another technique must be used to increase the memory system speed. The cache memory system improves overall performance of the memory systems for data that are accessed more than once. Note that the 80486 contains an internal cache called a level 1 cache and the 80386 can only contain an external cache called a level 2 cache.

A cache is a high-speed memory (SRAM) system that is placed between the microprocessor and the DRAM memory system. Cache memory devices are usually static RAM memory components with access times of less than 10 ns. In many cases, we see level 2 cache memory systems with sizes between 32K and 1M byte. The size of the cache memory is determined more by the application than by the microprocessor. If a program is small and refers to little memory data, a small cache is beneficial. If a program is large and references large blocks of memory, the largest cache size possible is recommended. In many cases, a 64K-byte cache improves speed sufficiently, but the maximum benefit is often derived from a 256K-byte cache. It has been found that increasing the cache size much beyond 256K provides little benefit to the operating speed of the system that contains an 80386 microprocessor.

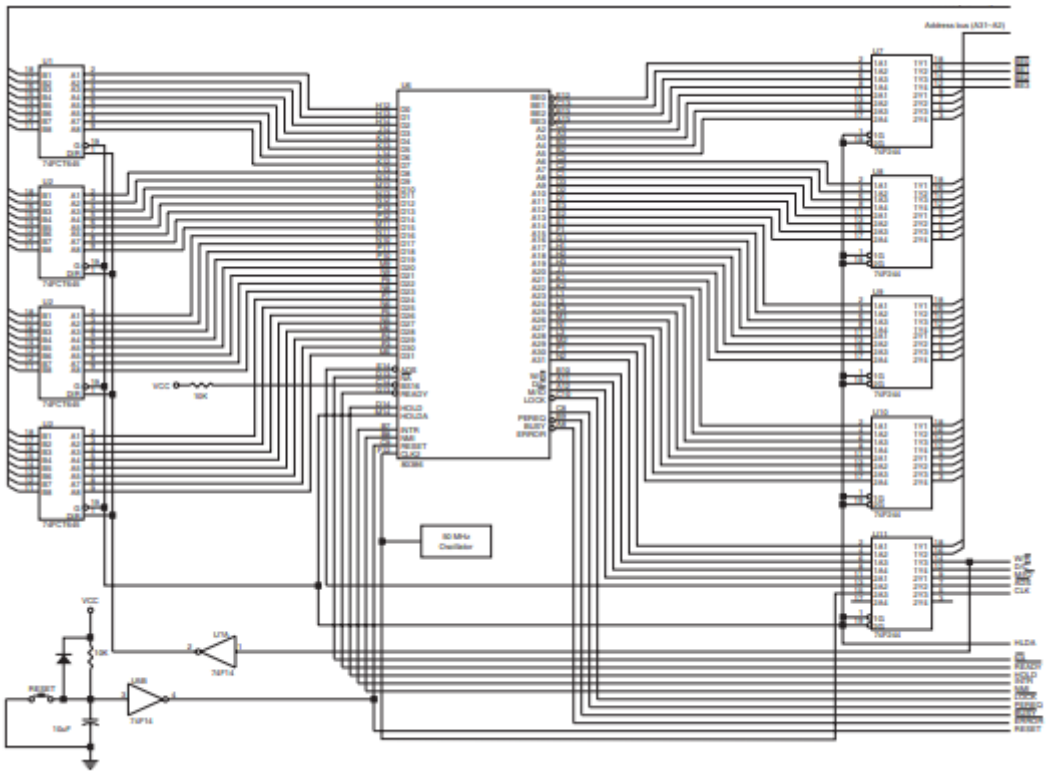


FIGURE 17–4 A fully buffered 25 MHz 80386DX.

Interleaved Memory Systems. An interleaved memory system is another method of improving the speed of a system. Its only disadvantage is that it costs considerably more memory because of its structure. Interleaved memory systems are present in some systems, so memory access times can be lengthened without the need for wait states. In some systems, an interleaved memory may still require wait states, but may reduce their number. An interleaved memory system requires two or more complete sets of address buses and a controller that provides addresses for each bus. Systems that employ two complete buses are called a two-way interleave; systems that use four complete buses are called a four-way interleave.

An interleaved memory is divided into two or four parts. For example, if an interleaved memory system is developed for the 80386SX microprocessor, one section contains the 16-bit addresses 000000H–000001H, 000004H–000005H, and so on; the other section contains addresses 000002–000003, 000006H–000007H, and so forth. While the microprocessor accesses locations 000000H–000001H, the interleave control logic generates the address strobe signal for locations 000002H–000003H. This selects and accesses the word at location 000002H–000003H, while the microprocessor processes the word at location 000000H–000001H. This process alternates memory sections, thus increasing the performance of the memory system

Interleaving increases the amount of access time provided to the memory because the address is generated to select the memory before the microprocessor accesses it. This is because the microprocessor pipelines memory addresses, sending the next address out before the data are read from the last address.

The problem with interleaving, although not major, is that the memory addresses must be accessed so that each section is alternately addressed. This does not always happen as a program executes. Under normal program execution, the microprocessor alternately addresses memory approximately 93% of the time. For the remaining 7%, the microprocessor addresses data in the same memory section, which means that in these 7% of the memory accesses, the memory system must cause wait states because of the reduced access time. The access time is reduced because the memory must wait until the previous data are transferred before it can obtain its address. This leaves the memory with less access time; therefore, a wait state is required for accesses in the same memory bank.

sses in the same memory bank. See Figure 17–5 for the timing diagram of the address as it appears at the microprocessor address pins. This timing diagram shows how the next address is output before the current data are accessed. It also shows how access time is increased by using interleaved memory addresses for each section of memory compared to a non-interleaved access, which requires a wait state. Figure 17–6 pictures the interleave controller. Admittedly, this is a complex logic circuit, which needs some explanation. First, if the SEL input (used to select this section of the memory) is inactive (logic 0), then the signal is a logic 1. Also, both ALE0 and ALE1, used to strobe the address to the memory sections, are both logic 1s, causing the latches connected to them to become transparent. As soon as the SEL input becomes a logic 1, this circuit begins to function. The A1 input is used to determine which latch (U2B or U5A) becomes a logic 0, selecting a section of the memory. Also the ALE pin that becomes a logic 0 is compared with the previous state of the ALE pins. If the same section of memory is accessed a second time, the signal becomes a logic 0, requesting a wait state.

Figure 17–7 illustrates an interleaved memory system that uses the circuit of Figure 17–6. Notice how the ALE0 and ALE1 signals are used to capture the address for either section of memory. The memory in each bank is 16 bits wide. If accesses to memory require 8-bit data, the system causes wait states, in most cases. As a program executes, the 80386SX fetches instructions 16 bits at a time from normally sequential memory locations. Program execution uses interleaving in most cases. If a system is going to access mostly 8-bit data, it is doubtful that memory interleaving will reduce the number of wait states.

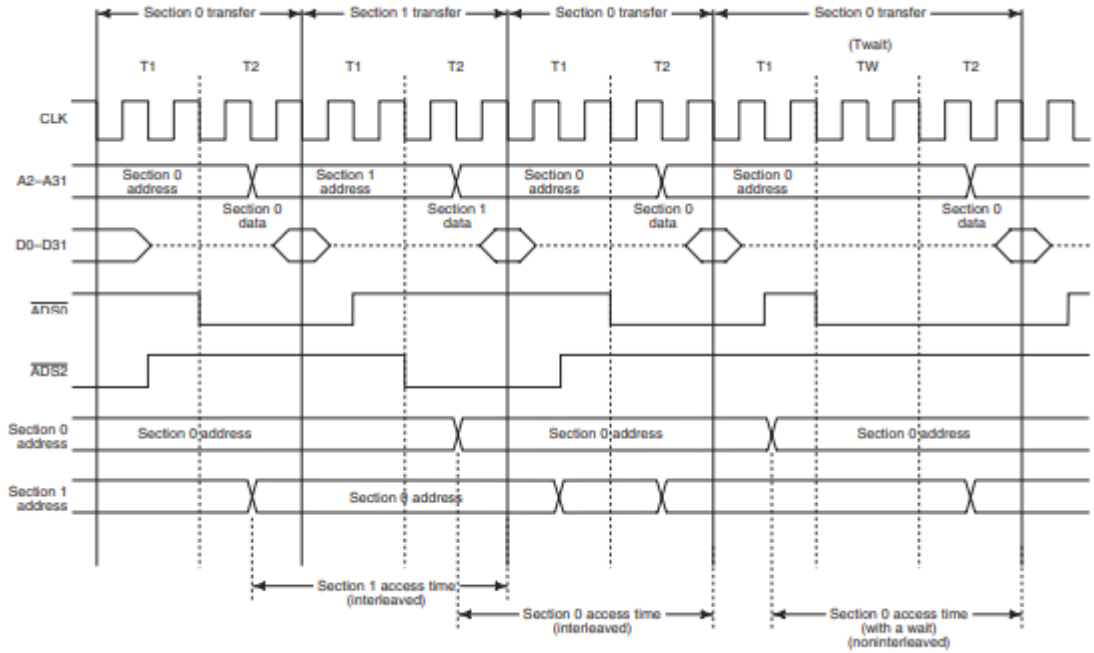


FIGURE 17–5 The timing diagram of an interleaved memory system showing the access times and address signals for both sections of memory.

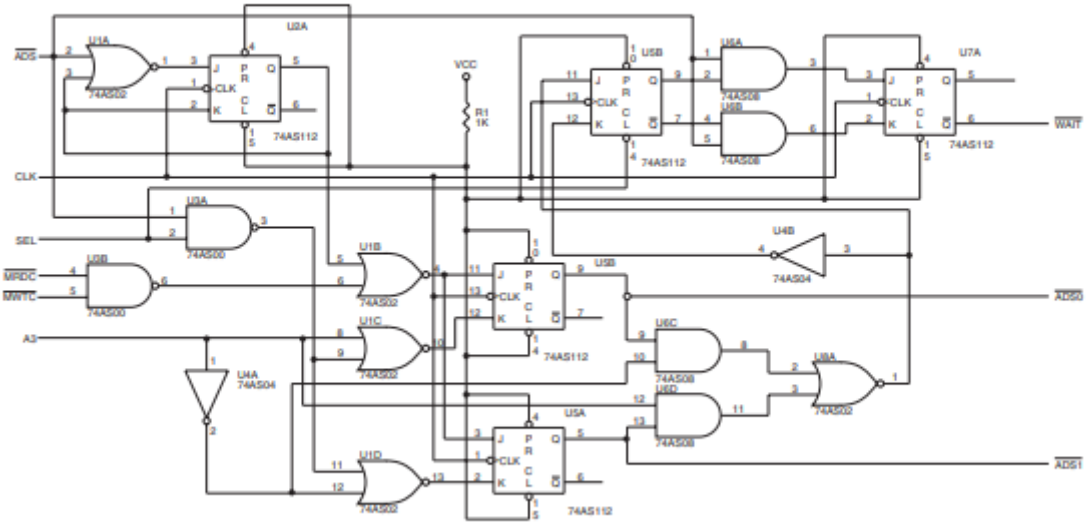


FIGURE 17–6 The interleaved control logic, which generates separate ADS signals and a WAIT signal used to control interleaved memory.

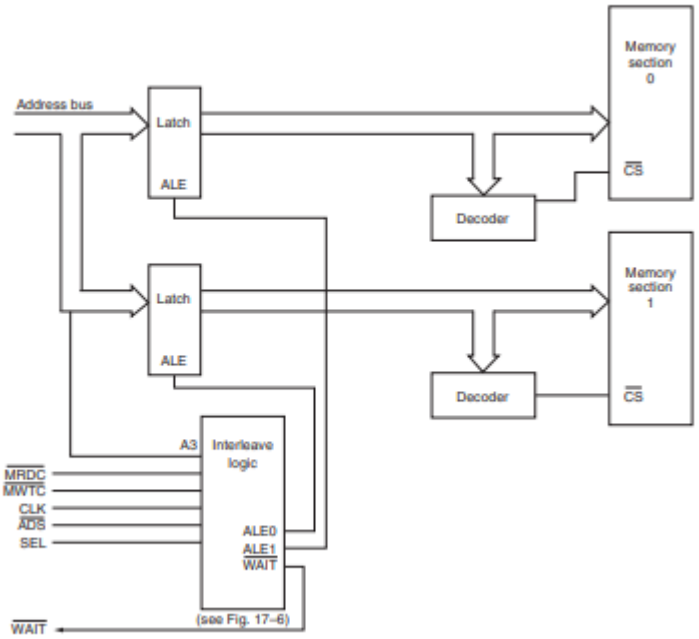


FIGURE 17–7 An interleaved memory system showing the address latches and the interleaved logic circuit.

The access time allowed by an interleaved system, such as the one shown in Figure 17–7, is increased to 112 ns from 69 ns by using a 16 MHz system clock. (If a wait state is inserted, access time with a 16 MHz clock is 136 ns, which means that an interleaved system performs at about the same rate as a system with one wait state.) If the clock is increased to 20 MHz, the interleaved memory requires 89.6 ns, where standard, non interleaved memory interfaces allow 48 ns for memory access. At this higher clock rate, 80 ns DRAMs function properly without wait states when the memory addresses are interleaved. If an access to the same section occurs, a wait state is inserted.

The Input/Output System

The 80386 input/output system is the same as that found in any Intel 8086 family microprocessorbased system. There are 64K different bytes of I/O space available if isolated I/O is implemented. With isolated I/O, the IN and OUT instructions are used to transfer I/O data between the microprocessor and I/O devices. The I/O port address appears on address bus connections A15–A2, with used to select a byte, word, or doubleword of I/O data. If memorymapped I/O is implemented, then the number of I/O locations can be any amount up to 4G bytes. With memory-mapped I/O, any instruction that transfers data between the microprocessor and memory system can be used for I/O transfers because the I/O device is treated as a memory

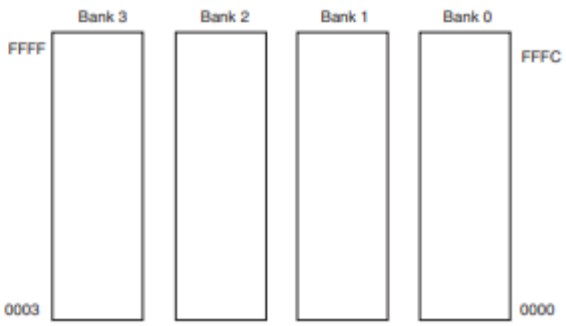


FIGURE 17–8 The isolated I/O map for the 80386 microprocessor. Here four banks of 8 bits each are used to address 64K different I/O locations. I/O is numbered from location 0000H to FFFFH.

device. Almost all 80386 systems use isolated I/O because of the I/O protection scheme provided by the 80386 in protected mode operation.

Figure 17–8 shows the I/O map for the 80386 microprocessor. Unlike the I/O map of earlier Intel microprocessors, which were 16 bits wide, the 80386 uses a full 32-bit-wide I/O system divided into four banks. This is identical to the memory system, which is also divided into four banks. Most I/O transfers are 8 bits wide because we often use ASCII code (a 7-bit code) for transferring alphanumeric data between the microprocessor and printers and keyboards. This may change if Unicode, a 16-bit alphanumeric code, becomes common and replaces ASCII code. Recently, I/O devices that are 16 and even 32 bits wide have appeared for systems such as disk memory and video display interfaces. These wider I/O paths increase the data transfer rate between the microprocessor and the I/O device when compared to 8-bit transfers.

The I/O locations are numbered from 0000H to FFFFH. A portion of the I/O map is designated for the 80387 arithmetic coprocessor. Although the port numbers for the coprocessor are well above the normal I/O map, it is important that they be taken into account when decoding I/O space (overlaps). The coprocessor uses I/O locations 800000F8H–800000FFH for communications between the 80387 and 80386. The 80287 numeric coprocessor designed for use with the 80286 uses the I/O addresses 00F8H–00FFH for coprocessor communications. Because we often decode only address connections A15–A2 to select an I/O device, be aware that the coprocessor will activate devices 00F8H–00FFH unless address line A31 is also decoded. This should present no problem because you really should not be using I/O ports 00F8H–00FFH for any purpose.

The only new feature that was added to the 80386 with respect to I/O is the I/O privilege information added to the tail end of the TSS when the 80386 is operated in protected mode. As described in the section on memory management, an I/O location can be blocked or inhibited in the protected mode. If the blocked I/O location is addressed, an interrupt (type 13, general fault) is generated. This scheme is added so that I/O access can be prohibited in a multiuser environment. Blocking is an extension of the protected mode operation, as are privilege levels.

INTRODUCTION TO THE 80486 MICROPROCESSOR

The 80486 microprocessor is a highly integrated device, containing well over 1.2 million transistors. Located within this device circuit are a memory-management unit (MMU), a complete numeric coprocessor that is compatible with the 80387, a high-speed level 1 cache memory that contains 8K bytes of space, and a full 32-bit microprocessor that is upward-compatible with the 80386 microprocessor. The 80486 is currently available as a 25 MHz, 33 MHz, 50 MHz, 66 MHz, or 100 MHz device. Note that the 66 MHz version is double-clocked and the 100 MHz version is triple-clocked. In 1990, Intel demonstrated a 100 MHz version (not double-clocked) of the 80486 for Computer Design magazine, but it has yet to be released. Advanced Micro Devices (AMD) has produced a 40 MHz version that is also available in an 80 MHz (double-clocked) and a 120 MHz (triple-clocked) form. The 80486 is available as an 80486DX or an 80486SX. The only difference between these devices is that the 80486SX does not contain the numeric coprocessor, which reduces its price. The 80487SX numeric coprocessor is available as a separate component for the 80486SX microprocessor. This section details the differences between the 80486 and 80386 microprocessors. These differences are few, as shall be seen. The most notable differences apply to the cache memory system and parity generator.

Pin-Out of the 80486DX and 80486SX Microprocessors

Figure 17–29 illustrates the pin-out of the 80486DX microprocessor, a 168-pin PGA. The 80486SX, also packaged in a 168-pin PGA, is not illustrated because only a few differences exist. Note that pin B15 is NMI on the 80486DX and pin A15 is NMI on the 80486SX. The only other differences are that pin A15 is (ignore numeric error) on the 80486DX (not present on the 80486SX), pin C14 is (floating-point error) on the 80486DX, and pins B15 and C14 on the 80486SX are not connected. When connecting the 80486 microprocessor, all VCC and VSS pins must be connected to the power supply for proper operation. The power supply must be capable of supplying 5.0 V 10%, with up to 1.2 A of surge current for the 33 MHz version. The average supply current is 650 mA for the 33 MHz version. Intel has also produced a 3.3 V version that requires an average of 500 mA at a triple-clock speed of 100 MHz. Logic 0 outputs allow up to 4.0 mA of current, and logic 1 outputs allow up to 1.0 mA. If larger currents are required, as they often are, then the 80486 must be buffered. Figure 17–30 shows a buffered 80486DX system. In the circuit shown, only the address, data, and parity signals are buffered.

Pin Definitions

A31–A2 Address outputs A31–A2 provide the memory and I/O with the address during normal operation; during a cache line invalidation, A31–A4 are used to drive the microprocessor.

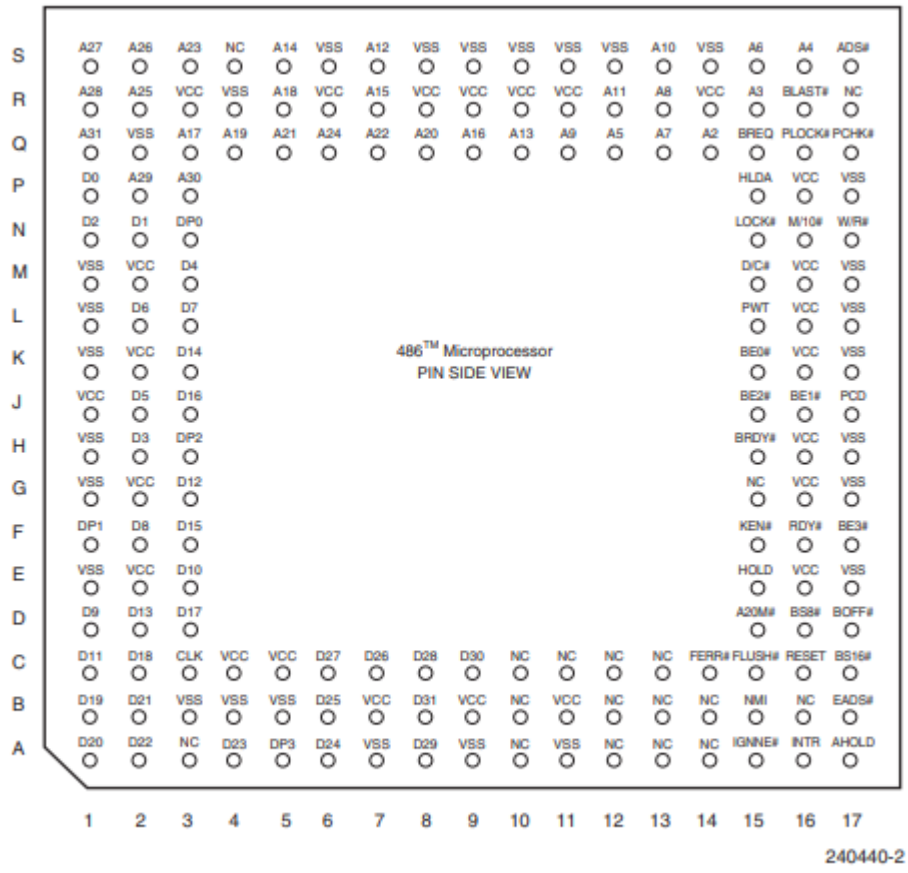


FIGURE 17–29 The pin-out of the 80486. (Courtesy of Intel Corporation.)

$\overline{\text{A20M}}$ **Address bit 20 mask** causes the 80486 to wrap its address around from location 000FFFFFH to 00000000H, as does the 8086 microprocessor. This provides a memory system that functions like the 1M-byte real memory system in the 8086 microprocessor.

$\overline{\text{ADS}}$ **Address data strobe** becomes a logic 0 to indicate that the address bus contains a valid memory address.

AHOLD **Address hold** input causes the microprocessor to place its address bus connections at their high-impedance state, with the remainder of the buses staying active. It is often used by another bus master to gain access for a cache invalidation cycle.

$\overline{\text{BE3}}$–$\overline{\text{BE0}}$ **Byte enable** outputs select a bank of the memory system when information is transferred between the microprocessors and its memory and I/O space. The $\overline{\text{BE3}}$ signal enables D_{31}–D_{24}, $\overline{\text{BE2}}$ enables D_{23}–D_{16}, $\overline{\text{BE1}}$ enables D_{15}–D_8, and $\overline{\text{BE0}}$ enables D_7–D_0.

$\overline{\text{BLAST}}$ The **burst last** output shows that the burst bus cycle is complete on the next activation of the $\overline{\text{BRDY}}$ signal.

$\overline{\text{BOFF}}$ The **back-off** input causes the microprocessor to place its buses at their high-impedance state during the next clock cycle. The microprocessor remains in the bus hold state until the $\overline{\text{BOFF}}$ pin is placed at a logic 1 level.

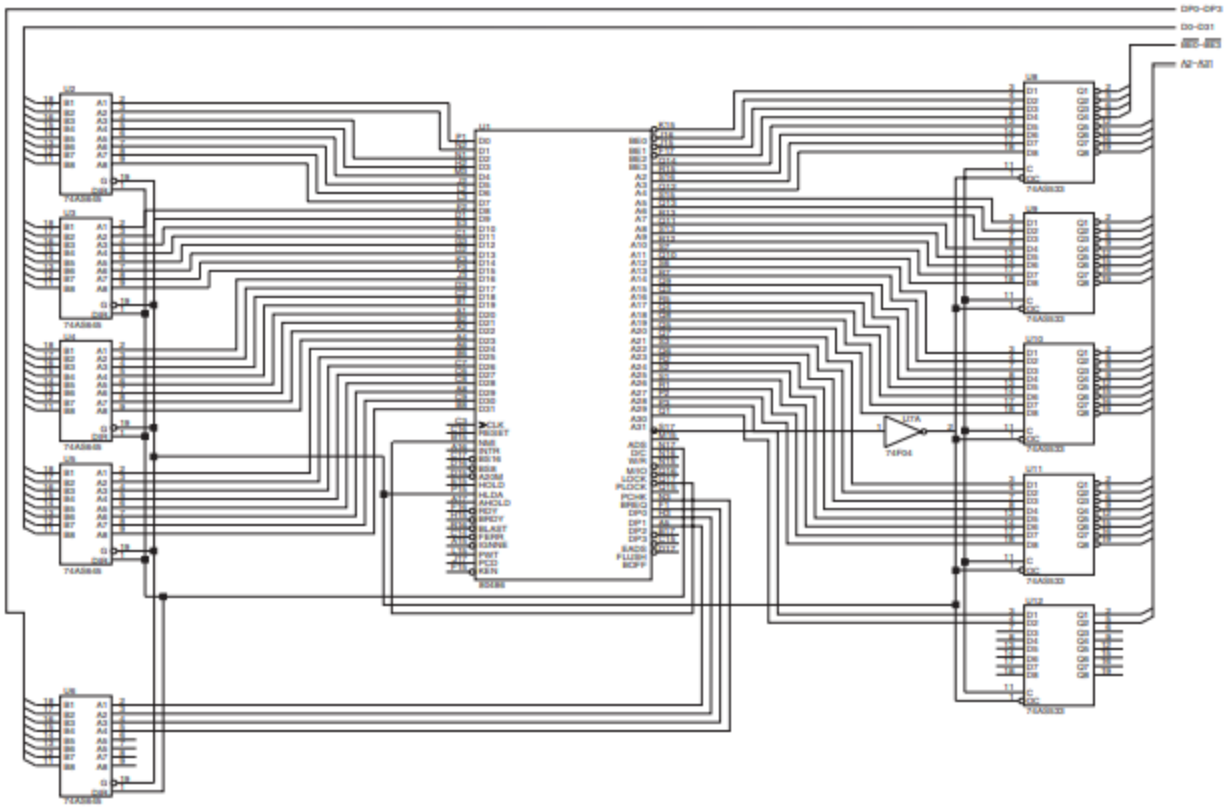


FIGURE 17–30 An 80486 microprocessor showing the buffered address, data, and parity buses.

$\overline{\text{BRDY}}$ The **burst ready** input is used to signal the microprocessor that a burst cycle is complete.

BREQ The **bus request** output indicates that the 80486 has generated an internal bus request.

$\overline{\text{BS8}}$ The **bus size 8** input causes the 80486 to structure itself with an 8-bit data bus to access byte-wide memory and I/O components.

$\overline{\text{BS16}}$ The **bus size 16** input causes the 80486 to structure itself with a 16-bit data bus to access word-wide memory and I/O components.

CLK The **clock** input provides the 80486 with its basic timing signal. The clock input is a TTL-compatible input that is 25 MHz to operate the 80486 at 25 MHz.

D_{31}–D_0 The **data bus** transfers data between the microprocessor and its memory and I/O system. Data bus connections D_7–D_0 are also used to accept the interrupt vector type number during an interrupt acknowledge cycle.

D/$\overline{\text{C}}$ The **data/control** output indicates whether the current operation is a data transfer or control cycle. See Table 17–3 for the function of D/$\overline{\text{C}}$, M/$\overline{\text{IO}}$, and W/$\overline{\text{R}}$.

DP_3–DP_0 **Data parity** I/O provides even parity for a write operation and check parity for a read operation. If a parity error is detected during a read, the $\overline{\text{PCHK}}$ output becomes a logic 0 to indicate a parity error. If parity is not used in a system, these lines must be pulled high to +5.0 V or to 3.3 V in a system that uses a 3.3 V supply.

$\overline{\text{EADS}}$ The **external address strobe** input is used with AHOLD to signal that an external address is used to perform a cache-invalidation cycle.

$\overline{\text{FERR}}$ The floating-point error output indicates that the floating-point coprocessor has detected an error condition. It is used to maintain compatibility with DOS software.

$\overline{\text{FLUSH}}$ The **cache flush** input forces the microprocessor to erase the contents of its 8K-byte internal cache.

HLDA The **hold acknowledge** output indicates that the HOLD input is active and that the microprocessor has placed its buses at their high-impedance state.

HOLD The **hold** input requests a DMA action. It causes the address, data, and control buses to be placed at their high-impedance state and also, once recognized, causes HLDA to become a logic 0.

$\overline{\text{IGNNE}}$ The **ignore numeric error** input causes the coprocessor to ignore floating-point errors and to continue processing data. This signal does not affect the state of the $\overline{\text{FERR}}$ pin.

TABLE 17–3 Bus cycle identification.

M/IO	*D/C*	*W/R*	*Bus Cycle Type*
0	0	0	Interrupt acknowledge
0	0	1	Halt/special
0	1	0	I/O read
0	1	1	I/O write
1	0	0	Opcode fetch
1	0	1	Reserved
1	1	0	Memory read
1	1	1	Memory write

INTR	The **interrupt request** input requests a maskable interrupt, as it does in all other family members.
$\overline{\text{KEN}}$	The **cache enable** input causes the current bus to be stored in the internal cache.
$\overline{\text{LOCK}}$	The **lock** output becomes a logic 0 for any instruction that is prefixed with the lock prefix.
M/$\overline{\text{IO}}$	**Memory/IO** defines whether the address bus contains a memory address or an I/O port number. It is also combined with the W/$\overline{\text{R}}$ signal to generate memory and I/O read and write control signals.
NMI	The **non-maskable interrupt** input requests a type 2 interrupt.
PCD	The **page cache disable** output reflects the state of the PCD attribute bit in the page table entry or the page directory entry.
$\overline{\text{PCHK}}$	The **parity check** output indicates that a parity error was detected during a read operation on the DP_3–DP_0 pins.
$\overline{\text{PLOCK}}$	The **pseudo-lock** output indicates that the current operation requires more than one bus cycle to perform. This signal becomes a logic 0 for arithmetic coprocessor operations that access 64- or 80-bit memory data.
PWT	The **page write through** output indicates the state of the PWT attribute bit in the page table entry or the page directory entry.
$\overline{\text{RDY}}$	The **ready** input indicates that a non-burst bus cycle is complete. The $\overline{\text{RDY}}$ signal must be returned, or the microprocessor places wait states into its timing until $\overline{\text{RDY}}$ is asserted.
RESET	The **reset** input initializes the 80486, as it does in other family members. Table 17–4 shows the effect of the RESET input on the 80486 microprocessor.
W/$\overline{\text{R}}$	**Write/read** signals that the current bus cycle is either a read or a write.

Basic 80486 Architecture

The architecture of the 80486DX is almost identical to the 80386. Added to the 80386 architecture inside the 80486DX is a math coprocessor and an 8K-byte level 1 cache memory. The 80486SX is almost identical to an 80386 with an 8K-byte cache, but no numeric coprocessor.

TABLE 17–4 State of the microprocessor after a RESET.

Register	*Initial Value with Self Test*	*Initial Value without Self Test*
EAX	00000000H	?
EDX	00000400H + ID*	00000400H + ID*
EFLAGS	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H
ES	0000H	0000H
CS	F000H	F000H
DS	0000H	0000H
SS	0000H	0000H
GS	0000H	0000H
FS	0000H	0000H
IDTR	Base = 0, limit = 3FFH	Base = 0, limit = 3FFH
CR_0	60000010H	60000010H
DR_7	00000000H	00000000H

*Note: Revision ID number is supplied by Intel for revisions to the microprocessor.

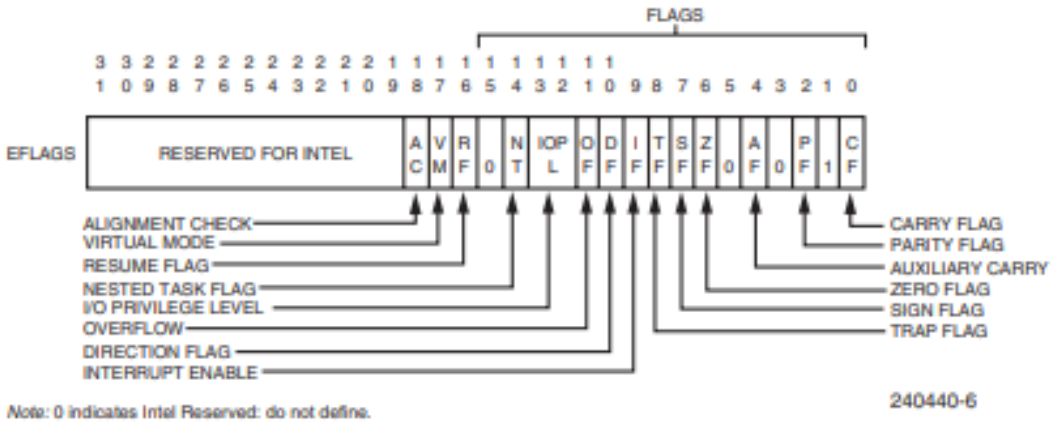


FIGURE 17–31 The EFLAG register of the 80486. (Courtesy of Intel Corporation.)

The extended flag register (EFLAG) is illustrated in Figure 17–31. As with other family members, the rightmost flag bits perform the same functions for compatibility. The only new flag bit is the AC (alignment check), used to indicate that the microprocessor has accessed a word at an odd address or a doubleword stored at a non-doubleword boundary. Efficient software and execution require that data be stored at word or doubleword boundaries.

80486 Memory System

The memory system for the 80486 is identical to the 80386 microprocessor. The 80486 contains 4G bytes of memory, beginning at location 00000000H and ending at location FFFFFFFFH. The major change to the memory system is internal to the 80486 in the form of an 8K-byte cache memory, which speeds the execution of instructions and the acquisition of data. Another addition is the parity checker/generator built into the 80486 microprocessor.

Parity Checker/Generator. Parity is often used to determine if data are correctly read from a memory location. To facilitate this, Intel has incorporated an internal parity generator/detector. Parity is generated by the 80486 during each write cycle. Parity is generated as even parity, and a parity bit is provided for each byte of memory. The parity check bits appear on pins DP0–DP3, which are also parity inputs as well as outputs. These are typically stored in memory during each write cycle and read from memory during each read cycle. On a read, the microprocessor checks parity and generates a parity check error, if it occurs, on the pin. A parity error causes no change in processing unless the user applies the signal to an interrupt input. Interrupts are often used to signal a parity error in DOS-based computer systems. Figure 17–32 shows the organization of the 80486 memory system that includes parity storage. Note that this is the same as for the 80386, except for the parity bit storage. If parity is not used, Intel recommends that the DP0–DP3 pins be pulled up to 5.0 V.

Cache Memory. The cache memory system caches (stores) data used by a program and also the instructions of the program. The cache is organized as a four-way set associative cache, with each location (line) containing 16 bytes or four double words of data. The cache operates as a

write-through cache. Note that the cache changes only if a miss occurs. This means that data written to a memory location not already cached are not written to the cache. In many cases, much of the active portion of a program is found completely inside the cache memory. This causes execution to occur at the rate of one clock cycle for many of the instructions that are

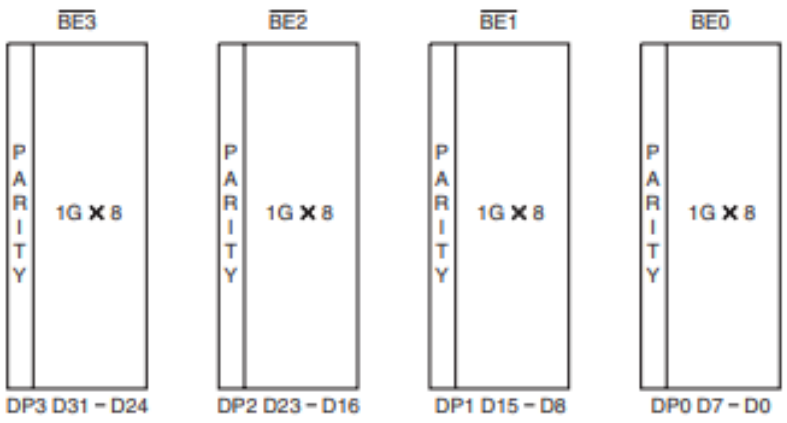


FIGURE 17–32 The organization of the 80486 memory, showing parity.

commonly used in a program. About the only way that these efficient instructions are slowed is when the microprocessor must fill a line in the cache. Data are also stored in the cache, but it has less of an impact on the execution speed of a program because data are not referenced repeatedly as many portions of a program are. Control register 0 (CR0) is used to control the cache with two new control bits not present in the 80386 microprocessor. (See Figure 17–33 for CR0 in the 80486 microprocessor.) The CD (cache disable) and NW (noncache write-through) bits are new to the 80486 and are used to control the 8K-byte cache. If the CD bit is a logic 1, all cache operations are inhibited. This setting is used only for debugging software and normally remains cleared. The NW bit is used to inhibit cache write-through operations. As with CD, cache write-through is inhibited only for testing. For normal program operation, CD = 0 and NW = 0. Because the cache is new to the 80486 microprocessor and the cache is filled by using burst cycles not present on the 80386, some detail is required to understand bus-filling cycles. When a bus line is filled, the 80486 must acquire four 32-bit numbers from the memory system to fill a line in the cache. Filling is accomplished with a burst cycle. The burst cycle is a special memory where four 32-bit numbers are fetched from the memory system in five clocking periods. This assumes that the speed of the memory is sufficient and that no wait states are required. If the clock frequency of the 80486 is 33 MHz, we can fill a cache line in 167 ns, which is very efficient considering that a normal, nonburst 32-bit memory read operation requires two clocking periods.

Memory Read Timing. Figure 17–34 illustrates the read timing for the 80486 for a nonburst memory operation. Note that two clocking periods are used to transfer data. Clocking period T1 provides the memory address and control signals, and clocking period T2 is where the data are transferred between the memory and the microprocessor. Note that the must become a logic 0 to

cause data to be transferred and to terminate the bus cycle. Access time for a nonburst access is determined by taking two clocking periods, minus the time required for the address to appear on the address bus connection, minus a setup time for the data bus connections. For the 20 MHz version of the 80486, two clocking periods require 100 ns minus 28 ns for address setup time and 6 ns for data setup time. This yields a nonburst access time of 100 ns - 34 ns, or 76 ns. Of course, if decoder time and delay times are included, the access time allowed the memory is

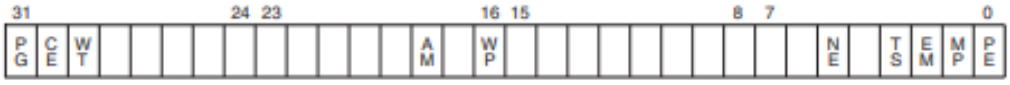
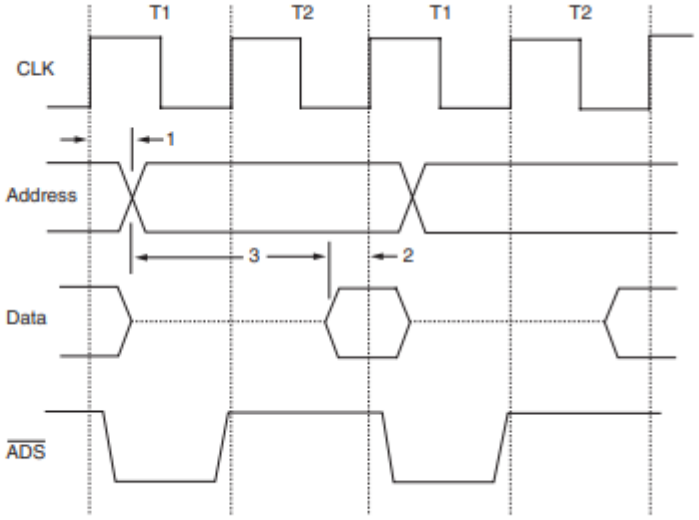


FIGURE 17–33 Control register zero (CR0) for the 80486 microprocessor.

FIGURE 17–34 The non-burst read timing for the 80486 microprocessor.



even less for no wait-state operation. If a higher frequency version of the 80486 is used in a system, memory access time is still less. The 80486 33 MHz, 66 MHz, and 100 MHz processors all access bus data at a 33 MHz rate. In other words, the microprocessor may operate at 100 MHz, but the system bus operates at 33 MHz. Notice that the nonburst access timing for the 33 MHz system bus allows 60 ns - 24 ns = 36 ns. It is obvious that wait states are required for operation with standard DRAM memory devices. Figure 17–35 illustrates the timing diagram for filling a cache line with four 32-bit numbers using a burst. Note that the addresses (A31–A4) appear during T1 and remain constant throughout the burst cycle. Also, note that A2 and A3 change during each T1 after the first to address four consecutive 32-bit numbers in the memory system. As mentioned, cache fills using bursts require only five clocking periods (one T1 and four T2s) to fill a cache line with four double words of data.

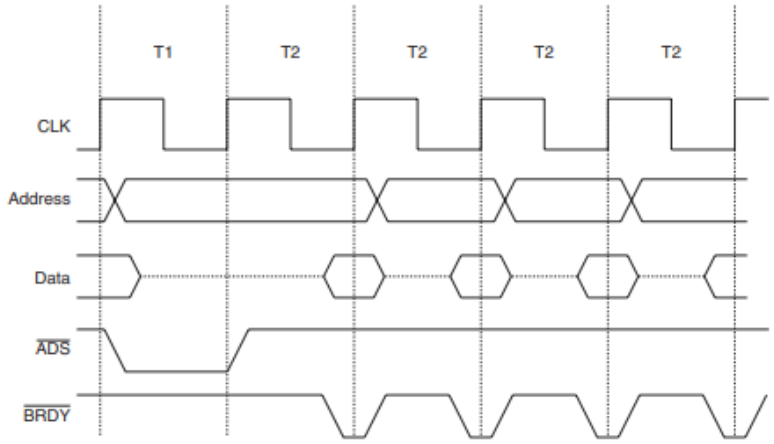


FIGURE 17–35 A burst cycle reads four doublewords in five clocking periods.

Access time using a 20 MHz version of the 80486 for the second and subsequent double words is 50 ns - 28 ns - 5 ns, or 17 ns, assuming no delays in the system. To use burst mode transfers, we need high-speed memory. Because DRAM memory access times are 40 ns at best, we are forced to use SRAM for burst cycle transfers. The 33 MHz system allows an access time of 30 ns - 19 ns - 5 ns, or 6 ns for the second and subsequent bytes. If an external counter is used in place of address bits A2 and A3, the 19 ns can be eliminated and the access time becomes 30 ns - 5 ns, or 25 ns, which is enough time for even the slowest SRAM connected to the system as a cache. This circuit is often called a synchronous burst mode cache if a SRAM cache is used with the system. Note that the pin acknowledges a burst transfer rather than the pin, which acknowledges a normal memory transfer. The PWT controls how the cache functions for a write operation of the external cache memory; it does not control writing to the internal cache. The logic level of this bit is found on the PWT pin of the 80486 microprocessor. Externally, it can be used to dictate the write-through policy of the external cache. The PCD bit controls the on-chip cache. If the PCD = 0, the on-chip cache is enabled for the current page of memory. Note that 80386 page table entries place a logic 0 in the PCD bit position, enabling caching. If PCD = 1, the on-chip cache is disabled. Caching is disabled, regardless of the condition of , CD, and NW.

Pentium Processors:

Intel introduced microprocessors in 1969

∗ 4-bit microprocessor 4004

∗ 8-bit microprocessors » 8080 » 8085

∗ 16-bit processors

» 8086 introduced in 1979 – 20-bit address bus, 16-bit data bus

» 8088 is a less expensive version – Uses 8-bit data bus

» Can address up to 4 segments of 64 KB

» Referred to as the real mode

* 80186

» A faster version of 8086

» 16-bit data bus and 20-bit address bus

» Improved instruction set

∗ 80286 was introduced in 1982

» 24-bit address bus

» 16 MB address space

» Enhanced with memory protection capabilities

» Introduced protected mode – Segmentation in protected mode is different from the real mode

» Backwards compatible

∗ 80386 was introduced 1985

» First 32-bit processor

» 32-bit data bus and 32-bit address bus

» 4 GB address space

» Segmentation can be turned off (flat model)

» Introduced paging

∗ 80486 was introduced 1989

» Improved version of 386

» Combined coprocessor functions for performing floating-point arithmetic

» Added parallel execution capability to instruction decode and execution units – Achieves scalar execution of 1 instruction/clock

» Later versions introduced energy savings for laptops

∗ Pentium (80586) was introduced in 1993

» Similar to 486 but with 64-bit data bus

» Wider internal data paths – 128- and 256-bit wide

» Added second execution pipeline – Superscalar performance – Two instructions/clock

» Doubled on-chip L1 cache – 8 KB data – 8 KB instruction

» Added branch prediction

∗ Pentium Pro was introduced in 1995

» Three-way superscalar – 3 instructions/clock

» 36-bit address bus – 64 GB address space

» Introduced dynamic execution – Out-of-order execution – Speculative execution

» In addition to the L1 cache – Has 256 KB L2 cache

∗ Pentium II was introduced in 1997

» Introduced multimedia (MMX) instructions

» Doubled on-chip L1 cache – 16 KB data – 16 KB instruction

» Introduced comprehensive power management features – Sleep – Deep sleep

» In addition to the L1 cache – Has 256 KB L2 cache

∗ Pentium III, Pentium IV…

UNIT –3

I/O INTERFACING

Memory Devices and Interfacing

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. Most of the peripheral devices are designed and interfaced with a CPU either to enable it to communicate with the user or an external process and to ease the circuit operations so that the microprocessor works more efficiently.

The use of peripheral integrated devices simplifies both the hardware circuits and software considerable. The following are the devices used in interfacing of Memory and General I/O devices.

• 74LS138 (Decoder / Demultiplexed).

• 74LS373 / 74LS374 3-STATE Octal D-Type Transparent Latches.

• 74LS245 Octal Bus Transceiver: 3-State.

74LS138 (Decoder / Demultiplexer)

The LS138 is a high speed 1-of-8 Decoder/ Demultiplexer fabricated with the low power Schottky barrier diode process. The decoder accepts three binary weighted inputs (A0, A1, A2) and when enabled provides eight mutually exclusive active LOW Outputs (O0–O7).

The LS138 can be used as an 8-output demultiplexer by using one of the activeLOW Enable inputs as the data input and the other Enable inputs as strobes. The Enableinputs which are not used must be permanently tied to their appropriate active HIGH oractive LOW state.

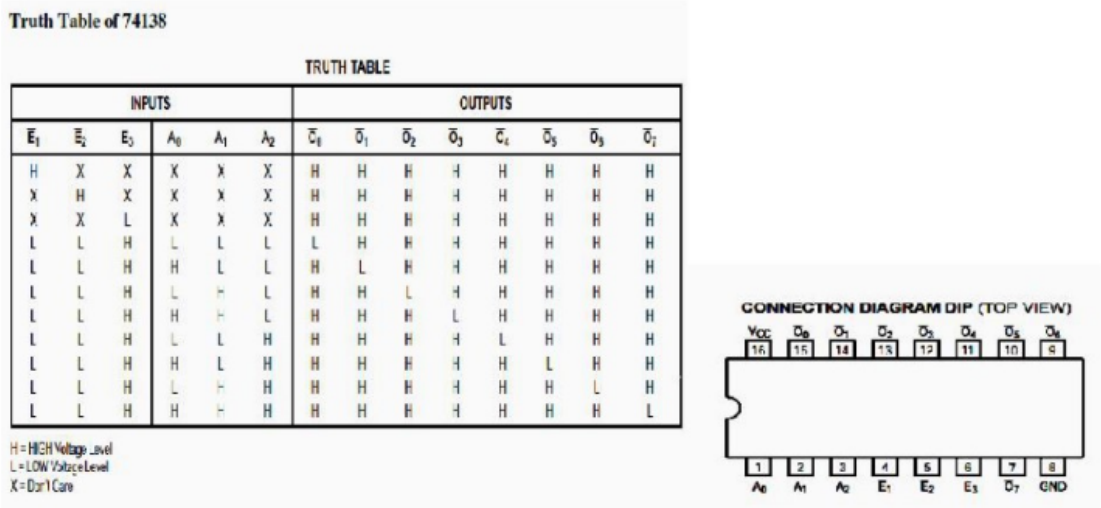
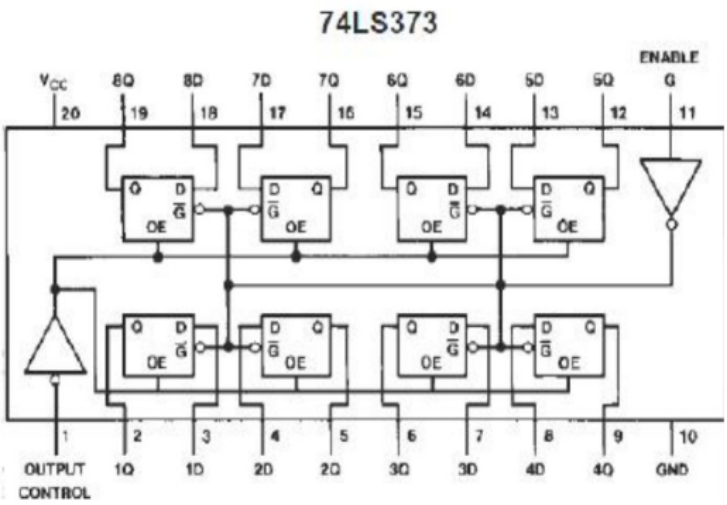


Fig. Pin diagram of 74138

74LS373 / 74LS374 3-STATE Octal D-Type Transparent Latches and Edge TriggeredFlip-Flops

These 8-bit registers feature totem-pole 3-STATE outputs designed specifically for implementing buffer registers, I/O ports, bidirectional bus drivers, and working registers. The eight latches of the 74LS373 are transparent D type latches meaning that while the enable (G) is HIGH the Q outputs will follow the data (D) inputs.When the enable is taken LOW the output will be latched at the level of the data that was set up. The eight flip-flops of the 74LS374 are edge-triggered D-type flip flops. On the positive transition of the clock, the Q outputs will be set to the logic states that were set up at the D inputs.



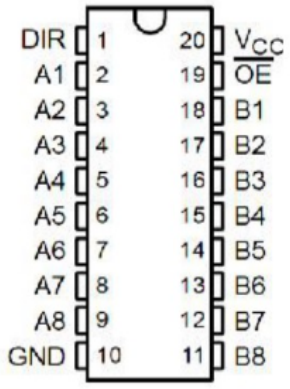


Fig. Connection diagram of 74LS373 **Fig .Pin of 74LS245**

Main Features

• Choice of 8 latches or 8 D-type flip-flops in a single package

• 3-STATE bus-driving outputs

• Full parallel-access for loading

• Buffered control inputs

• P-N-P inputs reduce D-C loading on data lines

74LS245 Octal Bus Transceiver: 3-State

The 74LS245 is a high-speed Si-gate CMOS device. The 74LS245 is an octal transceiver featuring non- inverting 3-state bus compatible outputs in both send and receive directions. The 74LS245 features an Output Enable (OE) input for easy cascading and a send/receive (DIR) input for direction control. OE controls the outputs so that the buses are effectively isolated. All inputs have a Schmitt-trigger action.

These octal bus transceivers are designed for asynchronous two-way communication between data buses. The 74LS245 is a high-speed Si-gate CMOS device. The 74LS245 is an octal transceiver featuring non-inverting 3-state bus compatible outputs in both send and receive directions.

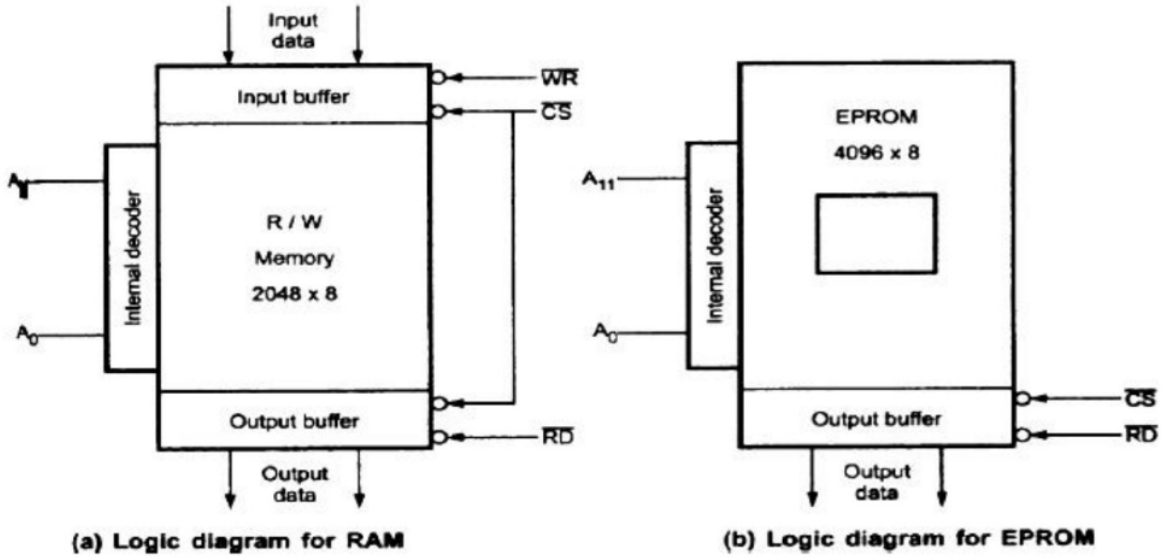
The 74LS245 features an Output Enable (OE) input for easy cascading and a send/receive (DIR) input for direction control. OE controls the outputs so that the buses are effectively isolated. All inputs have a Schmitt-trigger action. These octal bus transceivers are designed for asynchronous two-way communication between data buses.

Memory Devices and Interfacing

The memory interfacing circuit is used to access memory quit frequently to read instruction codes and data stored in the memory. The read / write operations are monitored by control signals. Semiconductor memories are of two types. Viz. RAM (Random Access Memory) and ROM (Read Only Memory) The Semiconductor RAM's are broadly two types- static Ram and dynamic RAM

Memory structure and its requirements

The read / write memories consist of an array of registers in which each register has unique address. The size of memory is N * M as shown in figure.



(a) Logic diagram for RAM (b) Logic diagram for EPROM

Where N is number of register and M is the word length, in number of bits. As shown in figure(a) memory chip has 12 address lines Ao–A11, one chip select (CS), and two control lines, Read (RD) to enable output buffer and Write (WR) to enable the input buffer.

The internal decoder is used to decoder the address lines. Figure (b) shows the logic diagramof a typical EPROM (Erasable Programmable Read-Only Memory) with 4096 (4K) register. It has 12 address lines Ao – A11, one chip selects (CS), one read control signal. Since EPROM does not require the (WR) signal

EPROM (or EPROMs) is used as a program memory and RAM (or RAMs) as a data memory. When both, EPROM and RAM are used, the total address space 1 Mbytes is shared by them.

Address Decoding Techniques

• Absolute decoding

• Linear decoding

• Block decoding

Absolute Decoding:

In the absolute decoding technique the memory chip is selected only for the specified logic level on the address lines: no other logic levels can select the chip. Below figure the memory interface with absolute decoding. Two 8K EPROMs (2764) are used to provide even and odd memory

banks. Control signals BHE and Ao are use to enable output of odd and even memory banks respectively. As each memory chip has 8K memory locations, thirteen address lines are required to address each locations, independently. All remaining address lines are used to generate an unique chip select signal. This address technique is normally used in large memory systems.

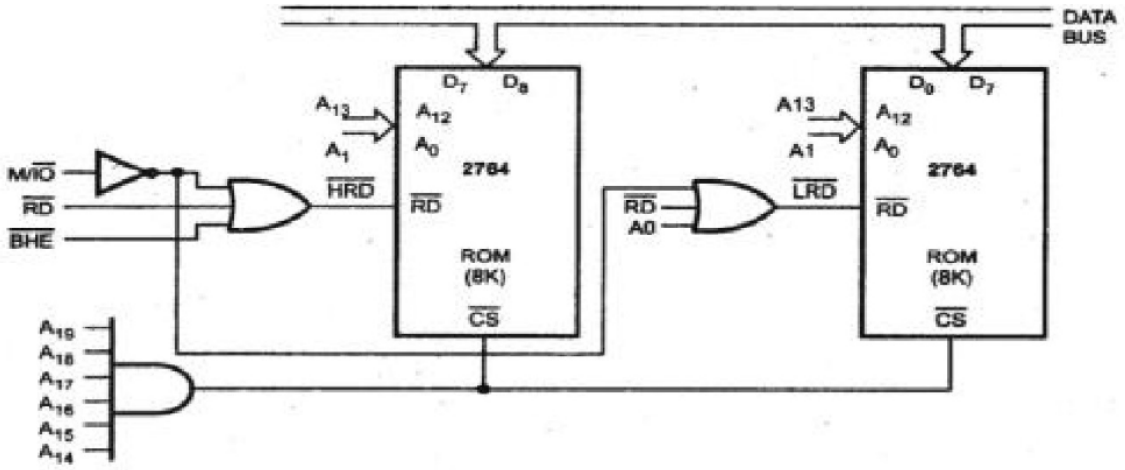


Fig. Linear decoding

Linear Decoding:

In small system hardware for the decoding logic can be eliminated by using only required number of addressing lines (not all). Other lines are simple ignored. This technique is referred as linear decoding or partial decoding. Control signals BHE and Ao are used to enable odd and even memory banks, respectively. Figure shows the addressing of 16K RAM (6264) with linear decoding. The address line A19 is used to select the RAM chips. When A19 is low, chip is selected, otherwise it is disabled. The status of A14 to A18 does not affect the chip selection logic. This gives you multiple addresses (shadow addresses).

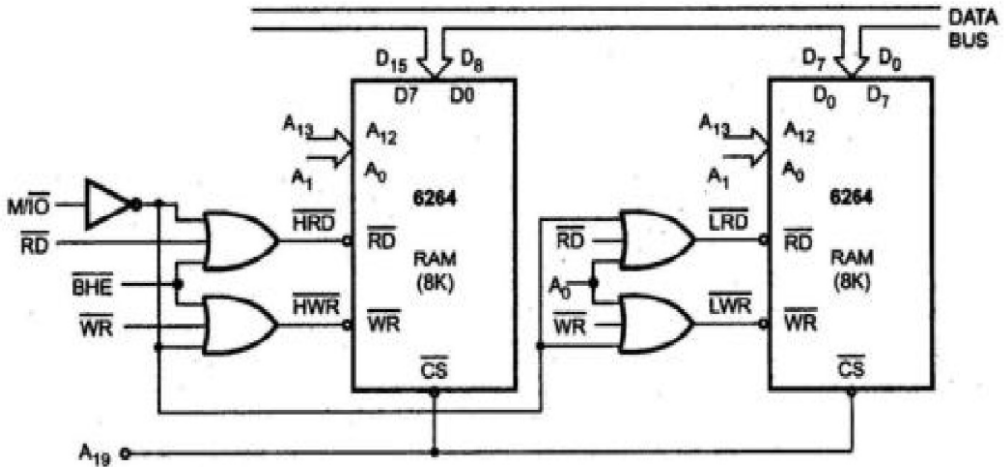


Fig.Block decoding

This technique reduces the cost of decoding circuit, but it gas drawback of multiple addresses

599999999999999, 0.
36666666666666Block Decoding:

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

Static Memory Interfacing

The general procedure of static memory interfacing with 8086 as follows:

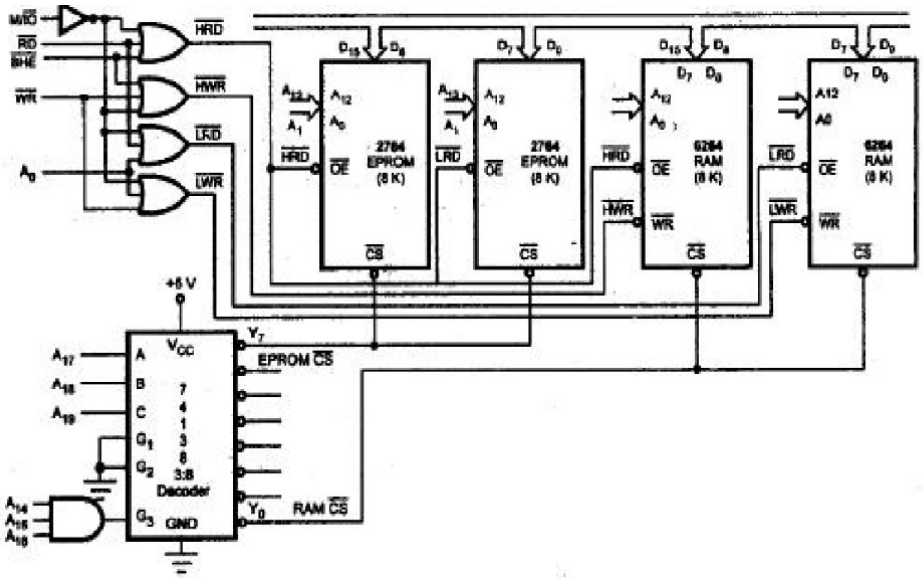


Fig. Static Memory interfacing

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called ‘odd address memory bank’ and the lower 8-bit bank is called ‘even address memory bank’.

2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory RD and WR inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.

3. The remaining address lines of the microprocessor, BHE and Ao are used for decoding the required chip select signals for the odd and even memory banks. The CS of memory is derived from the output of the decoding circuit.

4. As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible

Dynamic RAM Interfacing

The basic Dynamic RAM cell uses a capacitor to store the charge as a representation of data. This capacitor is manufactured as a diode that is reverse-biased so that the storage capacitance comes into the picture. This storage capacitance is utilized for storing the charge representation of data but the reverse-biased diode has a leakage current that tends to discharge the capacitor giving rise to the possibility of data loss.

To avoid this possible data loss, the data stored in a dynamic RAM cell must be refreshed after a fixed time interval regularly. The process of refreshing the data in the RAM is known as refresh cycle.

This activity is similar to reading the data from each cell of the memory, independent of the requirement of microprocessor, regularly. During this refresh period all other operations (accesses) related to the memory subsystem are suspended.

The advantages of dynamic RAM. Like low power consumption, higher packaging density and low cost, most of the advanced computer systems are designed using dynamic RAMs. Also the refresh mechanism and the additional hardware required makes the interfacing hardware, in case of dynamic RAM, more complicated, as compared to static RAM interfacing circuit.

Interfacing I/O Ports

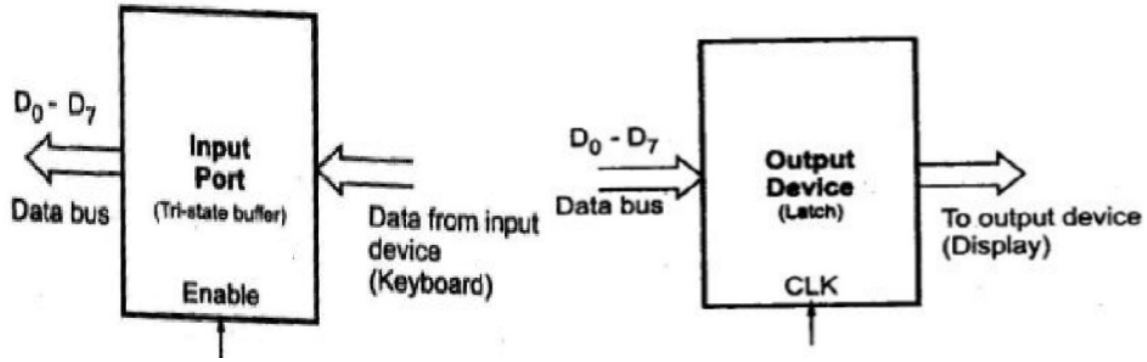


Fig. I/O interfacing

I/O ports or input/output ports are the devices through which the microprocessor communicates with other devices or external data sources/destinations. Input activity, as one may expect, is the activity that enables the microprocessor to read data from external devices, for example keyboard, joysticks, mouser etc. the devices are known as input devices as they feed data into a microprocessor system.

Output activity transfers data from the microprocessor top the external devices, for example CRT display, 7-segment displays, printer, etc, the devices that accept the data from a microprocessor system are called output devices.

Steps in Interfacing an I/O Device

The following steps are performed to interface a general I/O device with a CPU:

1. Connect the data bus of the microprocessor system with the data bus of the I/O port.

2. Derive a device address pulse by decoding the required address of the device and use it as the chip select of the device.

3. Use a suitable control signal, i.e. IORD and /or IOWR to carry out device operations, i.e. connect IORD to RD input of the device if it is an input devise, otherwise connector to WR input of the device. In some cases the RD or WRcontrol signals are combined with the device address pulse to generate the device select pulse.

Input Port

The input device is connected to the microprocessor through buffer. The simplest form of ainput port is a buffer as shown in the figure. This buffer is a tri-state buffer and its output isavailable only when enable signal is active.

When microprocessor wants to read data fromthe input device (keyboard), the control signals from the microprocessor activates the bufferby asserting enable input of the buffer. Once the buffer is enabled, data from the device isavailable on the data bus. Microprocessor reads this data by initiating read command.

Output Port

It is used to send the data to the output device such as display from the microprocessor. The simplest form of the output port is a latch.The output device is connected to the microprocessor through latch as shown in the figure.

When microprocessor wants to send data to the output device it puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device. I/O Interfacing Techniques Input/output devices can be interfaced with microprocessor systems in two ways:

1. I/O mapped I/O

2. Memory mapped I/O

1. I/O mapped I/O:

8086 has special instructions IN and OUT to transfer data through the input/output ports in I/O mapped I/O system. The IN instruction copies data from a port to the Accumulator. If an8-bit port is read data will go to AL and if 16-bit port is read the data will go to AX. The OUT instruction copies a byte from AL or a word from AX to the specified port. The M/IO signal is always low when 8086 is executing these instructions. In this address of I/O device is 8-bit or 16-bit. It is 8-bit for direct addressing and 16-bit for indirect addressing.

2. Memory mapped I/O

In this type of I/O interfacing, the 8086 uses 20 address lines to identify an I/O device. TheI/O device is connected as if it is a memory device. The 8086 uses same control signals and instructions to access I/O as those of memory, here RD and WR signals are activated indicating memory bus cycle.

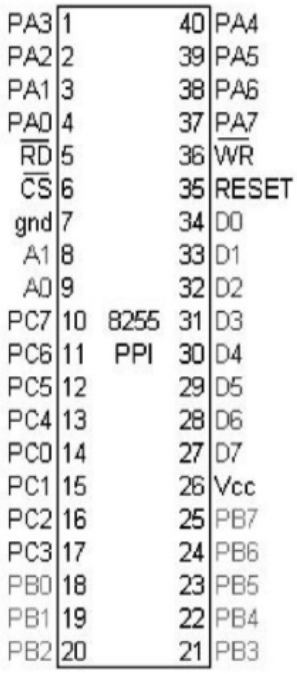
Parallel Communication Interface: 8255 Programmable Peripheral Interface and Interfacing

The 8255 is a widely used, programmable parallel I/O device. It can be programmed to transfer data under data under various conditions, from simple I/O to interrupt I/O. It is flexible, versatile and economical (when multiple I/O ports are required). It is an important general purpose I/O device that can be used with almost any microprocessor.

The 8255 has 24 I/O pins that can be grouped primarily into two 8 bit parallel ports: A and B, with the remaining 8 bits as Port C. The 8 bits of port C can be used as individual bits or be grouped into two 4 bit ports: CUpper (CU) and C Lower (CL). The functions of these ports are defined by writing a control word in the control register.

8255 can be used in two modes: Bit set/Reset (BSR) mode and I/O mode. The BSR mode is used to set or reset the bits in port C. The I/O mode is further divided into 3 modes: mode 0, mode 1 and mode 2. In mode 0, all ports function as simple I/O ports. Mode 1 is a handshake mode whereby Port A and/or Port B use bits from Port C as handshake signals. In the handshake

mode, two types of I/O data transfer can be implemented: status check and interrupt. In mode 2, Port A can be set up for bidirectional data transfer using handshake signals from Port C, and Port B can be set up either in mode 0 or mode 1.



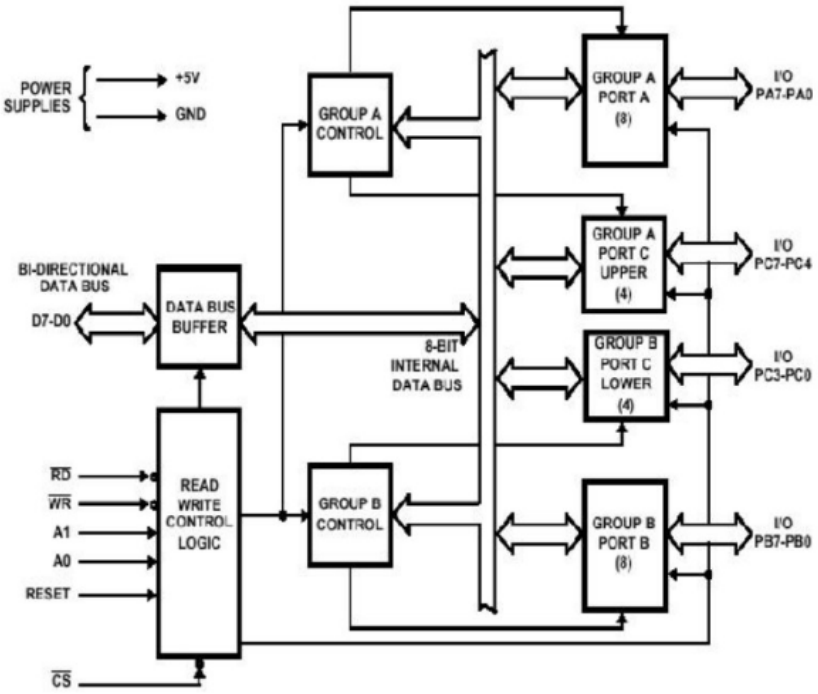


Fig. Pins of 8255 **Fig.Block diagram of 8255**

RD: (Read): This signal enables the Read operation. When the signal is low, microprocessor reads data from a selected I/O port of 8255.

WR: (Write): This control signal enables the write operation.

RESET (Reset): It clears the control registers and sets all ports in input mode.

CS, A0, A1: These are device select signals. is connected to a decoded address and A0, A1 are connected to A0, A1 of microprocessor.

$\overline{CS}$	A_1	A_0	Selected
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	x	x	8255 is not selected

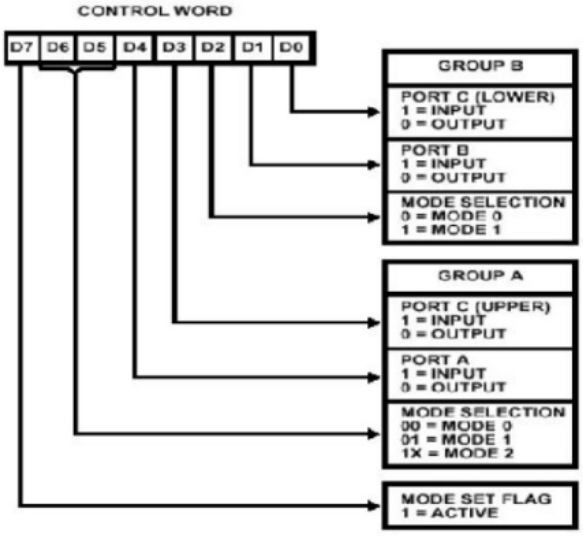


Fig. Control word format of8255

BSR Modes of 8255:

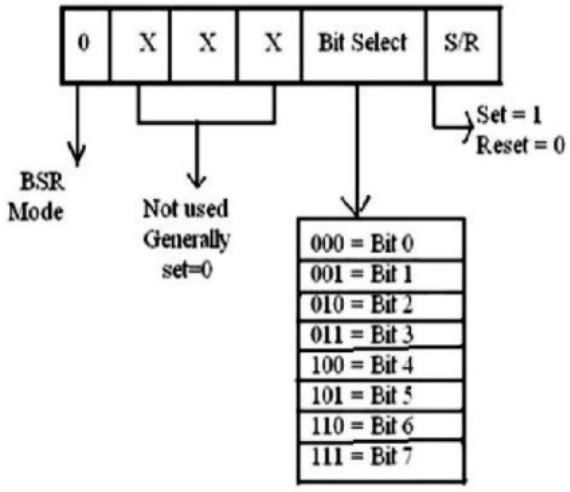


Fig. BSR mode of 8255

I/O Modes of 8255

Mode 0: Simple Input or Output

In this mode, Port A and Port B are used as two simple 8-bit I/O ports and Port C as two4- bit I/O ports. Each port (or half-port, in case of Port C) can be programmed to function as simply an input port or an output port. The input/output features in mode 0 are: Outputs are latched, Inputs are not latched. Ports do not have handshake or interrupt capability.

Mode 1: Input or Output with handshake

In mode 1, handshake signals are exchanged between the microprocessor and peripherals prior to data transfer. The ports (A and B) function as 8-bit I/O ports. Theycan be configured either as input or output ports. Each port (Port A and Port B) uses 3 lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions. Input and output data are latched and Interrupt logic is supported.

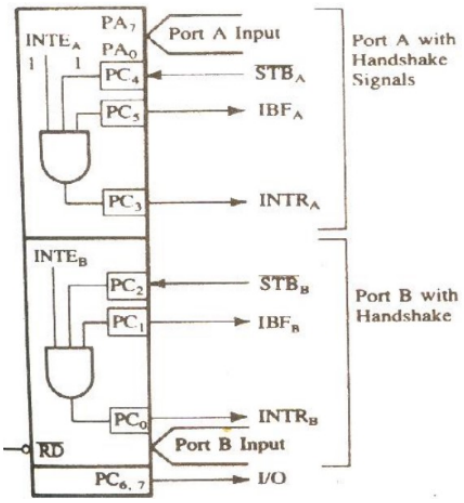


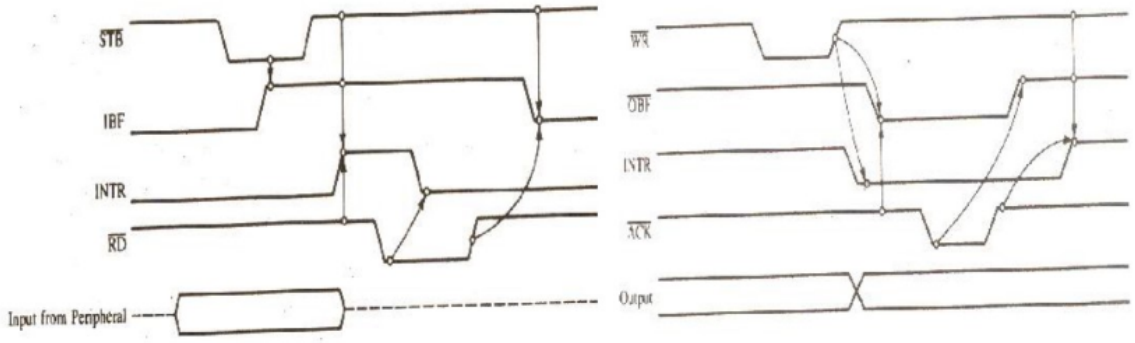
Fig. Mode 1 input control signals

STB Strobe Input): This signal (active low) is generated by a peripheral device that it has transmitted a byte of data. The 8255, in response to, generates IBF and INTR.

IBF (Input buffer full): This signal is an acknowledgement by the 8255 to indicate that the input latch has received the data byte. This is reset when the microprocessor reads the data.

INTR (Interrupt Request): This is an output signal that may be used to interrupt the microprocessor. This signal is generated if , IBF and INTE are all at logic 1.

INTE (Interrupt Enable): This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTEA and INTEB are set /reset using the BSR mode. The INTEA is enabled or disabled through PC4, and INTEB is enabled or disabled through PC2.



(a) input operation **(b) output operation**

Fig. Timing waveform for mode1 operation

(Output Buffer Full): This is an output signal that goes low when the microprocessor writes data into the output latch of the 8255. This signal indicates to an output peripheral that new data is ready to be read. It goes high again after the 8255 receives a signal from the peripheral. (Acknowledge): This is an input signal from a peripheral that must output a low when the peripheral receives the data from the 8255 ports.

INTR (Interrupt Request): This is an output signal, and it is set by the rising edge of thesignal. This signal can be used to interrupt the microprocessor to request the next data byte for output. The INTR is set and INTE are all one and reset by the rising edge of . .

INTE (Interrupt Enable): This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTEA and INTEB are set /reset using the BSRmode. The INTEA signal can be enabled or disabled through PC6, and INTEB is enabled or disabled through PC2.

Mode 2: Bidirectional Data Transfer

OBF This mode is used primarily in applications such as data transfer between the two computers or floppy disk controller interface. Port A can be configured as the bidirectional port and 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 lines from Port C can be used either as simple I/O or as handshake signals for Port B.

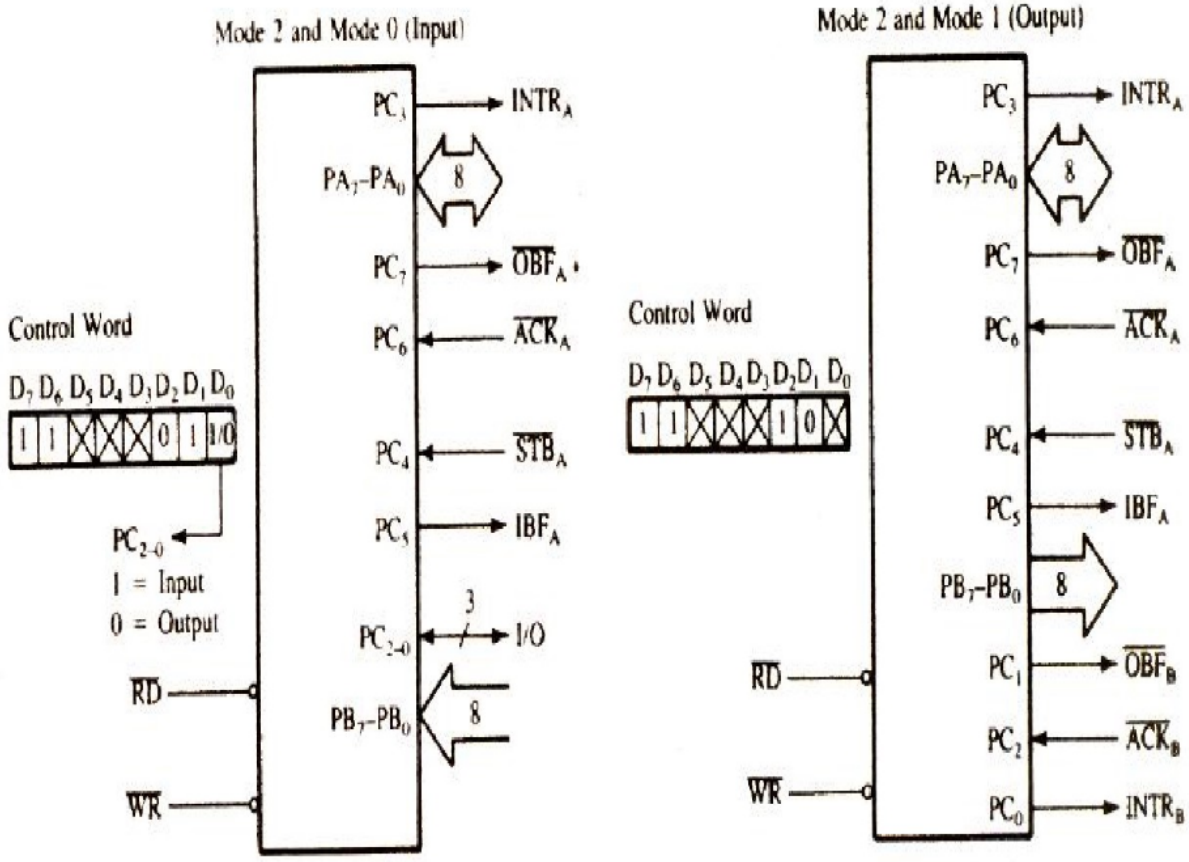


Fig.Mode 2 Control Signals

<u>Serial Communication: Using 8251</u>

8251 is a Universal Synchronous and Asynchronous Receiver and Transmittercompatible with Intel's processors. This chip converts the parallel data into a serial streamof bits suitable for serial transmission. It is also able to receive a serial stream of bits andconvert it into parallel data bytes to be read by a microprocessor.

Basic Modes of data transmission

a) Simplex

b) Duplex

c) Half Duplex

a) Simplex mode

Data is transmitted only in one direction over a single communication channel. For example,the processor may transmit data for a CRT display unit in this mode.

b) Duplex Mode

In duplex mode, data may be transferred between two transreceivers in both directions simultaneously.

c) Half Duplex mode

In this mode, data transmission may take place in either direction, but at a time data may betransmitted only in one direction. A computer may communicate with a terminal in thismode. It is not possible to transmit data from the computer to the terminal and terminal tocomputer simultaneously.

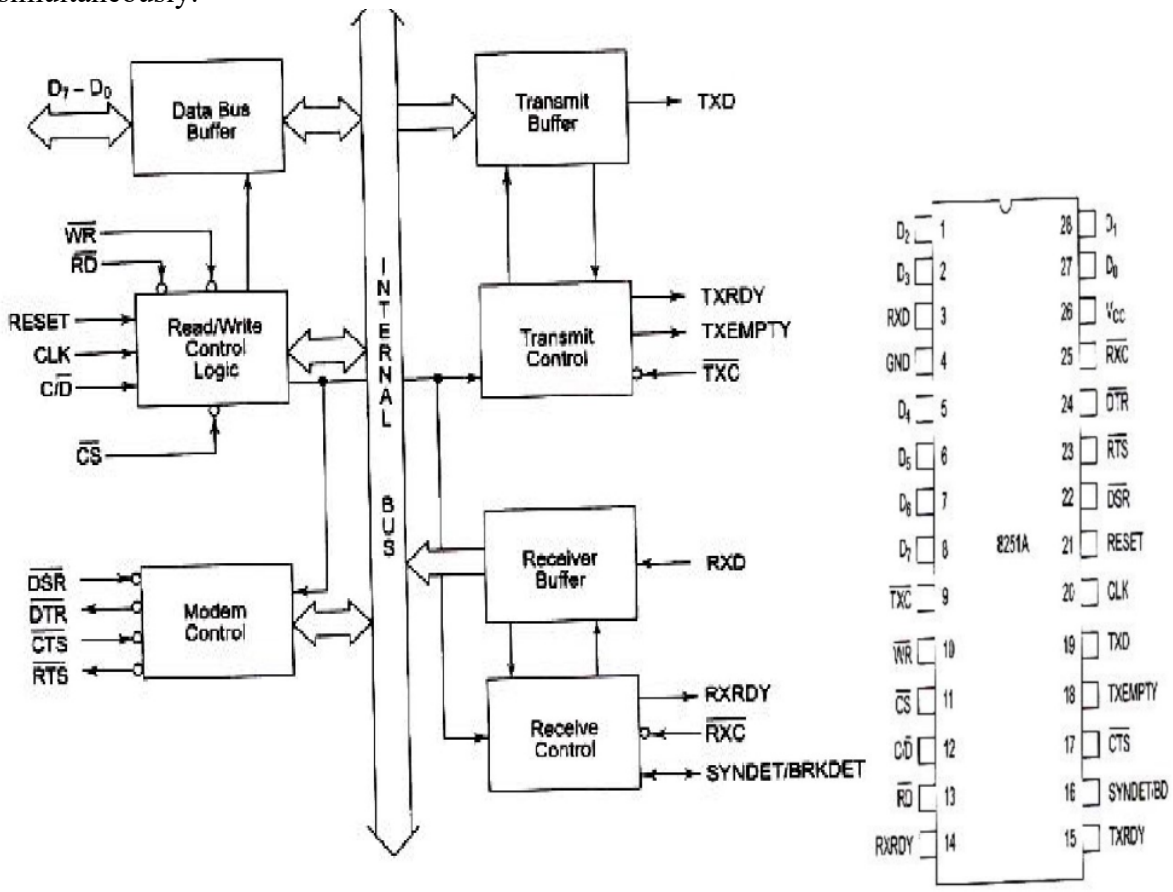


Fig.Serial communication interface 8251

The data buffer interfaces the internal bus of the circuit with the system bus. The read / write control logic controls the operation of the peripheral depending upon the operations initiated by the CPU decides whether the address on internal data bus is control address / data address. The modem control unit handles the modem handshake signals to coordinate the communication between modem and USART.

The transmit control unit transmits the data byte received by the data buffer from the CPU for serial communication. The transmission rate is controlled by the input frequency. Transmit control unit also derives two transmitter status signals namely TXRDY and TXEMPTY which may be used by the CPU for handshaking.

The transmit buffer is a parallel to serial converter that receives a parallel byte forconversion into a serial signal for further transmission. The receive control unit decides thereceiver frequency as controlled by the RXC input frequency. The receive control unitgenerates a receiver ready (RXRDY) signal that may be used by the CPU for handshaking.This unit also detects a break in the data string while the 8251 is in asynchronous mode. Insynchronous mode, the 8251 detects SYNC characters using SYNDET/BD pin.

Signal Description of 8251

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

C / D: (Control Word/Data): This input pin, together with RD and WR inputs, informs the 8251A that the word on the data bus is either a data or control word/status information. If this pin is 1, control / status is on the bus, otherwise data is on the bus.

RD: This active-low input to 8251A is used to inform it that the CPU is reading either data or status information from its internal registers. This active-low input to 8251A is used to inform it that the CPU is writing data or control word to 8251A.

WR: This is an active-low chip select input of 825lA. If it is high, no read or write operation can be carried out on 8251. The data bus is tristated if this pin is high.

CLK: This input is used to generate internal device timings and is normally connected to clock generator output. This input frequency should be at least 30 times greater than the receiver or transmitter data bit transfer rate.

RESET: A high on this input forces the 8251A into an idle state. The device will remain idle till this input signal again goes low and a new set of control word is written into it. The minimum required reset pulse width is 6 clock states, for the proper reset operation.

TXC (Transmitter Clock Input): This transmitter clock input controls the rate at which the character is to be transmitted. The serial data is shifted out on the successive negative edge of the TXC.

TXD (Transmitted Data Output): This output pin carries serial stream of the transmitted data bits along with other information like start bit, stop bits and parity bit, etc.

RXC (Receiver Clock Input): This receiver clock input pin controls the rate at which the character is to be received.

RXD (Receive Data Input): This input pin of 8251A receives a composite stream of the data to be received by 8251 A.

RXRDY (Receiver Ready Output): This output indicates that the 8251A contains a character to be read by the CPU.

TXRDY - Transmitter Ready: This output signal indicates to the CPU that the internal circuit of the transmitter is ready to accept a new character for transmission from the CPU.

DSR - Data Set Ready: This is normally used to check if data set is ready when communicating with a modem.

DTR - Data Terminal Ready: This is used to indicate that the device is ready to accept data when the 8251 is communicating with a modem.

RTS - Request to Send Data: This signal is used to communicate with a modem.

TXE- Transmitter Empty: The TXE signal can be used to indicate the end of a transmission mode.

Operating Modes of 8251

1. Asynchronous mode

2. Synchronous mode

Asynchronous Mode (Transmission)

When a data character is sent to 8251A by the CPU, it adds start bits prior to the serial data bits, followed by optional parity bit and stop bits using the asynchronous mode instruction control word format. This sequence is then transmitted using TXD output pin on the fallingedge of TXC.

Asynchronous Mode (Receive)

A falling edge on RXD input line marks a start bit. The receiver requires only onestop bit to mark end of the data bit string, regardless of the stop bit programmed at thetransmitting end. The 8-bit character is then loaded into the into parallel I/O buffer of 8251.

RXRDY pin is raised high to indicate to the CPU that a character is ready for it. Ifthe previous character has not been read by the CPU, the new character replaces it, and theoverrun flag is set indicating that the previous character is lost.

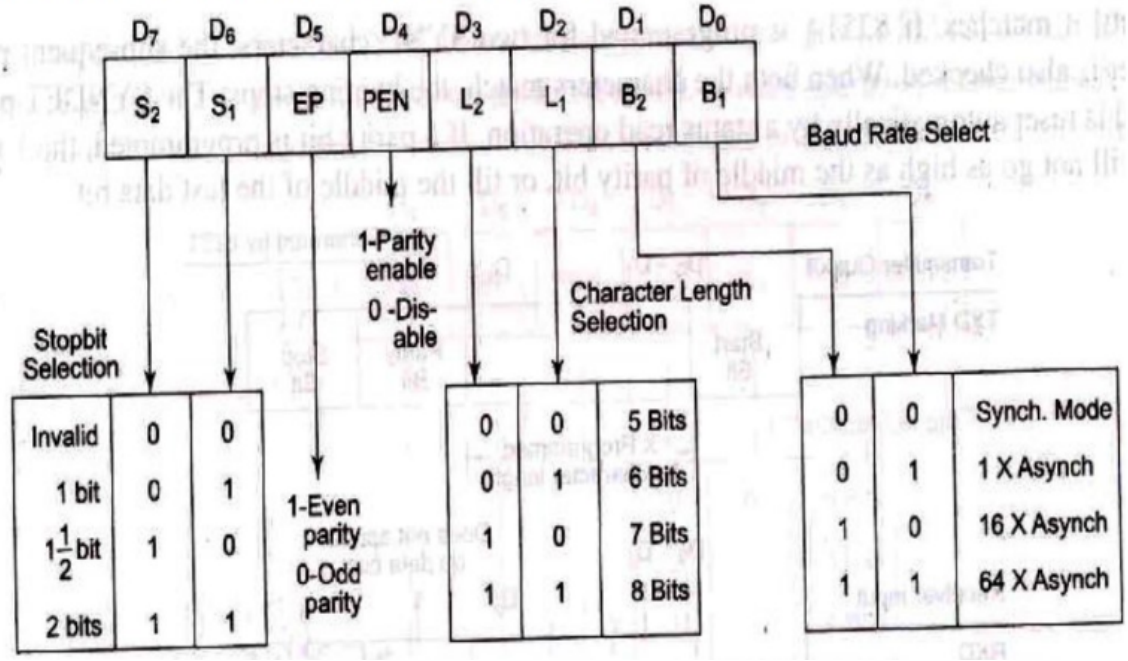


Fig. Mode instruction format-Async.

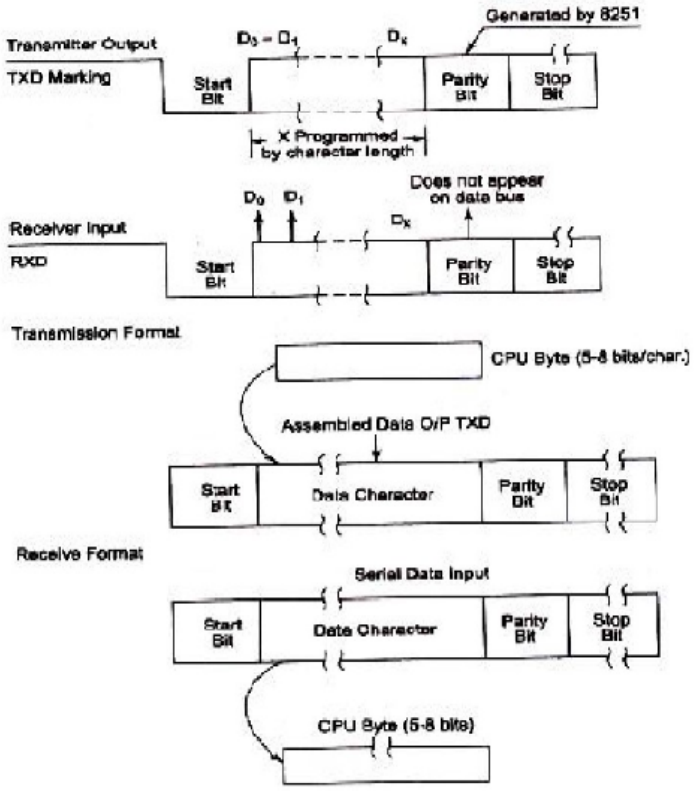


Fig. Communication format

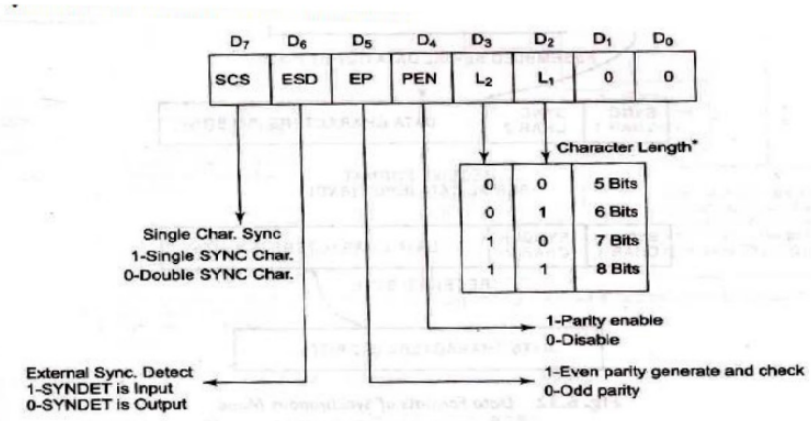


Fig. Synchronous mode Instruction format

Synchronous Mode (Transmission)

The TXD output is high until the CPU sends a character to 8251 which usually is a SYNC character. When CTS line goes low, the first character is serially transmitted out. Charactersare shifted out on the falling edge of TXC .Data is shifted out at the same rate as TXC , over TXD output line. If the CPU buffer becomes empty, the SYNC character or characters are inserted in the data stream over TXD output.

Synchronous Mode (Receiver)

In this mode, the character synchronization can be achieved internally or externally. The data on RXD pin is sampled on rising edge of the RXC. The content of the receiver buffer is compared with the first SYNC character at every edge until it matches. If 8251 is programmed for two SYNC characters, the subsequent received character is also checked. When the characters match, the hunting stops. The SYNDET pin set high and is reset automatically by a status read operation. In the external SYNC mode, the synchronization is achieved by applying a high level on the SYNDET input pin that forces 8251 out of HUNTmode. The high level can be removed after one RXC cycle. The parity and overrun error both are checked in the same way as in asynchronous mode.

Synchronous mode Transmit and Receive data format

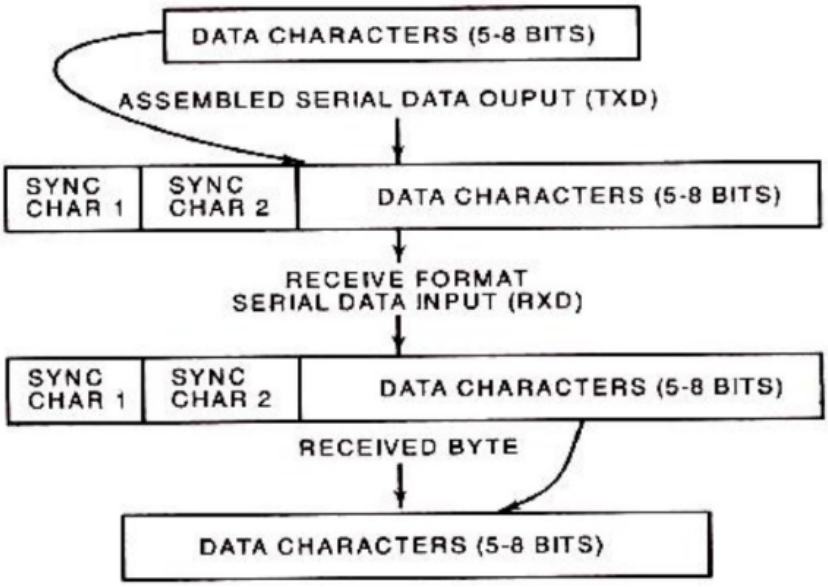


Fig.Data Formats of Synchronous Mode

Command Instruction Definition

The command instruction controls the actual operations of the selected format like enable transmit/receive, error reset and modem control. A reset operation returns 8251 back to mode instruction format.

Status Read Definition

This definition is used by the CPU to read the status of the active 8251 to confirm if any error condition or other conditions like the requirement of processor service has been detected during the operation.

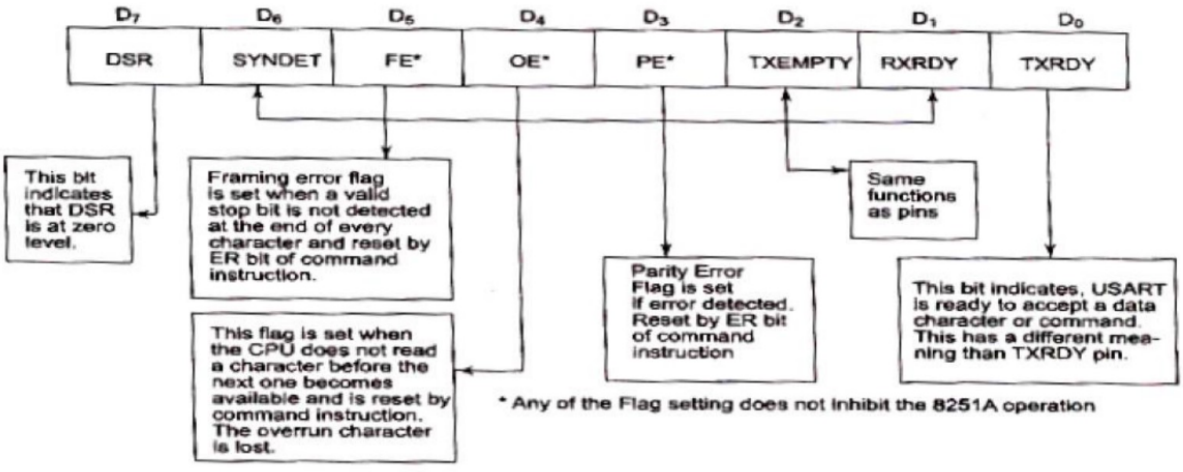


Fig.Status information

D/A and A/D Interface:

The function of an A/D converter is to produce a digital word which represents the magnitude of some analog voltage or current.

The specifications for an A/D converter are very similar to those for D/A converter:

1. The resolution of an A/D converter refers to the number of bits in the output binaryword. An 8-bit converter for example has a resolution of 1 part in 256.
2. Accuracy and linearity specifications have the same meaning for an A/D converter as they do for a D/A converter.
3. Another important specification for an ADC is its conversion time. - the time it takes the converter to produce a valid output binary code for an applied input voltage.When we refer to a converter as high speed, it has a short conversion time.

The analog to digital converter is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analog to digital data conversation process.

The start of conversion signal is a pulse of a specific duration. The process of analogto digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion (EOC) signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC.

These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports. The time taken by the ADC from the active edge of SOC pulse (the edge at which the conversion process actually starts) till the active edge of

EOC signal is called as the conversion delay of the ADC- the time taken by the converter to calculate the equivalent digital data output from the instant of the start of conversion is called conversion delay. It may range anywhere from a few microseconds in case of fast

ADCs to even a few hundred milliseconds in case of slow ADCs. A number of ADCs are available in the market, the selection of ADC for a particular application is done, keeping in mind the required speed, resolution range of operation, power supply requirements, sample and hold device requirements and the cost factors are considered.

The available ADCs in the market use different conversion techniques for the conversion of analog signals to digital signals.

- Parallel converter or flash converter,
- Successive approximation and
- dual slope integration

A general algorithm for ADC interfacing contains the following steps.

1. Ensure the stability of analog input, applied to the ADC.

2. Issue start of conversion (SOC) pulse to ADC.

3. Read end of conversion (EOC) signal to mark the end of conversion process.

4. Read digital data output of the ADC as equivalent digital output.

It may be noted that analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of 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 a specified 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. 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.

ADC 0808/0809

The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. Successive approximation technique is one of the fast techniques for analog to digital conversion.

The conversion delay is 100 μs at a clock frequency of 640 kHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.

These converters internally have a 3:8 analog multiplexer so that at a time eight different analog inputs can be connected to the chips. Out of these eight inputs only one can be selected for conversion by using address lines ADD A, ADD B and ADD C, as shown. Using these address inputs, multichannel data acquisition systems can be designed using a single ADC.

The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hard wired to select the proper input.

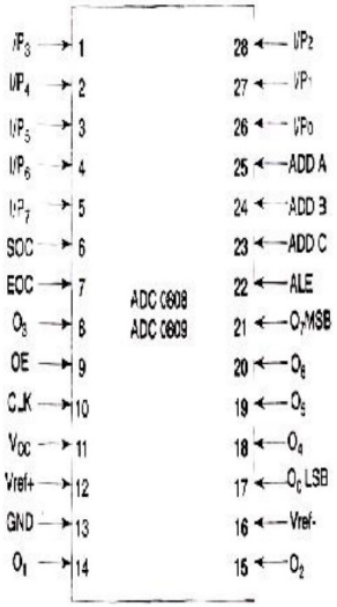


Fig. Pins of ADC0808/09

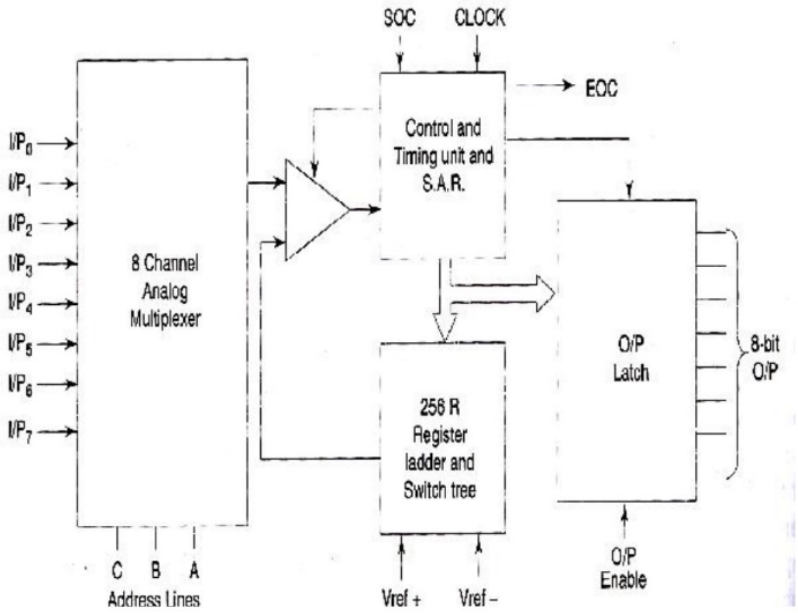
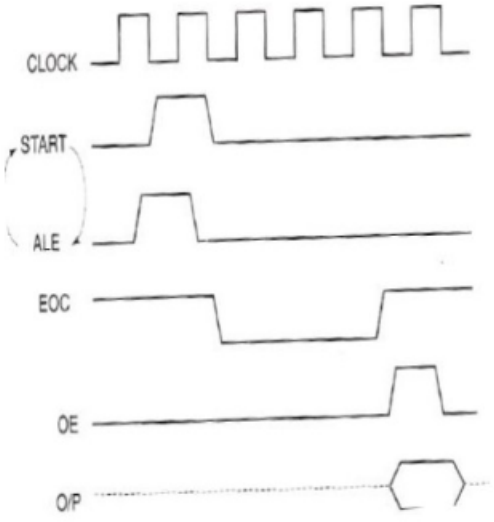


Fig. Block Diagram of ADC 0808/0809

only positive analog input voltages to their digital equivalents. These chips do not containany internal sample and hold circuit. If one needs a sample and hold circuit for theconversion of fast, signals into equivalent digital quantities, it has to be externally connected at each of the analog inputs.



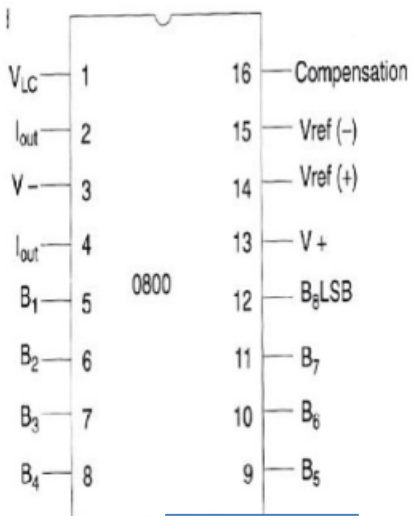


Fig. Timing Diagram of ADC 0808/0809 Fig. Pins of DAC 0800

INTERFACING DIGITAL TO ANALOG ONVERTERS:

The digital to analog converters convert binary numbers into their analog equivalent voltages or currents. Several techniques are employed for digital to analog conversion.

i. Weighted resistor network

ii. R-2R ladder network

iii. Current output D/A converter

Applications in areas like

1. Digitally controlled gains, motor speed control, programmable gain amplifiers,digital voltmeters, panel meters, etc.
2. In a compact disk audio player for example a 14-or16-bit D/A converter is used to convert the binary data read off the disk by a laser to an analog audio signal.
3. Most speech synthesizer integrated circuits contain a D/A converter to convert storedbinary data words into analog audio signals.

Characteristics:

1. Resolution: It is a change in analog output for one LSB change in digital input. It is given by (1/2n)*Vref. If n=8 (i.e.8-bit DAC)1/256*5V=39.06mV
2. Settling time: It is the time required for the DAC to settle for a full scale code change.

DAC 0800 8-bit Digital to Analog converter Features:

i. DAC0800 is a monolithic 8-bit DAC manufactured by National semiconductor.

ii. It has settling time around 100ms

iii. It can operate on a range of power supply voltage i.e. from 4.5V to +18V. Usually the supply V+ is 5V or +12V. The V- pin can be kept at a minimum of -12V.

iv. Resolution of the DAC is 39.06mV

Programmable timer device 8253

Intel's programmable counter/timer device (8253) facilitates the generation of accurate time delays. When 8253 is used as timing and delay generation peripheral, the microprocessor becomes free from the tasks related to the counting process and executes the programs in memory, while the timer device may perform the counting tasks. This minimizes the software overhead on the microprocessor.

Architecture and Signal Descriptions

- The programmable timer device 8253 contains three independent 16-bit counters, each with a maximum count rate of 2.6 MHz to generate three totally independent delays or maintain three independent counters simultaneously. All the three counters may be independently controlled by programming the three internal command word registers.

- The 8-bit, bidirectional data buffer interfaces internal circuit of8253 to microprocessor systems bus. Data is transmitted or received by the buffer upon the execution of IN or OUT instruction. The read/write logic controls the direction of the data buffer depending upon whether it is a read or a write operation. It may be noted that IN instruction reads data while OUT instruction writes data to a peripheral.

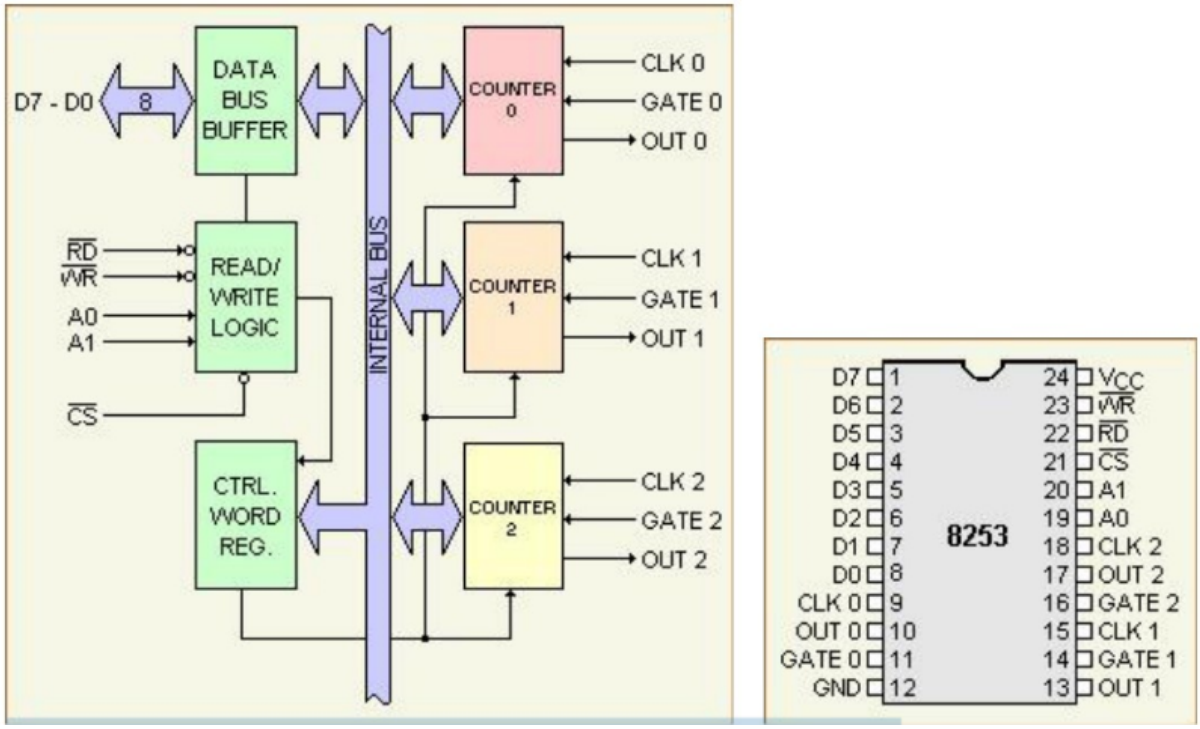


Fig.Internal blocks of 8253 and pin diagram

- The three counters all 16-bit presettable, down counters, able to operate either in BCD or in hexadecimal mode. The mode control word register contains the information that can be used for writing or reading the count value into or from the respective count register using the OUT and IN instructions. The specialty of the 8253 counters is that they can be easily read on line without disturbing the clock input to the counter. This facility is called as "on the fly" reading of counters, and is invoked using a mode control word.
- A0, Al pins are the address input pins and are required internally for addressing the mode control word registers and the three counter registers. A low on CS line enables the 8253. No operation will be performed by 8253 till it is enabled.

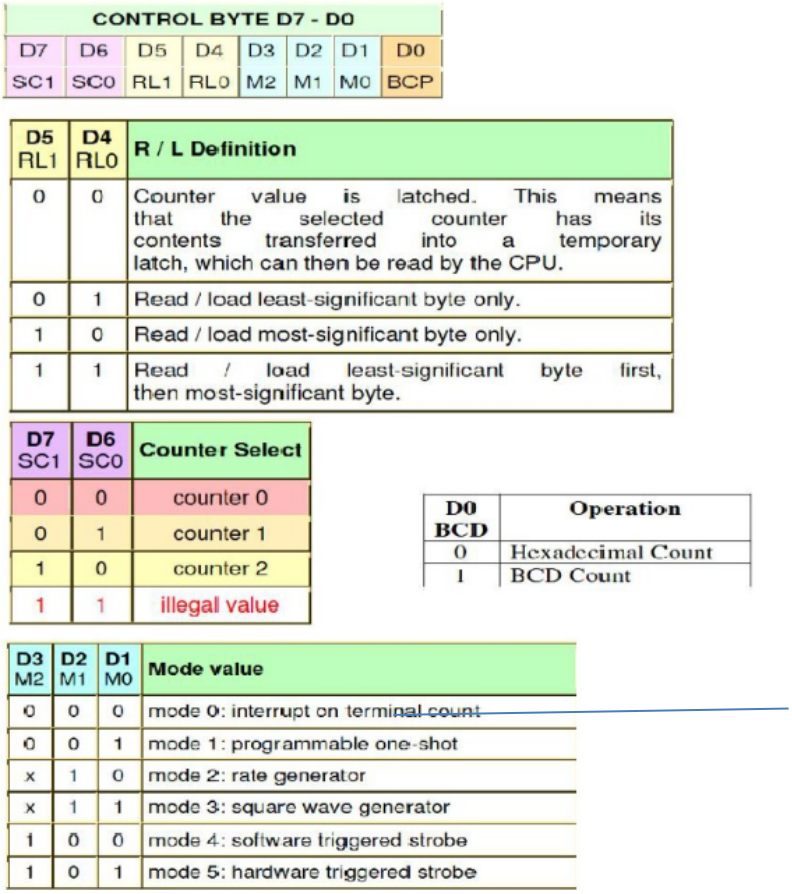
Table .Selected operations for various Control

CS	RD	WR	A_1	A_0	Selected Operations
0	1	0	0	0	Write Counter 0
0	1	0	0	1	Write Counter 1
0	1	0	1	0	Write Counter 2
0	1	0	1	1	Write control Word
0	0	1	0	0	Read Counter 0
0	0	1	0	1	Read Counter 1
0	0	1	1	0	Read Counter 2
0	0	1	1	1	No Operation
0	1	1	X	X	No Operation
1	X	X	X	X	Disabled

- A control word register accepts the 8-bit control word written by the microprocessor andstores it for controlling the complete operation of the specific counter. It may be noted that,the control word register can only be written and cannot be read as it is obvious from Table.The CLK, GATE and OUT pins are available for each of the three timer channels. Theirfunctions will be clear when we study the different operating modes of 8253.

Control Word Register

The 8253 can operate in anyone of the six different modes. A control word must be written in the respective control word register by the microprocessor to initialize each of the counters of 8253 to decide its operating mode. All the counters can operate in anyone of the modes or they may be even in different modes of operation, at a time. The control word format is presented, along with the definition of each bit, while writing a count in the counter, it should be noted that, the count is written in the counter only after thedata is put on the data bus and a falling edge appears at the clock pin of the peripheral thereafter. Any reading operation of the counter, before the falling edge appears may result in garbage data.



CONTROL BYTE D7 - D0							
D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCP

D5 RL1	D4 RL0	R / L Definition
0	0	Counter value is latched. This means that the selected counter has its contents transferred into a temporary latch, which can then be read by the CPU.
0	1	Read / load least-significant byte only.
1	0	Read / load most-significant byte only.
1	1	Read / load least-significant byte first, then most-significant byte.

D7 SC1	D6 SC0	Counter Select
0	0	counter 0
0	1	counter 1
1	0	counter 2
1	1	illegal value

D0 BCD	Operation
0	Hexadecimal Count
1	BCD Count

D3 M2	D2 M1	D1 M0	Mode value
0	0	0	mode 0: interrupt on terminal count
0	0	1	mode 1: programmable one-shot
x	1	0	mode 2: rate generator
x	1	1	mode 3: square wave generator
1	0	0	mode 4: software triggered strobe
1	0	1	mode 5: hardware triggered strobe

Fig. Control word format and bit definition

MODE 0 this mode of operation is called as interrupt on terminal count. In thismode, the output is initially low after the mode is set. The output remains low even after thecount value is loaded in the counter.

The counter starts decrementing the count value afterthe falling edge of the clock, if the GATE input is high. The process of decrementing thecounter continues at each falling edge of the clock till the terminal count is reached, i.e. thecount becomes zero. When the terminal count is reached, the output goes high and remainshigh till the selected control word register or the corresponding count register is reloadedwith a new mode of operation or a new count, respectively.

This high output may be used to interrupt the processor whenever required, bysetting suitable terminal count. Writing a count register while the previous counting is inprocess, generates the following sequence of response.

The first byte of the new count when loaded in the count register, stops the previouscount. The second byte when written, starts the new count, terminating the previous countthen and there. The GATE signal is active high and should be high for normal counting.When GATE goes low counting is terminated and the current count is latched till the GATEagain goes high.

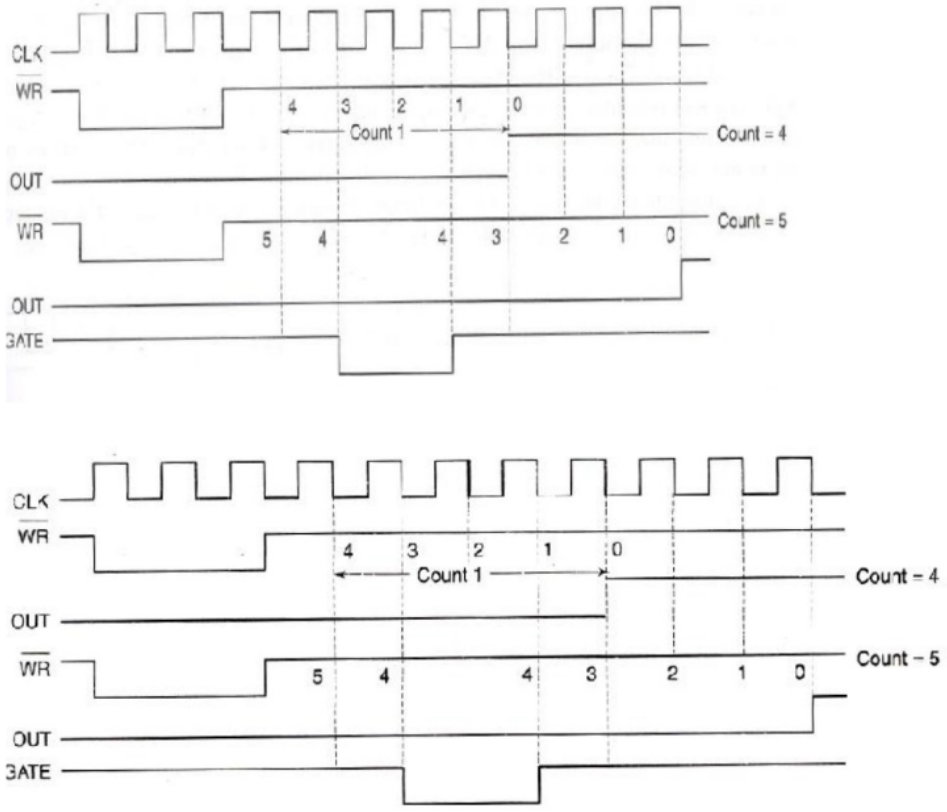


Fig. Waveforms WR, OUT and GATE in Mode 0

MODE 1 This mode of operation of 8253 is called as programmable one-shot mode.the 8253 can be used as a monostable multivibrator. The duration of the quasistable state ofthe monstable multivibrator is decided by the count loaded in the count register.The gate input is used as trigger input in this mode of operation.

Normally theoutput remains high till the suitable count is loaded in the count register and a trigger isapplied. After the application of a trigger (on the positive edge), the output goes low andremains low till the count becomes zero. If another count is loaded when the output isalready low, it does not disturb the previous count till a new trigger pulse is applied at theGATE input. The new counting starts after the new trigger pulse.

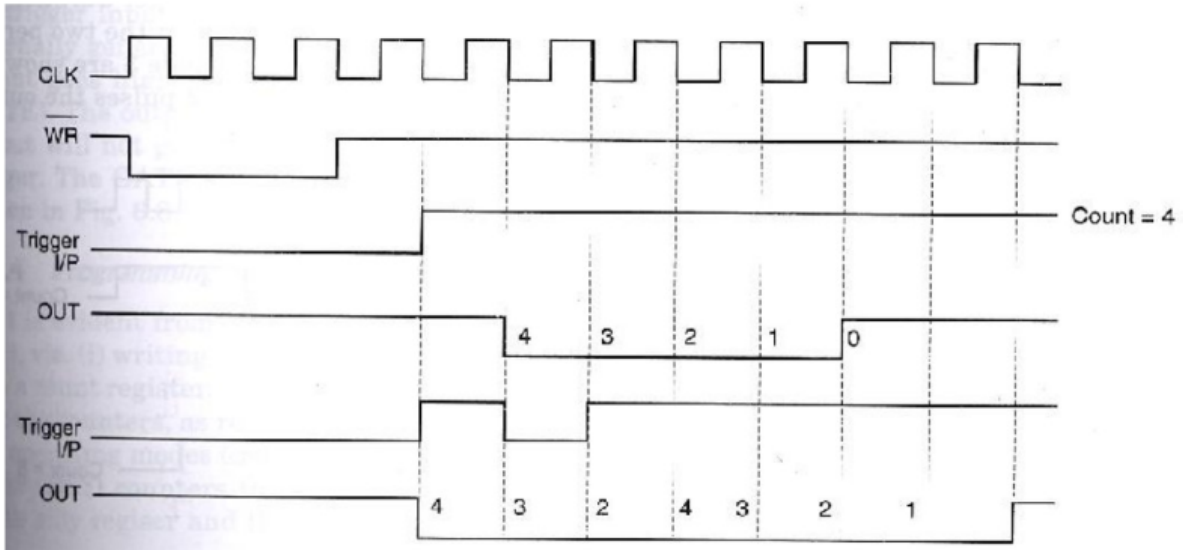


Fig. WR, GATE and OUT Waveforms in Mode 1

MODE 2

This mode is called either rate generator or divides by N counter. In thismode, if N is loaded as the count value, then, after N pulses, the output becomes low onlyfor one clock cycle. The count N is reloaded and again the output becomes high and remainshigh for N clock pulses. The output is normally high after initialisation or even a low signal on GATE input can force the output high. If GATE goes high, the counter starts counting down from the initial value.

The counter generates an active low pulse at the output initially, after the count register is loaded with a count value. Then count down starts and whenever the count becomes zero another active low pulse is generated at the output. The duration of these active low pulses are equal to one clock cycle. The number of input clock pulses between the two low pulses at the output is equal to the count loaded.

Figure shows the related waveforms for mode 2. Interestingly, the counting is inhibited when GATE becomes low.

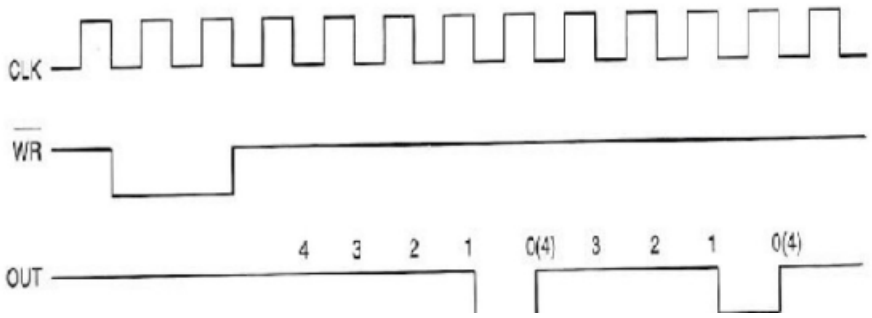


Fig.Waveforms at pin WR and OUT in Mode 2

MODE 3 In this mode, the 8253 can be used as a square wave rate generator. In terms of operation this mode is somewhat similar to mode 2. When, the count N loaded is

even, then for half of the count, the output remains high and for the remaining half it remains low0020.If the count loaded is odd, the first clock pulse decrements it by 1 resifting in an even count value (holding the output high). Then the output remains high for half of the new count and goes low for the remaining half. This procedure is repeated continuously resulting in the generation of a square wave.

In case of odd count, the output is high for longer duration and low for shorter duration. The difference of one clock cycle duration between the two periods is due to the initial decrementing of the odd count. The waveforms for mode 3 are shown in Fig. if theloaded count value 'N is odd, then for (N+l)/2 pulses the output remains high and for (N-l)/2pulses it remains low.

MODE 4 This mode of operation of 8253 is named as software triggered strobe.After the mode is set, the output goes high. When a count is loaded, counting down starts.On terminal count, the output goes low for one clock cycle, and then it again goes high. Thislow pulse can be used as a strobe, while interfacing the microprocessor with otherperipherals.

The count is inhibited and the count value is latched, when the GATE signal goeslow. If a new count is loaded in the count registers while the previous counting is in the nextclock cycle. The counting then proceeds according to the new count.

MODE 5 This mode of operation also generates a strobe in response to the rising edge atthe trigger input. This mode may be used to generate a delayed strobe in response to anexternally generated signal. Once this mode is programmed and the counter is loaded, theoutput goes high.

The counter starts counting after the rising edge of the trigger input (GATE). Theoutput goes low for one clock period, when the terminal count is reached. The output willnot go low until the counter content becomes zero after the rising edge of any trigger. TheGATE input in this mode is used as trigger input. The related waveforms are shown in Fig.

Programming and Interfacing 8253

There may be two types of write operations in 8253, viz.

(i) Writing a control word into a control word register and

(ii) Writing a count value into a count register.

The control word register accepts data from the data buffer and initializes the counters, asrequired. The control word register contents are used for (a) initializing the operating modes(mode0-mode4) (b) selection of counters (counter0-counter2) (c) choosing binary BCDcounters (d) loading of the counter registers.The mode control register is a write only register and the CPU cannot read its contents. Onecan directly write the mode control word for counter 2 or counter 1 prior to writing thecontrol word for counter0. Mode control word register has a separate address, so that it canbe written independently. A count register must be loaded with the count value with samebyte sequence that was programmed in the mode control word of that counter, using the bits RL0 and RL1.The loading of the count registers of different counters is again sequence independent. One can directly write the 16-bit count register for count 2 before writing count 0 and count 1, but the two bytes in a count must be written in the byte sequence programmed using RL0 and RL1 bits of the mode control word of the counter. All the counters in 8253 are down counters, hence their count values go on decrementing if the CLK input pin is applied with a valid clock signal. A maximum count is obtained by loading all zeros into a count register, i.e. 216 for binary counting and 104 for BCD counting. The 8253 responds to the negative clock edge of the clock input.

The maximum operating clock frequency of 8253 is 2.6 MHz For higher frequencies one can use timer 8254, which operates up to 10 MHz, maintaining pin compatibility with 8253. The following Table 6.2 shows the selection of different mode control words and counter register bytes depending upon address lines Ao and A1 In 8253, the 16-bit contents of the counter can simply be read using successive 8-bit IN operations. As stated earlier, the mode control register cannot be read for any of the counters. There are two methods for reading 8253 counter registers.

In the **first method,** either the clock or the counting procedure (using GATE) is inhibited toensure a stable count. Then the contents are read by selecting the suitable counterusing A0,Al and executing using IN instructions. The first IN instruction reads the least significant byte and the second IN instruction reads the most significant byte. Internal logic of 8253 is designed in such a way that the programmer has to complete the reading operation as programmed by him, using RL0 and RLl bits of control word.

In the **second method** of reading a counter, the counter can be read while counting is in progress. This method, as already mentioned is called as reading on fly. In this method, neither clock nor the counting needs to be inhibited to read the counter. The content of a counter can be read 'on fly' using a newly defined control word register format for online reading of the count

register. Writing a suitable control word, in the mode control register internally latches the contents of the counter. The control word format for 'read on fly' mode is given in Fig. 1.9 along with its bit definitions. After latching the content of a counter using this method, the programmer can read it using IN instructions, as discussed before.

Programmable Keyboard/Display Controller

Intel's 8279 is a general purpose Keyboard Display controller that simultaneously drives the display of a system and interfaces a Keyboard with the CPU. The Keyboard Display interface scans the Keyboard to identify if any key has been pressed and sends the code of the pressed key to the CPU. It also transmits the data received from the CPU, to the display device.

Both of these functions are performed by the controller in repetitive fashion without involving the CPU. The Keyboard is interfaced either in the interrupt or the polled mode. In the interrupt mode, the processor is requested service only if any key is pressed, otherwise the CPU can proceed with its main task.

In the polled mode, the CPU periodically reads an internal flag of 8279 to check for a key pressure. The Keyboard section can interface an array of a maximum of 64 keys with the CPU. The Keyboard entries (key codes) are debounced and stored in an 8-byte FIFORAM,that is further accessed by the CPU to read the key codes. If more than eight characters are entered in the FIFO (i.e. more that eight keys are pressed), before any FIFO read operation, the overrun status is set. If a FIFO contains a valid key entry, the CPU is interrupted (in interrupt mode) or the CPU checks the status (in polling) to read the entry. Once the CPU reads a key entry, the FIFO is updated, i.e. the key entry is pushed out of the FIFO to generate space for new entries. The 8279 normally provides a maximum of sixteen 7-seg display interface with CPU It contains a 16-byte display RAM that can be used either as an integrated block of 16x8-bits or two 16x4-bit block of RAM. The data entry to RAM block is controlled by CPU using the command words of the 8279.

- Architecture and Signal Descriptions of 8279

The Keyboard display controller chip 8279 provides

1. A set of four scan lines and eight return lines for interfacing keyboards.

2. A set of eight output lines for interfacing display.

- I/O Control and Data Buffer

The I/O control section controls the flow of data to/from the 8279. The data buffer interface the external bus of the system with internal bus of 8279 the I/O section is enabled only if Dis low.

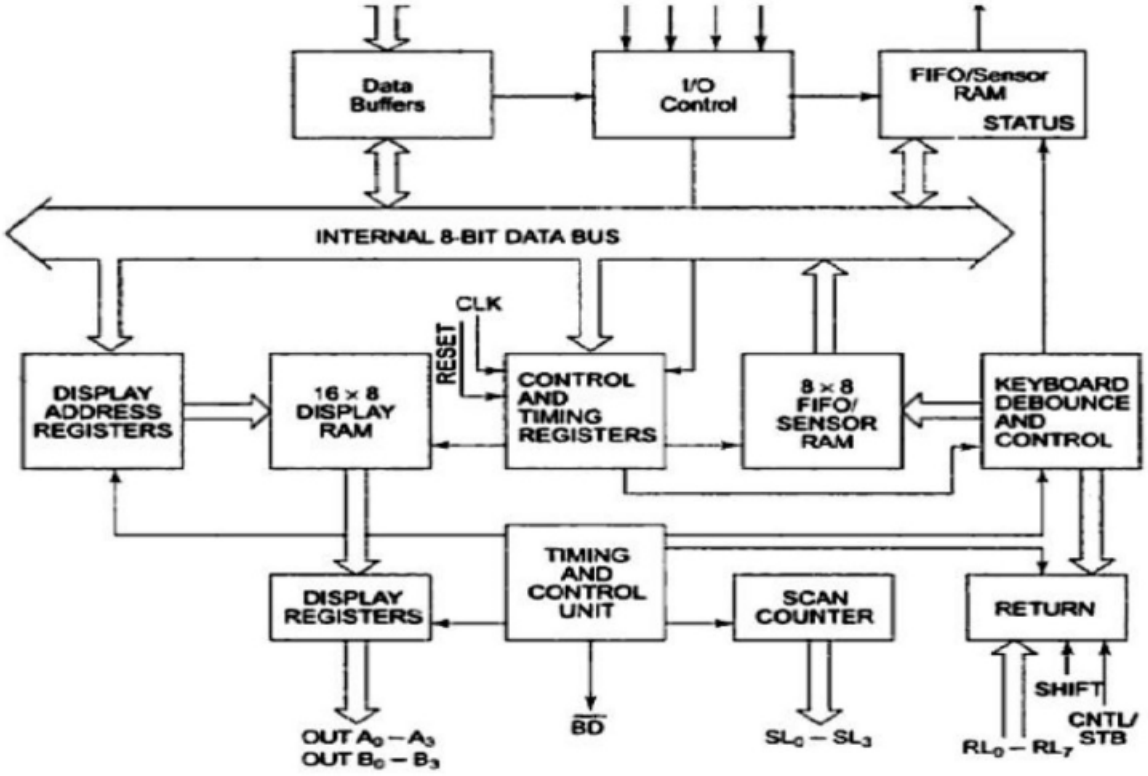


Fig. Internal blocks of Keyboard display controller

The pin Ao, RD and WR select the command, status or data read/write operations carried out by the CPU with 8279.

- Control and Timing Register and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by CPU. The registers are written with Ao=1 and WR =0. The timing and control unit controls the basic timings for the operation of the circuit. Scan Counter divide down the operating frequency of 8279 to derive scan keyboard and scan display frequencies.

- Scan Counter

The Scan Counter has two modes to scan the key matrix and refresh the display. In the Encoded mode, the counter provides a binary count that is to be externally decoded to provide the scan lines for keyboard and display (four externally decoded scan lines may drive up to 16 displays). In the decoded scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL0-SL3 (four internally decoded scan lines may drive up to 4 Displays). The Keyboard and Display both are in the same mode at a time.

Return Buffers and Keyboard Debounce and Control

This section scans for a Key closure row-wise. If it is detected, the Keyboard debounce unit debounces the key entry (i.e. wait for 10 ms). After the debounce period, if the key continues to be detected. The code of the Key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.

FIFO/Sensor RAM and Status Logic

In Keyboard or strobed input mode, this block acts as 8-byte first-in-first-out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry, and in the meantime, read by the CPU, till the RAM becomes empty. The status logic generates an interrupt request after each FIFO read operation till the FIFO is empty.

In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.

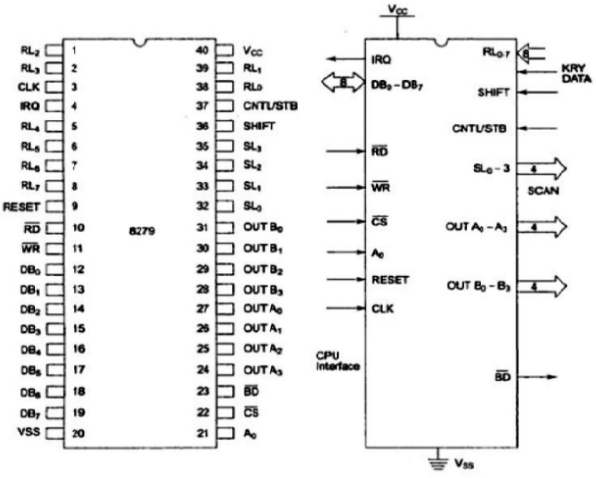
Display Address Registers and Display RAM.

The Display address registers hold the addresses of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU. The 16-byte display RAM contains the 16-byte of data to be displayed on the sixteen 7-seg displays in the encoded scan mode.

- Pin Diagram of 8279

DB0 - DB7: These are bidirectional data bus lines. The data and command words to and from the CPU are transferred on these lines.

CLK: This is a clock input used to generate internal timings required by 8279.



I

Fig. Pins and signals of keyboard controller

RESET: This pin is used to reset 8279. A high on this line resets 8279. After resetting 8279, its in sixteen 8-bit displays, left entry encoded scan, 2-key lock out mode. The clock prescaler isset to 31.

CS chip select: A low on this line enables 8279 for normal read or write operations. Otherwise this pinshould be high.

Ao: A high on the Ao line indicates the transfer of a command or status information. A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.

RD, WR: (Input/Output) READ/WRITE input pins enable the data buffer to receive or send data overthe data bus.

IRQ: This interrupt output line goes high when there is data in the FIFO sensor RAM. Theinterrupt line goes low with each FIFO RAM read operation. However, if the FIFO RAMfurther contains any Key-code entry to be read by the CPU, this pin again goes high togenerate an interrupt to the CPU.

Vss, Vcc: These are the ground and power supply lines for the circuit.

SL0-SL3 – Scan Lines: These lines are used to scan the keyboard matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.

RL0-RL7 – Return Lines: These are the input lines which are connected to one terminal of keys, while the other terminal of the keys is connected to the decoded scan lines. These are normally high, but pulled low when a key is pressed.

SHIFT: The status of the Shift input line is stored along with each key code in FIFO in the scanned keyboard mode. Till it is pulled low with a key closure it is pulled up internally to keep it high.

CNTL/STB-CONTROL/STROBED I/P Mode: In the Keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line is a strobe line that enters the data into FIFO RAM, in the strobed input mode. It has an internal pull up. The line is pulled down with a Key closure.

BD – Blank Display: This output pin is used to blank the display during digit switching or by a blanking command.

OUTA0 – OUTA3 and OUTB0 – OUTB3: These are the output ports for two 16x4 (or one 16 x 8) internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display andkeyboard. The two 4-bit ports may also be used as one 8-bit port.

Modes of Operation of 8279

The Modes of operation of 8279 are

i. Input (Keyboard) modes

ii. Output (Display) modes

Input (Keyboard) modes:

8279 provides three input modes, they are:

1. Scanned Keyboard Mode:

This mode allows a key matrix to be interfaced using either encoded or decoded scans. In the encoded scan, an 8 x 8 keyboard or in decoded scan, a 4 x 8 Keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.

2. Scanned Sensor Matrix:

In this mode, a sensor array can be interfaced with 8279 using either encoder or decoder scans. With encoder scan 8 x 8 sensor matrix or with decoder scan 4 x 8 sensor matrix can be interfaced. The sensor codes are stored in the CPU addressable sensor RAM.

3. Strobed Input:

In this mode, if the control line goes low, the data on return lines is stored in the FIFO byte by byte.

Output (Display) Modes:

8279 provides two output modes for selecting the display options.

1. Display Scan:

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

2. Display Entry:

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

Details of Modes of Operation

Keyboard Modes

1**. Scanned Keyboard Mode with 2 Key Lockout**

In this mode of operation, when a key is pressed, debounce logic comes into operation. The Key code of the identified key is entered into the FIFO with SHIFT and CNTL status, provided the FIFO is not full.

2. Scanned Keyboard with N-key Rollover

In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboard scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM. Any number of keys can be pressed simultaneously and recognized in the order, the Keyboard scan record them.

3. Scanned Keyboard Special Error Mode

This mode is valid only under the N-Key rollover mode. This mode is programmed usingend interrupt/error mode set command. If during a single debounce period (two Keyboard scan) two keys are found pressed, this is considered a simultaneous depression and an error flag is set. This flag, if set, prevents further writing in FIFO but allows generation of further interrupts to the CPU for FIFO read.

4. Sensor Matrix Mode

In the Sensor Matrix mode, the debounce logic is inhibited the 8-byte memory matrix. The status of the sensor switch matrix is fed directly to sensor RAM matrix Thus the sensor RAM bits contains the row-wise and column-wise status of the sensors in the sensor matrix.8

Display Modes

There are various options of data display the first one is known as left entry mode or type writer mode. Since in a type writer the first character typed appears at the left-most position, while the

subsequent characters appears successively to the right of the first one. The other display format is known as right entry mode, or calculator mode, since the calculator the first character entered appears to the right-most position and this character is shifted one position left when the next character is entered.

1. Left Entry Mode

In the Left entry mode, the data is entered from the left side of the display unit. Address 0 of the display RAM contains the leftmost display character and address 15 of the RAM contains the rightmost display character.

2. Right Entry Mode In the right entry mode, the first entry to be displayed is entered on the rightmost display. The next entry is also placed in the right most display but after the previous display is shifted left by one display position.

Command Words of 8279

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

a. Keyboard Display mode set

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

b. Programmable Clock

The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler. PPPPP is a 5-bit binary constant. The input frequency is divided by a decimal constant ranging from 2 to 31, decided by the bits of an internal prescalar, PPPPP.

c. Read FIFO/Sensor RAM

The format of this command is given as shown below

X - Don’t care

AI - Auto increment flag

AAA - Address pointer to 8 bit FIFO RAM

This word is written to set up 8279 for reading FIFO/Sensor RAM. In scanned keyboard mode, AI and AAA bits are of no use. The 8279 will automatically drive data bus for each subsequent read, in the same sequence, in which the data was entered.

d. Read Display RAM

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

e. Write Display RAM

The format of this command is given as shown below

AI - Auto increment flag

AAAA - 4-bit address for 16-bit display RAM to be written

Other details of this command are similar to the ‘Read Display RAM Command.

f. Display Write Inhibit/Blanking

The IW (Inhibit write flag) bits are used to mask the individual nibble Here Do and D2corresponds to OUTBo – OUTB3 while D1 and D3 corresponds to OUTAo-OUTA3 forblanking and masking respectively.

g. Clear Display RAM

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

CD CD1 CDo

1 0 x All Zeros (x don’t care) AB = 00

1 1 0 A3-Ao = 2(0010) and B3-Bo = 00(0000)

1 1 1 All ones (AB = FF), i.e. clear RAM

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

End Interrupt/Error Mode Set

For the sensor matrix mode, this command lowers the IRQ line and enables further writinginto the RAM. Otherwise, if a charge in sensor value is detected, IRQ goes high that inhibitswriting in the sensor RAM.

Key-code and status Data Formats

This briefly describes the formats of the Key-code/Sensor data in their respective modes ofoperation and the FIFO Status Word formats of 8279.

Key-code Data Formats:

After a valid Key closure, the key code is entered as a byte code into the FIFO RAM, in thefollowing format, in scanned keyboard mode. The Keycode format contains 3-bit contentsof the internal row counter, 3-bit contents of the column counter and status of the SHIFTand CNTL Keys The data format of the Keycode in scanned keyboard mode is given below.

In the sensor matrix mode, the data from the return lines is directly entered into anappropriate row of sensor RAM, that identifies the row of the sensor that changes its status.

The SHIFT and CNTL Keys are ignored in this mode. RL bits represent the return lines.Rn represents the sensor RAM row number that is equal to the row number of the sensorarray in which the status change was detected. Data Format of the sensor code in sensormatrix mode

FIFO Status Word:

The FIFO status word is used in keyboard and strobed input mode to indicate the error. Overrun error occurs, when an already full FIFO is attempted an entry, Under run erroroccurs when an empty FIFO read is attempted. FIFO status word also has a bit to show theunavailability of FIFO RAM because of the ongoing clearing operation.

In sensor matrix mode, a bit is reserved to show that at least one sensor closure indication isstored in the RAM, The S/E bit shows the simultaneous multiple closure error in specialerror mode. In sensor matrix mode, a bit is reserved to show that at least one sensor closureindication is stored in the RAM, The S/E bit shows the simultaneous multiple closure errorin special error mode.

Interfacing and Programming 8279

Problem:

Interface keyboard and display controller 8279 with 8086 at address 0080H. Write anALP to set up 8279 in scanned keyboard mode with encoded scan, N-Key rollover mode.

Use a 16 character display in right entry display format. Then clear the display RAM withzeros. Read the FIFO for key closure. If any key is closed, store its code to register CL.

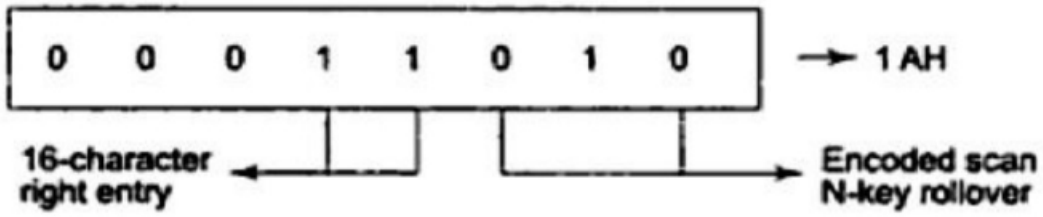
Then write the byte 55 to all the displays, and return to DOS. The clock input to 8279 is2MHz, operate it at 100 KHz.

Solution:

- The 8279 is interfaced with lower byte of the data bus, i.e. Do-D7. Hence the Ao input of 8279 is connected with address lineA1.
- The data register of 8279 is to be addressed as 0080H, i.e. Ao=0.
- For addressing the command or status word Ao input of 8279 should be 1.
- The next step is to write all the required command words for this problem.

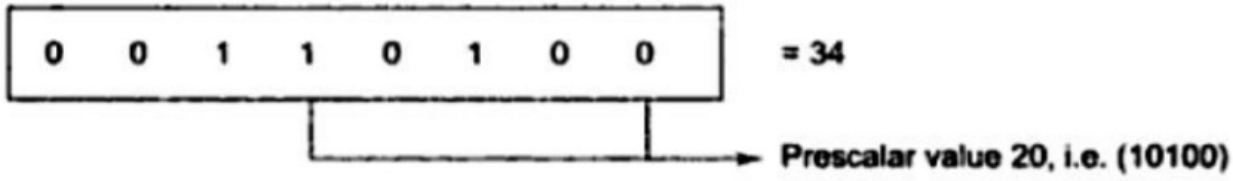
Keyboard/Display Mode Set CW:

This command byte sets the 8279 in 16-character right entry and encoded scan N-Key rollover mode**.**



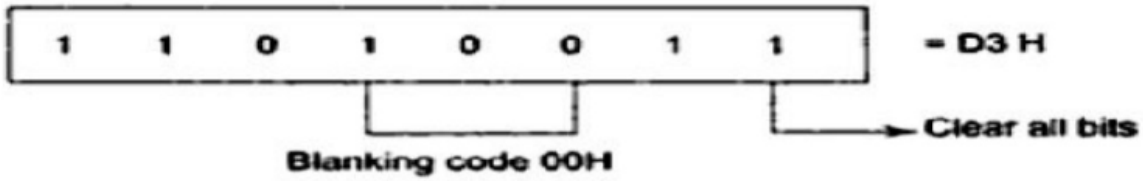
Program clock selection:

The clock input to 8279 is 2MHz, but the operating frequency is to be 100KHz, i.e. the clock input is to be divided by 20 (10100). Thus the prescalar value is 10100 and thecommand byte is set as given.



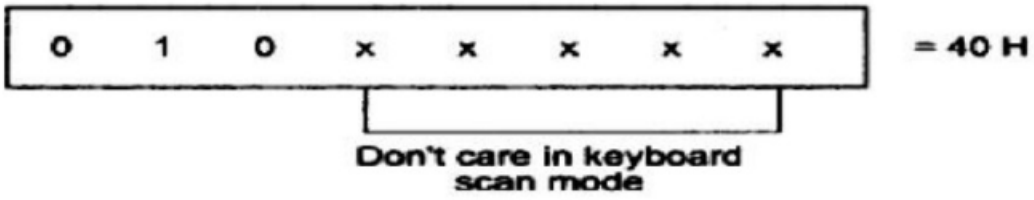
Clear Display RAM:

This command clears the display RAM with the programmable blanking code.



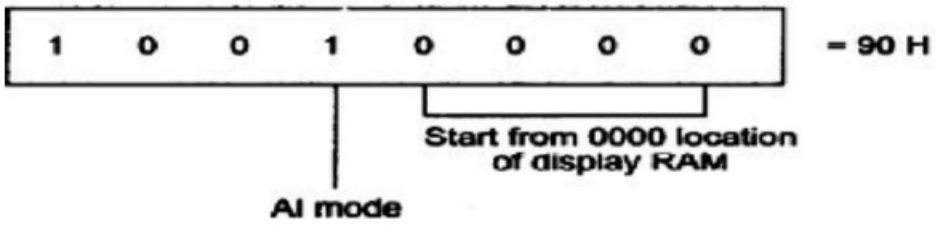
Read FIFO:

This command byte enables the programmer to read a key code from the FIFO RAM.



Write Display RAM:

This command enables the programmer to write the addressed display locations of the RAM as presented below.



<u>Interrupt Controller</u>

The Intel 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single a5V supply. Circuitry is static, requiring no clock input. The 8259A is designed to minimize the software and real time overhead in handling multi- level priority interrupts.

It has several modes, permitting optimization for a variety of system requirements. The 8259A is fully upward compatible with the Intel 8259. Software originally written for the8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered and Edge Triggered).

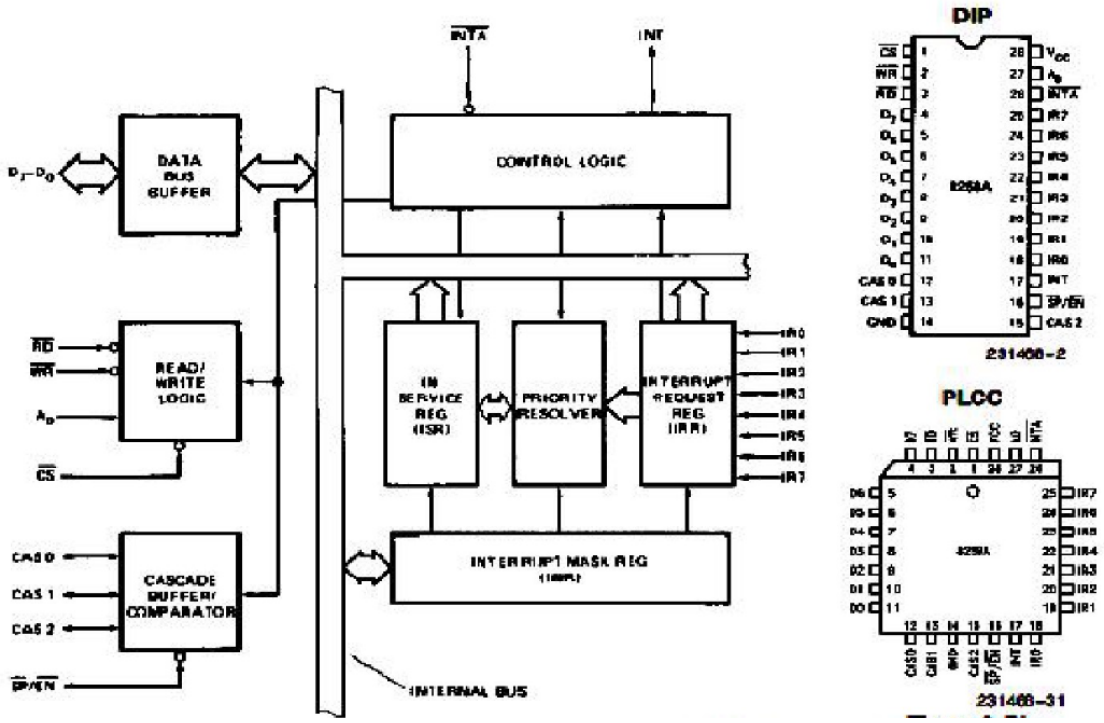


Fig .Functional block diagram of 8259

The microprocessor will be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off.This method is called Interrupt.

System throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness. The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

Each peripheral device or structure usually has a special program or ``routine" that is associated with its specific functional or operational requirements; this is referred to as a ``service routine". The PIC, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can ``point" the Program Counter to the service routine associated with the requesting device. This ``pointer" is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

- ✓ **Interrupt request register (IRR) AND in-service register (ISR):**

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt

- ✓ **Request Register (IRR) and the In-Service (ISR).**

The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

- ✓ **Priority resolver**

This logic block determines the priorities of the bits set in the IRR. The highest priority isselected and strobed into the corresponding bit of the ISR during INTA pulse. The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR.Masking of a higher priority input will not affect the interrupt request lines of lower quality.

- ✓ **INT (INTERRUPT)**

This output goes directly to the CPU interrupt input. The VOH level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

- ✓ **INTA (INTERRUPT ACKNOWLEDGE)**

INTA pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (mPM) of the 8259A.

- ✓ **Data bus buffer**

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

- ✓ **Read/write control logic**

The function of this block is to accept OUTput commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW)Registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

- ✓ **CS (CHIP SELECT)**

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless thedevice is selected.

- ✓ **WR (WRITE)**

A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the8259A.

- ✓ **RD (READ)**

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

- ✓ **A0**

This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tieddirectly to one of the address lines.

Interrupt sequence

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows:

1. One or more of the INTERRUPT REQUEST lines (IR7±0) are raised high, setting the corresponding IRR bit(s).

2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate.

3. The CPU acknowledges the INT and responds with an INTA pulse.

4. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7±0 pins.

5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.

6. These two INTA pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA pulse and the higher

8-bit address is released at the second INTA pulse.

7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence. The events occurring in an 8086 system are the same until step 4.

8. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the Data Bus during this cycle.

9. The 8086 will initiate a second INTA pulse. During this pulse, the 8259A releases an 8-bit pointer onto the Data Bus where it is read by the CPU.

10. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine. If no interrupt request is present at step 4 of either sequence (i.e., the request was too short in duration) the 8259A will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look like an interrupt level 7 was requested. When the 8259A PIC receives an interrupt, INT becomes active and an interrupt acknowledge cycle is started. If a higher priority interrupt occurs between the two INTA pulses, the INT line goes inactive immediately after the second INTA pulse. After an unspecified amount of time the INT line is activated again to signify the higher priority interrupt waiting for service.

DMA Controller -DMA Controller 8257

The Direct Memory Access or DMA mode of data transfer is the fastest amongst all the modes of data transfer. In this mode, the device may transfer data directly to/from memory without any interference from the CPU. The device requests the CPU (through a DMA controller) to hold its data, address and control bus, so that the device may transfer data directly to/from memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

Intel's 8257 is a four channel DMA controller designed to be interfaced with their family of microprocessors. The 8257, on behalf of the devices, requests the CPU for bus access using local bus request input i.e. HOLD in minimum mode.

In maximum mode of the microprocessor RQ/GT pin is used as bus request input. On receiving the HLDA signal (in minimum mode) or RQ/GT signal (in maximum mode) from the CPU, the requesting devices gets the access of the bus, and it completes the required number of DMA cycles for the data transfer and then hands over the control of the bus back to the CPU.

Internal Architecture of 8257

The internal architecture of 8257 is shown in figure. The chip support four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer through these channels at a time. The DMA controller has 8-bit internal data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers. The 8257 performs the

DMA operation over four independent DMA channels. Each of four channels of 8257 has a pair of two 16-bit registers, viz. DMA address register and terminal count register. There are two common registers for all the channels; namely, mode set register and status register. Thus there are a total of ten registers. The CPU selects one of these tenregisters using address lines Ao-A3. Table shows how the Ao-A3bits may be used forselecting one of these registers.

DMA Address Register

Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel.

Thus the starting address of the memory block which will be accessed by the device is first loaded in the DMA address register of the channel. The device that wants to transfer data over a DMA channel, will access the block of the memory with the starting address stored in the DMA Address Register.

Terminal Count Register

Each of the four DMA channels of 8257 has one terminal count register (TC). This 16-bit register issued for ascertaining that the data transfer through a DMA channel ceases or stops after the required number of DMA cycles. The low order 14-bits of the terminal count register are initialized with the binary equivalent of the number of required DMA cycles minus one.

After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over. The bits 14 and 15 of this register indicate the type of the DMA operation (transfer). If the device wants to write data into the memory, the DMA operation is called DMA write operation. Bit14 of the register in this case will be set to one and bit 15 will be set to zero.

Table gives detail of DMA operation selection and corresponding bit configurationof bits 14 and 15 of the TC register.

Table .DMA operation selection using A15/RD and A15/WR

Bit 15	*Bit 14*	*Type of DMA Operation*
0	0	Verify DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	(Illegal)

Mode Set Register

The mode set register is used for programming the 8257 as per the requirements of the system. The function of the mode set register is to enable the DMA channels individually and also to set the various modes of operation. The DMA channel should not be enabled till the DMA address register and the terminal count register contain valid information; otherwise, an unwanted DMA request may initiate a DMA cycle, probably destroying the valid memory data. The bits Do-D3enable one of the four DMA channels of 8257. For example, if Do is '1', channel 0 is enabled. If bit 4 is set, rotating priority is enabled, otherwise, the normal, i.e. fixed priority is enabled.

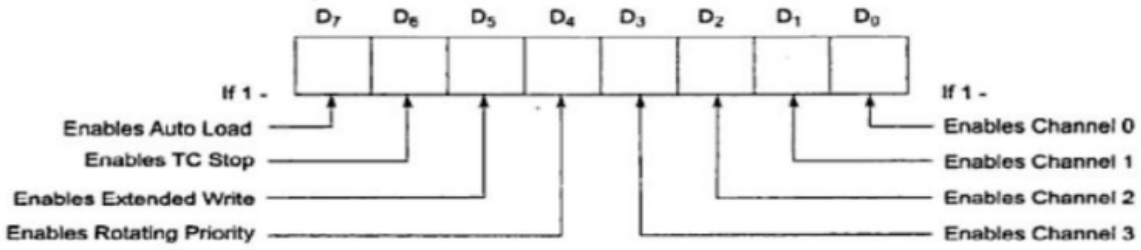


Fig. Bit definition of the mode set register

If the TC STOP bit is set, the selected channel is disabled after the terminal count condition is reached, and it further prevents any DMA cycle on the channel. To enable the channel again, this bit must be reprogrammed. If the TC STOP bit is programmed to be zero, the channel is not disabled, even after the count reaches zero and further request are allowed on the same channel.

The auto load bit, if set, enables channel 2 for the repeat block chaining operations, without immediate software intervention between the two successive blocks. The channel 2 registers are used as usual, while the channel 3 registers are used to store the block reinitialisation parameters, i.e. the DMA starting address and terminal count.

After the first block is transferred using DMA, the channel 2 registers are reloaded with the corresponding channel 3 registers for the next block transfer, if the update flag is set. The extended write bit, if set to '1', extends the duration of MEMW and IOW signals by activating them earlier, this is useful in interfacing the peripherals with different access times.

If the peripheral is not accessed within the stipulated time, it is expected to give the 'NOT READY' indication to 8257, to request it to add one or more wait states in the DMA CYCLE. The mode set register can only be written into.

Status Register

The status register of 8257 is shown in figure. The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition.

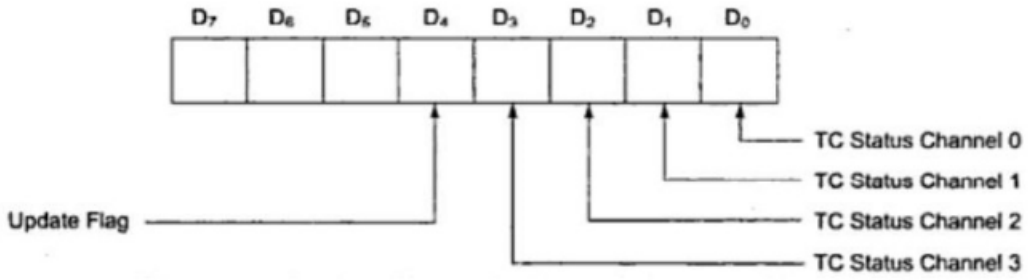


Fig. Status Register

These bits remain set till either the status is read by the CPU or the 8257 is reset. The update flag is not affected by the read operation. This flag can only be cleared by resetting 8257 or by resetting the auto load bit of the mode set register. If the update flag is set, the contents of the channel 3 registers are reloaded to the corresponding registers ofchannel 2 whenever the channel 2 reaches a terminal count condition, after transferring one block and the next block is to be transferred using the auto load feature of 8257. The update flag is set every time; the channel 2 registers are loaded with contents of the channel 3 registers. It is cleared by the completion of the first DMA cycle of the new block. This register can only read.

Data Bus Buffer, Read/Write Logic, Control Unit and Priority Resolver

The 8-bit. Tristate, bidirectional buffer interfaces the internal bus of 8257 with the external system bus under the control of various control signals.

In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the Ao-A3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending upon whether IOW or IOR signal is activated.

In master mode, the read/write logic generates the IOR and IOW signals to control the data flow to or from the selected peripheral. The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, MEMR, MEMW, TC and MARK along with the address lines A4-A7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.

Signal Description of 8257

DRQo-DRQ3:

These are the four individual channel DMA request inputs, used by the peripheral devices for requesting the DMA services. The DRQo has the highest priority while DRQ3 has the lowest one, if the fixed priority mode is selected.

DACKo-DACK3:

These are the active-low DMA acknowledge output lines which inform the requesting peripheral that the request has been honoured and the bus is relinquished by the CPU. theselines may act as strobe lines for the requesting devices.

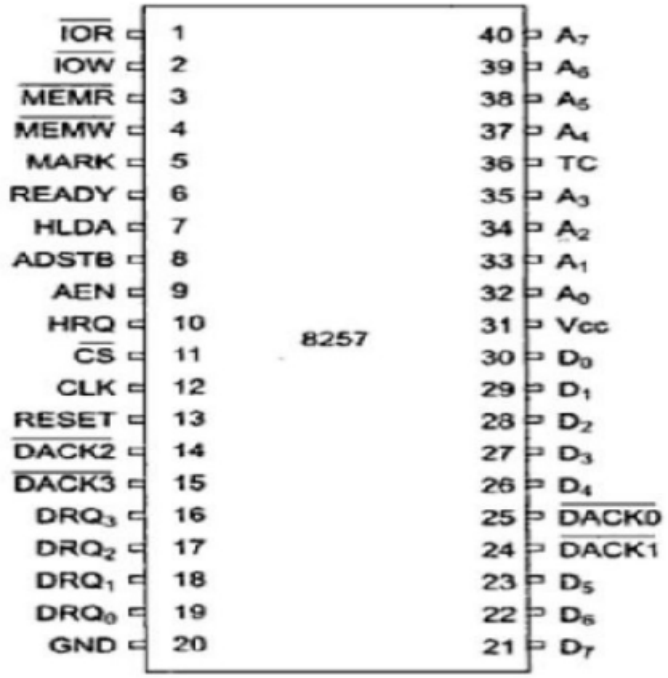


Fig Pin Diagram of 8257

Do-D7:

These are bidirectional, data lines used to interface the system bus with the internal data bus of 8257. These lines carry command words to 8257 and status word from 8257, in slave mode, i.e. under the control of CPU. The data over these lines may be transferred in both the directions. When the 8257 is the bus master (master mode, i.e. not under CPU control), it uses Do-D7lines to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal. the address is transferred over Do-D7during the first clock cycle of the DMA cycle. During the rest of the period, data is available on the data bus.

IOR:

This is an active-low bidirectional tristate input line that acts as an input in the slave mode. In slave mode, this input signal is used by the CPU to read internal registers of 8257.thisline acts output in master mode. In master mode, this signal is used to read data from a peripheral during a memory write cycle.

IOW:

This is an active low bidirection tristate line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register

or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK:

This is a clock frequency input required to derive basic system timings for the internal operation of 8257.

RESET:

This active-high asynchronous input disables all the DMA channels by clearing the mode register and tristates all the control lines.

Ao-A3:

These are the four least significant address lines. In slave mode, they act as input which select one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS:

This is an active-low chip select line that enables the read/write operations from/to 8257, in slave mode. In the master mode, it is automatically disabled to prevent he chip from getting selected (by CPU) while performing the DMA operation.

A4-A7:

This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.

READY:

This is an active-high asynchronous input used to stretch memory read and write cycles of8257 by inserting wait states. This is used while interfacing slower peripherals..

HRQ:

The hold request output requests the access of the system bus. In the noncascaded 8257systems, this is connected with HOLD pin of CPU. In the cascade mode, this pin of a slave is connected with a DRQ input line of the master 8257, while that of the master is connected with HOLD input of the CPU.

HLDA:

The CPU drives this input to the DMA controller high, while granting the bus to the device. This pin is connected to the HLDA output of the CPU. This input, if high, indicates to the DMA controller that the bus has been granted to the requesting peripheral by the CPU.

MEMR:

This active –low memory read output is used to read data from the addressed memory locations during DMA read cycles.

MEMW:

This active-low three state output is used to write data to the addressed memory location during DMA write operation.

ADST:

This output from 8257 strobes the higher byte of the memory address generated by the DMA controller into the latches.

AEN:

This output is used to disable the system data bus and the control the bus driven by the CPU,this may be used to disable the system address and data bus by using the enable input of the bus drivers to inhibit the non-DMA devices from responding during DMA operations. If the8257 is I/O mapped, this should be used to disable the other I/O devices, when the DMA controller addresses is on the address bus.

TC:

Terminal count output indicates to the currently selected peripherals that the present DMA cycle is the last for the previously programmed data block. If the TC STOP bit in the mode set register is set, the selected channel will be disabled at the end of the DMA cycle. The TC pin is activated when the 14-bit content of the terminal count register of the selected channel becomes equal to zero. The lower order 14 bits of the terminal count register are to be programmed with a 14-bit equivalent of (n-1), if n is the desired number of DMA cycles.

MARK:

The modulo 128 mark output indicates to the selected peripheral that the current DMA cycle is the 128thcycle since the previous MARK output. The mark will be activated after each128 cycles or integral multiples of it from the beginning if the data block (the first DMA cycle), if the total number of the required DMA cycles (n) is completely divisible by 128.

Vcc: This is a +5v supply pin required for operation of the circuit.

GND: This is a return line for the supply (ground pin of the IC).

Interfacing 8257 with 8086

Once a DMA controller is initialized by a CPU property, it is ready to take control of the system bus on a DMA request, either from a peripheral or itself (in case of memory-to memory transfer). The DMA controller sends a HOLD request to the CPU and waits for the CPU to assert the HLDA signal. The CPU relinquishes the control of the bus before asserting the HLDA signal

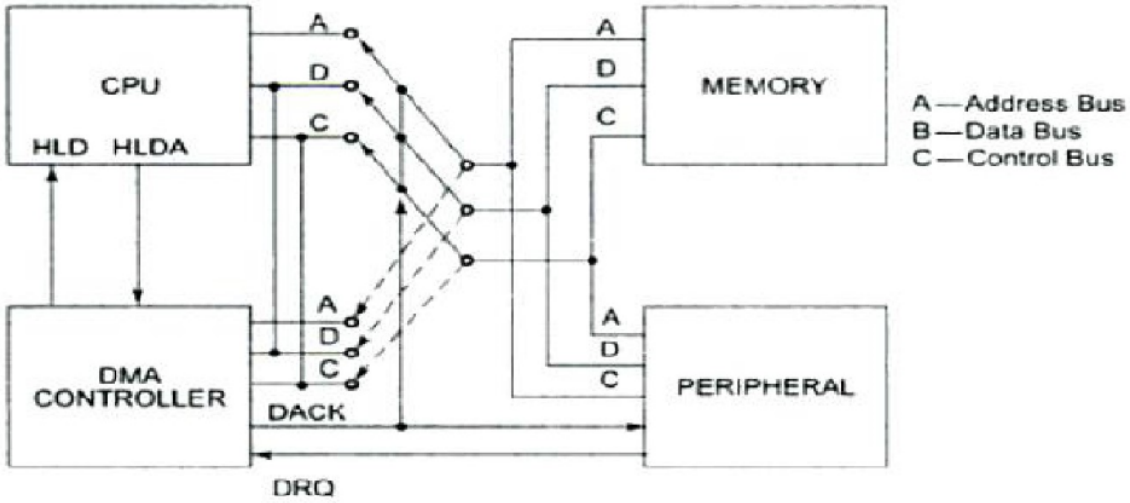


Fig Interfacing 8257 with CPU

Once the HLDA signal goes high, the DMA controller activates the DACK signal to the requesting peripheral and gains the control of the system bus. The DMA controller is the sole master of the bus, till the DMA operation is over. The CPU remains in the HOLD status (all of its signals are tristate except HOLD and HLDA), till the DMA controller is the master of the bus. In other words, the DMA controller interfacing circuit implements a switching arrangement for the address, data and control busses of the memory and peripheral subsystem from/to the CPU to/from the DMA controller.

Traffic Light Control: Traffic Light Controller Using 8086

Traffic light controller interface module is designed to simulate the function of four way traffic light controller.Combinations of red, amber and green LED's are provided to indicate Halt, Wait and Go signals for vehicles.

Combination of red and green LED's are provided for pedestrian crossing.36 LED's are arranged in the form of an intersection. A typical junction is represented on the PCB with comprehensive legend printing.At the left corner of each road, a group of five LED's (red, amber and 3 green) are arranged in the form of a T-section to control the traffic of that road.

Each road is named North (N), South(S), East (E) and West (W). LED's L1, L10, L19 &L28 (Red) are for the stop signal for the vehicles on the road N, S, W, & E respectively.L2, L11, L20 & L29 (Amber) indicates wait state for vehicles on the road N, S, W, & E respectively. L3, L4 & L5 (Green) are for left, strait and right turn for the vehicles on road S. similarly L12-L13-L14, L23-L22-L21 & L32-L31-L30 simulates same function for the roads E, N, W respectively.

A total of 16 LED's (2 Red & 2 Green at each road) are provided for pedestrian crossing.L7-L9.L16-L18, L25-L27 & L34-L36 (Green) when on allows pedestrians to cross and L6-L8, L15-L17, L24-L26 & L33-L35 (Red) when on alarms the pedestrians to wait.

To minimize the hardware pedestrian's indicator LED's (both red and green are connected to same port lines (PC4 to PC7) with red inverted. Red LED's L10 & L28 are connected to port lines PC2 & PC3 while L1 & L19are connected to lines PC0 & PC1 after inversion. All other LED's (amber and green) are connected to port A & B.

- ✓ **Working**:- 8255 is interfaced with 8086 in I/O mapped I/O and all ports are output ports. The basic operation of the interface is explained with the help of the enclosed program.

The enclosed program assumes no entry of vehicles from North to West, from road East to South. At the beginning of the program all red LED's are switch ON, and all other LED's are switched OFF. Amber LED is switched ON before switching over to precede state from Halt state.

The sequence of traffic followed in the program is given below.

a) From road north to East

From road east to north

From road south to west

From road west to south

From road west to north

b) From road north to East

From road south to west

From road south to north

From road south to east

From road north to south

From road south to north

Pedestrian crossing at roads west & east

d) From road east to west

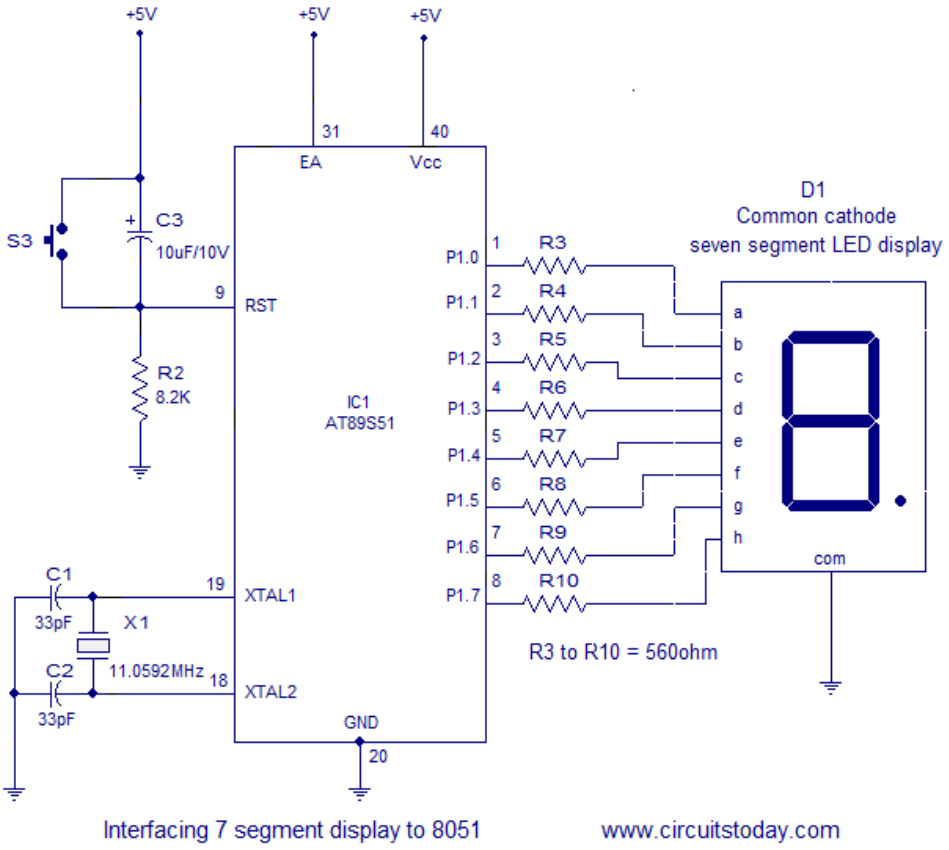
From road west to east

Pedestrian crossing at roads north & south

LED DISPLAY

A seven segment **display** consists of seven **LEDs** arranged in the form of a squarish '8' slightly inclined to the right and a single **LED** as the dot character. Different characters can be displayed by selectively glowing the required **LED** segments.

Interfacing seven segment displays to 8051.



This article is about how to interface a seven segment LED display to an 8051 microcontroller. 7 segment LED display is very popular and it can display digits from 0 to 9 and quite a few characters like A, b, C, ., H, E, e, F, n, o, t, u, y, etc. Knowledge about how to interface a seven segment display to a micro controller is very essential in designing embedded systems. A seven segment display consists of seven LEDs arranged in the form of a squarish **'8'** slightly inclined to the right and a single LED as the dot character. Different characters can be displayed by selectively glowing the required LED segments. Seven segment displays are of two types, ***common cathode and common anode.*** In common cathode type , the cathode of all LEDs are tied together to a single terminal which is usually labeled as '**com**' and the anode of all LEDs are left alone as individual pins labeled as a, b, c, d, e, f, g & h (or dot) . In common anode type, the anode of all LEDs is tied together as a single terminal and cathodes are left alone as individual pins. The pin out scheme and picture of a typical 7 segment LED display is shown in the image below.

Digit drive pattern.

Digit drive pattern of a seven segment LED display is simply the different logic combinations of its terminals **'a' to 'h'**in order to display different digits and characters. The common digit drive patterns (0 to 9) of a seven segment display are shown in the table below.

Digit	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

The circuit diagram shown above is of an AT89S51 microcontroller based 0 to 9 counter which has a 7 segment LED display interfaced to it in order to display the count. This simple circuit illustrates two things. How to setup simple 0 to 9 up counter using 8051 and more importantly how to interface a seven segment LED display to 8051 in order to display a particular result. The common cathode seven segment display D1 is connected to the Port 1 of the microcontroller (AT89S51) as shown in the circuit diagram. R3 to R10 are current limiting resistors. S3 is the reset switch and R2, C3 forms a de bouncing circuitry. C1, C2 and X1 are related to the clock circuit. The software part of the project has to do the following tasks.

- Form a 0 to 9 counter with a predetermined delay (around 1/2 second here).
- Convert the current count into digit drive pattern.
- Put the current digit drive pattern into a port for displaying.

All the above said tasks are accomplished by the program given below.

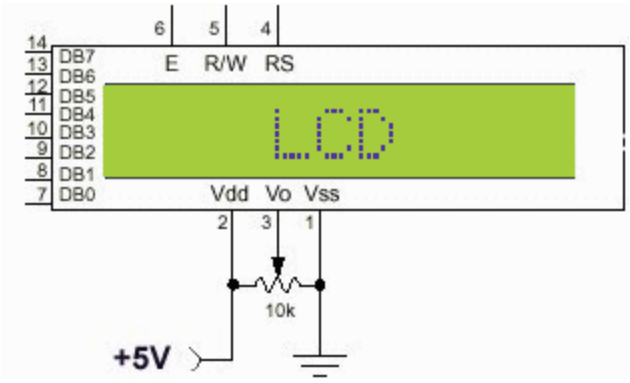
Program.

```
ORG 000H //initial starting address
START: MOV A,#00001001B // initial value of accumulator
MOV B,A
MOV R0,#0AH //Register R0 initialized as counter which counts from 10 to 0
```

```
LABEL: MOV A,B
INC A
MOV B,A
MOVC A,@A+PC // adds the byte in A to the program counters address
MOV P1,A
ACALL DELAY // calls the delay of the timer
DEC R0//Counter R0 decremented by 1
MOV A,R0 // R0 moved to accumulator to check if it is zero in next instruction.
JZ START //Checks accumulator for zero and jumps to START. Done to check if counting has been finished.
SJMP LABEL
DB 3FH // digit drive pattern for 0
DB 06H // digit drive pattern for 1
DB 5BH // digit drive pattern for 2
DB 4FH // digit drive pattern for 3
DB 66H // digit drive pattern for 4
DB 6DH // digit drive pattern for 5
DB 7DH // digit drive pattern for 6
DB 07H // digit drive pattern for 7
DB 7FH // digit drive pattern for 8
DB 6FH // digit drive pattern for 9
DELAY: MOV R4,#05H // subroutine for delay
WAIT1: MOV R3,#00H
WAIT2: MOV R2,#00H
WAIT3: DJNZ R2,WAIT3
DJNZ R3,WAIT2
DJNZ R4,WAIT1
RET
END
```

LCD display

The LCD display has two lines of characters, 16 characters per line. Each character is composed of matrix of pixels size 5x8. The matrix is controlled by Hitachi HD44780 controller, which performs all the operations that are required to run the matrix. Controller operation is done in accordance with the instructions it receives as described below:



- DB0 - DB7, the 8 data bus lines, which perform read/write of data
- Vss, Vdd - Voltage supply pins
- R/W – Pin writing/reading to/from - LCD
- RS - Pin selects registers between Instruction Register and Data Register
- E - "Enabling" pin; when this pin is set to logical low, the LCD does not care what is happening with R/W, RS, and the data bus lines; when this pin is set to logical high, the - LCD is processing the incoming data
- Vo - Pin for LCD contrast

LCD registers

The HD44780U controller has two 8-bit registers:

- *an instruction register (IR)* - the IR stores instruction codes, such as display clear and cursor shift, and address information for display data RAM (DDRAM) and character generator RAM (CGRAM).
- *a data register (DR)* - the DR temporarily stores data to be written into DDRAM or CGRAM and temporarily stores data to be read from DDRAM or CGRAM. The DR is also used for data storage when reading data from DDRAM or CGRAM.

The choice between the two registers is made by the register selector (RS) signal as detailed the following table:

Register Selector		
RS	R/W	
0	0	Sends a command to LCD
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	Sends information to LCD
1	1	Reads information from LCD

Busy Flag (BF)

BF gives an indication whether the LCD is finished the previous instruction and ready with the next.

DDRAM Memory (Display Data RAM)

Display data RAM (DDRAM) stores the information we send to LCD in ASCII Code. For each letter there is a special code that represents it: for example, the letter A in ASCII code, "receives" a value of 65 in base 10 or 01000001 in binary base, or 41 in the base 16. The memory can contain up to 80 letters.

Some of the addresses represent the lines of LCD (0x00-0x0F- first line; 0x40-0x4F – second line). The rest of the addresses represent the "non-visible" memory of the DRAM, which can be also used as a general memory. The DDRAM address is the position of the cursor on the display LCD (the received information will be written at the place where the cursor is).

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

CGRAM Memory (Character Generator RAM)

Using CGRAM memory the user can "build" and store their own letters. For 5x8 dots, eight character patterns can be written, and for 5x10 dots, four character patterns can be written. The difference between the memories is that the DDRAM memory displays on the screen the "ready" characters in accordance with the ASCII code, while the CGRAM memory displays the special characters that the user has created.

Address Counter (AC)

The address counter (AC) assigns addresses to both DDRAM and CGRAM. When an address of an instruction is written into the IR, the address information is sent from the IR to the AC. Selection of either DDRAM or CGRAM is also determined concurrently by the instruction. After writing into (reading from) DDRAM or CGRAM, the AC is automatically incremented by 1(decremented by 1). The AC contents are then output to DB0 to DB6 when RS = 0 and R/W = 1.

Writing a letter/character to the LCD display

To write a letter/character on the LCD display we have to do the following:

1. Perform an initialization.
2. Send the desired position to IR (DDRAM Address).
3. Send ASCII code of the letter to DR.

LCD display will show the letter that matches the code that was sent and the address counter AC will be updated (increment or decrement, depending on how it was initialized). You can write strings by sending characters in sequence.

LCD instruction set

The LCD instruction set consists of the commands you can send to LCD. Remember that the RS line needs to be set to zero to send instruction to the LCD. When the RS line is set to one, you are sending data to display memory or the character graphics (CG) memory. An "X" in any position means it does not matter what you enter there.

Clear Display:

This command clears the display and returns the cursor to the home position (address 0) and sets I/D to 1 in order to increment the cursor. Its line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

Home Cursor:

This returns the cursor to the home position, returns a shifted display to the correct position, and sets the display data (DD) RAM address to 0. Its line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	X

Entry Mode Set:

This command sets the cursor move direction and specifies whether to shift the display or not. These operations are performed during the data write/read of the CG or DD RAM. Its line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	I/D	S

I/D=0 means the cursor position is decremented (moves right to left). I/D=1 means the cursor position is incremented (moves left to right). S=0 means normal operation, the display remains still, and the cursor moves. S=1 means the display moves with the cursor.

Display On/Off Control:

This command sets the ON/OFF display as well as the cursor and blinking capabilities (0 equals OFF; 1 equals ON). D controls whether the display is ON or OFF, C controls whether the cursor is ON or OFF, B controls whether the blinking is ON or OFF. The line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0

0	0	0	0	0	0	1	D	C	B

Cursor or Display Shift:

This moves the cursor and shifts the display without changing DD RAM contents. The line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	S/C	R/L	X	X

S/C=0 means move the cursor.

S/C=1 means shift display.

R/L= 0 means shift to the left.

R/L= 1 means shift to the right.

Function Set:

This sets the interface data length (DL), the number of display lines (N), and character font (F). The line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	N	F	X	X

DL=0 means 4 bits are being used (the standard)

DL=1 means a full 8 bits being utilized

N=0 means 1 line

N=1 means 2 lines or more

F=0 means that 5x7 dot characters are used (which is how 99% of all LCDs are set up)

F=1 means 5x10 dot characters are used

Set CG RAM Address:

This command sets the custom graphics (CG) RAM address. Setting RS to 1 sends data to CG RAM instead of the DD RAM. Eight CG characters are available, and they reside in the ASCII codes 0 through 7. The is sent in the 8-bit bytes from the top row to the bottom row and is left justified, meaning that only the bottom 5 bits matter (it is a 5x7 dot matrix). The line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	MSB	CG	RAM	ADDRESS	LSB	

Set DD RAM Address:

This sets the DD RAM address. Setting RS to 1 sends data to the display RAM, and the cursor advances in the direction where the I/D bit was set to. The line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0

0	0	1	MSB DD RAM ADDRESS LSB

Read Busy Flag and Address:

This reads the busy flag (BF). If BF equals to 1, the LCD is busy and displays the location of the cursor. With the R/W line grounded, this command cannot be used. The line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	0	0	0	0	0	0	1

Write Data to CG or DD RAM:

This command's line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0	MSB ASCII code or CG bit pattern data LSB							

Read Data from CG or DD RAM:

This command's line settings are as follows:

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	1	MSB ASCII code or CG bit pattern data LSB							

The LCD initializing sequence:

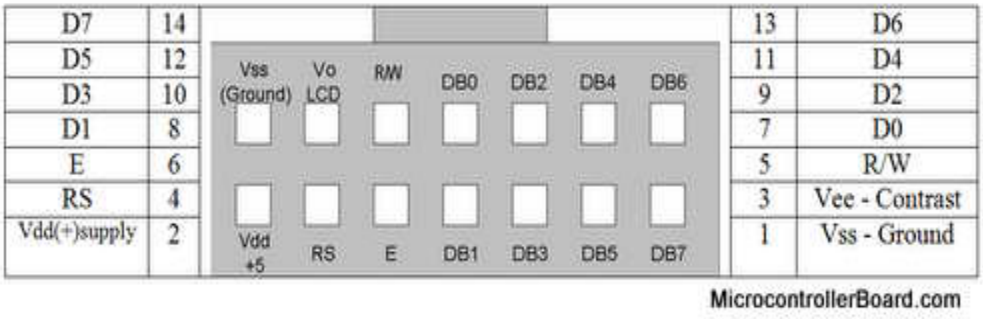
The initializing sequence includes the steps that need to be executed in order for the LCD to work. In fact, these sequences of steps define in what form we want the LCD to work: the data length (8 bit or 4 bit); size of the letters; activation of the cursor; and more. When you send out an instruction (command or information) to the LCD it takes some time to execute it, so it's important to make sure that the LCD is "ready" for the next instruction/operation.

You can check if the LCD is ready in the following 2 ways:

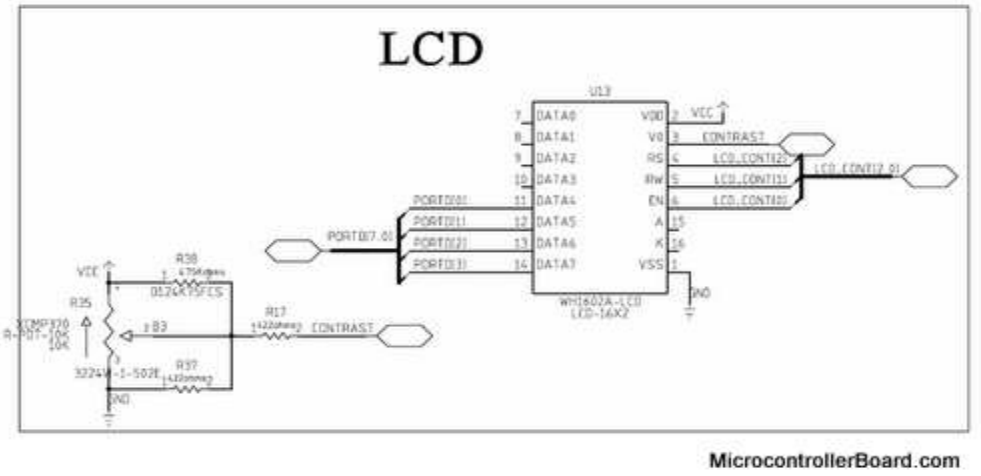
1. Create a delay subroutine to accommodate the minimum execution time.
2. Scanning BF (busy flag) bit – this bit gives an indication whether the LCD is finished working.

LCD hardware configuration

The traditional LCD connection is via a 14-pin dual in-line connector that works nicely with a 14-pin ribbon cable connector as show in the figure below:



Even though the cable pin out consists of 8 data lines (DB0-DB7), traditionally everyone uses the LCD in 4-bit mode to save on data lines and control signal lines. The following figure shows the LCD connection as it used with EduPIC development board.



We used 4 consecutive bits (PORTD0-PORTD3) in configurable nibble as the data lines. In addition, we used PORTE0-PORTE2 for the RS, EN and RW control signals lines.

KEYBOARD AND DISPLAY INTERFACE USING INTEL 8279 MICROPROCESSOR

In a microprocessor b system, when keyboard and 7-segment LED display is interfaced using ports or latches then the processor has to carry the following task.

• Keyboard scanning

• Key de bouncing

• Key code generation

• Sending display code to LED

• Display refreshing

Interfacing 8279 with 8085 processor:

• A typical Hexa keyboard and 7-segment LED display interfacing circuit using 8279 is shown.

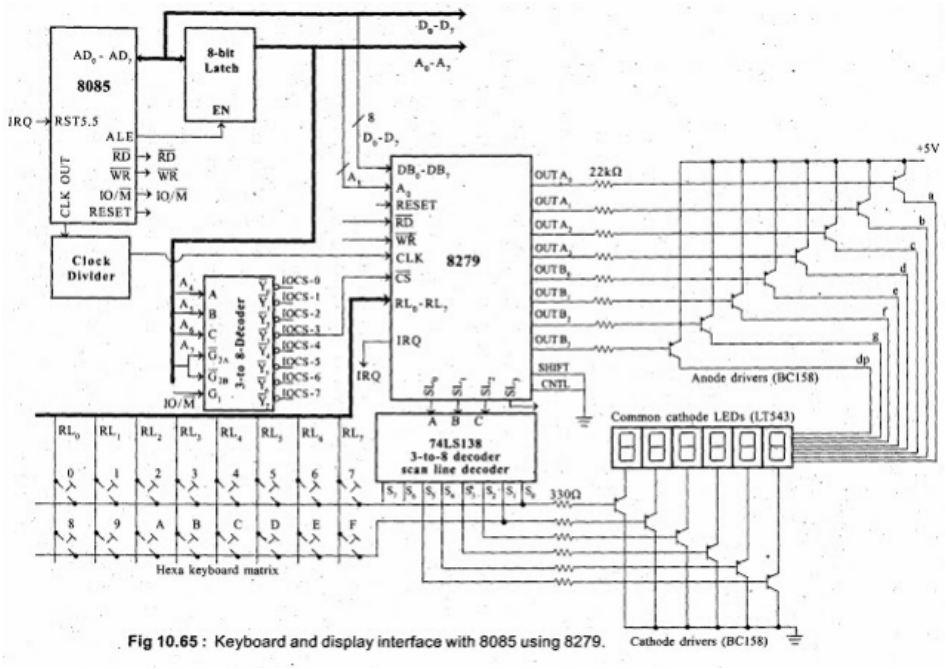


Fig 10.65 : Keyboard and display interface with 8085 using 8279.

• The circuit can be used in 8085 microprocessor system and consist of 16 numbers of hexa-keys and 6 numbers of 7-segment LEDs.

• The 7-segment LEDs can be used to display six digit alphanumeric characters.

• The 8279 can be either memory mapped or I/O mapped in the system. In the circuit shown is the 8279 is I/O mapped.

• The address line A0 of the system is used as A0 of 8279.

• The clock signal for 8279 is obtained by dividing the output clock signal of 8085 by a clock divider circuit.

• The chip select signal is obtained from the I/O address decoder of the 8085 system. The chip select signals for I/O mapped devices are generated by using a 3-to-8 decoder.

• The address lines A4, A5 and A6 are used as input to decoder.

• The address line A7 and the control signal IO/M (low) are used as enable for decoder.

• The chip select signal IOCS-3 is used to select 8279.

• The I/O address of the internal devices of 8279 is shown in table.

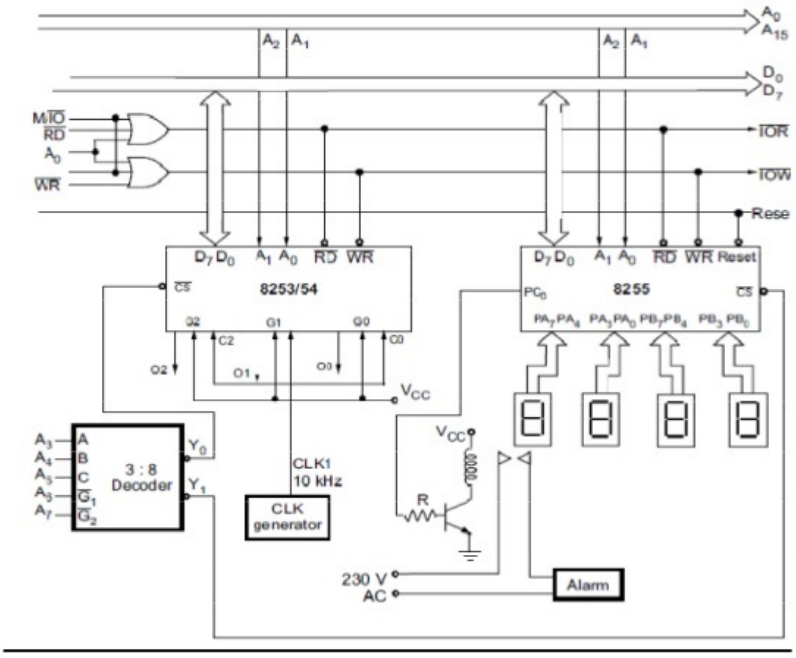
Internal Device	Binary Address								Hexa Address
	Decoder input and enable				Input to address line of 8279				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Data register	0	0	1	1	x	x	x	0	30
Control register	0	0	1	1	x	x	x	1	31

Note : ***Don't care "x" is considered as zero.***

• The circuit has 6 numbers of 7-segment LEDs and so the 8279 has to be programmed in encoded scan. (Because in decoded scan, only 4 numbers of 7-segment LEDs can be interfaced):
• in encoded scan the output of scan lines will be binary count. Therefore an external, 3-to8 decoder is used to decode the scan lines SL0, SL1 and SL2 of 8279 to produce eight scan lines S0 to S7.
• The decoded scan lines S0 and S1 are common for keyboard and display. • The decoded scan lines S2 to S5 are used only for display and the decoded scan lines S6 and S7 are not used in the system.
• Anode and Cathode drivers are provided to take care of the current requirement of LEDs.
• The pnp transistors, BC 158 are used as driver transistors.
• The anode drivers are called segment drivers and cathode drivers are called digit drivers.
• The 8279 output the display code for one digit through its output lines (OUT A0 to OUT A3 and OUT B0 to OUT B3) and send a scan code through, SL0- SL3.
• The display code is inverted by segment drivers and sent to segment bus.
• The scan code is decoded by the decoder and turns ON the corresponding digit driver. Now one digit of the display character is displayed. After a small interval (10 milli- second, typical), the display is turned OFF (i.e., display is blanked) and the above process is repeated for next digit. Thus multiplexed display is performed by 8279.
• The keyboard matrix is- formed using the return lines, RL0 to RL3 of 8279 as columns and decoded scan lines S0 and S1 as rows.
• A hexa key is placed at the crossing point of each row and column. A key press will short the row and column. Normally the column and row line will be high.
• During scanning the 8279 will output binary count on SL0 to SL3, which is decoded by decoder to make a row as zero. When a row is zero the 8279 reads the columns. If there is a key press then the corresponding column will be zero.
• If 8279 detects a key press then it waits for de bounce time and again read the columns to generate key code.
• In encoded scan keyboard mode, the 8279 stores an 8-bit code for each valid key press. The key code consist of the binary value of the column and row in which the key is found and the status of shift and control key.
• After a scan time, the next row is made zero and the above process is repeated and so on. Thus 8279 continuously scan the keyboard.

ALARAM CONTROLLER:

Pre settable alarm control:



Pin Diagram:

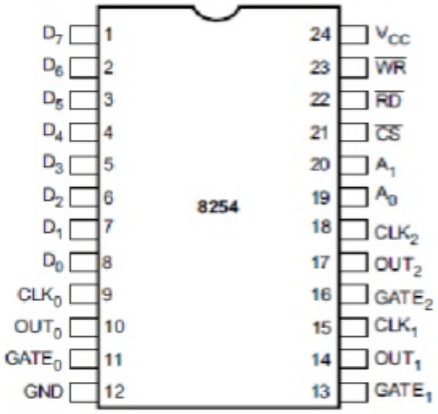


Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
D_7–D_0	1–8	I/O	**DATA:** Bi-directional three state data bus lines, connected to system data bus.
CLK 0	9	I	**CLOCK 0:** Clock input of Counter 0.
OUT 0	10	O	**OUTPUT 0:** Output of Counter 0.
GATE 0	11	I	**GATE 0:** Gate input of Counter 0.
GND	12		**GROUND:** Power supply connection.
V_{CC}	24		**POWER:** + 5V power supply connection.
$\overline{WR}$	23	I	**WRITE CONTROL:** This input is low during CPU write operations.
$\overline{RD}$	22	I	**READ CONTROL:** This input is low during CPU read operations.
$\overline{CS}$	21	I	**CHIP SELECT:** A low on this input enables the 8254 to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and $\overline{WR}$ are ignored otherwise.
A_1, A_0	20–19	I	**ADDRESS:** Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.

A_1	A_0	Selects
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word Register

CLK 2	18	I	**CLOCK 2:** Clock input of Counter 2.
OUT 2	17	O	**OUT 2:** Output of Counter 2.
GATE 2	16	I	**GATE 2:** Gate input of Counter 2.
CLK 1	15	I	**CLOCK 1:** Clock input of Counter 1.
GATE 1	14	I	**GATE 1:** Gate input of Counter 1.
OUT 1	13	O	**OUT 1:** Output of Counter 1.

Unit-4

Architecture of 8031/ 8051:

DEVICE	ON-CHIP DATA MEMORY (bytes)	ON-CHIP PROGRAM MEMORY (bytes)	16-BIT TIMER/COUNTER	NO. OF VECTORED INTERUPTS	FULL DUPLEX I/O
8031	128	None	2	5	1
8032	256	none	2	6	1
8051	128	4k ROM	2	5	1
8052	256	8k ROM	3	6	1
8751	128	4k EPROM	2	5	1
8752	256	8k EPROM	3	6	1
AT89C51	128	4k Flash Memory	2	5	1
AT89C52	256	8k Flash memory	3	6	1

It is a single chip

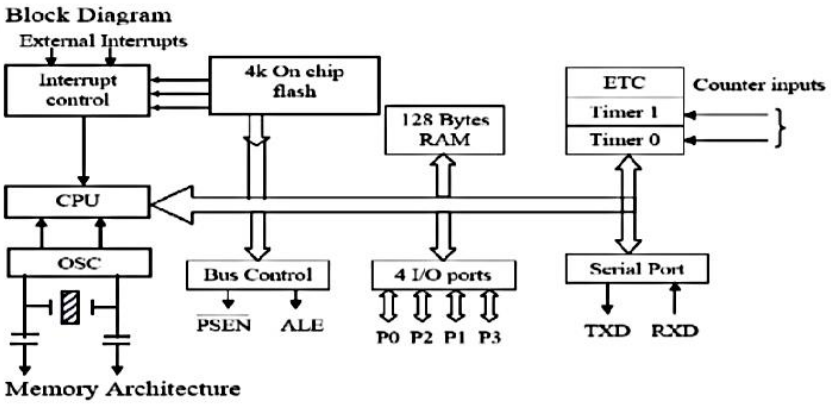
- ✓ Consists of CPU, Memory
- ✓ I/O ports, timers and other peripherals

- ✓ It is a CPU
- ✓ Memory, I/O Ports to be connected externally.
- ✓ Small size, low power, low cost;
- ✓ Harvard architecture with separate program and data memory;
- ✓ No data corruption or loss of data; but with complex circuit
- ✓ The 8051 has three very general types of memory.
- ✓ On-Chip Memory refers to any memory (Code, RAM, or other) that physically exists on the microcontroller itself. On-chip memory can be of several types.
- ✓ External Code Memory is code (or program) memory that resides off-chip. This isoften in the form of an external EPROM.
- ✓ External RAM is RAM memory that resides off-chip. This is often in the form ofstandard static RAM or flash RAM.

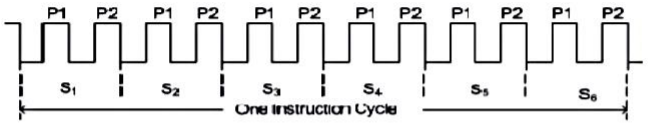
The 8051 is a flexible microcontroller with a relatively large number of modes of operations.

Your program may inspect and/or change the operating mode of the 8051 by manipulating the values of the 8051's Special Function Registers (SFRs).

SFRs are accessed as if they were normal Internal RAM. The only difference is that Internal RAM is from address 00h through 7Fh whereas SFR registers exist in the address range of 80h through FFh



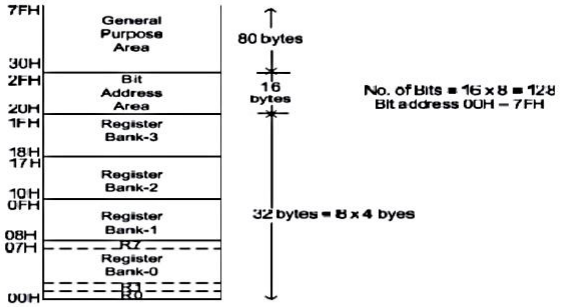
8051 Clock and Instruction Cycle In 8051, one instruction cycle consists of twelve(12) clock cycles. Instruction cycle is sometimes called as Machine cycle by some authors.



Instruction cycle of 8051

In 8051, each instruction cycle has six states (S1- S6). Each state has two pulses (P1 and P2)

128 bytes of Internal RAM Structure (lower address space).

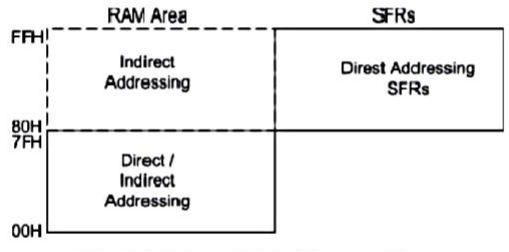


Internal RAM Structure

The lower 32 bytes are divided into 4 separate banks. Each register bank has 8 registers of one byte each. A register bank is selected depending upon two bank select bits in the PSW register. Next 16bytes are bit addressable. In total, 128bits (16X8) are availablein addressable area. Each bit can be accessed and modified by suitable instructions.

The bit addresses are from 00H (LSB of the first byte in 20H) to 7FH (MSB of the last byte in 2FH). Remaining 80bytes of RAM are available for general purpose.

Internal Data Memory and Special Function Register (SFR) Map



Internal data memory map

The special function registers (SFRs) are mapped in the upper 128 bytes of internal data memory address. Hence there is an address overlap between the upper 128 bytes of data RAM and SFRs. Please note that the upper 128 bytes of data RAM are present only in the 8052 family. The lower128 bytes of RAM (00H - 7FH) can be accessed both by direct or indirect addressing while the upper 128 bytes of RAM (80H - FFH) are accessed by indirect addressing. The SFRs (80H - FFH) are accessed by direct addressing only. This feature distinguishes the upper 128 bytes of memory from the SFRs, as shown in figure.

Processor Status Word (PSW) Address=D0H

CY	AC	F0	RS1	RS0	OV	-	P

Processor status word

PSW register stores the important status conditions of the microcontroller. It also stores the bank select bits (RS1 & RS0) for register bank selection.

Special Function Registers:

The 8051 is a flexible microcontroller with a relatively large number of modes of operations. Your program may inspect and/or change the operating mode of the 8051 by manipulating the values of the 8051's Special Function Registers (SFRs).

SFRs are accessed as if they were normal Internal RAM. The only difference is that Internal RAM is from address 00h through 7Fh whereas SFR registers exist in the address range of 80h through FFh. Each SFR has an address (80h through FFh) and a name.

The following chart provides a graphical presentation of the 8051's SFRs, theirnames, and their address. As you can see, although the address ranges of 80h through FFh offer 128 possible addresses, there are only 21 SFRs in a standard 8051. All other addresses in the SFR range (80h through FFh) are considered invalid. Writing to or reading from these registers may produce undefined values or behavior.

SFR Types:

SFRs related to the I/O ports: The 8051 has four I/O ports of 8 bits, for a total of 32 I/O lines. Whether a given I/O line is high or low and the value read from the line are controlled by the SFRs.

The SFRs control the operation or the configuration of some aspect of the 8051. For example, TCON controls the timers, SCON controls the serial port, the remaining SFRs, are auxiliary SFRs in the sense that they don't directly configure the 8051 but obviously the 8051 cannot operate without them. For example, once the serial port has been configured using SCON, the program may read or write to the serial port using the SBUF register.

SFR Descriptions:

P0 (Port 0, Address 80h, and Bit-Addressable): This is input/output port 0. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 0 is pin P0.0, bit 7 is in P0.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level. SP (Stack Pointer, Address 81h): This is the stack pointer of the microcontroller. This SFR indicates where the next value to be taken from the stack will be read from in Internal RAM. If you push a value onto the stack, the value will be written to the address of SP + 1. This SFR is modified by all instructions which modify the stack, such as PUSH, POP, LCALL, RET, RETI, and whenever interrupts are provoked by the microcontroller. The Stack Pointer, like all registers except DPTR and PC, may hold an 8-bit (1-byte) value.

When you pop a value off the stack, the 8051 returns the value from the memory location indicated by SP and then decrements the value of SP.

This order of operation is important. When the 8051 is initialized SP will be initialized to 07h. If you immediately push a value onto the stack, the value will be stored in Internal RAM address 08h. First the 8051 will increment the value of SP (from 07h to 08h) and then will store the pushed value at that memory address (08h). It is also used intrinsically whenever an interrupt is triggered .DPL/DPH (Data Pointer Low/High, Addresses 82h/83h): The SFRs DPL and DPH work together to represent a 16-bit value called the Data Pointer. The data pointer is used in

operations regarding external RAM and some instructions involving code memory. Since it is an unsigned two-byte integer value, it can represent values from 0000h to FFFFh (0 through 65,535 decimal).

PCON (Power Control, Addresses 87h): The Power Control SFR is used to control the8051's power control modes. Certain operation modes of the 8051 allow the 8051 to go into a type of "sleep" mode which requires much less power. These modes of operation are controlled through PCON. Additionally, one of the bits in PCON is used to double the effective baud rate of the 8051's serial port.

TCON (Timer Control, Addresses 88h, and Bit-Addressable): The Timer Control SFR is used to configure and modify the way in which the 8051's two timers operate. This SFR controls whether each of the two timers is running or stopped and contains a flag to indicatethat each timer has overflowed. Additionally, some non-timer related bits are located in the TCON SFR. These bits are used to configure the way in which the external interrupts are activated and also contain the external interrupt flags which are set when an external interrupt has occurred.

TMOD (Timer Mode, Addresses 89h): The Timer Mode SFR is used to configure the mode of operation of each of the two timers. Using this SFR your program may configure each timer to be a 16-bit timer, an 8-bit auto reload timer, a 13-bit timer, or two separate timers. Additionally, you may configure the timers to only count when an external pin is activated or to count "events" that are indicated on an external pin.

TL0/TH0 (Timer 0 Low/High, Addresses 8Ah/8Bh): These two SFRs, taken together, represent timer 0. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

TL1/TH1 (Timer 1 Low/High, Addresses 8Ch/8Dh): These two SFRs, taken together, represent timer 1. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

P1 (Port 1, Address 90h, and Bit-Addressable): This is input/output port 1. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 1 is pin P1.0, bit 7 is pin P1.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

SCON (Serial Control, Addresses 98h, Bit-Addressable): The Serial Control SFR is used to configure the behavior of the 8051's on-board serial port. This SFR controls the baud rate of the serial port, whether the serial port is activated to receive data, and also contains flags that are set when a byte is successfully sent or received.

SBUF (Serial Control, Addresses 99h): The Serial Buffer SFR is used to send and receive data via the on-board serial port. Any value written to SBUF will be sent out the serial port'sTXD pin. Any value which the 8051 receives via the serial port's RXD pin will be deliveredto the user program via SBUF. In other words, SBUF serves as the output port when writtento and as an input port when read from.

P2 (Port 2, Address A0h, and Bit-Addressable): This is input/output port 2. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 2 is pin P2.0, bit 7 is pin P2.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

IE (Interrupt Enable, Addresses A8h): The Interrupt Enable SFR is used to enable and disable specific interrupts. The low 7 bits of the SFR are used to enable/disable the specific interrupts, where as the highest bit is used to enable or disable ALL interrupts. Thus, if the high bit of IE is 0 all interrupts are disabled regardless of whether an individual interrupt is enabled by setting a lower bit.

P3 (Port 3, Address B0h, and Bit-Addressable): This is input/output port 3. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 3 is pin P3.0, bit 7 is pin P3.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

IP (Interrupt Priority, Addresses B8h, and Bit-Addressable): The Interrupt Priority SFR is used to specify the relative priority of each interrupt. On the 8051, an interrupt may either be of low (0) priority or high (1) priority. An interrupt may only interrupt interrupts of lower priority. For example, if we configure the 8051 so that all interrupts are of low priority except the serial interrupt, the serial interrupt will always be able to interrupt the system, even if another interrupt is currently executing. However, if a serial interrupt is executing no other interrupt will be able to interrupt the serial interrupt routine since the serial interrupt routine has the highest priority.

PSW (Program Status Word, Addresses D0h, Bit-Addressable): The Program Status Word is used to store a number of important bits that are set and cleared by 8051 instructions. The PSW SFR contains the carry flag, the auxiliary carry flag, the overflow flag, and the parity flag. Additionally, the PSW register contains the register bank select flags which are used to select which of the "R" register banks are currently selected.

ACC (Accumulator, Addresses E0h, and Bit-Addressable): The Accumulator is one of the most used SFRs on the 8051 since it is involved in so many instructions. The Accumulator resides as an SFR at E0h, which means the instruction MOV A,#20h is really the same as

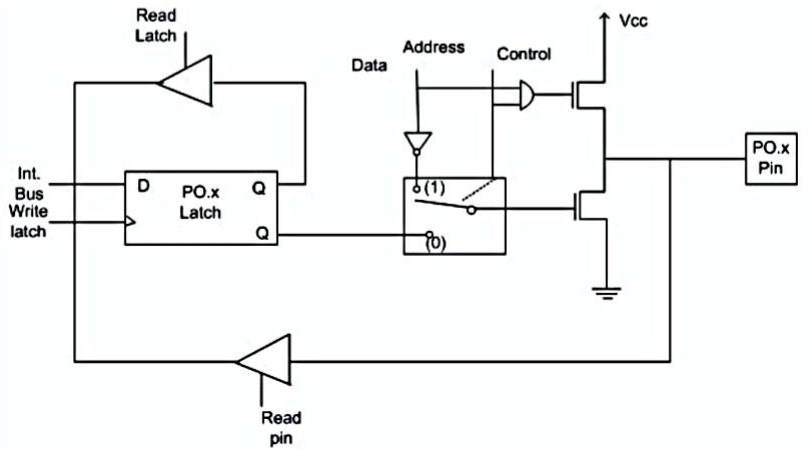
MOV E0h,#20h. First method requires two bytes whereas the second option requires three bytes. It can hold an 8-bit (1-byte) value and More than half of the 8051’s 255 instructions manipulate or use the accumulator in some way.

For example, if you want to add the number 10 and 20, the resulting 30 will be store in the Accumulator. Once you have a value in the Accumulator you may continue processing the value or you may store it in another register or in memory.

B (B Register, Addresses F0h, Bit-Addressable): The "B" register is used in two instructions:the multiply and divide operations. The B register is also commonly used by programmers as an auxiliary register to temporarily store values. Thus, if you want to quickly and easily multiply or divide A by another number, you may store the other number in "B" and make use of these two instruction Aside from the MUL and DIV instructions, the "B" register is often used as yet another temporary storage register much like a ninth "R" register.

I/O ports and circuits:

Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats (tristated) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'. Port-0 Pin Structure Port -0 has 8 pins (P0.0-P0.7).



Port-0 structure

Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port.

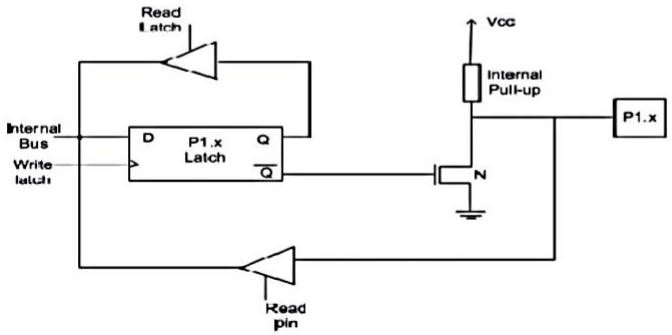
Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats.

This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.

When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. Theoutput becomes '0'. If the address/data bus is '1', the upper transistor is 'on' and the lowertransistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.

Port-0 latch is written to with 1's when used for external memory access.

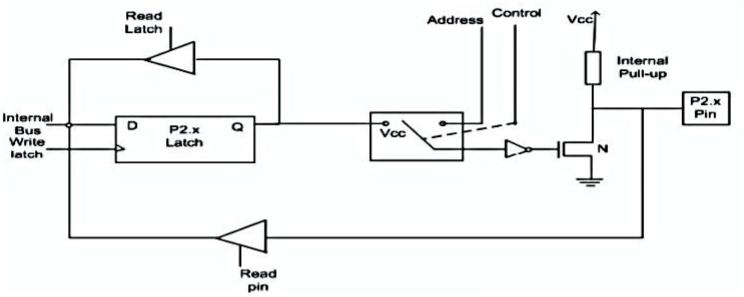
Port-1 Pin Structure Port-1 has 8 pins (P1.1-P1.7) .The structure of a port-1 pin is shown in fig below.



Port-1 structure

Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input

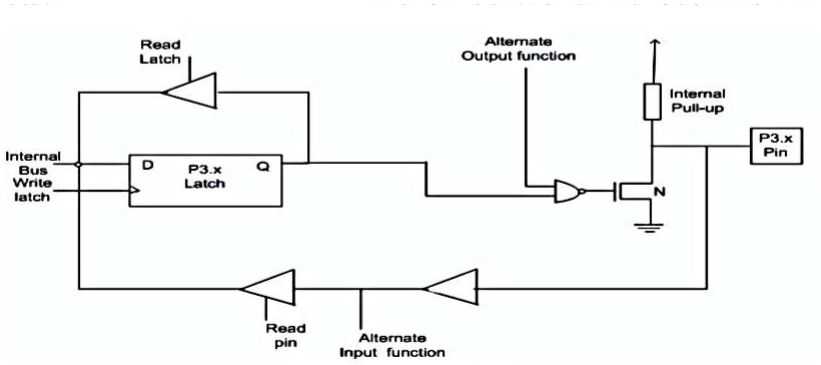
port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading. PORT 2 Pin Structure Port-2 has 8-pins (P2.0-P2.7) . The structure of a port-2 pin is shown in figure below:



Port-2 structure

Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

PORT 3 Pin StructurePort-3 has 8 pin (P3.0-P3.7). Port-3 pins have alternate functions. The structure of a port-3 pin is shown in figure

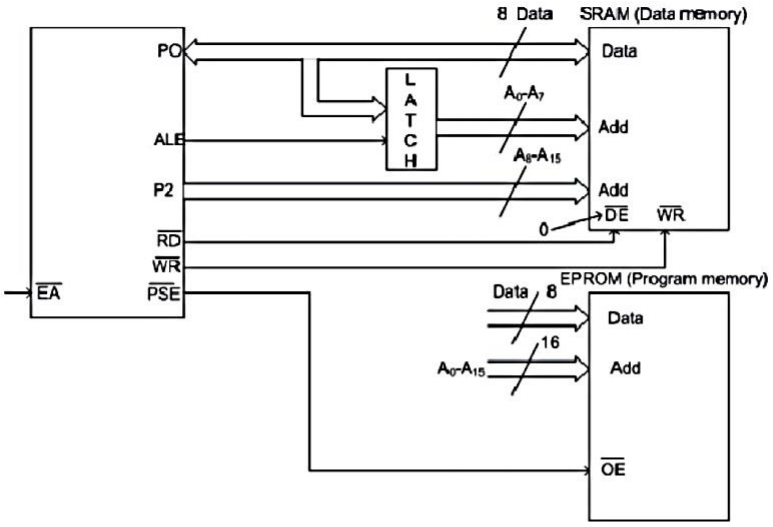


Port-3 structure

Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been written to '1'. To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.

Interfacing External Memory:

If external program/data memory is to be interfaced, they are interfaced in the following way.



External program memory is fetched if either of the following two conditions is satisfied.

1. (Enable Address) is low. The microcontroller by default starts searching for program from external program memory.

2. PC is higher than FFFH for 8051 or 1FFFH for 8052. PSEN tells the outside world whether the external memory fetched is program memory or data memory.

8051 Instructions:

8051 has about 111 instructions. These can be grouped into the following categories

1. Arithmetic Instructions

2. Logical Instructions

3. Data Transfer instructions

4. Boolean Variable Instructions

5. Program Branching Instructions

The following nomenclatures for register, data, address and variables are used while

Write instructions.

A: Accumulator

B: "B" registers

C: Carry bit

Rn: Register R0 - R7 of the currently selected register bank

Direct: 8-bit internal direct address for data. The data could be in lower 128bytes of RAM (00 - 7FH) or it could be in the special function register (80 - FFH).

@Ri: 8-bit external or internal RAM address available in register R0 or R1. This is used for indirect addressing mode.

#data8: Immediate 8-bit data available in the instruction.

#data16: Immediate 16-bit data available in the instruction.

Addr11: 11-bit destination address for short absolute jump. Used by instructions

AJMP & ACALL. Jump range is 2 Kbyte (one page).

Addr16: 16-bit destination address for long call or long jump.

Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and allconditional jumps.

Bit: Directly addressed bit in internal RAM or SFR

✓ **Arithmetic Instructions**

Mnemonics	Description	Bytes	Instruction Cycles
ADD A, Rn	A A + Rn	1	1
ADD A, direct	A A + (direct)	2	1
ADD A, @Ri	A A + @Ri	1	1
ADD A, #data	A A + data	2	1
ADDC A, Rn	A A + Rn + C	1	1
ADDC A, direct	A A + (direct) + C	2	1
ADDC A, @Ri	A A + @Ri + C	1	1
ADDC A, #data	A A + data + C	2	1
DA A	Decimal adjust accumulator	1	1
DIV AB	Divide A by B A quotient B remainder	1	4
DEC A	A A -1	1	1
DEC Rn	Rn Rn - 1	1	1
DEC direct	(direct) (direct) - 1	2	1
DEC @Ri	@Ri @Ri - 1	1	1
INC A	A A+1	1	1
INC Rn	Rn Rn + 1	1	1
INC direct	(direct) (direct) + 1	2	1
INC @Ri	@Ri @Ri +1	1	1
INC DPTR	DPTR DPTR +1	1	2
MUL AB	Multiply A by B A low byte (A*B)	1	4

	B	high byte (A* B)		
SUBB A, Rn	A	A - Rn - C	1	1
SUBB A, direct	A	A - (direct) - C	2	1
SUBB A, @Ri	A	A - @Ri - C	1	1
SUBB A, #data	A	A - data - C	2	1

✓ Logical Instructions

Mnemonics		Description	Bytes	Instruction Cycles
ANL A, Rn	A	A AND Rn	1	1
ANL A, direct	A	A AND (direct)	2	1
ANL A, @Ri	A	A AND @Ri	1	1
ANL A, #data	A	A AND data	2	1
ANL direct, A	(direct)	(direct) AND A	2	1
ANL direct, #data	(direct)	(direct) AND data	3	2
CLR A	A	00H	1	1
CPL A	A	A	1	1
ORL A, Rn	A	A OR Rn	1	1
ORL A, direct	A	A OR (direct)	1	1
ORL A, @Ri	A	A OR @Ri	2	1
ORL A, #data	A	A OR data	1	1
ORL direct, A	(direct)	(direct) OR A	2	1
ORL direct, #data	(direct)	(direct) OR data	3	2
RL A	Rotate accumulator left		1	1
RLC A	Rotate accumulator left through carry		1	1
RR A	Rotate accumulator right		1	1
RRC A	Rotate accumulator right through carry		1	1
SWAP A	Swap nibbles within Acumulator		1	1
XRL A, Rn	A	A EXOR Rn	1	1
XRL A, direct	A	A EXOR (direct)	1	1
XRL A, @Ri	A	A EXOR @Ri	2	1
XRL A, #data	A	A EXOR data	1	1
XRL direct, A	(direct)	(direct) EXOR A	2	1
XRL direct, #data	(direct)	(direct) EXOR data	3	2

Data Transfer Instructions

Mnemonics		Description	Bytes	Instruction Cycles
MOV A, Rn	A	Rn	1	1
MOV A, direct	A	(direct)	2	1
MOV A, @Ri	A	@Ri	1	1
MOV A, #data	A	data	2	1
MOV Rn, A	Rn	A	1	1
MOV Rn, direct	Rn	(direct)	2	2
MOV Rn, #data	Rn	data	2	1
MOV direct, A	(direct)	A	2	1
MOV direct, Rn	(direct)	Rn	2	2
MOV direct1, direct2	(direct1)	(direct2)	3	2

MOV direct, @Ri	(direct) @Ri	2	2
MOV direct, #data	(direct) #data	3	2
MOV @Ri, A	@Ri A	1	1
MOV @Ri, direct	@Ri (direct)	2	2
MOV @Ri, #data	@Ri data	2	1
MOV DPTR, #data16	DPTR data16	3	2
MOVC A, @A+DPTR	A Code byte pointed by A + DPTR	1	2
MOVC A, @A+PC	A Code byte pointed by A + PC	1	2
MOVC A, @Ri	A Code byte pointed by Ri 8-bit address)	1	2
MOVX A, @DPTR	A External data pointed by DPTR	1	2
MOVX @Ri, A	@Ri A (External data - 8bit address)	1	2
MOVX @DPTR, A	PUSH direct (SP) (direct)	2	2
@DPTR	A(External data - 16bit address)	1	2

POP direct	(direct) ←(SP)	2	2
XCH Rn	Exchange A with Rn	1	1
XCH direct	Exchange A with direct byte	2	1
XCH @Ri	Exchange A with indirect RAM	1	1
XCHD A, @Ri	Exchange least significant nibble of A with that of indirect RAM	1	1

Boolean Variable Instructions

Mnemonics	Description	Bytes	Instruction Cycles
CLR C	C-bit 0	1	1
CLR bit	bit 0	2	1
SET C	C 1	1	1
SET bit	bit 1	2	1
CPL C	C	1	1
CPL bit	bit	2	1
ANL C, /bit	C C .	2	1
ANL C, bit	C C. bit	2	1
ORL C, /bit	C C +	2	1
ORL C, bit	C C + bit	2	1
MOV C, bit	C bit	2	1
MOV bit, C	bit C	2	2

Program Branching Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ACALL addr11	PC + 2 (SP); addr 11 PC	2	2
AJMP addr11	Addr11 PC	2	2
CJNE A, direct, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE A, #data, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE Rn, #data, rel	Compare with Rn, jump (PC + rel) if not equal	3	2
CJNE @Ri, #data, rel	Compare with @Ri A, jump (PC + rel) if not equal	3	2
DJNZ Rn, rel	Decrement Rn, jump if not zero	2	2
DJNZ direct, rel	Decrement (direct), jump if not zero	3	2
JC rel	Jump (PC + rel) if C bit = 1	2	2
JNC rel	Jump (PC + rel) if C bit = 0	2	2
JB bit, rel	Jump (PC + rel) if bit = 1	3	2
JNB bit, rel	Jump (PC + rel) if bit = 0	3	2
JBC bit, rel	Jump (PC + rel) if bit = 1	3	2
JMP @A+DPTR	A+DPTR PC	1	2
JZ rel	If A=0, jump to PC + rel	2	2
JNZ rel	If A 0 , jump to PC + rel	2	2
LCALL addr16	PC + 3 (SP), addr16 PC	3	2
LJMP addr 16	Addr16 PC	3	2
NOP	No operation	1	1
RET	(SP) PC	1	2
RETI	(SP) PC, Enable Interrupt	1	2
SJMP rel	PC + 2 + rel PC	2	2
JMP @A+DPTR	A+DPTR PC	1	2
JZ rel	If A = 0. jump PC+ rel	2	2
JNZ rel	If A 0, jump PC + rel	2	2
NOP	No operation	1	1

8051 Addressing Modes:

8051 has four addressing modes.

1. Immediate Addressing: Data is immediately available in the instruction. For example - ADD A, #77; Adds 77 (decimal) to A and stores in A ADD A, #4DH; Adds 4D (hexadecimal) to A and stores in A MOV DPTR, #1000H; Moves 1000 (hexadecimal) to data pointer

2. Bank Addressing or Register Addressing:

This way of addressing accesses the bytes in the current register bank. Data is available in the register specified in the instruction. The register bank is decided by 2 bits of Processor Status Word (PSW). For example- ADD A, R0; Adds content of R0 to A and stores in A.

3. Direct Addressing:

The address of the data is available in the instruction. For example - MOV A, 088H; Moves content of SFR TCON (address 088H)to A

4. Register Indirect Addressing:

The address of data is available in the R0 or R1 registers as specified in the instruction. For example - MOV A, @R0 moves content of address pointed by R0 to A .

5. External Data Addressing:

Pointer used for external data addressing can be either R0/R1 (256 byte access) or DPTR (64kbyte access).

For example -

MOVX A, @R0; Moves content of 8-bit address pointed by R0 to A

MOVX A, @DPTR; Moves content of 16-bit address pointed by DPTR to A

6. External Code Addressing:

Sometimes we may want to store non-volatile data into the ROM e.g. look-up tables. Such data may require reading the code memory. This may be done as follows -

MOVC A, @A+DPTR; Moves content of address pointed by A+DPTR to A

MOVC A, @A+PC; Moves content of address pointed by A+PC to A.

Assembly language Programming:

Character transmission using a time delay:

A program shown below takes the character in 'A' register, transmits it, delays for transmission time, and returns to the calling program. Timer-1 is used to set the baud rate, which is 1200 baud in this program.

The delay for one character transmission (in Mode 1 i.e.10 bits) is 10/2400 = 0.00833 seconds

Or, 8.33 milliseconds

Hence software delay of 10ms is used. Timer-1 generates a baud rate close to 1200. Using a 12MHz crystal, the reload value is

$$256 - \frac{12X10^6}{32X12X2400} = 229.958$$

Or, 230 i.e. E6H .

This gives rise to an actual baud rate of 1202. SMOD is programmed to be 0.

Assembly language Program is as follows:

```
RELOAD   EQU 0E6H  ; defining constant for reload value for baudrate generation
DELAY    EQU 0A6H  ; defining constant for 1 millisecond
DLYLSB   EQU 0AH   ; defining constant for 10 millisecond
DLYMSB   EQU 00H   ; defining constant

ORG 0C00H          ; Org directive
ANL PCON, #7FH     ; SMOD = 0
ANL TMOD, #0FH     ; Alter only Timer-1
ORL TMOD, #20H     ; program Timer-1 in mode-2
MOV TH1,  #RELOAD  ; program reload value to TH1
SETB TR1           ; enable Timer-1 run bit (start timer)
MOV SCON, #40H     ; Serial port in mode-1 (receive not enabled)

TRMIT: MOV SBUF, #'A'   ; send the ASCII value of 'A'
       ACALL TRMITTIME  ; wait for DELAY
       SJMP TRMIT       ; again transmit
```

; Code to wait for the transmission to complete

The subroutine TRMITTIME generates a delay of about 10ms. With a clock of 12MHz, one instruction cycle time is

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

The loop "MILSEC" generates a delay of about 1×10^{-3} sec. This gets executed 10 times for a total delay of 10×10^{-3} sec or 10ms

```
TRITTIME:       MOV A, #DLYLSB
                MOV B, #DLYMSB
                ACALL SOFTIME
                RET
SOFTIME:        PUSH 07H
                PUSH A
                ORL A, B
                CJNE A, #00H, OK
                POP A
                SJMP DONE

OK:             POP A

TIMER:          MOV R7, #DELAY  ; Generate delay for 1 millisecond
MILSEC:         NOP
                NOP
                NOP
                NOP
                DJNZ R7, MILSEC
                NOP
                NOP
                DEC A
                CJNE A, #0FFH, NO ROLL
                DEC B

NO ROLL:        CJNE A, #00H, TIMER
DONE:           POP 07H
                RET
END
```

Introduction to 16-bit microcontrollers:

A microcontroller is a small, low-cost computer-on-a-chip which usually includes: –

- ✓ An 8 or 16 bit microprocessor (CPU).
- ✓ A small amount of RAM.
- ✓ Programmable ROM and/or flash memory.
- ✓ Parallel and/or serial I/O.
- ✓ Timers and signal generators.
- ✓ Analog to Digital (A/D) and/or Digital to Analog (D/A) conversion.

• Often used to run dedicated code that controls one or more tasks in the operation of a device or a system. • Also called embedded controllers, because the microcontroller and support circuits are often built into, or embedded in, the devices they control.

• Devices that utilize microcontrollers include car engines, consumer electronics (VCRs, microwaves, cameras, pagers, cell phones ...), computer peripherals (keyboards, printers, and modems.), test/measurement equipment (signal generators, multimeters, oscilloscopes …).

• Microcontrollers usually must have low-power requirements (~. 05 - 1 W as opposed to ~10 - 50 W for general purpose desktop CPUs) since many devices they control are battery-operated.

Examples: Motorola's 68HC11, 68HC12, AMD 29K, Zilog's Z8, Z80, Intel's 8052, Microchip's PIC Low-power, embedded versions of desktop CPUs: e.g Intel's 80486

A Typical 68HC12 has the following components on the chip:

A 16-bit central processing unit (CPU12):

- ✓ 20-Bit ALU. Instruction Queue.
- ✓ Enhanced Indexed Addressing.
- ✓ Fuzzy Logic Instructions.

32-Kbyte Flash EEPROM with 2-Kbyte Erase-Protected Boot Block.

768-Byte EEPROM.

1-Kbyte RAM with Single-Cycle Access for Aligned or Misaligned Read/Write.

8-Channel, 8-Bit Analog-to-Digital (A/D) Converter.

8-Channel Timer.

16-Bit Pulse Accumulator:

- ✓ External Event Counting,Gated Time Accumulation.

Pulse-Width Modulator:

- ✓ 8-Bit, 4-Channel or 16-Bit, 2-Channel
- ✓ Separate Control for Each Pulse Width and Duty Cycle.

UNIT V

ADVANCED TOPICS

Programming 8051 Timers: Using Timers to Measure Time:

One of the primary uses of timers is to measure time. When a timer is in interval timer mode (as opposed to event counter mode) and correctly configured, it will increment by 1 every machine cycle. A single machine cycle consists of 12 crystal pulses. Thus a running timer will be incremented: 11,059,000 / 12 = 921,583 times per second.

Unlike instructions which require 1 machine cycle, others 2, and others 4--the timers are consistent: They will always be incremented once per machine cycle. Thus if a timer has counted from 0 to 50,000 you may calculate: 50,000 / 921,583 = .0542.0542 seconds have passed. To execute an event once per second you'd have to wait for the timer to count from 0 to 50,000 18.45times. To calculate how many times the timer will be incremented in .05 seconds, a simple multiplication can be done: 0 .05 * 921,583 = 46,079.15. This tells us that it will take .05 seconds (1/20th of a second) to count from 0 to 46.0. To work with timers is to control the timers and initialize them.

The TMOD SFR:

TMOD (Timer Mode): The TMOD SFR is used to control the mode of operation of both timers. Each bit of the SFR gives the microcontroller specific information concerning how to run a timer. The high four bits (bits 4 through 7) relate to Timer 1whereas the low four bits (bits 0 through 3) perform the exact same functions, but for timer 0. The modes of operation are:

Table 5.1 modes of Timer

TxM1	TxM0	Timer Mode	Description of Mode
0	0	0	13-bit Timer.
0	1	1	16-bit Timer
1	0	2	8-bit auto-reload
1	1	3	13-bit Time Mode (mode 0)

Timer mode "0" is a 13-bit timer. When the timer is in 13-bit mode, TLx will count from 0 to 31. When TLx is incremented from 31, it will "reset" to 0 and increment THx. Thus, effectively, only 13 bits of the two timer bytes are being used: bits 0-4 of TLx and bits 0-7 of THx. The timer can only contain 8192 values. If you set a 13-bit timer to 0, it will overflow back to zero 8192 machine cycles later.

16-bit Time Mode (mode 1) Timer mode "1" is a 16-bit timer. TLx is incremented from 0 to 255. When TLx is incremented from 255, it resets to 0 and causes THx to be incremented by 1. Since this is a full 16-bit timer, the timer may contain up to 65536 distinct values. If you set a 16-bit timer to 0, it will overflow back to 0 after 65,536 machine cycles. 8-bit Time Mode (mode 2) Timer mode "2" is an 8-bit auto-reload mode. When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself. Thus, TLx starts counting up. When TLx reaches 255 and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be reset to the value stored in THx. For example, if TH0 holds the value FDh and TL0 holds the value FEh values of TH0 and TL0 for a few machine cycles: The value of TH0 never changed. When we use mode 2 you almost always set THx to a known value and TLxis the SFR that is constantly incremented. The benefit of auto-reload mode is the timer always has a value from 200 to 255. If you use mode 0 or 1, you'd have to check in code to see if the timer had overflowed and, if so, reset the timer to 200. This takes precious instructions of execution time to check the value and/or to reload it. Whenyou use mode 2 the microcontroller takes care of this. Auto-reload mode is very commonly used for establishing a baud rate in Serial Communications.

Table 5.2 mode 2 operation

Machine Cycle	TH0 Value	TL0 Value

1	FDh	FEh
2	FDh	FEh
3	FDh	FEh
4	FDh	FEh
5	FDh	FEh
6	FDh	FEh
7	FDh	FEh

Split Timer Mode (mode 3)

Timer mode "3" is a split-timer mode. When Timer 0 is placed in mode 3, it essentially becomes two separate 8-bit timers. Timer 0 is TL0 and Timer 1 is TH0. Both timers count from 0 to 255 and overflow back to 0. All the bits that are related to Timer 1 will now be tied to TH0. While Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be put into modes 0, 1 or 2 normally--however, you may not start or stop the real timer 1 since the bits that do that are now linked to TH0. The real timer 1,e, will be incremented every machine cycle always. The only real use in split timer mode is if you need to have two separate timers and, additionally, a baud rate generator you can use the real Timer 1 as a baud rate generator and use TH0/TL0 as two separate timers.

Reading the Timer

There are two common ways of reading the value of a 16-bit timer; which you use depends on your specific application. You may either read the actual value of the timer as a 16-bit number, or you may simply detect when the timer has overflowed.

Reading the value of a Timer

If timer is in an 8-bit mode either 8-bit Auto Reload mode or in split timer mode, you simply read the 1-byte value of the timer. With a 13-bit or16-bit timer the timer value was 14/255 (High byte 14, low byte 255) but you read 15/255.Because you read the low byte as 255. But when you executed the next instruction a small amount of time passed--but enough for the timer to increment again at which time the value rolled over from 14/255 to 15/0. But in the process you've read the timer as being 15/255.

You read the high byte of the timer, then read the low byte, then read the high byte again. If the high byte read the second time is not the same as the high byte read the first time you repeat the cycle. In code, this would appear as:

```
REPEAT: MOV A, TH0
MOV R0, TL0
CJNE A, TH0, REPEAT
```

In this case, we load the accumulator with the high byte of Timer 0. We then load R0 with the low byte of Timer 0. Finally, we check to see if the high byte we read out of Timer 0--which is now stored in the Accumulator--is the same as the current Timer 0 high byte. We do by going back to REPEAT. When the loop exits we will have the low byte of the timer in R0 and the high byte in the Accumulator.

Another much simpler alternative is to simply turn off the timer run bit (i.e. CLR TR0), read the timer value, and then turn on the timer run bit (i.e. SETB TR0).

Detecting Timer Overflow

Whenever a timer overflows from its highest value back to 0, the microcontroller automatically sets the TFx bit in the TCON register. if TF1 is set it means that timer 1 has overflowed.

We can use this approach to cause the program to execute a fixed delay. it takes the 8051 1/20thof a second to count from 0 to 46,079. However, the TFx flag is set when the timer overflows back to 0. Thus, if we want to use the TFx flag to indicate when 1/20th of a second has passed we must set the timer initially to 65536 less 46079, or 19,457. If we set the timer to 19,457, 1/20th of a second later the timer will overflow.

The following code to execute a pause of 1/20th of a second:

```
MOV TH0, #76; High byte of 19,457 (76 * 256 = 19,456)
MOV TL0, #01; Low byte of 19,457 (19,456 + 1 = 19,457)
MOV TMOD, #01; Put Timer 0 in 16-bit mode
SETB TR0; Make Timer 0 start counting
 JNB TF0, $; If TF0 is not set, jump back to this same instruction
```

In the above code the first two lines initialize the Timer 0 starting value to 19,457. The next two instructions configure timer 0 and turn it on. Finally, the last instruction JNB TF0, $, reads "Jump, if TF0 is not set, back to this same instruction." The "$" operand means, in most assemblers, the address of the current instruction. Thus as long as the timer has not overflowed and the TF0 bit has not been set the program will keep executing this same instruction. After 1/20th of a second timer 0 will overflow, set the TF0 bit, and program execution will then break out of the loop.

Serial Port Programming: 8051 Serial Communication:

One of the 8051's many powerful features -integrated UART, known as a serial port to easily read and write values to the serial port instead of turning on and off one of the I/O lines in rapid succession to properly "clock out" each individual bit, including start bits, stop bits and parity bits.

- Setting the Serial Port Mode configures it by specifying 8051 how many data bits we want, the baud rate we will be using and how the baud rate will be determined. First, let's present the "Serial Control" (SCON) SFR and define what each bit of the SFR represents:

Table 5.3 Definition of SCON SFR

Bit	Name	Bit Address	Explanation of Function
7	SM0	9Fh	Serial port mode bit 0
6	SM1	9Eh	Serial port mode bit 1.
5	SM2	9Dh	Mutli processor Communications Enable
4	REN	9Ch	Receiver Enable. This bit must be set in order to receive Characters.

3	TB8	9Bh	Transmit bit 8. The 9th bit to transmit in mode 2 and 3.
2	RB8	9AH	Receive bit 8. The 9th bit received in mode 2 and 3.
1	T1	99h	Transmit Flag. Set when a byte has been completely Transmitted.
0	RI	98h	Receive Flag. Set when a byte has been completely Received.

Additionally, it is necessary to define the function of SM0 and SM1 by an additional table: Table 5.4 SCON as serial Port

Table 5.4 Modes of SCON

SM0	SM1	Serial Mode	Explanation Baud Rate
0	0	0	0 8-bit Shift Register Oscillator / 12
0	1	1	8-bit UART Set by Timer 1 (*)
1	0	2	9-bit UART Oscillator / 32 (*)
1	1	3	9-bit UART Set by Timer 1 (*)

The SCON SFR allows us to configure the Serial Port. The first four bits (bits 4 through 7) are configuration bits:

Bits SM0 and SM1 is to set the serial mode to a value between 0 and 3, inclusive as in table above selecting the Serial Mode selects the mode of operation (8-bit/9-bit, UART or Shift Register) and also determines how the baud rate will be calculated. In modes 0 and 2 the baud rate is fixed based on the oscillator's frequency. In modes 1 and 3 the baud rate is variable based on how often Timer 1 overflows.

The next bit, SM2, is a flag for " Multiprocessor communication whenever a byte has been received the 8051 will set the "RI" (Receive Interrupt) flag to let the program know that a byte has been received and that it needs to be processed.

However, when SM2 is set the "RI" flag will only be triggered if the 9th bit received was a "1". if SM2 is set and a byte is received whose 9th bit is clear, the RI flag will never be set .You will almost always want to clear this bit so that the flag is set upon reception of any character.

The next bit, REN, is "Receiver Enable." is set indicate to data received via the serial port.

The last four bits (bits 0 through 3) are operational bits. They are used when actually sending and receiving data--they are not used to configure the serial port.

The TB8 bit is used in modes 2 and 3. In modes 2 and 3, a total of nine data bits are transmitted. The first 8 data bits are the 8 bits of the main value, and the ninth bit is taken from TB8. If TB8 is set and a value is written to the serial port, the data's bits will be written to the serial line followed by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear."

The RB8 also operates in modes 2 and 3and functions essentially the same way as TB8, but on the reception side. When a byte is received in modes 2 or 3, a total of nine bits are received. In

this case, the first eight bits received are the data of the serial byte received and the value of the ninth bit received will be placed in RB8.TI means "Transmit Interrupt."

When a program writes a value to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" the serial port. If the program were to write another byte to the serial port before the first byte was completely output, the data being sent would be garbled. Thus, the8051 lets the program know that it has "clocked out" the last byte by setting the TI bit.

When the TI bit is set, the program may assume that the serial port is "free" and ready to send the next byte. Finally, the RI bit means "Receive Interrupt." It functions similarly to the "TI" bit, but it indicates that a byte has been received. Whenever the 8051 has received a complete byte it will trigger the RI bit to let the program know that it needs to read the value quickly, before another byte is read.

- Setting the Serial Port Baud Rate

Once the Serial Port Mode has been configured, the program must configure the serial port's baud rate. This only applies to Serial Port modes 1 and 3. The Baud Rate is determined based on the oscillator's frequency when in mode 0 and 2. In mode 0, the baud rate is always the oscillator frequency divided by 12. This means if you're crystal is 1.059 MHz; mode 0 baud rate will always be 921,583 baud. In mode 2 the baud rate is always the oscillator frequency divided by 64, so a 11.059Mhz crystal speed will yield a baud rate of172, 797.

In modes 1 and 3, the baud rate is determined by how frequently timer 1 overflows. The more frequently timer 1 overflows, the higher the baud rate. There are many ways one can cause timer 1 to overflow at a rate that determines a baud rate, but the most common method is to put timer 1 in 8-bit auto-reload mode (timer mode2) and set a reload value (TH1) that causes Timer 1 to overflow at a frequency appropriate to generate a baud rate.

To determine the value that must be placed in TH1 to generate a given baud rate,

(Assuming PCON.7 is clear).

TH1 = 256 - ((Crystal / 384) / Baud)

ITH1 = 256 - ((Crystal / 192) / Baud)

If PCON.7 is set then the baud rate is effectively doubled, thus the equation becomes:

TH1 = 256 - ((Crystal / 192) / Baud)

For example, if we have an 11.059 MHz crystal and we want to configure the serial port to

19,200 baud we try plugging it in the first equation:

TH1 = 256 - ((Crystal / 384) / Baud)

TH1 = 256 - ((11059000 / 384) / 19200)

TH1 = 256 - ((28,799) / 19200)

TH1 = 256 - 1.5 = 254.5

To obtain 19,200 baud on a 11.059Mhz crystal we'd have to set TH1 to 254.5. If we set it to 254 we will have achieved 14,400 baud and if we set it to 255 we will have achieved 28,800 baud.

To achieve 19,200 baud we simply need to set PCON.7 (SMOD). When we do this we double the baud rate and utilize the second equation mentioned above. Thus we have:

TH1 = 256 - ((Crystal / 192) / Baud)

TH1 = 256 - ((11059000 / 192) / 19200)

TH1 = 256 - ((57699) / 19200)

TH1 = 256 - 3 = 253

Therefore, to obtain 19,200 baud with an 11.059MHz crystal we must:

1) Configure Serial Port mode 1 or 3.

2) Configure Timer 1 to timer mode 2 (8-bit auto reload).

3) Set TH1 to 253 to reflect the correct frequency for 19,200 baud.

4) Set PCON.7 (SMOD) to double the baud rate.

Writing to the Serial Port

Once the Serial Port has been properly configured as explained above, the serial port is ready to be used to send data and receive data.

To write a byte to the serial writes the value to the SBUF (99h) SFR. For example, if you wanted to send the letter "A" to the serial port, it could be accomplished as easily as:

```
MOV SBUF, #'A'
```

Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port. Obviously transmission is not instantaneous--it takes a measureable amount of time to transmit. And since the 8051 does not have a serial output buffer we need to be sure that a character is completely transmitted before we try to transmit the next character.

The 8051 lets us know when it is done transmitting a character by setting the TI bit in SCON. When this bit is set the last character has been transmitted and that send the next character, if any. Consider the following code segment:

```
CLR TI; be sure the bit is initially clear

MOV SBUF, #'A'; Send the letter 'A' to the serial port

JNB TI, $; Pause until the RI bit is set.
```

The above three instructions will successfully transmit a character and wait for the TI bit to be set before continuing. The last instruction says "Jump if the TI bit is not set to $"— $, in most

assemblers, means "the same address of the current instruction." Thus the 8051 will pause on the JNB instruction until the TI bit is set by the 8051 upon successful transmission of the character.

- **Reading the Serial Port**

Reading data received by the serial port is equally easy. To read a byte from the serial port one just needs to read the value stored in the SBUF (99h) SFR after the 8051 has automatically set the RI flag in SCON.

For example, if your program wants to wait for a character to be received and subsequently read it into the Accumulator, the following code segment may be used:

JNB RI, $; Wait for the 8051 to set the RI flag

MOV A, SBUF; Read the character from the serial port

The first line of the above code segment waits for the 8051 to set the RI flag; again, the8051 sets the RI flag automatically when it receives a character via the serial port. So as long as the bit is not set the program repeats the "JNB" instruction continuously. Once the RI bit is set upon character reception the above condition automatically fails and program flow falls through to the "MOV" instruction which reads the value.

5.1 Interrupt Programming:

What Events can trigger interrupts, and where do they go?

The following events will cause an interrupt:

- Timer 0 Overflow
- Timer 1 Overflow.
- Reception/Transmission of Serial Character.
- External Event 0.
- External Event 1.

To distinguish between various interrupts and executing different code depending on what interrupt was triggered 8051may be jumping to a fixed address when a given interrupt occurs.

Table 5.5 Interrupt handling

Interrupt	Flag	Interrupt Handler Address
External 0	IE0	0003h
Timer 0	TF0	000Bh
External 1	IE1	0013h
Timer 1	TF1	001Bh
Serial	RI/TI	0023h

If Timer 0 overflows (i.e., the TF0 bit is set), the main program will be temporarily suspended and control will jump to 000BH if we have code at address 0003H that handles the situation of Timer 0 overflowing.

- ✓ Setting Up Interrupts

By default at power up, all interrupts are disabled. Even if, for example, the TF0 bit is set, the 8051 will not execute the interrupt. Your program must specifically tell the 8051 that it wishes to enable interrupts and specifically which interrupts it wishes to enable. Your program may enable and disable interrupts by modifying the IE SFR (A8h):

Table 5.6 Interrupt and address

Bit	Name Bit	Address	Explanation of Function
7	EA	AFh	Global Interrupt Enable/Disable
6		AFh	Undefined
5		ADh	Undefined
4	ES	ACh	Enable Serial Interrupt
3	ET1	ABh	Enable Timer 1 Interrupt
2	EX1	AAh	Enable External 1 Interrupt
1	ET0	A9h	Enable Timer 0 Interrupt
0	EX0	A8h	Enable External 0 Interrupt

Each of the 8051'sinterrupts has its own bit in the IE SFR. You enable a given interrupt by setting the corresponding bit. For example, if you wish to enable Timer 1 Interrupt, you would execute either:

MOV IE, #08h || SETB ET1

Both of the above instructions set bit 3 of IE, thus enabling Timer 1 Interrupt. Once Timer 1 Interrupt is enabled, whenever the TF1 bit is set, the 8051 will automatically put "on hold" the main program and execute the Timer 1 Interrupt Handler at address 001Bh. However, before Timer 1 Interrupt (or any other interrupt) is truly enabled, you must also set bit 7 of IE.

Bit 7, the Global Interrupt Enable/Disable, enables or disables all interrupts simultaneously. That is to say, if bit 7 is cleared then no interrupts will occur, even if all the other bits of IE are set. Setting bit 7 will enable all the interrupts that have been selected by setting other bits in IE. This is useful in program execution if you have time-critical code that needs to execute. In this case, you may need the code to execute from start to finish without any interrupt getting in the way. To accomplish this you can simply clear bit 7 of IE (CLR EA) and then set it after your time critical code is done.

To enable the Timer 1 Interrupt execute the following two instructions:

SETB ET1

SETB EA

Thereafter, the Timer 1 Interrupt Handler at 01Bh will automatically be called whenever the TF1 bit is set (upon Timer 1 overflow).

- ✓ Polling Sequence

The 8051 automatically evaluates whether an interrupt should occur after every instruction. When checking for interrupt conditions, it checks them in the following order:

1) External 0 Interrupt

2) Timer 0 Interrupt

3) External 1 Interrupt

4) Timer 1 Interrupt

5) Serial Interrupt

- ✓ Interrupt Priorities

The 8051 offers two levels of interrupt priority: high and low. By using interrupt priorities you may assign higher priority to certain interrupt conditions. For example, you may have enabled Timer 1 Interrupt which is automatically called every time Timer 1 overflows. Additionally, you may have enabled the Serial Interrupt which is called every time a character is received via the serial port. However, you may consider that receiving a character is much more important than the timer interrupt. In this case, if Timer 1 Interrupt is already executing you may wish that the serial interrupt itself interrupts the Timer 1 Interrupt. When the serial interrupt is complete, control passes back to Timer 1 Interrupt and finally back to the main program. You may accomplish this by assigning a high priority to the Serial Interrupt and a low priority to the Timer 1 Interrupt.

Interrupt priorities are controlled by the IPSFR (B8h). The IP SFR has the following format:

Bit Name Bit Address Explanation of Function

7	Undefined
6	Undefined
5	Undefined
4	PS BCh Serial Interrupt Priority
3	PT1 BBh Timer 1 Interrupt Priority
2	PX1 BAh External 1 Interrupt Priority
1	PT0 B9h Timer0 Interrupt Priority
0	PX0 B8h External 0 Interrupt Priority.

When considering interrupt priorities, the following rules apply:

- ✓ Nothing can interrupt a high-priority interrupt--not even another high priority interrupt.
- ✓ A high-priority interrupt may interrupt a low priority interrupt.
- ✓ A low-priority interrupt may only occur if no other interrupt is already executing.
- ✓ If two interrupts occur at the same time, the interrupt with higher priority will execute first. If both interrupts are of the same priority the interrupt which is serviced first by polling sequence will be executed first.

What Happens When an Interrupt Occurs?

When an interrupt is triggered, the following actions are taken automatically by the microcontroller:

- The current Program Counter is saved on the stack, low-byte first.

- Interrupts of the same and lower priority are blocked.
- In the case of Timer and External interrupts, the corresponding interrupt flag is set.
- Program execution transfers to the corresponding interrupt handler vector address.
- The Interrupt Handler Routine executes. Take special note of the third step: If the interrupt being handled is a Timer or External interrupt, the microcontroller automatically clears the interrupt flag before passing control to your interrupt handler routine.

✓ What Happens When an Interrupt Ends?

An interrupt ends when your program executes the RETI instruction. When the RETI instruction is executed the following actions are taken by the microcontroller:

- Two bytes are popped off the stack into the Program Counter to restore normal program execution.
- Interrupt status is restored to its pre-interrupt status.
- **Serial Interrupts**

Serial Interrupts are slightly different than the rest of the interrupts. This is due to the fact that there are two interrupt flags: RI and TI. If either flag is set, a serial interrupt is triggered. As you will recall from the section on the serial port, the RI bit is set when a byte is received by the serial port and the TI bit is set when a byte has been sent. This means that when your serial interrupt is executed, it may have been triggered because the RI flag was set or because the TI flag was set--or because both flags were set. Thus, your routine must check the status of these flags to determine what action is appropriate. Also, since the 8051does not automatically clear the RI and TI flags you must clear these bits in your interrupt handler.

pt handler. INT_SERIAL: JNB RI, CHECK_TI; If the RI flag is not set, we jump to check TI

MOV A, SBUF; If we got to this line, it's because the RI bit *was* set

CLR RI; Clear the RI bit after we've processed it

CHECK_TI: JNB TI, EXIT_INT; if the TI flag is not set, we jump to the exit point

CLR TI; Clear the TI bit before we send another character

MOV SBUF, #'A'; Send another character to the serial port

EXIT_INT: RETI

As you can see, our code checks the status of both interrupts flags. If both flags were set, both sections of code will be executed. Also note that each section of code clears its corresponding interrupt flag. If you forget to clear the interrupt bits, the serial interrupt will be executed over and over until you clear the bit. Thus it is very important that you always clear the interrupt flags in a serial interrupt.

✓ **Important Interrupt Consideration: Register Protection**

One very important rule applies to all interrupt handlers: Interrupts must leave the processor in the same state as it was in when the interrupt initiated. Remember, the idea behind interrupts is that the main program isn't aware that they are executing in the "background." However, consider the following code:

```
CLR C; Clear carry
MOV A, #25h; Load the accumulator with 25h
ADDC A, #10h; Add 10h, with carry
```

After the above three instructions are executed, the accumulator will contain a value of35h. But what would happen if right after the MOV instruction an interrupt occurred. During this interrupt, the carry bit was set and the value of the accumulator was changed to 40h. When the interrupt finished and control was passed back to the main program, the ADDC would add 10h to40h, and additionally add an additional 1h because the carry bit is set. In this case, the accumulator will contain the value 51h at the end of execution. In this case, the main program has seemingly calculated the wrong answer. How can25h + 10h yield 51h as a result? It doesn't make sense. A programmer that was unfamiliar with interrupts would be convinced that the microcontroller was damaged in some way, provoking problems with mathematical calculations.

What has happened, in reality, is the interrupt did not protect the registers it used.

To insure that the value of the accumulator is the same at the end of the interrupt as it was at the beginning. This is generally accomplished with a PUSH and POP sequence. For example:

```
PUSH ACC
PUSH PSW
 MOV A, #0FFh
ADD A, #02h
POP PSW
POP ACC
```

The guts of the interrupt is the MOV instruction and the ADD instruction. However, these two instructions modify the Accumulator (the MOV instruction) and also modify the value of the carry bit (the ADD instruction will cause the carry bit to be set). Since an interrupt routine must guarantee that the registers remain unchanged by the routine, the routine pushes the original values onto the stack using the PUSH instruction. It is then free to use the registers it protected to its heart's content. Once the interrupt has finished its task, it pops the original values back into the registers. When the interrupt exits, the main program will never know the difference because the registers are exactly the same as they were before the interrupt executed.

In general, your interrupt routine must protect the following registers:

- PSW
- DPTR (DPH/DPL)
- PSW
- ACC
- B
- Registers R0-R7

PSW consists of many individual bits that are set by various 8051instructions. Always protect PSW by pushing and popping it off the stack at the beginning and end of your interrupts. It will not be allow executing the instruction: PUSH R0

Because depending on which register bank is selected, R0 may refer to either internal ram address 00h, 08h, 10h, or 18h.R0, in and of itself, is not a valid memory address that the PUSH and POP instructions can use. Thus, if you are using any "R" register in your interrupt routine, you will have to push that register's absolute address onto the stack instead of just saying PUSH R0. For example, instead of PUSH R0 you would execute: PUSH 00h.

Interfacing a Microprocessor to Keyboard

When you press a key on your computer, you are activating a switch. There are many different ways of making these switches. An overview of the construction and operation of some of the most common types.

- ✓ **Mechanical key switches**: In mechanical-switch keys, two pieces of metal are pushed together when you press the key. The actual switch elements are often made of a phosphor-bronze alloy with gold platting on the contact areas. The key switch usually contains a spring to return the key to the non pressed position and perhaps a small piece of foam to help damp out bouncing.
  Some mechanical key switches now consist of a molded silicon dome with a small piece of conductive rubber foam short two trace on the printed-circuit board to produce the key pressed signal.
- ✓ Mechanical switches are relatively inexpensive but they have several disadvantages. First, they suffer from contact bounce. A pressed key may make and break contact several times before it makes solid contact.
  Second, the contacts may become oxidized or dirty with age so they no longer make a dependable connection.
  Higher- quality mechanical switches typically have a rated life time of about 1 million keystrokes. The silicone dome type typically last 25 million keystrokes.
- ✓ **Membrane key switches**: These switches are really a special type of mechanical switches. They consist of a three-layer plastic or rubber sandwich.
  The top layer has a conductive line of silver ink running under each key position. The bottom layer has a conductive line of silver ink running under each column of keys.

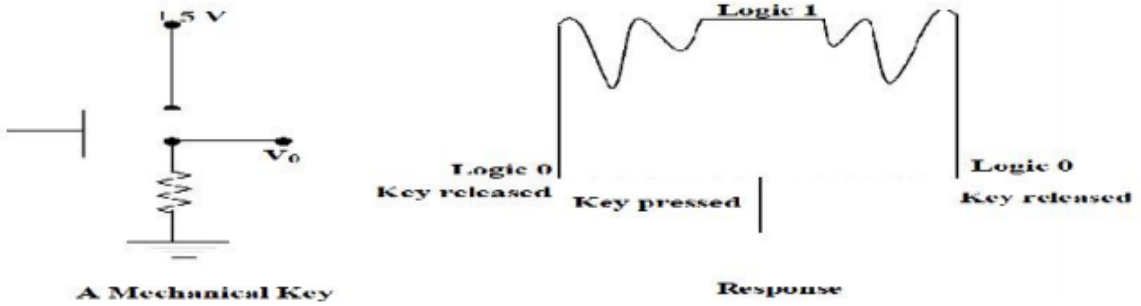


Fig .5.1 Mechanical key and its response to key press

The key board interfaced is a matrix keyboard. This key board is designed with a particular rows and columns. These rows and columns are connected to the microcontroller through its ports of

the micro controller 8051. We normally use 8*8 matrix key boards. So only two ports of 8051 can be easily connected to the rows and columns of the key board.

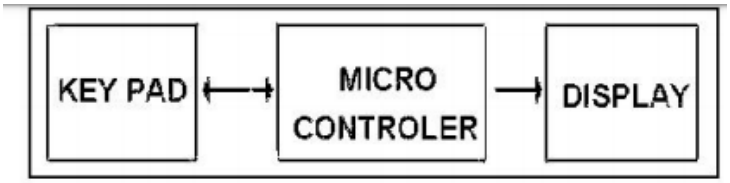
Whenever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a key is pressed only a bit in the port goes high which indicates microcontroller that the key is pressed. By this high on the bit key in the corresponding column is identified.

Once we are sure that one of key in the key board is pressed next our aim is to identify that key. To do this we firstly check for particular row and then we check the corresponding column the key board.

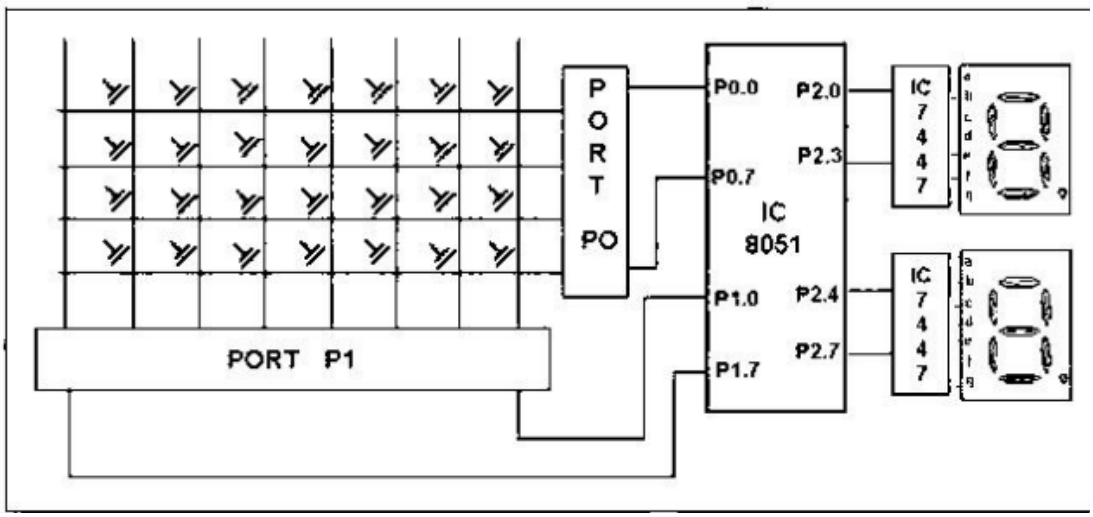
To check the row of the pressed key in the keyboard, one of the row is made high by making one of bit in the output port of 8051 high. This is done until the row is found out.

Once we get the row next out job is to find out the column of the pressed key. The column is detected by contents in the input ports with the help of a counter. The content of the input port is rotated with carry until the carry bit is set.

The contents of the counter is then compared and displayed in the display. This display is designed using a seven segment display and a BCD to seven segment decoder IC 7447. The BCD equivalent number of counter is sent through output part of 8051 displays the number of pressed key.



Interfacing keyboard with 8051



Interfacing to alpha-numeric display

- To give directions or data values to users, many microprocessor-controlled instruments and machines need to display letters of the alphabet and numbers. In systems where a large amount of data needs to be displayed a CRT is used to

display the data. In system where only a small amount of data needs to be displayed, simple digit-type displays are often used. • There are several technologies used to make these digit-oriented displays but we are discussing only the two major types.

- These are light emitting diodes (LED) and liquid-crystal displays (LCD)
- LCD displays use very low power, so they are often used in portable, battery-powered instruments. They do not emit their own light, they simply change the reflection of available light. Therefore, for an instrument that is to be used in low-light conditions, you have to include a light source for LCDs or use LEDs which emit their own light.

Interfacing Analog to Digital Data Converters

- In most of the cases, the PPI 8255 is used for interfacing the analog to digital converters with microprocessor
- The analog to digital converters is treaded as an input device by the microprocessor that sends an initializing signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
- The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.
- The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
- It may range anywhere from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
- The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.
- General algorithm for ADC interfacing contains the following steps:
  1. Ensure the stability of analog input, applied to the ADC.
  2. Issue start of conversion pulse to ADC
  3. Read end of conversion signal to mark the end of conversion processes.
  4. Read digital data output of the ADC as equivalent digital output.
  5. 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.

6. 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.

ADC 0808/0809:

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100μs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.

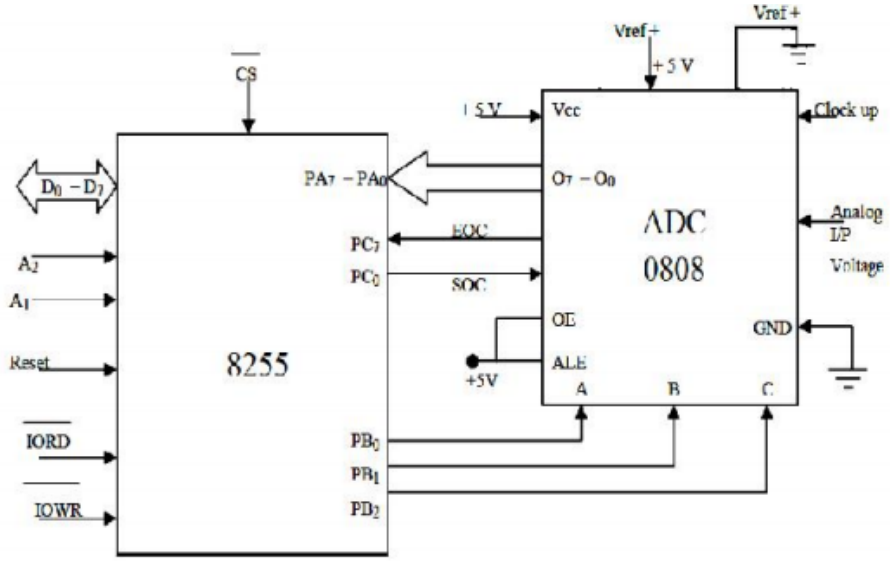
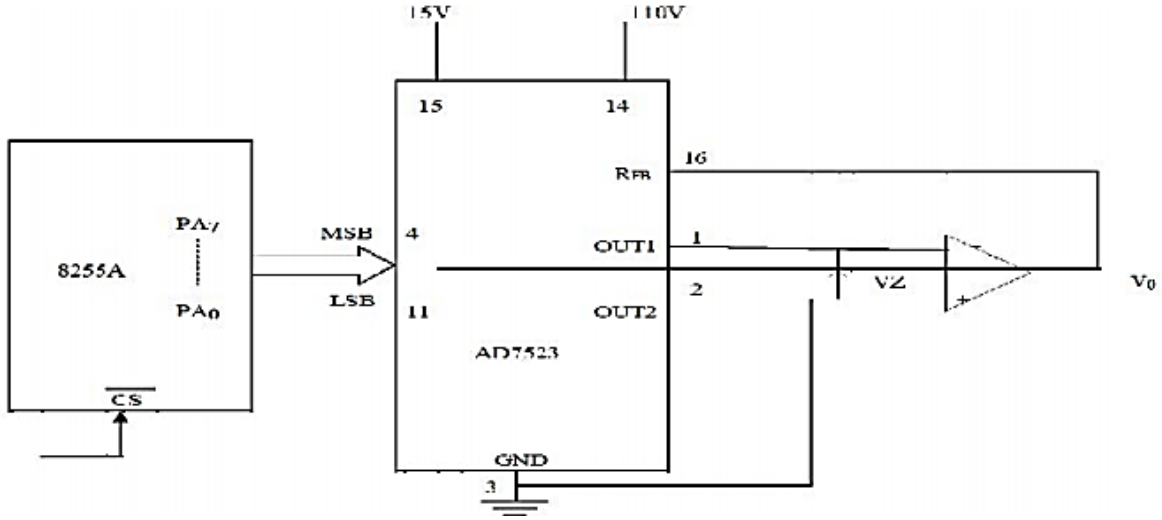


Fig. 5.4 Interfacing ADC with 8255 of microcontroller

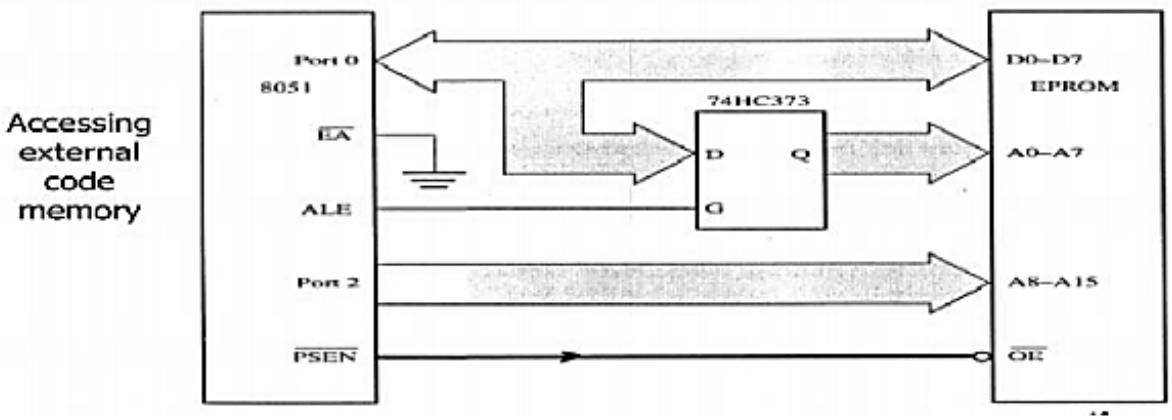
- These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.
- There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do no contain any internal sample and hold circuit. If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

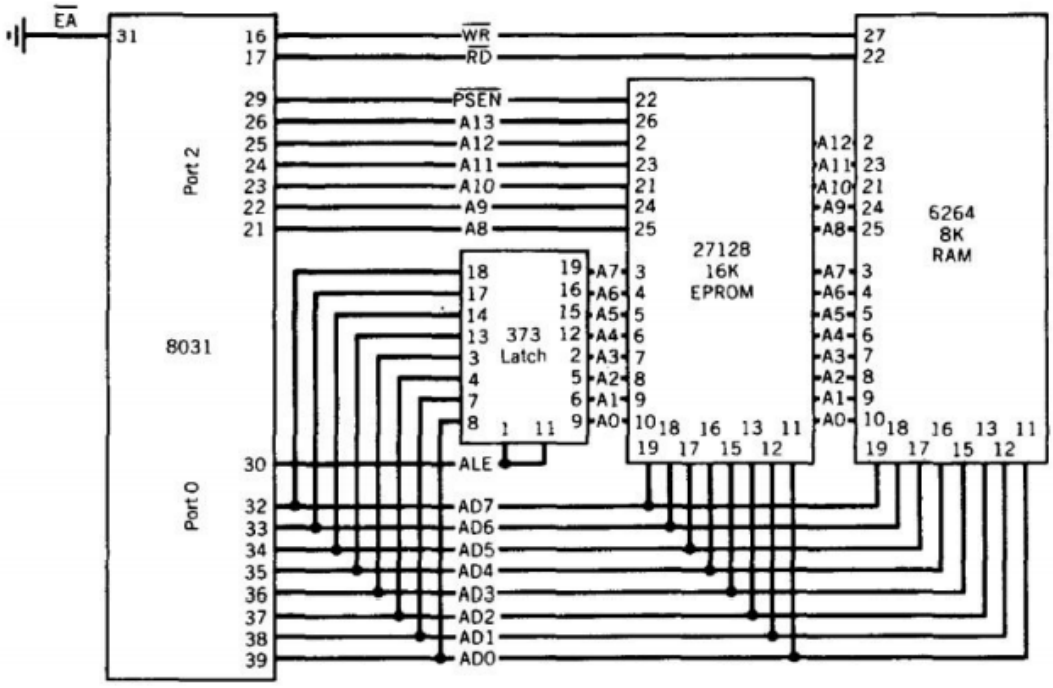
Interfacing Digital to Analog Converters:

The digital to analog converters convert binary number into their equivalent voltages. The DAC find applications in areas like digitally controlled gains, motors speed controls, programmable gain amplifiers etc. AD 7523 8-bit Multiplying DAC: This is a 16 pin DIP, multiplying digital to analog converter, containing R- 2R ladder for D-A conversion along with single pole double thrown NMOS switches to connect the digital inputs to the ladder.

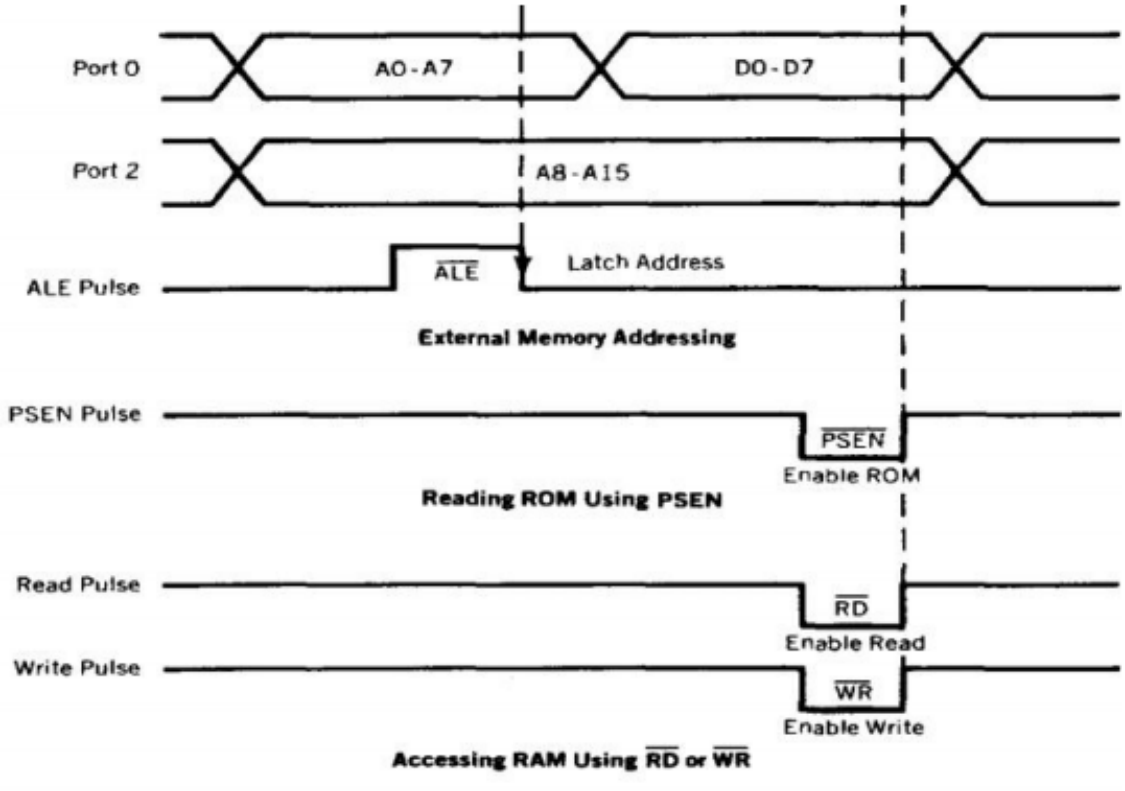


External Memory Interface:





Interfacing external memories with micro-controller



External memory timing

Stepper Motor Interface

The complete board consists of transformer, control circuit, keypad and stepper motor as shown in snap.

The circuit has inbuilt 5 V power supply so when it is connected with transformer it will give the supply to circuit and motor both. The 8 Key keypad is connected with circuit through which user can give the command to control stepper motor. The control circuit includes micro controller 89C51, indicating LEDs, and current driver chip ULN2003A. One can program the controller to control the operation of stepper motor. He can give different commands through keypad like, run clockwise, run anticlockwise, increase/decrease RPM, increase/decrease revolutions, stop motor, change the mode, etc. Unipolar stepper motor:- unipolar stepper motor has four coils. One end of each coil is tied together and it gives common terminal which is always connected with positive terminal of supply. The other ends of each coil are given for interface. Specific color code may also be given. Like in my motor orange is first coil (L1), brown is second (L2), yellow is third (L3), black is fourth (L4) and red for common terminal.

By means of controlling a stepper motor operation we can

1. Increase or decrease the RPM (speed) of it
2. Increase or decrease number of revolutions of it
3. Change its direction means rotate it clockwise or anticlockwise

To vary the RPM of motor we have to vary the PRF (Pulse Repetition Frequency). Number of applied pulses will vary number of rotations and last to change direction we have to change pulse sequence.

So all these three things just depends on applied pulses. Now there are three different modes to rotate this motor

1. Single coil excitation
2. Double coil excitation
3. Half step excitation

The table given below will give you the complete idea that how to give pulses in each mode

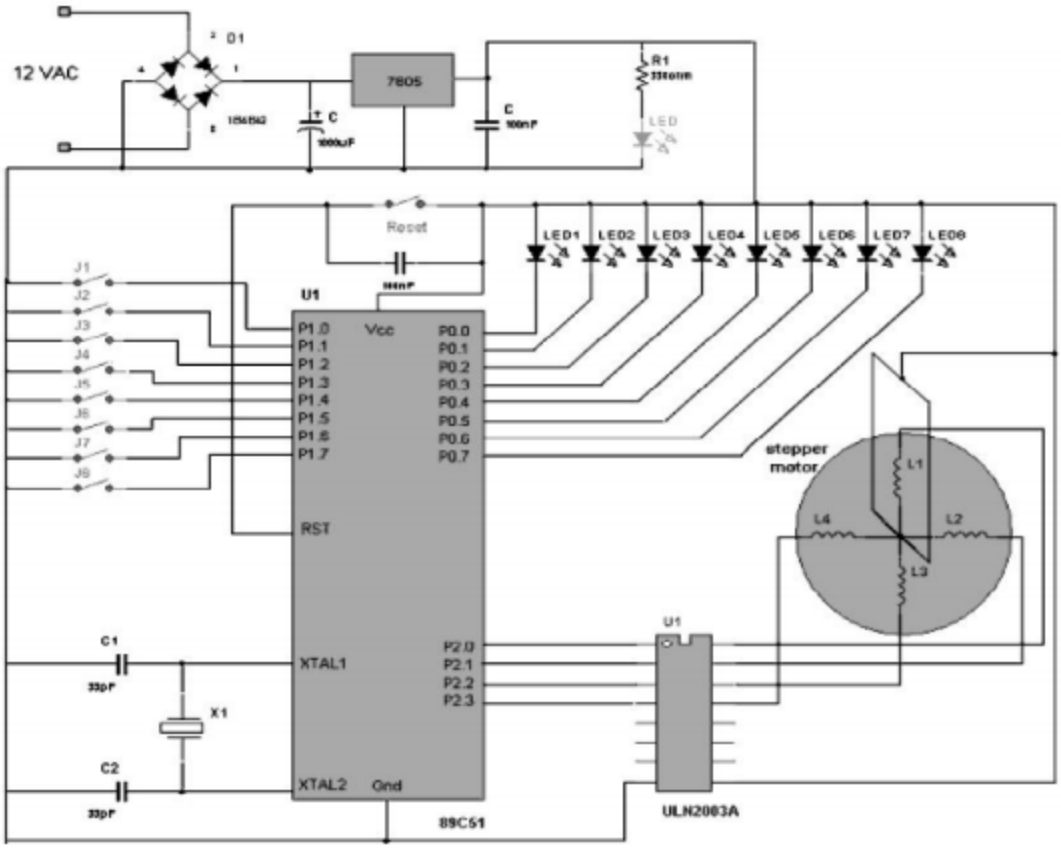
Table 5.7 Pulses for stepper motor module

Single coil excitation		Double coil excitation		Half step excitation	
Clockwise	Anticlockwise	Clockwise	Anticlockwise	Clockwise	Anticlockwise
L4 L3 L2 L1	L4 L3 L2 L1	L4 L3 L2 L1	L4 L3 L2 L1	L4 L3 L2 L1	L4 L3 L2 L1
0 0 0 1	0 0 0 1	0 0 1 1	0 0 1 1	0001	0001
0 0 1 0	1 0 0 0	0 1 1 0	1 0 0 1	0011	0011
0 1 0 0	0 1 0 0	1 1 0 0	1 1 0 0	0010	1000
1 0 0 0	0 0 1 0	1 0 0 1	0 1 1 0	0110	1001
				0100	0100
				1100	1100
				1000	0010
				1001	0110

The circuit consists of very few components. The major components are 7805, 89C51 and ULN2003A.

Connections:-

1. The transformer terminals are given to bridge rectifier to generate rectified DC.
2. It is filtered and given to regulator IC 7805 to generate 5 V pure DC. LED indicates supply is ON.
3. All the push button micro switches J1 to J8 are connected with port P1 as shown to form serial keyboard.
4. 12 MHz crystal is connected to oscillator terminals of 89C51 with two biasing capacitors.
5. All the LEDs are connected to port P0 as shown
6. Port P2 drives stepper motor through current driver chip ULN2003A.
7. The common terminal of motor is connected to Vcc and rest all four terminals are connected to port P2 pins in sequence through ULN chip.



Stepper motor control board